



INSTITUTO POLITECNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

REDES NEURONALES ALFA-BETA SIN PESOS: TEORÍA Y
FACTIBILIDAD DE IMPLEMENTACIÓN

T E S I S

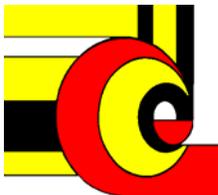
QUE PARA OBTENER EL GRADO DE
DOCTOR EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

AMADEO JOSÉ ARGÜELLES CRUZ

DIRECTORES DE TESIS:

DR. CORNELIO YÁÑEZ MÁRQUEZ
DR. MIGUEL LINDIG BOS



MÉXICO, D.F.

DICIEMBRE DE 2007



INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

SIP-14

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D. F. siendo las 13:06 horas del día 7 del mes de Diciembre de 2007 se reunieron los miembros de la Comisión Revisora de Tesis designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis de grado titulada:

“REDES NEURONALES ALFA-BETA SIN PESOS: TEORÍA Y FACTIBILIDAD DE IMPLEMENTACIÓN”

Presentada por el alumno:

ARGÜELLES

Apellido paterno

CRUZ

materno

AMADEO JOSÉ

nombre(s)

Con registro:

B	0	3	1	2	3	2
---	---	---	---	---	---	---

aspirante al grado de: **DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN**

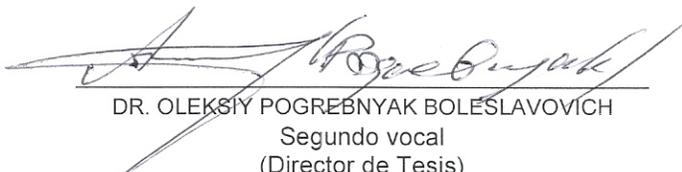
Después de intercambiar opiniones los miembros de la Comisión manifestaron **SU APROBACIÓN DE LA TESIS**, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

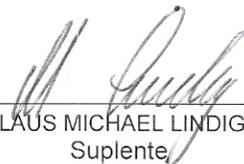
Presidente

Secretario

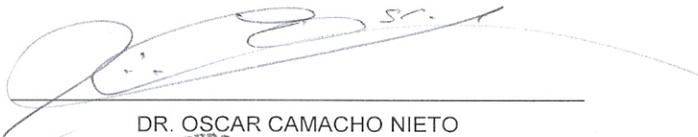

 DR. SÉRGIO SUÁREZ GUERRA
 Primer vocal
 (Director de Tesis)


 DR. OLEKSIY POGREBNYAK BOLESLAVOVICH
 Segundo vocal
 (Director de Tesis)


 DR. CORNELIO YÁÑEZ MÁRQUEZ
 Tercer vocal


 DR. KLAUS MICHAEL LINDIG BOS
 Suplente


 DR. LUIS PÁSTOR SÁNCHEZ FERNÁNDEZ


 DR. OSCAR CAMACHO NIETO

EL PRESIDENTE DEL COLEGIO




 DR. JAIME ÁLVAREZ GALLEGOS
 INSTITUTO POLITÉCNICO NACIONAL
 SECRETARÍA DE INVESTIGACIÓN
 EN COMPUTACIÓN
 DIRECCIÓN



INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México D.F. el día 10 del mes de diciembre del año 2007, el (la) que suscribe Amadeo José Argüelles Cruz alumno (a) del Programa de Doctorado en Ciencias de la Computación con número de registro B031232, adscrito a Centro de Investigación en Computación, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de Dr. Cornelio Yáñez Márquez y Dr. Miguel Lindig Bos y cede los derechos del trabajo intitulado Redes Neuronales Alfa-Beta sin pesos: Teoría y Factibilidad de Implementación, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección jamadeo@cic.ipn.mx. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Amadeo José Argüelles Cruz

Resumen

En este trabajo de tesis se presenta un nuevo modelo de redes neuronales sin pesos basado en las operaciones conocidas Alfa y Beta, y en tres operaciones originales del autor de la tesis. El nuevo modelo de redes neuronales Alfa-Beta sin pesos se ha denominado CAINN, por sus siglas en inglés: *Computing Artificial Intelligence Neural Network*.

Como primera aportación de la tesis, se definen y ejemplifican las operaciones Alfa Generalizada, Sigma-Alfa y Sigma-Beta. Acto seguido, en términos de las operaciones Alfa, Beta, Alfa Generalizada, Sigma-Alfa y Sigma-Beta se crean y diseñan los algoritmos de aprendizaje y recuperación de patrones de CAINN, tomando como base los algoritmos de aprendizaje y recuperación de la red neuronal sin pesos denominada ADAM. Esta es la contribución central.

Un aspecto que se ha considerado importante en este trabajo de tesis, por las ventajas de rapidez de operación, es la incorporación del hardware. En este sentido, se diseñan e implementan en FPGAs las tres operaciones: Alfa, Beta y Alfa Generalizada y, con base en ello, se diseña e implementa una arquitectura hardware para las fases de aprendizaje y recuperación de patrones de CAINN.

Los aspectos formal y experimental están presentes de manera importante en el contenido de esta tesis. Por el lado formal, con base en la teoría de los circuitos booleanos y de la clase de complejidad NC, se establecen formalmente y se sustentan teóricamente las condiciones necesarias y suficientes de equivalencia entre las redes neuronales sin pesos y los circuitos booleanos; se realiza, además, un estudio completo de factibilidad de implementación de CAINN.

El aspecto experimental se cubre ampliamente dado que se diseñan y realizan experimentos de aplicación de CAINN en bases de datos conocidas, cuyos resultados se reportan en el capítulo de experimentos. Adicionalmente, se llevan a cabo y se reportan, estudios comparativos del rendimiento de CAINN respecto de ADAM y otros modelos, cuyos resultados exhiben la superioridad de CAINN respecto de ADAM, su contraparte como red neuronal sin pesos, y sobre otros modelos inmersos en el estado del arte de las redes neuronales o las memorias asociativas, sin olvidar la presencia de los efectos del Teorema *No free lunch*.

Con el desarrollo de este trabajo de tesis se muestra claramente la posibilidad real de que en México se ideen, diseñen e implementen nuevas formas de atacar exitosamente problemas importantes en diversos ámbitos del quehacer humano. Se ilustra, de manera específica, uno de los resultados relevantes que ha obtenido el Grupo Alfa-Beta, a través del trabajo creativo de uno e sus miembros, el autor de esta tesis. Asimismo, se proporciona nuevo material original relacionado con los modelos asociativos Alfa-Beta a los grupos de investigación que desarrollan proyectos relacionados con las ciencias de la computación coadyuvando, con ello, al logro de la máxima del IPN: “La Técnica al Servicio de la Patria”.

Abstract

In this thesis a new weightless neural network model is presented, based on the known operations Alpha and Beta, and three original operations proposed by the author of this thesis. The new model of weightless Alpha-Beta neural network has been called CAINN – Computing Artificial Intelligence Neural Network.

As a first contribution of the current thesis, Generalized Alpha, Sigma-Alpha, and Sigma-Beta operations are defined and exemplified. Then, CAINN's pattern learning and recalling algorithms are created and designed in terms of Generalized Alpha, Sigma-Alpha, and Sigma-Beta operations, based on the ADAM weightless neural network model algorithms for learning and recalling of patterns. This is the main contribution.

One aspect that has been considered relevant in this thesis, by the advantage of speed of operation, is the incorporation of hardware. In this sense, the Alpha, Beta and Generalized Alpha operations are designed and implemented in FPGAs and, based on the former, a hardware architecture for the learning and recalling phases of the CAINN model is designed and implemented.

The experimental aspect is fully covered, given that a series of experiments by applying the CAINN model to several known databases were reported in the chapter of experiments. Also, comparative studies about the performance of the CAINN model concerning ADAM model and other models are reported. Results exhibit the superiority of the CAINN model over the ADAM model, its counterpart as a weightless neural network; and over other models immersed in the state of the art of the neural networks or the associative memories; taking into account the *No free lunch* theorem effects.

The development of this thesis clearly shows the feasibility of devising, designing, and implementing new ways to successfully solve, in Mexico, major problems in various fields of human endeavour. It illustrates, specifically, one relevant result which has been obtained by the Alpha-Beta Group, through the creative work by one of its members: the author of this thesis. Also, this thesis provides new original material, related to Alpha-Beta associative models, that can be used by research teams to develop projects dealing with computer sciences, and contributing with IPN's motto "*La Técnica al Servicio de la Patria*" ("Technique at the service of the Nation").

Índice General

Índice de tablas	xi
Índice de figuras	xii
Capítulo 1: Introducción	1
1.1 Antecedentes	1
1.2 Justificación	3
1.3 Hipótesis	4
1.4 Objetivos	4
1.5 Contribuciones	5
1.6 Organización del documento	5
Capítulo 2: Estado del arte	6
2.1 Redes neuronales sin pesos	6
2.1.1 Nodo neuronal basado en RAM	7
2.1.2 Red neuronal WISARD	8
2.1.3 Red neuronal ADAM	9
2.1.4 Red neuronal PLN	9
2.1.5 MPLN y <i>p</i> RAM	11
2.2 Memorias asociativas	11
2.2.1 Conceptos básicos	12
2.2.2 Lernmatrix de Steinbuch	14
2.2.3 Correlograph de Willshaw, Buneman y Longuet-Higgins	15
2.2.4 Linear Associator de Anderson-Kohonen	16
2.2.5 La memoria asociativa Hopfield	16
2.2.6 Memorias Asociativas Morfológicas	18
2.2.7 Memorias Asociativas Alfa-Beta	20
2.3 Lógica reconfigurable	20
2.3.1 Tecnología de fusible	21
2.3.2 Tecnología de anti-fusible	22
2.3.3 Tecnología PROM	22
2.3.4 Tecnología basada en EPROM	23
2.3.5 Tecnología FLASH	23
2.3.6 Tecnología basada en SRAM	24

Capítulo 3: Materiales y métodos 25

3.1 El modelo ADAM	25
3.1.1 Fase de aprendizaje	28
3.1.2 Fase de recuperación	29
3.2 Memorias Asociativas Alfa-Beta	30
3.2.1 Operaciones binarias α y β	30
3.2.2 Operaciones matriciales Alfa-Beta	31
3.2.3 Memorias Heteroasociativas Alfa-Beta	32
3.2.4 Memorias Autoasociativas Alfa-Beta	33
3.3 Empleo de FPGAs en redes neuronales	34
3.4 Teoría de los circuitos Booleanos	35

Capítulo 4: Modelo propuesto 37

4.1 Tres operaciones originales	38
4.1.1 Operación α_g	38
4.1.2 Operaciones σ_α y σ_β	38
4.2 Algoritmo de la nueva red neuronal Alfa-Beta sin pesos	39
4.2.1 Fase de aprendizaje de CAINN	39
4.2.2 Fase de recuperación de CAINN	40
4.3 Implementación en hardware del modelo CAINN	57
4.3.1 Operaciones Alfa y Beta	57
4.3.2 Arquitectura propuesta	60
4.4 Factibilidad de implementación	62
4.4.1 Recursos presentes en una red neuronal Alfa-Beta sin pesos	64
4.4.2 Familias de redes neuronales uniformes Alfa-Beta sin pesos	66
4.4.3 La clase de Complejidad NC	67

Capítulo 5: Resultados 69

5.1 Bases de datos y equipo utilizado	69
5.1.1 Iris Plants Database	69
5.1.2 Contraceptive Method Choice Database (CMC)	70
5.1.3 Equipo de cómputo utilizado	70
5.2 Recuperación de patrones	70
5.2.1 Recuperación del conjunto fundamental completo	70
5.2.2 Recuperación por particiones del conjunto fundamental	73
5.3 Clasificación de patrones	74
5.3.1 Clasificación del conjunto fundamental completo	74
5.3.2 Clasificación por particiones del conjunto fundamental	76

Capítulo 6: Conclusiones y trabajo futuro	79
6.1 Conclusiones	79
6.2 Trabajo futuro	80
Apéndice A - Simbología	82
Apéndice B – Virtex-II Pro and Virtex-II Pro X Platform FPGAs	84
Apéndice C – Spartan-3 FPGA Family	89
Referencias	94
Trabajos publicados	103

Índice de tablas

<i>Tabla</i>	<i>Descripción</i>	<i>Página</i>
3.1	<i>Operación binaria $\alpha : A \times A \rightarrow B$</i>	31
3.2	<i>Operación binaria $\beta : B \times A \rightarrow A$</i>	31
4.1	<i>Operación binaria $\alpha : A \times A \rightarrow B$</i>	58
4.2	<i>Operación binaria $\beta : B \times A \rightarrow A$</i>	59
5.1	<i>Características del primer conjunto de datos usado.</i>	69
5.2	<i>Características del segundo conjunto de datos usado</i>	70
5.3	<i>Número de intentos de recuperación</i>	71
5.4	<i>Índice de Recuperación (conjunto completo)</i>	71
5.5	<i>Índice de Recuperación (por particiones 70 / 30)</i>	73
5.6	<i>Índice de Clasificación</i>	75
5.7	<i>Índice de Clasificación (por particiones 70 / 30)</i>	76
5.8	<i>Comparación de Índices de Clasificación</i>	77

Índice de figuras

<i>Figura</i>	<i>Descripción</i>	<i>Página</i>
2.1	<i>Nodo RAM</i>	7
2.2	<i>Discriminador basado en memoria RAM.</i>	8
2.3	<i>Red neuronal PLN.</i>	10
2.4	<i>Nodo pRAM con entradas binarias y el proceso de cómputo a la salida.</i>	11
2.5	<i>Memoria asociativa.</i>	12
2.6	<i>Programación de una función sencilla.</i>	20
2.7	<i>Selección de fusibles.</i>	21
2.8	<i>Programación de tecnología anti-fusible.</i>	22
2.9	<i>Tecnología basada en celdas programables de solo lectura (PROM). Configuración de celdas lógicas. (a) Control por compuerta de paso, (b) Multiplexor controlado por bits de estado.</i>	23
3.1	<i>Arquitectura del modelo de red neuronal ADAM.</i>	26
3.2	<i>Etapa de aprendizaje de la red ADAM.</i>	26
3.3	<i>Primera etapa de recuperación.</i>	27
3.4	<i>Segunda etapa de recuperación.</i>	27
3.5	<i>Arquitectura General de un FPGA.</i>	34
4.1	<i>Tricotomía que sustenta a las redes neuronales Alfa-Beta sin pesos.</i>	
4.2	<i>Bloque Alfa.</i>	58
4.3	<i>Implementación de la Operación Alfa a nivel de compuertas lógicas.</i>	58
4.4	<i>Bloque Beta.</i>	59
4.5	<i>Implementación de la Operación Beta a nivel de compuertas lógicas.</i>	59
4.6	<i>Arquitectura Propuesta para la Fase de Aprendizaje.</i>	61
4.7	<i>Arquitectura Propuesta para la Fase de Recuperación.</i>	62
4.8	<i>Ejemplo de una red neuronal Alfa-Beta sin pesos κ codificada.</i>	64
4.9	<i>Arquitectura de una red neuronal Alfa-Beta sin pesos de tamaño $tmo(\kappa)$.</i>	65
5.1	<i>Índice de recuperación para el primer conjunto de datos.</i>	72
5.2	<i>Índice de recuperación para el segundo conjunto de datos.</i>	72
5.3	<i>Índice de recuperación para el primer conjunto de datos. (70 / 30)</i>	73
5.4	<i>Índice de recuperación para el segundo conjunto de datos. (70 / 30)</i>	74
5.5	<i>Índice de clasificación para el primer conjunto de datos.</i>	75
5.6	<i>Índice de clasificación para el segundo conjunto de datos.</i>	75
5.7	<i>Índice de clasificación para el primer conjunto de datos. (70 / 30)</i>	76
5.8	<i>Índice de clasificación para el segundo conjunto de datos. (70 / 30)</i>	77
5.9	<i>Comparación del Índice de Clasificación con otros clasificadores</i>	78

CAPÍTULO 1

Introducción

En este trabajo de tesis doctoral se crea, diseña e implementa un nuevo modelo de redes neuronales sin pesos (redes neuronales Alfa-Beta sin pesos), aprovechando el formalismo de las memorias asociativas Alfa-Beta. Adicionalmente, se presenta un estudio de factibilidad de implementación del nuevo modelo, y se ilustran, mediante ejemplos de aplicación, su operación, rendimiento, eficacia y eficiencia.

1.1 Antecedentes

Las redes neuronales artificiales (RNA) son modelos computacionales inspirados por el intento del ser humano de crear máquinas que simulen los procesos de cómputo, que tienen lugar en las redes neuronales biológicas [1]. El procesamiento que se genera en 10^{12} neuronas y 10^{17} interconexiones sinápticas presentes en el cerebro es principalmente paralelo y de forma distribuida, y el cerebro adapta durante su vida los patrones que provienen del mundo exterior y los asimila para su uso posterior [2].

Las RNA poseen características interesantes como son la adaptabilidad, la habilidad para aprender reglas mediante el empleo de datos de entrenamiento y, con base en la información asimilada previamente, la capacidad de responder a nuevos patrones de entrada [3]. Los trabajos pioneros de McCulloch y Pitts hicieron posible la creación del primer modelo de neurona artificial [4], el cual evolucionó de manera sorprendente hasta dar lugar a los sofisticados modelos de RNA que se conocen hoy en día.

Las RNA tuvieron uno de sus momentos más fructíferos con el trabajo de Rosenblatt, quien inventó la primera máquina de aprendizaje automático, el *perceptron* [5], modelo que dio lugar a la creación de ADALINE [84] y posteriormente a las RNA tipo *backpropagation* [85]. Por otro lado, científicos como Anderson y Kohonen llevaron el tema de las memorias asociativas al mundo de la investigación de punta [6, 55, 56], pero el mérito de haber relacionado íntimamente las redes neuronales y las memorias asociativas, se debe a John Hopfield, quien en 1982 presentó un importante modelo que funciona como red neuronal y memoria asociativa [7]. A mediados de la década de los 90, Ritter *et. al.* crearon las memorias asociativas morfológicas [8], mediante la incorporación de los conceptos y operaciones básicas de la morfología matemática a la teoría de las RNA.

El propósito fundamental de una memoria asociativa es recuperar correctamente patrones completos a partir de patrones de entrada, los cuales pueden estar alterados [17]; en su operación, previamente a la fase de recuperación de patrones, se requiere de la fase de aprendizaje, que es el proceso mediante el cual se crea la memoria asociativa a través de la formación de asociaciones de patrones, las cuales son parejas de patrones, uno de entrada y

uno de salida. Si en cada asociación sucede que el patrón de entrada es igual al de salida, la memoria es autoasociativa; en caso contrario, la memoria es heteroasociativa [46, 47].

Dentro del estado del arte en el área de memorias asociativas, se encuentra el modelo de memoria asociativa Alfa-Beta, que fue desarrollado en el CIC-IPN en el año 2002 [18]; el modelo Alfa-Beta fundamenta teóricamente una buena parte de este trabajo de tesis. En diversas investigaciones recientes, se ha mostrado que este modelo es sumamente sencillo, eficaz y eficiente al momento de abordar los problemas en los que se aplica, incluidos los problemas de clasificación o reconocimiento de patrones [19-24]; se basa en dos operadores originales, Alfa y Beta, mismos que le dan nombre. El operador Alfa es básico en la fase de aprendizaje, mientras que el operador Beta se utiliza durante la fase de recuperación de patrones.

Históricamente, otra vertiente en el desarrollo de las RNA se nutrió del proceso de reconocimiento de patrones denominado método n-tupla binario, el cual fue descrito en los reportes de investigación presentados por Bledsoe y Browning en 1959 [9]. Dicho método fue retomado posteriormente por investigadores como Aleksander [32, 35, 42-44], Morciniec y Austin [10-12], quienes utilizaron memorias de acceso aleatorio (RAM) como elementos básicos para implementar el método n-tupla, el cual se ha aplicado en diversos problemas de interés, con énfasis en problemas de reconocimiento tridimensional de patrones [12, 13]. En la literatura científica actual, al método n-tupla también se le conoce como “redes neuronales basadas en RAM” (*RAM-Based Neural Networks* [11, 29]) o “redes neuronales sin pesos” (*Weightless Neural Networks* [12, 14, 36, 37, 41]).

Las redes neuronales sin pesos (WNN), modelo que sirve como uno de los pilares conceptuales del presente trabajo de tesis, tienen como base de funcionamiento el hecho de que el proceso de aprendizaje se genera a partir de la construcción de un conjunto de funciones lógicas, las cuales son capaces de describir el problema que se desea resolver. Con este conjunto de funciones lógicas se validarán aquellas funciones que corresponden a las clases que representan, y se evaluarán como falsas todas las funciones restantes al momento de ejecutarse la fase de recuperación [11, 15, 16, 29, 36].

La importancia del surgimiento, desarrollo y aplicaciones de las WNN está fundamentada en que posee estrechas ligas conceptuales y pragmáticas con el papel que han jugado las RNA, tanto en la creación de ambientes propicios para generar nuevas y útiles teorías, como para propiciar aplicaciones en diversas actividades del ser humano [10]. Las WNN definen un importante paradigma en el campo de las RNA, dado que proporcionan flexibilidad, facilidad en la implementación de los nodos y, por ende, algoritmos de aprendizaje muy rápidos [36].

Las ventajas de las WNN sobre las RNA con pesos han sido mostradas por una gran cantidad de equipos de investigación científica internacionales. En lo que respecta a los algoritmos de aprendizaje, las WNN son capaces de soportar algoritmos *one-shot* [89], a diferencia de las RNA cuyos algoritmos son iterativos; por otro lado, las WNN son fácilmente adaptables como sistemas direccionables por contenido [90], lo cual no sucede con las redes con pesos. Una de las facetas más atractivas de las WNN, que en parte ha motivado la investigación en este trabajo de tesis, es la facilidad con la que se pueden

implementar en hardware las redes neuronales sin pesos [91]. La muy alta velocidad de las WNN se debe a la independencia mutua entre los nodos al momento del cambio en las entradas, característica que se aprovecha en esta tesis para proponer el nuevo modelo, aunado a la premisa de que los operadores Alfa y Beta reforzarán el rendimiento.

Las RNA, las redes neuronales sin pesos y las memorias asociativas se han aplicado de manera exitosa en diversas áreas, entre las cuales se encuentran: procesamiento de señales [3, 45], análisis de imágenes [10, 14], detección y predicción de contaminantes atmosféricos [86, 87] e igualación industrial de colores [88], entre otras. Un gran número de modelos de RNA, redes neuronales sin pesos y de memorias asociativas han sido implementadas en software que se ejecuta en máquinas secuenciales, pero en la actualidad es posible generar procesos que involucren paralelismo como el que las neuronas biológicas implícitamente utilizan para su operación mediante el empleo de dispositivos reconfigurables, cuya alta densidad de circuitos y líneas de interconexiones hacen posible colocar redes neuronales densas en hardware [25-28].

La evolución de las WNN, desde el nodo RAM elemental hasta los complejos esquemas de aprendizaje ideados para el diseño y operación de los modelos recientes, ha evidenciado la relevancia que adquirida por este tipo de modelos en el quehacer intelectual y tecnológico en los ambientes científicos e ingenieriles a nivel mundial [92-95].

Los retos que representan, a nivel mundial, algunas aplicaciones específicas de alta especialización, colocan a las WNN en el foco de atención de los equipos de investigación que realizan desarrollos científicos y tecnológicos de punta, en el afán de resolver problemas que enfrenta el ser humano, de cuyas áreas de acción se ilustra con una pequeña muestra: filtrado no lineal [96], representación semántica [97], procesamiento de datos tomográficos [98], reconocimiento automático de rostros [99], monitoreo inteligente de estaciones de ensamblaje industrial [100], estimación automática de solvencia financiera de clientes [101], autenticación de personas usando biométricos [102], caracterización y clasificación de proteínas y otras biomoléculas [103], entre muchos otros campos donde se presentan interesantes problemas de actualidad, cuya solución precisa de la generación de nuevos modelos que exhiban rendimientos y prestaciones que generen expectativas de solución, a la manera del modelo presentado en este trabajo de tesis.

1.2 Justificación

En la sección anterior se ha evidenciado la importancia de las WNN y las ventajas que ofrecen estos modelos sobre las RNA con pesos, y se han fundamentado estos hechos con referencias bibliográficas correspondientes a trabajos de investigación científica de punta realizada por equipos internacionales. Por otro lado, se ha enfatizado la existencia, la importancia y la aplicación de los modelos asociativos Alfa-Beta, cuyas características más importantes son similares a las que exhiben las WNN, en cuanto a eficacia, eficiencia y facilidad de implementación.

Ciertamente algunos connotados autores en el área de las redes neuronales sin pesos han creado modelos relacionados con memorias asociativas [6, 8, 19, 30, 56, 79]; otros, han

incursionado en los modelos asociativos Alfa-Beta [19-24, 70], o bien, han realizado implementaciones en hardware de algunos modelos conocidos de redes neuronales sin pesos [25-29, 41, 60, 67, 72, 74-77, 91]. Sin embargo, hasta noviembre de 2007, no se han reportado en la literatura trabajos científicos donde se relacionen específicamente las redes neuronales sin pesos con los modelos de memorias asociativas Alfa-Beta.

El presente trabajo de tesis queda justificado porque incluye, de manera central, la creación y diseño de un nuevo modelo de redes neuronales sin pesos, basado en los operadores Alfa y Beta, los cuales son tomados de los modelos de memorias asociativas Alfa-Beta. Por otro lado, se establecen las condiciones necesarias y suficientes de equivalencia de las redes neuronales sin pesos y los circuitos booleanos, y se presenta un estudio completo de factibilidad de implementación en hardware del nuevo modelo, el cual exhibe un alto desempeño, derivado del hecho de que es un hecho ya probado que la densidad aritmética de los modelos de memorias asociativas Alfa y Beta es menor que la otros modelos relevantes de memorias asociativas, como las morfológicas [18].

1.3 Hipótesis

La reflexión previa da pie al surgimiento de la hipótesis de este trabajo de tesis: es posible crear, diseñar e implementar redes neuronales sin pesos, cuyos algoritmos de aprendizaje y recuperación de patrones estén basados en las operaciones binarias Alfa y Beta, lo cual redunde en la obtención de un nuevo modelo de WNN que conjugue las ventajas de ambos campos: las WNN y los modelos asociativos Alfa-Beta. El nuevo modelo de **redes neuronales Alfa-Beta sin pesos** exhibirá un rendimiento competitivo con los modelos actuales de redes neuronales, con pesos o sin pesos, y de memorias asociativas.

1.4 Objetivos

Objetivo General:

Crear, diseñar e implementar un nuevo modelo de redes neuronales sin pesos basado en los algoritmos conocidos de WNN, específicamente de ADAM, y en los operadores Alfa y Beta, los cuales son tomados de los modelos de memorias asociativas Alfa-Beta. El nuevo modelo de redes neuronales Alfa-Beta sin pesos se ha denominado CAINN (*Computing Artificial Intelligence Neural Network*).

Objetivos Específicos:

1. Definir y ejemplificar las operaciones Alfa Generalizada, Sigma-Alfa y Sigma-Beta
2. Tomando como base los algoritmos de aprendizaje y recuperación de la red neuronal sin pesos denominada ADAM, crear y diseñar los algoritmos de aprendizaje y recuperación de patrones de CAINN, en términos de las operaciones Alfa, Beta, Alfa Generalizada, Sigma-Alfa y Sigma-Beta
3. Diseñar e implementar en FPGAs las operaciones Alfa, Beta, Alfa Generalizada, Sigma-Alfa y Sigma-Beta

4. Con base en los resultados del objetivo particular 3, diseñar e implementar una arquitectura hardware para las fases de aprendizaje y recuperación de patrones de CAINN
5. Establecer formalmente las condiciones necesarias y suficientes de equivalencia entre las redes neuronales sin pesos y los circuitos booleanos
6. Realizar un estudio completo de factibilidad de implementación de CAINN
7. Realizar experimentos de aplicación de CAINN en bases de datos conocidas
8. Realizar estudios comparativos del rendimiento de CAINN respecto de ADAM y otros modelos

1.5 Contribuciones

Las contribuciones de este trabajo de tesis son:

- La creación de un nuevo modelo de redes neuronales sin pesos (redes neuronales Alfa-Beta sin pesos), basado en los operadores Alfa y Beta, los cuales son tomados de los modelos de memorias asociativas Alfa-Beta, y de una nueva operación definida específicamente para el desarrollo de los algoritmos de esta tesis, que ha sido llamada Alfa Generalizada.
- El desarrollo de un estudio completo de factibilidad de implementación del nuevo modelo.
- La sustentación teórica de las condiciones necesarias y suficientes de la equivalencia de las redes neuronales sin pesos y los circuitos booleanos.

1.6 Organización del documento

El resto del documento de tesis está organizado de la siguiente manera: en el capítulo 2 se presenta el nodo RAM y los modelos más importantes de redes neuronales sin pesos y de memorias asociativas; además de algunos tópicos históricos y conceptuales relacionados con la tecnología en que se basa el desarrollo de la lógica reconfigurable.

En materiales y métodos, materia del capítulo 3, se incluyen: el modelo ADAM, las memorias asociativas Alfa-Beta, un resumen del empleo de los FPGAs en redes neuronales y la teoría de los circuitos booleanos. Estos materiales y métodos soportan teóricamente el algoritmo de CAINN el cual, aunado a la factibilidad de implementación en hardware, constituye el contenido del capítulo 4.

Los resultados experimentales y las discusiones correspondientes se muestran en el capítulo 5, mientras que en el 6 se presentan las conclusiones y recomendaciones para trabajo futuro. Finalmente, se anexan los apéndices y las referencias bibliográficas.

CAPÍTULO 2

Estado del arte

Este capítulo está constituido por tres secciones. En la sección 2.1 se hace una descripción del nodo RAM y de los modelos más importantes de redes neuronales sin pesos. La sección 2.2, por su parte, trata sobre los principales modelos de memorias asociativas; y finalmente, en la sección 2.3, se incluyen algunos tópicos históricos y conceptuales relacionados con la tecnología en que se basa el desarrollo de la lógica reconfigurable.

2.1 Redes neuronales sin pesos

El interés de los investigadores en el campo de la inteligencia artificial e ingeniería de cómputo en relación con el estudio de las redes neuronales, es desarrollar modelos matemáticos simplificados de las redes neuronales presentes en el cerebro. Mediante su utilización, se ha comprendido cómo resolver problemas computacionales creando herramientas de cómputo que reproducen en cierto modo el comportamiento del cerebro para reconocer patrones, o la construcción de máquinas con inteligencia similar a la de organismos biológicos [1-3].

Los modelos de cómputo de las redes neuronales artificiales sin pesos (WNNs) comparten la característica de presentar entradas y salidas binarias. Las funciones que realiza cada neurona están contenidas en tablas de búsqueda conformadas por memorias de acceso aleatorio (RAM) [32].

Generalmente, las redes neuronales sin pesos llevan a cabo el aprendizaje mediante la modificación de los valores almacenados en las tablas de búsqueda. Se han empleado en técnicas de aprendizaje supervisado (reconocimiento de imágenes y análisis de oclusión) y no supervisado (reconocimiento de gramáticas regulares, imágenes en dos tonos, generación de mapas topológicos, reconocimiento óptico de caracteres, entre otros) [10-16, 35-41].

Las redes neuronales sin pesos no presentan pesos sinápticos adaptables asociados a sus conexiones, y esto hace que no se requiera del ajuste de pesos, lo que realza su flexibilidad y velocidad en el aprendizaje de algoritmos. Su origen proviene de los trabajos de Bledsoe y Browning [9] a finales de la década de los 50, quienes crearon el clasificador n-tupla. Este clasificador se concibe como una memoria distribuida que almacena información, de la cual se derivan subpatrones, los cuales están relacionados con determinadas clases. Al clasificar nuevos patrones, la clase de salida se define como aquella cuyos ejemplos de aprendizaje son más comunes con el patrón presentado [34].

Aleksander y Stonham emplearon las máquinas de muestreo n-tuplas para su aplicación en redes de aprendizaje adaptivas, sugiriendo circuitos lógicos universales como nodos de la red de aprendizaje [32]; también emplearon los nodos RAM y el Microcircuito Adaptivo Lógico de Almacenamiento (SLAM), diseñados para propósitos de investigación antes de que estuvieran disponibles las memorias de circuitos integrados.

La aparición de memorias RAM de mayor capacidad y bajo costo, propició que se construyeran redes más grandes al terminar la década de los 70. El diseño y construcción de la máquina WISARD (Wilkie, Stonham & Alexander Recognition Device) se convirtió en una realidad, fue un hecho, y en 1981 se logró obtener el primer prototipo. En 1984 se presenta la versión comercial del dispositivo de reconocimiento de patrones WISARD [35].

Posteriormente, otros modelos de redes neuronales sin pesos (tales como PLN, Prat, GSN y GRAM) fueron desarrollados. A continuación se da una mención breve de algunos modelos, que son relevantes para el presente trabajo de tesis.

2.1.1 Nodo Neuronal basado en RAM

De acuerdo con [36], la figura 2.1 muestra un nodo RAM con N entradas, el cual contiene 2^N localidades de memoria y un número N de líneas de dirección $E = \{E_1, E_2, \dots, E_N\}$ para la identificación de las localidades de almacenamiento dentro de la neurona. Una señal binaria presente en las líneas de dirección tendrá acceso sólo a una de las localidades del nodo neuronal. El bit $C[E]$, almacenado durante la fase de entrenamiento, se envía al exterior activando la terminal EA y empleando la terminal S durante la fase de recuperación. Esto indica que la función booleana que genera la neurona se determina por el contenido de la RAM.

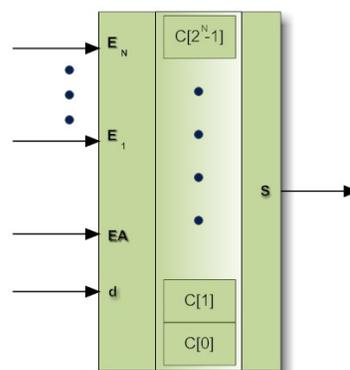


Figura 2.1 Nodo RAM, donde se aprecian las líneas de direcciones E_N , las localidades de memoria $C[E]$, la estrategia de aprendizaje EA (entrenamiento/recuperación), la conectividad N (no. de entradas), la entrada de datos d y la salida de datos S .

El nodo RAM aprende un patrón al presentarlo en la entrada de datos (d), el cual se deposita en la localidad de memoria $C[E]$. Sin embargo, el nodo RAM no puede realizar generalización alguna por sí mismo (se entiende la generalización como la habilidad de proporcionar una salida correcta para patrones no presentados durante la fase de entrenamiento), y sólo produce una salida correcta únicamente para aquellos patrones almacenados en la fase de entrenamiento.

En una red neuronal RAM típica (con arquitectura de una sola capa), cada RAM aprende a responder con un valor de 1 para aquellos patrones presentes en el conjunto de aprendizaje. Al presentarse un patrón no contenido en la red neuronal, se clasifica en la misma clase del conjunto de aprendizaje si todas las salidas en las RAM dan 1. Esta arquitectura divide el conjunto de todos los patrones posibles en aquellos que sí se encuentran en la generalización y aquellos que no [36].

2.1.2 Red Neuronal WISARD

Esta máquina de reconocimiento de patrones de propósito general, desarrollada por Aleksander y Stonham [35], se construyó considerando como elemento básico al discriminador RAM de bit direccionable. Este discriminador, como se muestra en la figura 2.2, se conforma de una capa de k RAMs de N entradas, donde cada RAM almacena 2^N palabras de 1 bit. Cada una de las k RAMs se direcciona utilizando N -tuplas, seleccionadas de forma aleatoria al emplear un vector de valores binarios presentes en la entrada a la red. Esto se conoce como mapeo de entrada o patrón de conectividad. Aunque el mapeo de entrada se elige al azar, tal mapeo es un parámetro fijo de la red. En la salida del discriminador se tiene conectado un sumador que recibe las salidas de cada RAM, y produce lo que se denomina la respuesta del discriminador r . Previo a la fase de aprendizaje, todas las $k2^N$ palabras de 1 bit se colocan en cero.

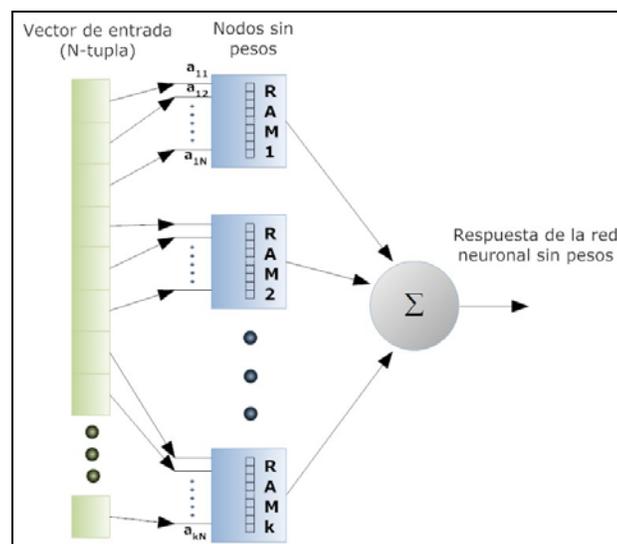


Figura 2.2 Discriminador basado en memoria RAM

El aprendizaje se realiza cuando un patrón \mathbf{v} de entrada kN se presenta al discriminador, y coloca en 1 todas las palabras a las que \mathbf{v} tuvo acceso en todos los \mathbf{k} nodos. Si se presenta a \mathbf{v} un tiempo después, las localidades que se escribieron previamente se vuelven a acceder y cada nodo responde con un 1, dando un máximo ($r = k$) como respuesta del discriminador. En el caso de presentarse una versión ruidosa \mathbf{u} de \mathbf{v} , \mathbf{r} es una función del número de localidades previamente escritas a las que \mathbf{u} accede, lo cual da una proporción de la similitud entre \mathbf{v} y \mathbf{u} . Así, aunque de manera individual una RAM no generaliza, un discriminador WISARD sí lo hace y es capaz de clasificar patrones no conocidos dado el conocimiento adquirido al entrenarse con el conjunto de patrones presentado en la fase de aprendizaje.

Un sistema WISARD se construye agrupando un conjunto de discriminadores, donde cada uno reconoce una diferente clase de patrones. Durante el entrenamiento de un patrón \mathbf{v} , se escribe solamente en la memoria RAM que corresponde al discriminador D_i . Al concluir el aprendizaje de forma individual para todas las clases, el reconocimiento de un patrón desconocido \mathbf{u} se realiza mediante la presentación concurrente de \mathbf{u} en todos los discriminadores y revisando cada respuesta \mathbf{r} . El patrón se asigna a la clase contenida en el discriminador que presenta la respuesta \mathbf{r} más alta.

La respuesta que el método N-tupla ofrece respecto del rendimiento depende principalmente de la elección del tamaño del N-tupla [32, 37-39]. Resultados experimentales muestran que a mayor N la respuesta es mejor [37, 40].

Otro elemento a considerar es el proceso de aprendizaje. Al momento de presentarse un patrón de entrada, varias localidades de memoria se colocan en 1 en un N-tupla específico. Cuando esto sucede, el patrón produce que $r=k$ en otros discriminadores, lo que origina el problema de saturación [13]. Entonces, para mejorar el desempeño del método N-tupla se propuso el uso de otros algoritmos de aprendizaje [38].

2.1.3 Red neuronal ADAM

El fundamento teórico de la red neuronal ADAM (Advanced Distributed Associative Memory) se presenta en el siguiente capítulo, debido a que este modelo es una de las bases que sustentan este trabajo de tesis.

2.1.4 Red neuronal PLN

En este modelo, el nodo utiliza un número de dos bits alojado en una localidad de memoria. El contenido de las localidades de memoria se convierte en la probabilidad de disparo, es decir cuando se presenta un nivel lógico 1 a la salida del nodo [41]. Considerado como un generador estocástico de salida, esta característica aumenta la versatilidad en cada nodo RAM. La figura 2.3 muestra a manera de bloques cómo está constituida la red neuronal PLN.

Para el caso de las N entradas binarias, una dirección en el nodo PLN corresponde a una de las 2^N localidades. El contenido de la localidad de memoria se transfiere a la función de salida, que se encarga de convertirla en binaria a la salida del nodo. Dicho contenido puede ser 1, 0 ó u , donde u es un estado indefinido que permite que el nodo intercambie su salida entre 0 y 1 con igual probabilidad. El empleo del estado no definido para este tercer valor lógico hace posible el uso de estados no conocidos en la operación de redes neuronales sin pesos. Este valor es el estado que contienen todas las localidades de memoria, previo a realizarse la fase de aprendizaje, creando un estado de “ignorancia” en la red neuronal. La salida del nodo PLN se describe de la manera siguiente:

$$r = \begin{cases} 0 & \text{si } C[I]=0 \\ 1 & \text{si } C[I]=1 \\ \text{aleatorio}(0,1) & \text{si } C[I]=u \end{cases}$$

$C[I]$ indica el contenido de la dirección asociada con el patrón de entrada I y $\text{aleatorio}(0,1)$ es una función aleatoria que genera ceros y unos con igual probabilidad.

Las redes neuronales con arquitecturas multicapa denominadas *pirámides*, que emplean al nodo PLN, fueron sugeridas por Alexander [42]. Este tipo de estructura presenta un número fijo de neuronas por capa, con un conjunto de conexiones de entrada (fan-in) y una conexión de salida (fan-out). Al utilizar los mismos patrones de entrada, se observó que la funcionalidad presente en las arquitecturas de pirámide es menor que la presentada por un nodo RAM [43].

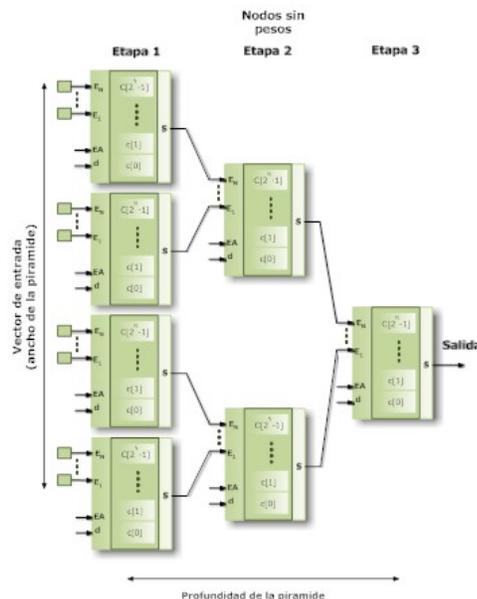


Figura 2.3 Red neuronal PLN

Al iniciar el aprendizaje, las localidades de memoria tienen un valor de u , con lo que se asegura que el comportamiento de la red sea imparcial. El proceso de aprendizaje en las

redes PLN reemplaza los estados indeterminados u por ceros y unos; así, y de manera regular, es como la red produce patrones de salida correctos en respuesta a los patrones de aprendizaje en la entrada.

2.1.5 MPLN y p RAM

Una red PLN de m -estados o MPLN, como el mostrado en la figura 2.4, almacena un extenso rango de valores discretos en cada componente de memoria [44]. La función de salida puede manejar funciones probabilísticas, lineales o sigmoidales. Las localidades del nodo pueden almacenar probabilidades de salida las cuales están clasificadas.

Por ejemplo, una red MPLN puede presentar una salida 1 con un 15% de probabilidad dependiendo de su entrada. Esto hace que en el aprendizaje se permitan cambios incrementales en los valores almacenados.

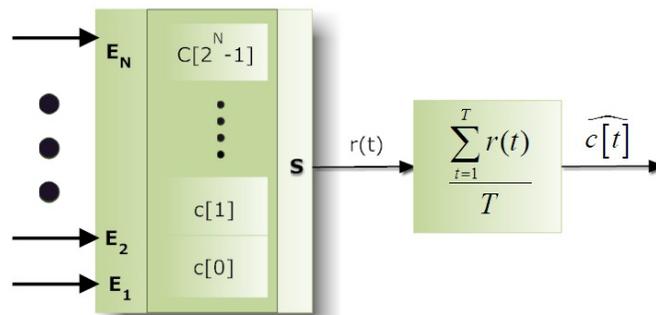


Figura 2.4 Nodo p RAM con entradas binarias y el proceso de cómputo a la salida.

Taylor propuso un modelo de neuronas ruidosas, el cual presenta ecuaciones que incorporan y formalizan muchas de las propiedades presentes en neuronas reales. La evolución de dicho modelo muestra ser equivalente a las redes ruidosas (probabilísticas) RAM o p RAM [45]. En el caso del nodo p RAM, y al igual que en la definición del nodo RAM, presenta 2^N localidades de memoria a las que se accede utilizando un vector \mathbf{a} y el valor presente en la entrada E sólo accede a una localidad. El valor almacenado en $C[E]$ representa la probabilidad de que un pulso de corta duración y de amplitud 1 se produzca en la línea de salida S , dada la entrada E . El contenido de la localidad de memoria proporciona el valor promedio, es decir $y = C[E]$.

2.2 Memorias asociativas

Los conceptos presentados en esta sección se han tomado de las referencias que, a nuestro juicio, son las más representativas [17, 46-49].

2.2.1 Conceptos básicos

Una **Memoria Asociativa** puede formularse como un sistema de entrada y salida, idea que se esquematiza en la figura 2.5. En este esquema, los patrones de entrada y salida están representados por vectores columna denotados por \mathbf{x} y \mathbf{y} , respectivamente. Cada uno de los patrones de entrada forma una asociación con el correspondiente patrón de salida, la cual es similar a una pareja ordenada; por ejemplo, los patrones \mathbf{x} y \mathbf{y} del esquema anterior forman la asociación (\mathbf{x}, \mathbf{y}) .



Figura 2.5 Memoria asociativa.

A continuación se propone una notación que se usará en la descripción de los conceptos básicos sobre memorias asociativas, y en el resto de los capítulos de esta tesis.

Los patrones de entrada y salida se denotarán con las letras negrillas, \mathbf{x} y \mathbf{y} , agregándoles números naturales como superíndices para efectos de discriminación simbólica. Por ejemplo, a un patrón de entrada \mathbf{x}^1 le corresponderá el patrón de salida \mathbf{y}^1 , y ambos formarán la asociación $(\mathbf{x}^1, \mathbf{y}^1)$; del mismo modo, para un número entero positivo k específico, la asociación correspondiente será $(\mathbf{x}^k, \mathbf{y}^k)$.

La memoria asociativa \mathbf{M} se representa mediante una matriz, la cual se genera a partir de un conjunto finito de asociaciones conocidas de antemano: este es el **conjunto fundamental de aprendizaje**, o simplemente **conjunto fundamental**.

El conjunto fundamental se representa de la siguiente manera:

$$\{(x^\mu, y^\mu) \mid \mu = 1, 2, \dots, p\}$$

donde p es un número entero positivo que representa la cardinalidad del conjunto fundamental.

A los patrones que conforman las asociaciones del conjunto fundamental se les llama **patrones fundamentales**. La naturaleza del conjunto fundamental proporciona un importante criterio para clasificar las memorias asociativas:

Una memoria es **Autoasociativa** si se cumple que $\mathbf{x}^\mu = \mathbf{y}^\mu \forall \mu \in \{1, 2, \dots, p\}$, por lo que uno de los requisitos que se debe de cumplir es que $n = m$.

Una memoria **Heteroasociativa** es aquella en donde $\exists \mu \in \{1, 2, \dots, p\}$ para el que se cumple que $\mathbf{x}^\mu \neq \mathbf{y}^\mu$. Nótese que puede haber memorias heteroasociativas con $n = m$.

En los problemas donde intervienen las memorias asociativas, se consideran dos fases importantes: La fase de aprendizaje, que es donde se genera la memoria asociativa a partir de las p asociaciones del conjunto fundamental, y la fase de recuperación que es donde la memoria asociativa opera sobre un patrón de entrada, a la manera del esquema que aparece al inicio de esta sección.

A fin de especificar las componentes de los patrones, se requiere la notación para dos conjuntos a los que llamaremos arbitrariamente A y B . Las componentes de los vectores columna que representan a los patrones, tanto de entrada como de salida, serán elementos del conjunto A , y las entradas de la matriz \mathbf{M} serán elementos del conjunto B .

No hay requisitos previos ni limitaciones respecto de la elección de estos dos conjuntos, por lo que no necesariamente deben ser diferentes o poseer características especiales. Esto significa que el número de posibilidades para escoger A y B es infinito.

Por convención, cada vector columna que representa a un patrón de entrada tendrá n componentes cuyos valores pertenecen al conjunto A , y cada vector columna que representa a un patrón de salida tendrá m componentes cuyos valores pertenecen también al conjunto A ; es decir:

$$x^\mu \in A^n \text{ y } y^\mu \in A^m \forall \mu \in \{1, 2, \dots, p\}$$

La j -ésima componente de un vector columna se indicará con la misma letra del vector, pero sin negrilla, colocando a j como subíndice ($j \in \{1, 2, \dots, n\}$ o $j \in \{1, 2, \dots, m\}$ según corresponda). La j -ésima componente del vector columna \mathbf{x}^μ se representa por: x_j^μ

Con los conceptos básicos ya descritos y con la notación anterior, es posible expresar las dos fases de una memoria asociativa:

1. **Fase de Aprendizaje** (Generación de la memoria asociativa). Encontrar los operadores adecuados y una manera de generar una matriz \mathbf{M} que almacene las p asociaciones del conjunto fundamental $\{(\mathbf{x}^1, \mathbf{y}^1), (\mathbf{x}^2, \mathbf{y}^2), \dots, (\mathbf{x}^p, \mathbf{y}^p)\}$, donde $\mathbf{x}^\mu \in A^n$ y $\mathbf{y}^\mu \in A^m \forall \mu \in \{1, 2, \dots, p\}$. Si $\exists \mu \in \{1, 2, \dots, p\}$ tal que $\mathbf{x}^\mu \neq \mathbf{y}^\mu$, la memoria será heteroasociativa; si $m = n$ y $\mathbf{x}^\mu = \mathbf{y}^\mu \forall \mu \in \{1, 2, \dots, p\}$, la memoria será autoasociativa.
2. **Fase de Recuperación** (Operación de la memoria asociativa). Hallar los operadores adecuados y las condiciones suficientes para obtener el patrón fundamental de salida \mathbf{y}^μ , cuando se opera la memoria \mathbf{M} con el patrón fundamental de entrada \mathbf{x}^μ ,

lo anterior para todos los elementos del conjunto fundamental y para ambos modos: autoasociativo y heteroasociativo.

Se dice que una memoria asociativa \mathbf{M} exhibe **recuperación correcta** si al presentarle como entrada, en la fase de recuperación, un patrón \mathbf{x}^ω con $\omega \in \{1, 2, \dots, p\}$, ésta responde con el correspondiente patrón fundamental de salida \mathbf{y}^ω .

A continuación, en esta sección haremos un breve recorrido por los modelos de memorias asociativas más representativos.

2.2.2 *Lernmatrix* de Steinbuch

Karl Steinbuch fue uno de los primeros investigadores en desarrollar un método para codificar información en arreglos cuadrículados conocidos como *crossbar*. La importancia de la *Lernmatrix* [50, 51] se evidencia en una afirmación que hace Kohonen [52] en su artículo de 1972, donde apunta que las matrices de correlación, base fundamental de su innovador trabajo, vinieron a sustituir a la *Lernmatrix* de Steinbuch.

La *Lernmatrix* es una memoria heteroasociativa que puede funcionar como un clasificador de patrones binarios si se escogen adecuadamente los patrones de salida; es un sistema de entrada y salida que al operar acepta como entrada un patrón binario $\mathbf{x}^\mu \in A^n$, $A = \{0, 1\}$ y produce como salida la clase $\mathbf{y}^\mu \in A^p$ que le corresponde (de entre p clases diferentes), codificada ésta con un método que en la literatura se le ha llamado *one-hot* [53].

La codificación *one-hot* funciona así: para representar la clase $k \in \{1, 2, \dots, p\}$, se asignan a las componentes del vector de salida \mathbf{y}^μ los siguientes valores: $y_k^\mu = 1$ y $y_j^\mu = 0$ para $j = 1, 2, \dots, k-1, k+1, \dots, p$.

Algoritmo de la *Lernmatrix*

Fase de Aprendizaje

Se genera el esquema (*crossbar*) al incorporar la pareja de patrones de entrenamiento $(\mathbf{x}^\mu, \mathbf{y}^\mu) \in A^n \times A^p$. Cada uno de los componentes m_{ij} de \mathbf{M} , la *Lernmatrix* de Steinbuch, tiene valor cero al inicio, y se actualiza de acuerdo con la regla $m_{ij} + \Delta m_{ij}$, donde:

$$\Delta m_{ij} = \begin{cases} +\varepsilon & \text{si } x_j^\mu = 1 = y_i^\mu \\ -\varepsilon & \text{si } x_j^\mu = 0 \text{ y } y_i^\mu = 1 \\ 0 & \text{en otro caso} \end{cases}$$

donde ε una constante positiva escogida previamente: es usual que ε es igual a 1.

Fase de Recuperación

La i -ésima coordenada y_i^ω del vector de clase $\mathbf{y}^\omega \in A^p$ se obtiene como lo indica la siguiente expresión, donde \vee es el operador *máximo*:

$$y_i^\omega = \begin{cases} 1 & \text{si } \sum_{j=1}^n m_{ij} \cdot x_j^\omega = \bigvee_{h=1}^p \left[\sum_{j=1}^n m_{hj} \cdot x_j^\omega \right] \\ 0 & \text{en otro caso} \end{cases}$$

2.2.3 Correlograph de Willshaw, Buneman y Longuet-Higgins

El *correlograph* es un dispositivo óptico elemental capaz de funcionar como una memoria asociativa [54, 56]. En palabras de los autores “el sistema es tan simple, que podría ser construido en cualquier laboratorio escolar de física elemental”.

Algoritmo del Correlograph

Fase de Aprendizaje

La *red asociativa* se genera al incorporar la pareja de patrones de entrenamiento $(\mathbf{x}^\mu, \mathbf{y}^\mu) \in A^n \times A^m$. Cada uno de los componentes m_{ij} de la *red asociativa* \mathbf{M} tiene valor cero al inicio, y se actualiza de acuerdo con la regla:

$$m_{ij} = \begin{cases} 1 & \text{si } y_i^\mu = 1 = x_j^\mu \\ \text{valor anterior} & \text{en otro caso} \end{cases}$$

Fase de Recuperación

Se le presenta a la *red asociativa* \mathbf{M} un vector de entrada $\mathbf{x}^\omega \in A^n$. Se realiza el producto de la matriz \mathbf{M} por el vector \mathbf{x}^ω y se ejecuta una operación de umbralizado, de acuerdo con la siguiente expresión:

$$y_i^\omega = \begin{cases} 1 & \text{si } \sum_{j=1}^n m_{ij} \cdot x_j^\omega \geq u \\ 0 & \text{en otro caso} \end{cases}$$

donde u es el valor de umbral. Una estimación aproximada del valor de umbral u se puede lograr con la ayuda de un número indicador mencionado en el artículo [54] de Willshaw *et al.* de 1969: $\log_2 n$.

2.2.4 *Linear Associator* de Anderson-Kohonen

El *Linear Associator* tiene su origen en los trabajos pioneros de 1972 publicados por Anderson y Kohonen [52, 55, 56].

Para presentar el *Linear Associator* consideremos de nuevo el conjunto fundamental:

$$\{(x^\mu, y^\mu) \mid \mu = 1, 2, \dots, p\} \text{ con } A = \{0, 1\}, x^\mu \in A^n \text{ y } y^\mu \in A^m$$

Algoritmo del *Linear Associator*

Fase de Aprendizaje

- 1) Para cada una de las p asociaciones $(\mathbf{x}^\mu, \mathbf{y}^\mu)$ se encuentra la matriz $\mathbf{y}^\mu \cdot (\mathbf{x}^\mu)^t$ de dimensiones $m \times n$.
- 2) Se suman la p matrices para obtener la memoria

$$\mathbf{M} = \sum_{\mu=1}^p \mathbf{y}^\mu \cdot (\mathbf{x}^\mu)^t = [m_{ij}]_{m \times n}$$

de manera que la ij -ésima componente de la memoria \mathbf{M} se expresa así:

$$m_{ij} = \sum_{\mu=1}^p y_i^\mu x_j^\mu$$

Fase de Recuperación

Esta fase consiste en presentarle a la memoria un patrón de entrada \mathbf{x}^ω , donde $\omega \in \{1, 2, \dots, p\}$ y realizar la operación

$$\mathbf{M} \cdot \mathbf{x}^\omega = \left[\sum_{\mu=1}^p \mathbf{y}^\mu \cdot (\mathbf{x}^\mu)^t \right] \cdot \mathbf{x}^\omega$$

2.2.5 La memoria asociativa Hopfield

El artículo de John J. Hopfield de 1982, publicado por la prestigiosa y respetada *National Academy of Sciences* (en sus *Proceedings*), impactó positivamente y trajo a la palestra internacional su famosa memoria asociativa [7].

En el modelo que originalmente propuso Hopfield, cada neurona x_i tiene dos posibles estados, a la manera de las neuronas de McCulloch-Pitts: $x_i = 0$ y $x_i = 1$; sin embargo, Hopfield observa que, para un nivel dado de exactitud en la recuperación de patrones, la capacidad de almacenamiento de información de la memoria se puede incrementar por un factor de 2, si se escogen como posibles estados de las neuronas los valores $x_i = -1$ y $x_i = 1$ en lugar de los valores originales $x_i = 0$ y $x_i = 1$.

Al utilizar el conjunto $\{-1,1\}$ y el valor de umbral cero, la fase de aprendizaje para la memoria Hopfield será similar, en cierta forma, a la fase de aprendizaje del *Linear Associator*. La intensidad de la fuerza de conexión de la neurona x_i a la neurona x_j se representa por el valor de m_{ij} , y se considera que hay simetría, es decir, $m_{ij} = m_{ji}$. Si x_i no está conectada con x_j entonces $m_{ij} = 0$; en particular, no hay conexiones recurrentes de una neurona a sí misma, lo cual significa que $m_{ij} = 0$. El estado instantáneo del sistema está completamente especificado por el vector columna de dimensión n cuyas coordenadas son los valores de las n neuronas.

La memoria Hopfield es autoasociativa, simétrica, con ceros en la diagonal principal. En virtud de que la memoria es autoasociativa, el conjunto fundamental para la memoria Hopfield es $\{(\mathbf{x}^\mu, \mathbf{x}^\mu) \mid \mu = 1, 2, \dots, p\}$ con $\mathbf{x}^\mu \in A^n$ y $A = \{-1, 1\}$.

Algoritmo Hopfield

Fase de Aprendizaje

La fase de aprendizaje para la memoria Hopfield es similar a la fase de aprendizaje del *Linear Associator*, con una ligera diferencia relacionada con la diagonal principal en ceros, como se muestra en la siguiente regla para obtener la ij -ésima componente de la memoria Hopfield \mathbf{M} :

$$m_{ij} = \begin{cases} \sum_{\mu=1}^p x_i^\mu x_j^\mu & \text{si } i \neq j \\ 0 & \text{si } i = j \end{cases}$$

Fase de Recuperación

Si se le presenta un patrón de entrada $\tilde{\mathbf{x}}$ a la memoria Hopfield, ésta cambiará su estado con el tiempo, de modo que cada neurona x_i ajuste su valor de acuerdo con el resultado que arroje la comparación de la cantidad $\sum_{j=1}^n m_{ij} x_j$ con un valor de umbral, el cual normalmente se coloca en cero.

Se representa el estado de la memoria Hopfield en el tiempo t por $\mathbf{x}(t)$; entonces $x_i(t)$ representa el valor de la neurona x_i en el tiempo t y $x_i(t+1)$ el valor de x_i en el tiempo siguiente $(t+1)$.

Dado un vector columna de entrada $\tilde{\mathbf{x}}$, la fase de recuperación consta de tres pasos:

- 1) Para $t = 0$, se hace $\mathbf{x}(t) = \tilde{\mathbf{x}}$; es decir, $x_i(0) = \tilde{x}_i, \forall i \in \{1, 2, \dots, n\}$
- 2) $\forall i \in \{1, 2, \dots, n\}$ se calcula $x_i(t+1)$ de acuerdo con la condición siguiente:

$$x_i(t+1) = \begin{cases} 1 & \text{si } \sum_{j=1}^n m_{ij}x_j(t) > 0 \\ x_i(t) & \text{si } \sum_{j=1}^n m_{ij}x_j(t) = 0 \\ -1 & \text{si } \sum_{j=1}^n m_{ij}x_j(t) < 0 \end{cases}$$

- 3) Se compara $x_i(t+1)$ con $x_i(t)$, $\forall i \in \{1,2,\dots,n\}$. Si $\mathbf{x}(t+1) = \mathbf{x}(t)$ el proceso termina y el vector recuperado es $\mathbf{x}(0) = \tilde{\mathbf{x}}$. De otro modo, el proceso continúa de la siguiente manera: los pasos 2 y 3 se iteran tantas veces como sea necesario hasta llegar a un valor $t = \tau$ para el cual $x_i(\tau+1) = x_i(\tau) \quad \forall i \in \{1,2,\dots,n\}$; el proceso termina y el patrón recuperado es $\mathbf{x}(\tau)$.

En el artículo original de 1982, Hopfield había estimado empíricamente que su memoria tenía una capacidad de recuperar $0.15n$ patrones, y en el trabajo de Abu-Mostafa & St. Jacques [57] se estableció formalmente que una cota superior para el número de vectores de estado arbitrarios estables en una memoria Hopfield es n .

2.2.6 Memorias Asociativas Morfológicas

La creación de las redes neuronales morfológicas en 1996 por Ritter & Sussner [8,112] marcó un hito en la historia de las redes neuronales artificiales, dado que cambiaron el esquema matemático con el que se habían trabajado tradicionalmente los modelos clásicos de redes neuronales artificiales.

Algo similar sucedió entre las memorias asociativas clásicas y las memorias asociativas morfológicas [30] surgidas a partir de las redes neuronales morfológicas. La diferencia fundamental entre las memorias asociativas clásicas (*Lernmatrix*, *Correlograph*, *Linear Associator* y Memoria Asociativa Hopfield) y las memorias asociativas morfológicas radica en los fundamentos operacionales de éstas últimas, que son las operaciones morfológicas de *dilatación* y *erosión*. El nombre de las memorias asociativas morfológicas está inspirado precisamente en estas dos operaciones básicas.

Estas memorias rompieron con el esquema utilizado a través de los años en los modelos de memorias asociativas clásicas, que utilizan operaciones convencionales entre vectores y matrices para la fase de aprendizaje y suma de productos para la recuperación de patrones. Las memorias asociativas morfológicas cambian los productos por sumas y las sumas por máximos o mínimos en ambas fases, tanto de aprendizaje como de recuperación de patrones [8, 30, 58, 59].

Hay dos tipos de memorias asociativas morfológicas: las memorias *max*, simbolizadas con \mathbf{M} , y las memorias *min*, cuyo símbolo es \mathbf{W} ; en cada uno de los dos tipos, las memorias pueden funcionar en ambos modos: heteroasociativo y autoasociativo.

Se definen dos nuevos productos matriciales:

El *producto máximo* entre \mathbf{D} y \mathbf{H} , denotado por $\mathbf{C} = \mathbf{D} \nabla \mathbf{H}$, es una matriz $[c_{ij}]_{m \times n}$ cuya ij -ésima componente c_{ij} es

$$c_{ij} = \bigvee_{k=1}^r (d_{ik} + h_{kj})$$

El *producto mínimo* de \mathbf{D} y \mathbf{H} denotado por $\mathbf{C} = \mathbf{D} \Delta \mathbf{H}$, es una matriz $[c_{ij}]_{m \times n}$ cuya ij -ésima componente c_{ij} es

$$c_{ij} = \bigwedge_{k=1}^r (d_{ik} + h_{kj})$$

Los productos máximo y mínimo contienen a los operadores máximo y mínimo, los cuales están íntimamente ligados con los conceptos de las dos operaciones básicas de la morfología matemática: *dilatación* y *erosión*, respectivamente.

Memorias Heteroasociativas Morfológicas

Algoritmo de las memorias morfológicas *max*

Fase de Aprendizaje

1. Para cada una de las p asociaciones $(\mathbf{x}^\mu, \mathbf{y}^\mu)$ se usa el producto mínimo para crear la matriz $y^\mu \Delta (-\mathbf{x}^\mu)^t$ de dimensiones $m \times n$, donde el negado transpuesto del patrón de entrada \mathbf{x}^μ se define como $(-\mathbf{x}^\mu)^t = (-x_1^\mu, -x_2^\mu, \dots, -x_n^\mu)$
2. Se aplica el operador máximo \bigvee a las p matrices para obtener la memoria \mathbf{M} .

$$\mathbf{M} = \bigvee_{\mu=1}^p [y^\mu \Delta (-\mathbf{x}^\mu)^t]$$

Fase de Recuperación

Esta fase consiste en realizar el producto mínimo Δ de la memoria \mathbf{M} con el patrón de entrada \mathbf{x}^ω , donde $\omega \in \{1, 2, \dots, p\}$, para obtener un vector columna \mathbf{y} de dimensión m :

$$\mathbf{y} = \mathbf{M} \Delta \mathbf{x}^\omega$$

Las fases de aprendizaje y de recuperación de las **memorias morfológicas *min*** se obtienen por dualidad. Es decir, se intercambian máximos por mínimos, y productos máximos por productos mínimos en todas y cada una de las operaciones de ambas fases.

Memorias Autoasociativas Morfológicas

Para este tipo de memorias se utilizan los mismos algoritmos descritos anteriormente y que son aplicados a las memorias heteroasociativas; lo único que cambia es el conjunto fundamental. Para este caso, se considera el siguiente conjunto fundamental:

$$\{(\mathbf{x}^\mu, \mathbf{x}^\mu) \mid \mathbf{x}^\mu \in A^n, \text{ donde } \mu = 1, 2, \dots, p\}$$

2.2.7 Memorias Asociativas Alfa-Beta

Las memorias asociativas Alfa-Beta [48] utilizan máximos y mínimos, y dos operaciones binarias originales α y β de las cuales heredan el nombre.

El fundamento teórico de las memorias asociativas Alfa-Beta se presenta en el siguiente capítulo de forma más completa, debido a que los operadores Alfa y Beta son la base fundamental para este trabajo de tesis.

2.3 Lógica reconfigurable

Un aspecto fundamental que distingue a un FPGA de un circuito integrado de aplicación específica (ASIC), resulta del hecho de que los FPGAs son dispositivos programables; es decir, mediante algún tipo de mecanismo es posible configurar (programar) el funcionamiento final de una pieza de silicio semiterminada [60-62].

Para ejemplificar lo anteriormente expuesto, tomaremos como punto de partida el circuito mostrado en la figura 2.6, que consta de dos entradas (a y b) y una salida (y). El valor que toma la señal de salida, resulta de la combinación lógica de las señales de entrada.

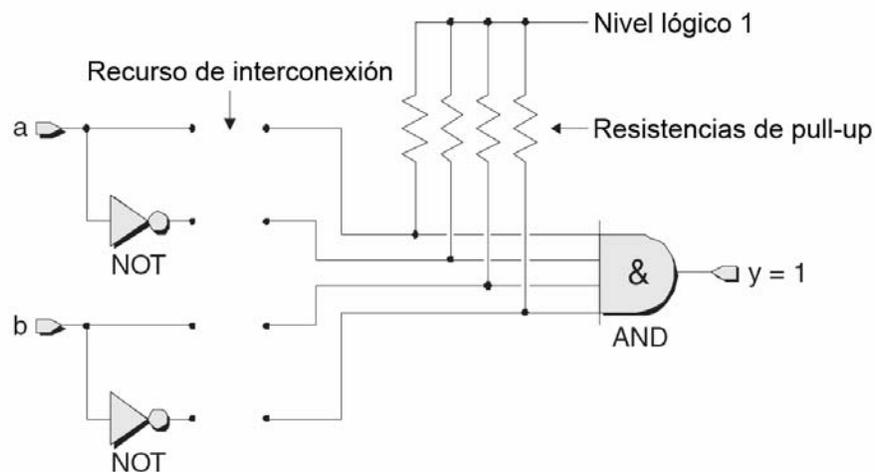


Figura 2.6 Programación de una función sencilla.

A la entrada del circuito tenemos tanto las señales originales como las complementadas, que resultan de la operación lógica NOT. En un principio, cuando no se ha llevado a cabo ningún tipo de configuración (programación), ambos puertos de entrada se encuentran en circuito abierto, lo cual provoca que, independientemente del valor que puedan tomar dichas señales, el resultado a la salida será el mismo. Particularmente, cuando no se ha llevado a cabo ningún tipo de configuración en el circuito mostrado en la figura 3.4, su resultado a la salida siempre será un nivel de voltaje “alto” provocado por las resistencias de *pull-up*.

2.3.1 Tecnología de fusible

Una de las primeras técnicas que permitieron programar estos dispositivos, fue conocida como tecnología de fusible (*Fusible link Technologies*). En este caso, el dispositivo es manufacturado con fusibles basados en el mismo proceso de fabricación de los transistores. Inicialmente, cuando un circuito no ha sido programado, todos los fusibles se encuentran intactos (estado cerrado), de manera que es posible elegir cuáles fusibles estarán habilitados y cuales estarán en estado abierto. Esta selección de la funcionalidad de los fusibles se lleva a cabo mediante la presencia o ausencia de pulsos de alta corriente al momento de la programación del dispositivo. Por ejemplo, al aplicar pulsos de alta corriente en las posiciones indicadas como a' y b de la figura 2.7, su resultado a la salida del circuito mostrado, es la función booleana $y = a \& !b$. El proceso de remover o inhabilitar fusibles, se conoce como programación.

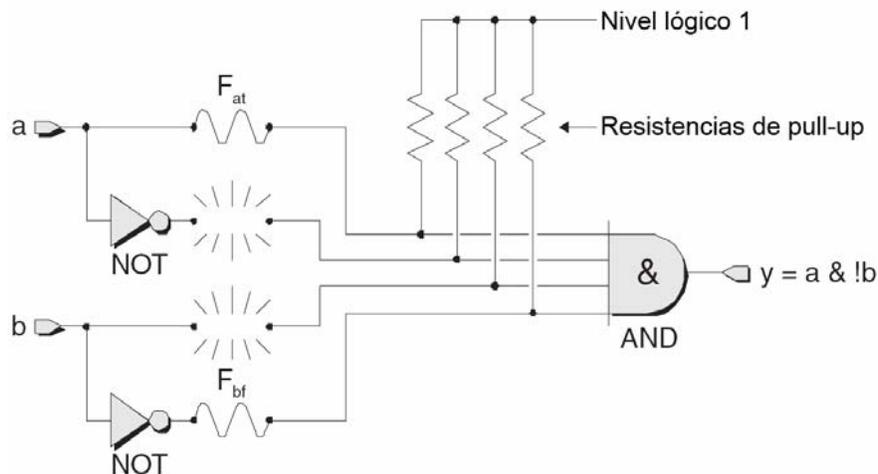


Figura 2.7 Selección de fusibles.

Los dispositivos basados en tecnologías de fusible, se dice que son dispositivos programables una sola vez (*One-time programmable*) ya que cuando un fusible ha sido inhabilitado por la presencia de un pulso de alta corriente, ya no es posible reestablecerlo [62].

2.3.2 Tecnología de anti-fusible

Diametralmente opuesto a la tecnología de fusible, tenemos la tecnología anti-fusible (*Anti-Fusible link Technologies*) [63]. Inicialmente, cuando un circuito no ha sido programado, todos los anti-fusibles se encuentran intactos, presentando tan alta resistencia al paso de corriente que pueden considerarse como circuitos abiertos.

Por ejemplo, al aplicar pulsos de alta corriente en las posiciones indicadas como a' y b de la figura 2.7, su resultado a la salida del circuito mostrado, es la función booleana $y = !a \& b$.

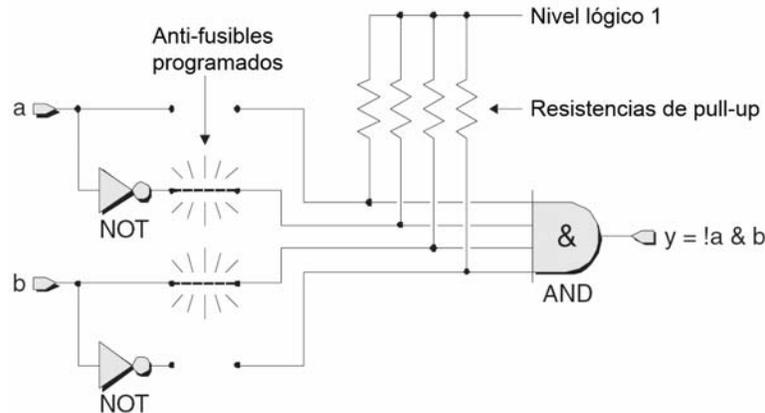


Figura 2.8 Programación de tecnología anti-fusible.

Cabe mencionar que la tecnología anti-fusible, también es considerada como dispositivos programables una sola vez (*One-time programmable*).

2.3.3 Tecnología PROM

La tecnología basada en celdas programables de solo lectura (PROM), pertenece a la gama de semiconductores no volátiles (una vez almacenado un dato, este permanece indefinidamente). Esta tecnología, mostrada en la figura 2.8, surge a principios de los años setenta para almacenar programas de cómputo y constantes numéricas; sin embargo, para finales de la década de los setentas ya eran ampliamente utilizadas no sólo para almacenar funciones lógicas, sino también para crear tablas de búsqueda así como para definir máquinas de estados. Uno de los factores que catapultaron su uso fue la posibilidad de corregir y actualizar las versiones de los programas con el simple hecho de programar un nuevo dispositivo y sustituir el anterior [62].

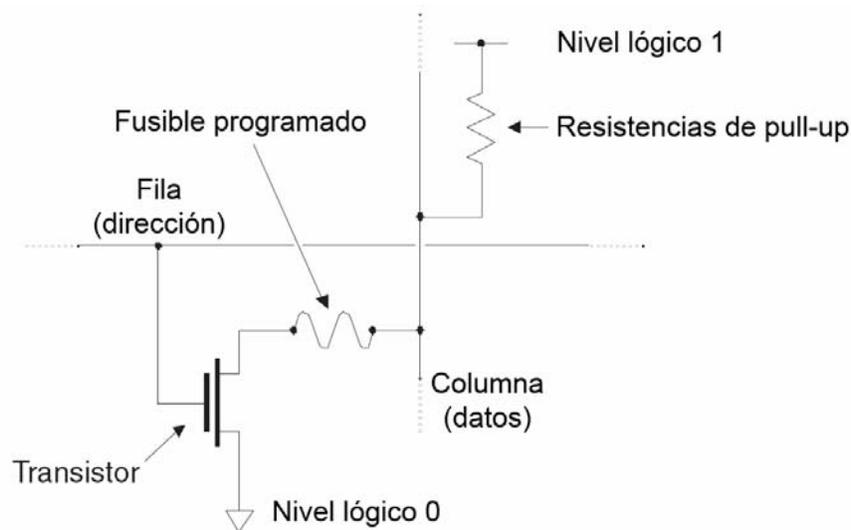


Figura 2.8 Tecnología basada en celdas programables de solo lectura (PROM).

2.3.4 Tecnología basada en EPROM

Anteriormente se mencionó que los dispositivos basados en tecnología de fusibles son dispositivos programables una sola vez. No obstante que su bajo costo de fabricación los posicionó como una alternativa eficaz para la implementación de programas y tablas de búsqueda, en el año de 1971, Intel lanzó al mercado un dispositivo capaz de ser programado, borrado y nuevamente programado (Erasable Programmable Read Only Memory). Además de la inherente ventaja que proporciona la posibilidad de reprogramar un dispositivo, las celdas EPROM pueden ser borradas mediante la descarga de electrones al interior del dispositivo. Físicamente un dispositivo EPROM consiste en un empaquetado de silicio con una ventana central (de cuarzo), el cual al ser expuesto a una fuente de luz ultravioleta intensa, pierde su contenido. Existen dos claras desventajas respecto al uso de dispositivos EPROM, la primera resulta del elevado costo de fabricación, mientras que la segunda tiene que ver con los largos periodos de exposición a los que tienen que someterse estos dispositivos para borrar su contenido (generalmente 20 minutos).

2.3.5 Tecnología FLASH

La tecnología FLASH, además de poder ser implementada mediante diversos tipos de arquitecturas (transistor único, transistores por pares, almacenamiento discreto por nodos), tiene la ventaja de poder eliminar su contenido eléctricamente en un lapso de tiempo menor al de las tecnologías anteriormente expuestas.

Las primeras versiones de tecnología basada en FLASH, únicamente eran capaces de almacenar un bit por celda, sin embargo en el año 2002 surge un esquema de almacenamiento discreto, el cual permite identificar la información mediante niveles variables de voltaje agrupados en un mismo transistor, lo cual conlleva al hecho de que dos bits sean almacenados en la misma celda FLASH [62].

2.3.6 Tecnología basada en SRAM

Existen dos versiones de dispositivos semiconductores basados en tecnología RAM: dinámica (DRAM) y estática (SRAM) [61]. En el caso de la tecnología basada en DRAM, cada celda de almacenamiento es formada por un transistor y un capacitor. El calificador dinámico aparece debido a que todo elemento capacitor pierde su carga con el paso del tiempo. Para mantener la información almacenada, cada celda debe ser recargada periódicamente, lo cual provoca grandes cantidades de circuitos adicionales, además, esta tecnología es de poco interés para fines de dispositivos programables.

Diametralmente opuesta es la tecnología SRAM, la cual almacena información sin necesidad de circuitos adicionales ni ciclos asociados de recarga, además, actualmente la mayoría de los FPGAs basan su funcionamiento sobre esta tecnología.

La funcionalidad de cada celda lógica de tecnología SRAM, es controlada mediante compuertas de paso que actúan como elementos de conmutación para formar rutas de señalización, Figura 2.10. (a). Cuando se tiene almacenado en la celda lógica un nivel de voltaje alto (valor lógico “1”), la compuerta de paso actúa como corto circuito, permitiendo la interconexión de dos segmentos independientes. La salida de cada multiplexor, es controlada mediante la interconexión de celdas lógicas y líneas de entrada del elemento en cuestión, de modo tal, que a la salida del multiplexor se tenga sólo una de las entradas del mismo, Figura 2.10 (b) .

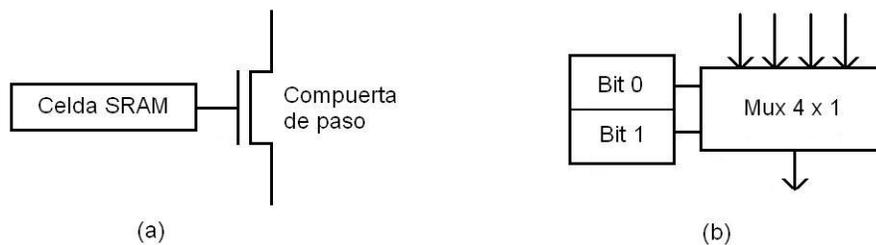


Figura 2.10 Configuración de celdas lógicas. (a) Control por compuerta de paso, (b) Multiplexor controlado por bits de estado.

CAPÍTULO 3

Materiales y métodos

En este capítulo se describen los materiales y los métodos que sirven de soporte para el algoritmo central de este trabajo de tesis. El capítulo consta de cuatro secciones; en la primera se expone el modelo de red neuronal sin pesos ADAM y sus propiedades.

En la segunda sección se describen las memorias asociativas Alfa-Beta, así como los operadores α y β que dan nombre al modelo asociativo Alfa-Beta y que fundamentan el nuevo algoritmo creado en esta tesis.

La tercera sección consiste en un resumen condensado del empleo de los FPGAs en redes neuronales, a través del tiempo. Por último, la cuarta sección comprende la teoría de los circuitos booleanos, que apoya el desarrollo del aporte formal en relación con la complejidad y el tipo de problemas que puede decidir la red neuronal Alfa-Beta sin pesos.

3.1 El modelo ADAM

El empleo de redes basadas en RAM que incluyen métodos N-tupla en memorias asociativas se investigaron en [64]. Esta arquitectura se denominó ADAM (Advance Distributed Associative Memory, Memoria Asociativa Distribuida Avanzada), y en realidad es un arreglo de dos redes asociativas de Willshaw, conectadas a través de un conjunto de vectores llamados códigos de clase [11]. Es una red neuronal binaria empleada en el procesamiento y análisis de imágenes, que se construye a partir de la componente principal denominada memoria de matriz de correlación (CMM).

La arquitectura básica de este modelo está compuesta de distintas etapas, las cuales se muestran en la figura 3.1. El empleo de dos redes asociativas de Willshaw permite que la capacidad de almacenamiento sea independiente del tamaño de los patrones de entrada y de salida.

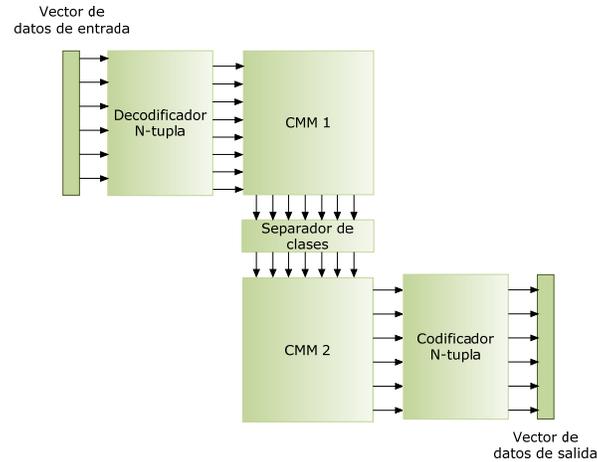


Figura 3.1 Arquitectura del modelo de red neuronal ADAM

El decodificador de funciones N-tupla recibe el patrón proporcionado al sistema ADAM durante la fase de aprendizaje, y su salida se emplea para entrenar a la primera CMM junto con el código de clase. Dicho código de clase contiene k bits de valor 1 (el valor de k se determina por el usuario). La asociación se habilita en las líneas de cruce activas, como se muestra en la figura 3.2. El patrón a recordar se almacena en la segunda CMM, y se realiza el mismo procedimiento de aprendizaje para esta memoria que el mencionado para la primera.

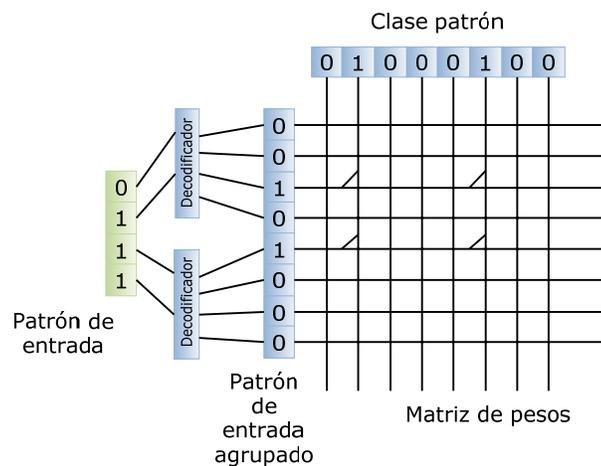


Figura 3.2. Etapa de aprendizaje de la red ADAM

La fase de aprendizaje, realizada en la primera etapa, se inicia al presentar el patrón de entrada, y luego se suma el número de asociaciones activas en cada línea. Ejemplo de ello se muestra en la figura 3.3. Se necesita luego recuperar la clase patrón. El valor de k se usa de nuevo para la recuperación del patrón al colocar en k el valor sumado mayor a 1, y el residuo se coloca en 0.

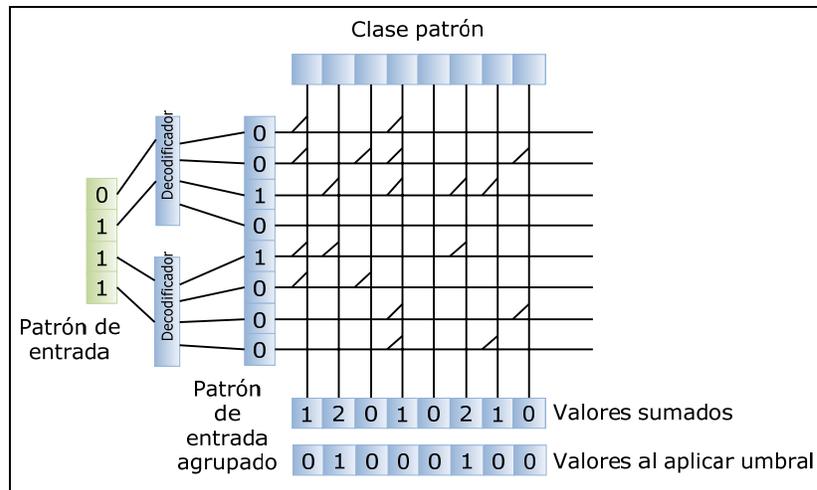


Figura 3.3 Primera etapa de recuperación

Luego, el código de clase recuperado se utiliza para recobrar el patrón de aprendizaje contenido en la segunda red asociativa de Willshaw. Esto se lleva a cabo al sumar las asociaciones activadas en las líneas horizontales. Los valores obtenidos al realizar la suma, se convierten al valor de umbral, empleando el método de umbral Willshaw; si el resultado de la suma iguala a k , entonces se proporciona el valor de 1, y si el resultado de la suma es menor a k , entonces se proporciona el valor de 0. Si el patrón a recordar ha sido agrupado, entonces el patrón recordado ahora podría ser codificado para recuperar el patrón asociado. Esto se muestra en la figura 3.4.

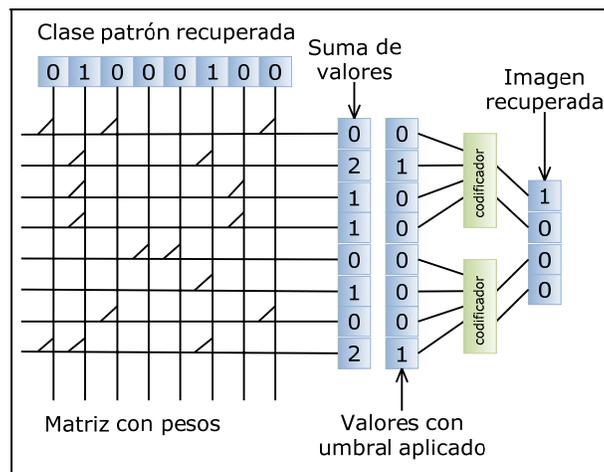


Figura 3.4 Segunda etapa de recuperación

A fin de describir el modelo ADAM de manera analítica, consideremos las constantes fijas $m \in \mathbb{Z}^+$, $n \in \mathbb{Z}^+$, $p \in \mathbb{Z}^+$ y el conjunto de los dígitos binarios $A = \{0, 1\}$; así, el conjunto fundamental es $\{(\mathbf{x}^\mu, \mathbf{y}^\mu) \mid \mathbf{x}^\mu \in A^n \text{ y } \mathbf{y}^\mu \in A^m, \text{ donde } \mu = 1, 2, \dots, p\}$. ADAM es un

sistema de entrada y salida que al operar acepta como entrada un patrón binario $\mathbf{x}^\mu \in A^n$, y produce como salida un patrón $\mathbf{y}^\mu \in A^m$.

En ambas fases, aprendizaje y recuperación, para cada pareja de patrones de entrada y salida se requiere un código de clase, el cual es un patrón $\mathbf{z}^\mu \in A^k$, donde k es un número entero positivo; cada código de clase contiene r componentes con valor 1 y $k-r$ componentes con valor 0, donde r es un número entero positivo fijo con la condición $r < k$.

Lo anterior significa que habrá $\frac{k!}{r!(k-r)!}$ diferentes códigos de clase disponibles.

El cálculo de los códigos de clase \mathbf{z}^μ se realiza a través de un proceso en el que se considera el patrón de entrada \mathbf{x}^μ , y se hace una distribución, por sectores, empleando r componentes.

Antes de iniciar las fases de aprendizaje y recuperación, es preciso llevar a cabo dos pasos previos [18]:

- 1) Se escoge el valor $r < k$ para crear los códigos de clase
- 2) Se crean dos matrices nulas (llenas de valores 0): $\mathbf{P} = [p_{ij}]_{k \times n}$ y $\mathbf{Q} = [q_{ij}]_{m \times k}$

3.1.1 Fase de aprendizaje

En la *fase de aprendizaje* de ADAM, para cada $\mu = 1, 2, \dots, p$ se realiza lo siguiente:

1. Se propone un código de clase \mathbf{z}^μ (el cual contiene r componentes con valor 1)
2. Se actualiza la matriz \mathbf{P} de acuerdo con el siguiente esquema:

	x_1^μ	x_2^μ	\dots	x_j^μ	\dots	x_n^μ
z_1^μ	p_{11}	p_{12}	\dots	p_{1j}	\dots	p_{1n}
z_2^μ	p_{21}	p_{22}	\dots	p_{2j}	\dots	p_{2n}
\vdots	\vdots	\vdots		\vdots		\vdots
z_i^μ	p_{i1}	p_{i2}	\dots	p_{ij}	\dots	p_{in}
\vdots	\vdots	\vdots		\vdots		\vdots
z_k^μ	p_{k1}	p_{k2}	\dots	p_{kj}	\dots	p_{kn}

donde la regla para actualizar los componentes p_{ij} es:

$$p_{ij} = \begin{cases} 1 & \text{si } z_i^\mu = 1 = x_j^\mu \\ \text{valor anterior} & \text{en otro caso} \end{cases}$$

3. Se actualiza la matriz \mathbf{Q} de acuerdo con el siguiente esquema:

	z_1^μ	z_2^μ	\dots	z_j^μ	\dots	z_k^μ
y_1^μ	q_{11}	q_{12}	\dots	q_{1j}	\dots	q_{1k}
y_2^μ	q_{21}	q_{22}	\dots	q_{2j}	\dots	q_{2k}
\vdots	\vdots	\vdots		\vdots		\vdots
y_i^μ	q_{i1}	q_{i2}	\dots	q_{ij}	\dots	q_{ik}
\vdots	\vdots	\vdots		\vdots		\vdots
y_m^μ	q_{m1}	q_{m2}	\dots	q_{mj}	\dots	q_{mk}

donde la regla para actualizar los componentes q_{ij} es:

$$q_{ij} = \begin{cases} 1 & \text{si } y_i^\mu = 1 = z_j^\mu \\ \text{valor anterior} & \text{en otro caso} \end{cases}$$

Al final de la fase de aprendizaje resultan dos redes asociativas de Willshaw \mathbf{P} y \mathbf{Q} .

3.1.2 Fase de recuperación

En la *fase de recuperación* se considera un patrón de entrada $\tilde{\mathbf{x}}^\omega \in A^n$, donde $\omega \in \{1, 2, \dots, p\}$ (si el ruido es cero, se cumple que $\tilde{\mathbf{x}}^\omega = \mathbf{x}^\omega$ pertenece al conjunto fundamental de entrada) y se llevan a cabo las siguientes acciones:

1. Se realiza la operación $\mathbf{P} \cdot \tilde{\mathbf{x}}^\omega$; es decir,

$$(\mathbf{P} \cdot \tilde{\mathbf{x}}^\omega)_i = \sum_{j=1}^n p_{ij} \tilde{x}_j^\omega, \forall i \in \{1, 2, \dots, k\}$$

2. Se calcula el correspondiente código de clase \mathbf{z}^ω , de la siguiente manera:

$$z_i^\omega = \begin{cases} 1 & \text{si } \sum_{j=1}^n p_{ij} \tilde{\mathbf{x}}_j^\omega = \vee_{h=1}^k \left[\sum_{j=1}^n p_{hj} \tilde{\mathbf{x}}_j^\omega \right] \\ 0 & \text{en otro caso} \end{cases}$$

3. Ya conocido el código de clase \mathbf{z}^ω , se realiza la operación $\mathbf{Q} \cdot \mathbf{z}^\omega$, es decir,

$$\left(\mathbf{Q} \cdot \mathbf{z}^\omega \right)_i = \sum_{j=1}^k q_{ij} z_j, \forall i \in \{1, 2, \dots, m\}$$

4. Se realiza la siguiente operación:

$$\tilde{y}_i^\omega = \begin{cases} 1 & \text{si } \sum_{j=1}^m q_{ij} \mathbf{z}_j^\omega = \vee_{h=1}^k \left[\sum_{j=1}^m q_{hj} \mathbf{z}_j^\omega \right] \\ 0 & \text{en otro caso} \end{cases}$$

Se espera que el vector $\tilde{\mathbf{y}}^\omega$ sea precisamente el patrón fundamental de salida \mathbf{y}^ω .

3.2 Memorias Asociativas Alfa-Beta

En esta sección se presentan los conceptos elementales relacionados con los operadores que permiten el diseño y operación de las memorias asociativas Alfa-Beta [18]; para ello, se presentan las definiciones y propiedades de las operaciones α y β , las operaciones matriciales que se derivan de estas operaciones originales, y se describen las fases de aprendizaje y recuperación de las memorias heteroasociativas Alfa-Beta, tanto \mathbf{V} (*max*) como $\mathbf{\Lambda}$ (*min*).

3.2.1 Operaciones binarias α y β

Las memorias Alfa-Beta utilizan máximos y mínimos, y dos operaciones binarias originales α y β de las cuales heredan el nombre.

Para la definición de las operaciones binarias α y β se deben especificar los conjuntos A y B , los cuales son:

$$A = \{0,1\} \quad \text{y} \quad B = \{0,1,2\}$$

La operación binaria $\alpha : A \times A \rightarrow B$ se define como se muestra en la Tabla 3.1.

Tabla 3.1 Operación binaria $\alpha : A \times A \rightarrow B$

x	y	$\alpha(x, y)$
0	0	1
0	1	0
1	0	2
1	1	1

La operación binaria $\beta : B \times A \rightarrow A$ se define como se muestra en la Tabla 3.2

Tabla 3.2 Operación binaria $\beta : B \times A \rightarrow A$

x	y	$\beta(x, y)$
0	0	0
0	1	0
1	0	0
1	1	1
2	0	1
2	1	1

Los conjuntos A y B , las operaciones binarias α y β junto con los operadores \wedge (mínimo) y \vee (máximo) usuales conforman el sistema algebraico $(A, B, \alpha, \beta, \wedge, \vee)$ en el que están inmersas las memorias asociativas Alfa-Beta [18, 65].

3.2.2 Operaciones matriciales Alfa-Beta

Se requiere la definición de cuatro operaciones matriciales, de las cuales se usarán sólo 4 casos particulares:

Operación **αmax** : $P_{m \times r} \nabla_{\alpha} Q_{r \times n} = |f_{ij}^{\alpha}|_{m \times n}$, donde $f_{ij}^{\alpha} = \bigvee_{k=1}^r \alpha(p_{ik}, q_{kj})$

Operación **βmax** : $P_{m \times r} \nabla_{\beta} Q_{r \times n} = |f_{ij}^{\beta}|_{m \times n}$, donde $f_{ij}^{\beta} = \bigvee_{k=1}^r \beta(p_{ik}, q_{kj})$

Operación **αmin** : $P_{m \times r} \Delta_{\alpha} Q_{r \times n} = |h_{ij}^{\alpha}|_{m \times n}$, donde $h_{ij}^{\alpha} = \bigwedge_{k=1}^r \alpha(p_{ik}, q_{kj})$

Operación **βmin** : $P_{m \times r} \Delta_{\beta} Q_{r \times n} = |h_{ij}^{\beta}|_{m \times n}$, donde $h_{ij}^{\beta} = \bigwedge_{k=1}^r \beta(p_{ik}, q_{kj})$

$\mathbf{y} \nabla_{\alpha} \mathbf{x}^t$ es una matriz de dimensiones $m \times n$, y además $\mathbf{y} \nabla_{\alpha} \mathbf{x}^t = \mathbf{y} \Delta_{\alpha} \mathbf{x}^t$; por ello, es conveniente escoger un símbolo único, el símbolo \otimes , que represente a las dos operaciones ∇_{α} y Δ_{α} cuando se opera un vector columna de dimensión m con un vector fila de dimensión n :

$$\mathbf{y} \nabla_{\alpha} \mathbf{x}^t = \mathbf{y} \otimes \mathbf{x}^t = \mathbf{y} \Delta_{\alpha} \mathbf{x}^t$$

La ij -ésima componente de la matriz está $\mathbf{y} \otimes \mathbf{x}^t$ dada por:

$$[\mathbf{y} \otimes \mathbf{x}^t]_{ij} = \alpha(y_i, x_j)$$

Dado un índice de asociación μ , la expresión anterior indica que la ij -ésima componente de la matriz $\mathbf{y}^{\mu} \otimes (\mathbf{x}^{\mu})^t$ se expresa de la siguiente manera:

$$[\mathbf{y}^{\mu} \otimes (\mathbf{x}^{\mu})^t]_{ij} = \alpha(y_i^{\mu}, x_j^{\mu})$$

A continuación se presenta el caso en el que se opera una matriz de dimensiones $m \times n$ con un vector columna de dimensión n usando la operación ∇_{β} .

La operación $\mathbf{P}_{m \times n} \nabla_{\beta} \mathbf{x}$ da como resultado un vector columna de dimensión m , cuya i -ésima componente tiene la siguiente forma: $(\mathbf{P}_{m \times n} \nabla_{\beta} \mathbf{x})_i = \bigvee_{j=1}^n \beta(p_{ij}, x_j)$.

Los resultados son similares para la operación $\mathbf{P}_{m \times n} \Delta_{\beta} \mathbf{x}$.

3.2.3 Memorias Heteroasociativas Alfa-Beta

Se tienen dos tipos de memorias heteroasociativas Alfa-Beta: tipo \mathbf{V} y tipo $\mathbf{\Lambda}$. En la generación de ambos tipos de memorias se usará el operador \otimes el cual tiene la siguiente forma:

$$[\mathbf{y}^{\mu} \otimes (\mathbf{x}^{\mu})^t]_{ij} = \alpha(y_i^{\mu}, x_j^{\mu}); \mu \in \{1, 2, \dots, p\}, i \in \{1, 2, \dots, m\}, j \in \{1, 2, \dots, n\}$$

Algoritmo Memorias Alfa-Beta tipo \mathbf{V}

Fase de Aprendizaje

Paso 1. Para cada $\mu = 1, 2, \dots, p$, a partir de la pareja $(\mathbf{x}^{\mu}, \mathbf{y}^{\mu})$ se construye la matriz

$$[\mathbf{y}^{\mu} \otimes (\mathbf{x}^{\mu})^t]_{m \times n}$$

Paso 2. Se aplica el operador binario máximo \bigvee a las matrices obtenidas en el paso 1:

$$\mathbf{V} = \bigvee_{\mu=1}^p [\mathbf{y}^{\mu} \otimes (\mathbf{x}^{\mu})^t]$$

La entrada ij -ésima está dada por la siguiente expresión:

$$v_{ij} = \bigvee_{\mu=1}^p \alpha(y_i^{\mu}, x_j^{\mu})$$

Fase de Recuperación

Se presenta un patrón \mathbf{x}^{ω} , con $\omega \in \{1, 2, \dots, p\}$, a la memoria heteroasociativa $\alpha\beta$ tipo \mathbf{V} y se realiza la operación $\Delta_{\beta} : \mathbf{V}\Delta_{\beta}\mathbf{x}^{\omega}$.

Dado que las dimensiones de la matriz \mathbf{V} son de $m \times n$ y \mathbf{x}^{ω} es un vector columna de dimensión n , el resultado de la operación anterior debe ser un vector columna de dimensión m , cuya i -ésima componente es:

$$(\mathbf{V}\Delta_{\beta}\mathbf{x}^{\omega})_i = \bigwedge_{j=1}^n \beta(v_{ij}, x_j^{\omega})$$

El **algoritmo para las memorias asociativas Alfa-Beta tipo Λ** es similar al anterior en ambas fases; sólo se cambian los máximos por mínimos y los mínimos por máximos, y en lugar de la operación $\mathbf{V}\Delta_{\beta}\mathbf{x}^{\omega}$ se ejecuta la operación $\Lambda\nabla_{\beta}\mathbf{x}^{\omega}$.

3.2.4 Memorias Autoasociativas Alfa-Beta

Si a una memoria heteroasociativa se le impone la condición de que $\mathbf{y}^{\mu} = \mathbf{x}^{\mu} \forall \mu \in \{1, 2, \dots, p\}$ entonces, deja de ser heteroasociativa y ahora se le denomina autoasociativa.

A continuación se enlistan algunas de las características de las memorias autoasociativas Alfa-Beta:

1. El conjunto fundamental toma la forma $\{(\mathbf{x}^{\mu}, \mathbf{x}^{\mu}) \mid \mu \in 1, 2, \dots, p\}$
2. Los patrones fundamentales de entrada y salida son de la misma dimensión; denotémosla por n .
3. La memoria es una matriz cuadrada, para ambos tipos, \mathbf{V} y Λ . Si $\mathbf{x}^{\mu} \in A^n$ entonces

$$\mathbf{V} = [v_{ij}]_{n \times n} \quad \text{y} \quad \Lambda = [\lambda_{ij}]_{n \times n}$$

Las fases de aprendizaje y recuperación son similares a las memorias heteroasociativas Alfa-Beta, considerando los tres puntos anteriores.

Una propiedad muy importante de las memorias autoasociativas Alfa-Beta es que, sin condiciones, recuperan de manera correcta el conjunto fundamental completo.

3.3 Empleo de FPGAs en redes neuronales

Los arreglos de compuertas programables son dispositivos semiconductores principalmente estructurados por celdas lógicas interconectadas mediante una matriz de interruptores programables (Figura 3.5). Típicamente, cada una de estas celdas es capaz de llevar a cabo tareas combinatorias o secuenciales, de manera que cualquier diseño puede ser implementado mediante la descripción funcional de cada celda lógica y la selección de los interruptores adecuados que permiten la interconexión entre los elementos involucrados [62].

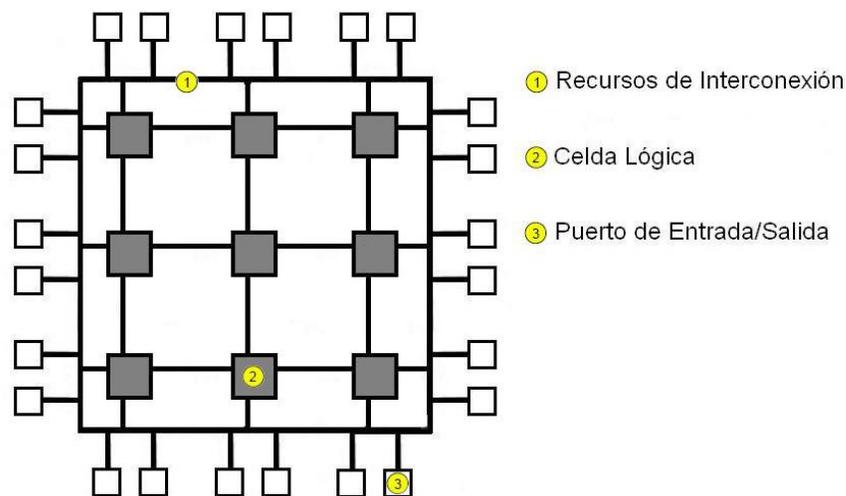


Figura 3.5 Arquitectura General de un FPGA.

Cuando los primeros FPGAs aparecieron en la década de los ochenta [66], principalmente fueron usados como recursos de implementación de máquinas de estados de mediana complejidad en tareas de procesamiento limitado de datos. En el inicio de la década posterior, el incremento en el tamaño y la sofisticación de estos dispositivos hicieron posible que grandes bloques de información pudieran ser procesados, trayendo consigo un crecimiento nunca antes visto en el mercado de las telecomunicaciones y de las redes de cómputo. Para finales de la década de los noventa, estos dispositivos gozaron de gran importancia no sólo en el sector automotriz, sino también en el campo de las aplicaciones industriales. Al comienzo de la década siguiente, gracias a los avances en las ciencias de los materiales, aparecieron los llamados FPGAs de alto desempeño, los cuales ya contaban con

millones de compuertas lógicas disponibles e interfaces de entrada/salida de alta velocidad; permitiendo la implantación de núcleos de microprocesadores. El resultado final, es que los FPGAs de hoy en día, pueden ser utilizados para implementar casi cualquier elemento de cómputo; de ahí que, estos sean frecuentemente usados para el diseño de prototipos [67], dispositivos de comunicación [68, 69], sistemas embebidos, así como para diagnóstico y verificación de la implementación física de nuevos algoritmos relacionados con la inteligencia artificial [41, 60, 70, 71].

La implementación en hardware de las redes neuronales ha presentado un progreso creciente en las últimas dos décadas [25, 60, 72], enfrentando retos de construir sistemas que presenten técnicas de aprendizaje eficientes, mostrando mejoras en características como velocidad de cómputo o reducción en consumo de energía, explotando o desarrollando de manera eficiente las técnicas de representación de señal/estado (basadas en pulso, continuas en el tiempo o de transición repentina), que aprovechen las propiedades de ruido o las estadísticas. Los progresos en hardware actuales soportan los requerimientos mencionados.

El empleo de arreglo de compuertas programables en campo (FPGAs) ha proporcionado ventajas significativas, entre ellas la capacidad de reconfiguración. Esta característica permite utilizar los recursos presentes en el FPGA para implementar varios tipos de redes neuronales. Actualmente las características de rendimiento y capacidad, presentes en los FPGAs, son una alternativa viable en la implementación de neurocomputadoras (una neurocomputadora es una herramienta esencial empleada en investigación y desarrollo de redes neuronales artificiales inspiradas biológicamente y sus aplicaciones). Diversos grupos de investigación alrededor del mundo desarrollan modelos de cómputo basados en hardware para la realización de tareas en el procesamiento de información (clasificación, decisión, predicción y regresión, entre otros) [73-77]

Al implementarse en hardware, las redes neuronales artificiales toman ventaja de la arquitectura presente en los dispositivos, realizando cómputo con un rango de velocidad mayor al mostrado en la simulación por software. Sin embargo, se debe señalar que el número de operadores y la complejidad de las conexiones se hacen posibles gracias a la flexibilidad presente en los dispositivos lógicos reconfigurables de modificar líneas de interconexión internas para enlazar las neuronas creadas in situ.

La tecnología de FPGA, denominada Virtex, utilizada en la implementación del modelo de redes neuronales Alfa-Beta sin pesos propuesto en esta tesis, se exhibe en el anexo B.

3.4 Teoría de los circuitos *Booleanos*

Existen varios modelos abstractos que describen el comportamiento del cómputo paralelo, y uno de ellos es el modelo de los circuitos booleanos. En la complejidad de circuitos booleanos se realiza una medición de recursos con base en el tamaño y longitud o profundidad de los mismos [78].

De manera general, se entiende a un circuito booleano como un modelo matemático de computación empleado en la teoría de la complejidad computacional. Un circuito booleano es en esencia una gráfica acíclica dirigida, que se construye asociando cada nodo con una variable, una constante o una compuerta (\neg , \wedge , \vee), que a su vez se unen por medio de líneas de conexión. Cada nodo presenta un rango definido de una o dos entradas (bounded fan-in), y un rango de salida definido de uno (bounded fan-out) para el caso de las compuertas mencionadas previamente.

CAPÍTULO 4

Modelo propuesto

Este capítulo es el más relevante del presente documento de tesis, y consta de cuatro secciones. En la sección 4.1 se presentan las definiciones y ejemplos de tres operaciones que representan el primer resultado original de esta tesis: la operación Alfa Generalizada y las operaciones Sigma-Alfa y Sigma-Beta, las cuales resultan de gran utilidad en el diseño del nuevo modelo.

La sección 4.2 contiene el resultado principal de esta tesis: el algoritmo de CAINN; y las dos secciones restantes incluyen, respectivamente, la implementación en hardware de CAINN y su factibilidad de implementación.

El nuevo modelo se sustenta en la tricotomía que se muestra en la figura 4.1

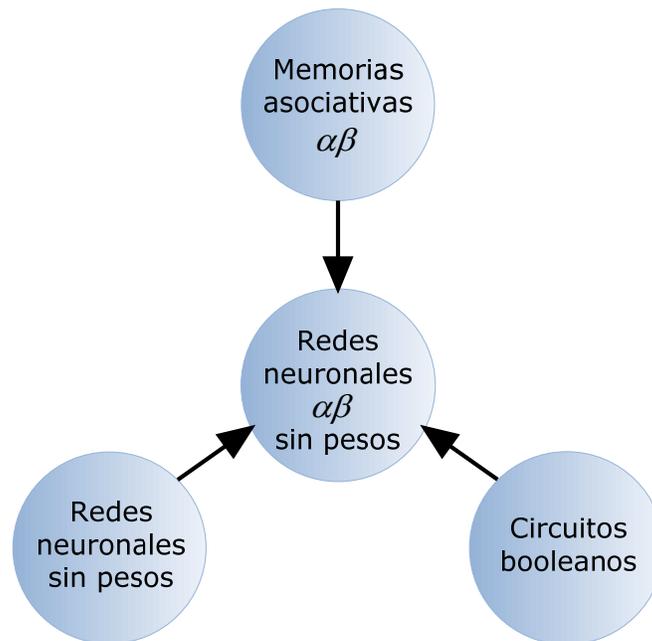


Figura 4.1. Tricotomía que sustenta a las redes neuronales Alfa-Beta sin pesos

4.1 Tres operaciones originales

En esta sección se presenta el primer resultado original de esta tesis. Las tres operaciones aquí definidas y ejemplificadas, fueron creadas por el autor y resultan de gran utilidad en el desarrollo, aplicación y prueba del algoritmo propuesto.

4.1.1 Operación α_g

Con el fin de caracterizar la fase de recuperación del nuevo modelo usando las operaciones Alfa y Beta, se requiere la definición de una nueva operación, a la que se denomina Operación Alfa Generalizada y la denotamos con el símbolo α_g . Esta nueva operación, como su nombre lo indica, es una generalización de la operación binaria Alfa original, de modo que acepte valores reales como argumentos, entregando a la salida valores en el conjunto $B = \{00,01,10\}$; es decir, $\alpha_g : \mathbb{R} \times \mathbb{R} \rightarrow B$, y se define de la siguiente manera:

Sean $x \in \mathbb{R}$, $y \in \mathbb{R}$ dos números reales cualesquiera, entonces

$$\alpha_g = \begin{cases} 00 & \text{si } x < y \\ 01 & \text{si } x = y \\ 10 & \text{si } x > y \end{cases}$$

Por ejemplo, si $a=2.1$, $b=50$, $c=2.1$ y $d=-0.3$, se tiene lo siguiente:

- $\alpha_g(a, b) = 00$
- $\alpha_g(a, c) = 01$
- $\alpha_g(b, d) = 10$
- $\alpha_g(c, d) = 10$
- $\alpha_g(b, c) = 10$

4.1.2 Operaciones σ_α y σ_β

Estas dos operaciones se definen y ejemplifican como sigue:

Operación **Sigma-Alfa**: $P_{m \times r} \sigma_\alpha Q_{r \times n} = \left| f_{ij}^\alpha \right|_{m \times n}$, donde $f_{ij}^\alpha = \sum_{k=1}^r \alpha(p_{ik}, q_{kj})$

Operación **Sigma-Beta**: $P_{m \times r} \sigma_\beta Q_{r \times n} = \left| f_{ij}^\beta \right|_{m \times n}$, donde $f_{ij}^\beta = \sum_{k=1}^r \beta(p_{ik}, q_{kj})$

Dados los vectores columna $\mathbf{x} \in A^n$ y $\mathbf{y} \in A^m$, con el conjunto $A = \{0,1\}$, y $\sigma_\alpha \mathbf{x}^t$ es una

matriz de dimensiones $m \times n$. La ij -ésima componente de la matriz $\mathbf{y} \sigma_\alpha \mathbf{x}^t$ está dada por:

$$[\mathbf{y} \sigma_\alpha \mathbf{x}^t]_{ij} = \alpha(y_i, x_j)$$

A continuación se presenta el caso en el que se opera una matriz de dimensiones $m \times n$ con un vector columna de dimensión n usando la operación σ_β .

La operación $\mathbf{P}_{m \times n} \sigma_\beta \mathbf{x}$ da como resultado un vector columna de dimensión m , cuya i -ésima

componente tiene la siguiente forma: $(\mathbf{P}_{m \times n} \sigma_\beta \mathbf{x})_i = \sum_{j=1}^n \beta(p_{ij}, x_j)$.

Los resultados son similares para la operación $\mathbf{P}_{m \times n} \sigma_\beta \mathbf{x}$.

Por ejemplo, si $\mathbf{P} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$ y $\mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$, entonces $\mathbf{P} \sigma_\alpha \mathbf{x} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$ y $\mathbf{P} \sigma_\beta \mathbf{x} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$.

4.2 Algoritmo de la nueva red neuronal Alfa-Beta sin pesos

Para describir el nuevo modelo, al que le he asignado el acrónimo CAINN (Computing Artificial Intelligence Neural Network), sea el conjunto $A = \{0,1\}$ de donde toman los valores los patrones tanto de entrada como de salida, y sea el siguiente conjunto fundamental $\{(\mathbf{x}^\mu, \mathbf{y}^\mu) \mid \mathbf{x}^\mu \in A^n \text{ y } \mathbf{y}^\mu \in A^m, \text{ donde } \mu = 1, 2, \dots, p\}$.

En ambas fases, aprendizaje y recuperación, para cada pareja de patrones de entrada y salida se requiere un código de clase, el cual es un patrón $z^\mu \in A^k$, donde k es la dimensión de un vector one-hot que corresponde al valor decimal del patrón binario \mathbf{x}^μ , considerando el bit superior como el más significativo.

4.2.1 Fase de aprendizaje de CAINN

1. Se crean dos matrices nulas (llenas de valores 0): $\mathbf{P} = [p_{ij}]_{k \times n}$ y $\mathbf{Q} = [q_{ij}]_{m \times k}$
2. Se denota por $p_{ij}(0)$ y $q_{ij}(0)$ a los valores 0 iniciales de las componentes ij -ésimas de las matrices \mathbf{P} y \mathbf{Q} , respectivamente.
3. Para cada $\mu = 1, 2, \dots, p$ se denotan por $p_{ij}(\mu)$ y $q_{ij}(\mu)$, respectivamente, los valores que adquieren las componentes ij -ésimas de las matrices \mathbf{P} y \mathbf{Q} , como consecuencia de las actualizaciones realizadas en la fase de aprendizaje, de acuerdo con las reglas descritas a continuación:

- 3.1. Se propone el código de clase \mathbf{z}^μ de dimensión k
 3.2. Se actualiza la matriz \mathbf{P} de acuerdo con la siguiente regla de actualización de los componentes p_{ij} :

$$p_{ij}(\mu) = \beta\left(\alpha\left[p_{ij}(\mu-1), 0\right], \beta(z_i^\mu, x_j^\mu)\right)$$

- 3.3. Se actualiza la matriz \mathbf{Q} de acuerdo con la siguiente regla de actualización de los componentes q_{ij} :

$$q_{ij}(\mu) = \beta\left(\alpha\left[q_{ij}(\mu-1), 0\right], \beta(y_i^\mu, z_j^\mu)\right)$$

4. Se genera un vector columna adicional $s \in A^n$, que contiene en su i -ésima componente, la suma de los valores positivos en el i -ésimo renglón de la matriz \mathbf{P} [104, 105]. Es decir

$$s_i = \sum_{j=1}^k p_{ij}, \text{ tal que } p_{ij} > 0$$

4.2.2 Fase de recuperación de CAINN

Se considera un patrón de entrada $\tilde{\mathbf{x}}^\omega \in A^n$, donde $\omega \in \{1, 2, \dots, p\}$ (si se cumple que $\tilde{\mathbf{x}}^\omega = \mathbf{x}^\omega$, el patrón pertenece al conjunto fundamental de entrada) y se llevan a cabo las siguientes acciones:

1. Se realiza la operación $\mathbf{P}\sigma_\beta\tilde{\mathbf{x}}^\omega$; es decir,

$$\left(\mathbf{P}\sigma_\beta\tilde{\mathbf{x}}^\omega\right)_i = \sum_{j=1}^n \beta(p_{ij}, \tilde{x}_j^\omega), \forall i \in \{1, 2, \dots, k\}$$

2. Se calcula el correspondiente vector de transición t^ω del código de clase \mathbf{z}^ω , de la siguiente manera:

$$t_i^\omega = \alpha_g \left(\sum_{j=1}^n \beta(p_{ij}, \tilde{x}_j^\omega), \bigvee_{h=1}^k \left[\sum_{j=1}^n \beta(p_{hj}, \tilde{x}_j^\omega) \right] \right), \forall i \in \{1, 2, \dots, k\}$$

3. Se define el conjunto H como sigue:

$$H = \left\{ h \left| \sum_{j=1}^n p_{hj} \tilde{x}_j^\omega = \bigvee_{i=1}^k \left(\sum_{j=1}^n p_{ij} \tilde{x}_j^\omega \right) \right. \right\}$$

4. Se calcula el código de clase z^ω de la siguiente manera:

$$z_i^\omega = \beta \left\{ \alpha_g(t_i^\omega, 1), \alpha_g \left[\bigwedge_{h \in H} \left(\sum_{j=1}^n p_{hj} \right), \sum_{j=1}^n p_{ij} \right] \right\}, \forall i \in \{1, 2, \dots, k\}$$

5. Ya conocido el código de clase z^ω , se realiza la operación $Q_{\sigma_{\beta} z^\omega}$, es decir,

$$\left(Q_{\sigma_{\beta} z^\omega} \right)_i = \sum_{j=1}^k \beta(q_{ij}, z_j), \forall i \in \{1, 2, \dots, m\}$$

6. Finalmente, se calcula el vector y^ω así:

$$y_i^\omega = \alpha_g \left(\sum_{j=1}^k \beta(q_{ij}, z_j), \bigvee_{h=1}^m \left[\sum_{j=1}^k \beta(q_{hj}, z_j) \right] \right), \forall i \in \{1, 2, \dots, m\}$$

Se espera que el vector y^ω sea precisamente el patrón fundamental de salida correspondiente a x^ω .

A continuación se realiza un ejemplo donde se emplea el algoritmo CAINN. Los patrones de entrada y salida son los siguientes:

$$x^1 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad y^1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad x^2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \quad y^2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad x^3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad y^3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

CAINN – Fase de aprendizaje:

$$P_{k \times n} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad Q_{m \times k} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

4. Se genera el vector columna s .

$$s = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \end{pmatrix}$$

4. Se calcula el código de clase z^ω .

$$z_i^\omega = \beta \left\{ \alpha_g(t_i^\omega, 1), \alpha_g \left[\bigwedge_{h \in H} \left(\sum_{j=1}^n p_{hj} \right), \sum_{j=1}^n p_{ij} \right] \right\}, \forall i \in \{1, 2, \dots, k\}$$

$$\begin{aligned} z_0^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,0) \} = \beta(0,2) = 0 & z_8^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,0) \} = \beta(0,2) = 0 \\ z_1^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,0) \} = \beta(0,2) = 0 & z_9^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,0) \} = \beta(0,2) = 0 \\ z_2^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,0) \} = \beta(0,2) = 0 & z_{10}^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,0) \} = \beta(0,2) = 0 \\ z_3^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,1) \} = \beta(0,1) = 0 & z_{11}^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,0) \} = \beta(0,2) = 0 \\ z_4^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,0) \} = \beta(0,2) = 0 & z_{12}^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,0) \} = \beta(0,2) = 0 \\ z_5^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,0) \} = \beta(0,2) = 0 & z_{13}^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,3) \} = \beta(0,0) = 0 \\ z_6^\omega &= \beta \{ \alpha_g(1,1), \alpha_g(2,2) \} = \beta(1,1) = 1 & z_{14}^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,0) \} = \beta(0,2) = 0 \\ z_7^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,0) \} = \beta(0,2) = 0 & z_{15}^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(2,0) \} = \beta(0,2) = 0 \end{aligned}$$

De lo anterior, se tiene que:

$$z^\omega = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

5. Se ejecuta la operación $Q_{\sigma_\beta} z^\omega$.

$$\begin{aligned}
 y_1^\omega &= \alpha_g \left(1, \sqrt{[1, 0, 0]} \right) = \alpha_g(1, 1) = 1 \\
 y_2^\omega &= \alpha_g \left(0, \sqrt{[1, 0, 0]} \right) = \alpha_g(0, 1) = 0 \\
 y_3^\omega &= \alpha_g \left(0, \sqrt{[1, 0, 0]} \right) = \alpha_g(0, 1) = 0
 \end{aligned}$$

$$y^\omega = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = y^2 \quad \therefore \text{recuperación correcta.}$$

Por último, tenemos la fase de recuperación para el último vector de entrada $\tilde{\mathbf{x}}^\omega = \mathbf{x}^3$.

CAINN – Fase de recuperación:

2. Operación $\mathbf{P}\sigma_\beta\tilde{\mathbf{x}}^\omega$:

$$\left(\mathbf{P}\sigma_\beta\tilde{\mathbf{x}}^\omega \right) = \begin{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} \sigma_\beta \\ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \end{matrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \mathbf{2} \\ 0 \\ 0 \\ 0 \\ \mathbf{1} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \mathbf{1} \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

2. Calculo del vector de transición t^ω :

$$t^\omega = \begin{bmatrix} \alpha_g(0,2) \\ \alpha_g(0,2) \\ \alpha_g(0,2) \\ \alpha_g(2,2) \\ \alpha_g(0,2) \\ \alpha_g(0,2) \\ \alpha_g(1,2) \\ \alpha_g(0,2) \\ \alpha_g(1,2) \\ \alpha_g(0,2) \\ \alpha_g(0,2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

3. Se define el conjunto H como $H = \{2\}$

4. Se calcula el código de clase z^ω .

$$\begin{aligned} z_0^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,0)\} = \beta(0,2) = 0 & z_8^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,0)\} = \beta(0,2) = 0 \\ z_1^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,0)\} = \beta(0,2) = 0 & z_9^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,0)\} = \beta(0,2) = 0 \\ z_2^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,0)\} = \beta(0,2) = 0 & z_{10}^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,0)\} = \beta(0,2) = 0 \\ z_3^\omega &= \beta\{\alpha_g(1,1), \alpha_g(2,2)\} = \beta(1,1) = 1 & z_{11}^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,0)\} = \beta(0,2) = 0 \\ z_4^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,0)\} = \beta(0,2) = 0 & z_{12}^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,0)\} = \beta(0,2) = 0 \\ z_5^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,0)\} = \beta(0,2) = 0 & z_{13}^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,1)\} = \beta(0,0) = 0 \\ z_6^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,1)\} = \beta(0,1) = 0 & z_{14}^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,0)\} = \beta(0,2) = 0 \\ z_7^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,0)\} = \beta(0,2) = 0 & z_{15}^\omega &= \beta\{\alpha_g(0,1), \alpha_g(2,0)\} = \beta(0,2) = 0 \end{aligned}$$

4. Se genera el vector columna s .

$$s = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 2 \\ 3 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \end{pmatrix}$$

CAINN – Fase de recuperación

En esta fase, se prueba el algoritmo con los patrones fundamentales. Para este caso, sea $\tilde{\mathbf{x}}^\omega = \mathbf{x}^4$

2. Se aplica la operación $\mathbf{P}_{\sigma_\beta} \tilde{\mathbf{x}}^\omega$:

$$\left(\mathbf{P}_{\sigma_\beta} \tilde{\mathbf{x}}^\omega \right) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} \sigma_\rho \\ \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \end{matrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 2 \\ 3 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{pmatrix}$$

2. Calculo del vector de transición t^ω :

$$t^\omega = \begin{bmatrix} \alpha_g(0,3) \\ \alpha_g(0,3) \\ \alpha_g(0,3) \\ \alpha_g(2,3) \\ \alpha_g(0,3) \\ \alpha_g(0,3) \\ \alpha_g(2,3) \\ \alpha_g(3,3) \\ \alpha_g(0,3) \\ \alpha_g(2,3) \\ \alpha_g(0,3) \\ \alpha_g(0,3) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

3. Se define el conjunto H como: $H = \{3\}$

4. Se calcula el código de clase z^ω .

$$z_i^\omega = \beta \left\{ \alpha_g(t_i^\omega, 1), \alpha_g \left[\bigwedge_{h \in H} \left(\sum_{j=1}^n p_{hj} \right), \sum_{j=1}^n p_{ij} \right] \right\}, \forall i \in \{1, 2, \dots, k\}$$

$$\begin{aligned} z_0^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,0) \} = \beta(0,2) = 0 & z_8^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,0) \} = \beta(0,2) = 0 \\ z_1^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,0) \} = \beta(0,2) = 0 & z_9^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,0) \} = \beta(0,2) = 0 \\ z_2^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,0) \} = \beta(0,2) = 0 & z_{10}^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,0) \} = \beta(0,2) = 0 \\ z_3^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,2) \} = \beta(0,2) = 0 & z_{11}^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,0) \} = \beta(0,2) = 0 \\ z_4^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,0) \} = \beta(0,2) = 0 & z_{12}^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,0) \} = \beta(0,2) = 0 \\ z_5^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,0) \} = \beta(0,2) = 0 & z_{13}^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,2) \} = \beta(0,1) = 0 \\ z_6^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,2) \} = \beta(0,2) = 0 & z_{14}^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,0) \} = \beta(0,2) = 0 \\ z_7^\omega &= \beta \{ \alpha_g(1,1), \alpha_g(3,3) \} = \beta(1,1) = 1 & z_{15}^\omega &= \beta \{ \alpha_g(0,1), \alpha_g(3,0) \} = \beta(0,2) = 0 \end{aligned}$$

De lo anterior, se tiene que:

$$z^{\omega} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

5. Se ejecuta la operación $Q_{\sigma_{\beta}} z^{\omega}$.

$$Q_{\sigma_{\beta}} z^{\omega} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}_{\sigma_{\beta}} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

6. Por último, se calcula el vector de salida y^ω .

$$\begin{aligned} y_1^\omega &= \alpha_g \left(\sum_{j=1}^{16} \beta(q_{1j}, z_j), \bigvee_{h=1}^3 \left[\sum_{j=1}^k \beta(q_{hj}, z_j) \right] \right) \\ y_2^\omega &= \alpha_g \left(\sum_{j=1}^{16} \beta(q_{2j}, z_j), \bigvee_{h=1}^3 \left[\sum_{j=1}^k \beta(q_{hj}, z_j) \right] \right) \\ y_3^\omega &= \alpha_g \left(\sum_{j=1}^{16} \beta(q_{3j}, z_j), \bigvee_{h=1}^3 \left[\sum_{j=1}^k \beta(q_{hj}, z_j) \right] \right) \end{aligned}$$

Sustituyendo valores,

$$\begin{aligned} y_1^\omega &= \alpha_g \left(1, \bigvee [1, 0, 1] \right) = \alpha_g (1, 1) = 1 \\ y_2^\omega &= \alpha_g \left(0, \bigvee [1, 0, 1] \right) = \alpha_g (0, 1) = 0 \\ y_3^\omega &= \alpha_g \left(1, \bigvee [1, 0, 1] \right) = \alpha_g (1, 1) = 1 \end{aligned}$$

$$y^\omega = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = y^1 \quad \therefore \text{recuperación correcta.}$$

4.3 Implementación en hardware del modelo CAINN

Esta sección consta de dos partes. En la 4.3.1 se describen las implementaciones en hardware de las operaciones Alfa y Beta, tomadas de la tesis [70] (la cual dirigi), mientras que en la parte 4.3.2 se hace uso de los resultados de la parte 4.1, con el fin de implementar en hardware el modelo CAINN, producto central de la presente tesis doctoral.

4.3.1 Operaciones Alfa y Beta

Retomando algunos conceptos presentados en el capítulo 3, tenemos que: para la definición de las operaciones binarias α y β se deben especificar los conjuntos A y B , los cuales son:

$$A = \{0,1\} \quad \text{y} \quad B = \{0,1,2\}$$

Los cuales para fines de implementación en hardware se tienen que expresar de la siguiente forma:

$$A = \{0, 1\} \quad \text{y} \quad B = \{00, 01, 10\}$$

La operación binaria $\alpha : A \times A \rightarrow B$ se presenta en la Tabla 4.1

Tabla 4.1 Operación binaria $\alpha : A \times A \rightarrow B$

x	y	$\alpha(x, y)$	
0	0	0	1
0	1	0	0
1	0	1	0
1	1	0	1

Por definición, esta operación recibe dos bits como parámetros de entrada y entrega otros dos bits a la salida, siendo $a_out(1)$ el bit más significativo a la salida, como se muestra en la figura 4.2.

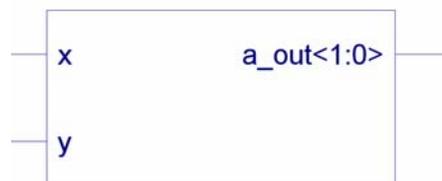


Figura 4.2 Bloque Alfa.

La implementación de la operación binaria α , a nivel de compuertas lógicas se muestra en la Figura 4.3.

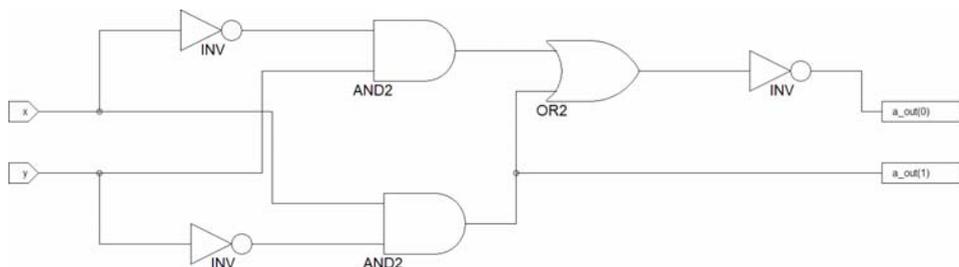


Figura 4.2 Implementación de la Operación Alfa a nivel de compuertas lógicas.

La operación binaria $\beta : (B \times A) \rightarrow A$, se presenta en la tabla 4.2.

Tabla 4.2 Operación binaria $\beta : (B \times A) \rightarrow A$

x		y	$\beta(x, y)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1

Por definición, esta operación recibe tres bits como parámetros de entrada y entrega un solo bit a la salida, siendo $x(1)$ el bit más significativo a la entrada, como se muestra en la figura 4.4.



Figura 4.4 Bloque Beta.

La implementación de la operación binaria β , a nivel de compuertas lógicas se muestra en la Figura 4.5.

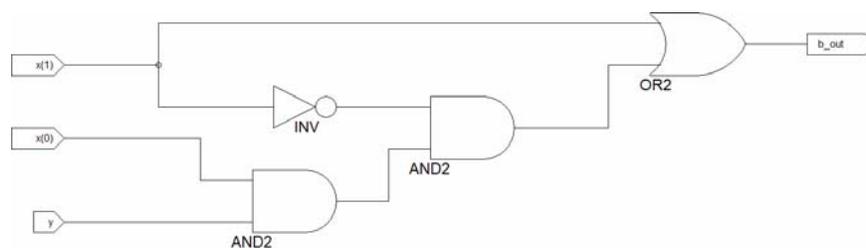


Figura 4.4. Implementación de la Operación Beta a nivel de compuertas lógicas.

La implementación de la operación α_g no ha sido realizada al momento de reportar este avance en el trabajo doctoral. Se planea su presentación correspondiente al momento de concluir la experimentación y generar la versión de tesis final.

4.3.2 Arquitectura Propuesta

El objetivo principal de la red neuronal Alfa-Beta sin pesos es poder llevar a cabo tareas de aprendizaje y recuperación de patrones de manera eficaz. Sin embargo, cuando la dimensionalidad de los datos es elevada, la fase de aprendizaje puede ser particularmente demandante. De ahí que, para lograr este tipo de tareas, es preciso contar con arquitecturas de cómputo especialmente diseñadas para ejecutar las operaciones necesarias de manera eficiente. En esta sección se presentan por separado las dos fases.

Fase de aprendizaje

La arquitectura de cómputo propuesta para la fase de aprendizaje, se muestra en la Figura 4.6. Consta de dos registros R_1 y R_2 , los cuales almacenan un patrón de entrada y uno de salida respectivamente; es decir, R_1 contiene el patrón x^μ , mientras que R_2 contiene el patrón y^μ .

Un aspecto interesante de toda implementación en hardware, es la posibilidad de poder ejecutar operaciones en paralelo.

Particularmente, son dos aspectos fundamentales, los que hacen posible instanciar múltiples unidades Alfa funcionando en paralelo:

- El número de compuertas lógicas necesarias para la implementación en hardware de un bloque Alfa (el cual, para el caso presente, es igual a ocho).
- El número de compuertas disponibles en un FPGA, el cual varía de acuerdo a la serie (Spartan y/o Virtex de Xilinx) y al componente dentro de dicha serie.

Los registros R_1 , R_2 y R_3 funcionan como argumentos de entrada del bloque de operaciones $\beta(\alpha, \beta)$. Los registros $p_{ij}(\mu)$ y $q_{ij}(\mu)$ mantienen los datos que resultan de la operación de la fase de aprendizaje controlada por el bloque $\beta(\alpha, \beta)$ y entregan los resultados a las rutas de datos que se dirigen a los arreglos de memoria $P_{k \times n}$ y $Q_{m \times k}$ respectivamente, mientras que los registros $p_{ij}(\mu-1)$ y $q_{ij}(\mu-1)$ mantiene el dato proveniente de memoria.

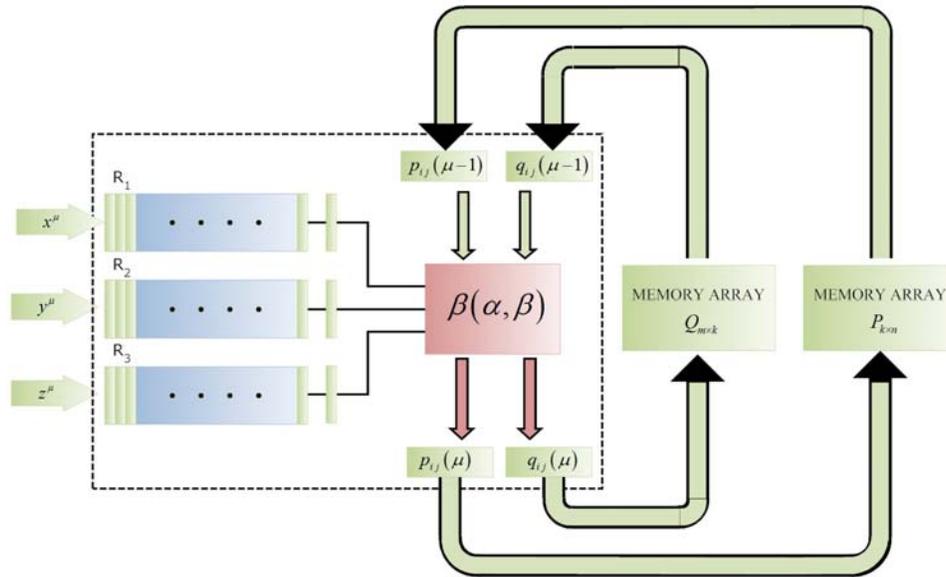


Figura 4.6 Arquitectura Propuesta para la Fase de Aprendizaje.

La conclusión general es que, la arquitectura propuesta para la fase de aprendizaje, permite aprender n veces más rápido que el modelo propuesto en [20], siempre que sea posible instanciar n unidades $\beta(\alpha, \beta)$ operando en paralelo.

Fase de recuperación

La arquitectura de cómputo propuesta para la fase de recuperación, se muestra en la Figura 4.7. Consta de los registros R_1 , y R_2 los cuales sirven como argumentos de entrada para el modulo de operación σ_β . R_2 contiene el patrón x^μ , mientras que R_1 contiene una línea de la matriz de memoria que contiene los patrones que fueron obtenidos durante la fase de aprendizaje.

Las operaciones de umbral y de obtención del vector de salida y^μ se realizan por medio de los bloques α_g , **MAX** y z^ω . Particularmente, son dos aspectos fundamentales, los que hacen posible instanciar múltiples unidades σ_β funcionando en paralelo:

- El número de compuertas lógicas necesarias para la implementación en hardware de un bloque σ_β .
- El número de compuertas disponibles en un FPGA, el cual varía de acuerdo a la familia o serie empleada (Spartan y/o Virtex de Xilinx, cuyas especificaciones técnicas se presentan en los apéndices B y C) y al componente dentro de dicha serie.

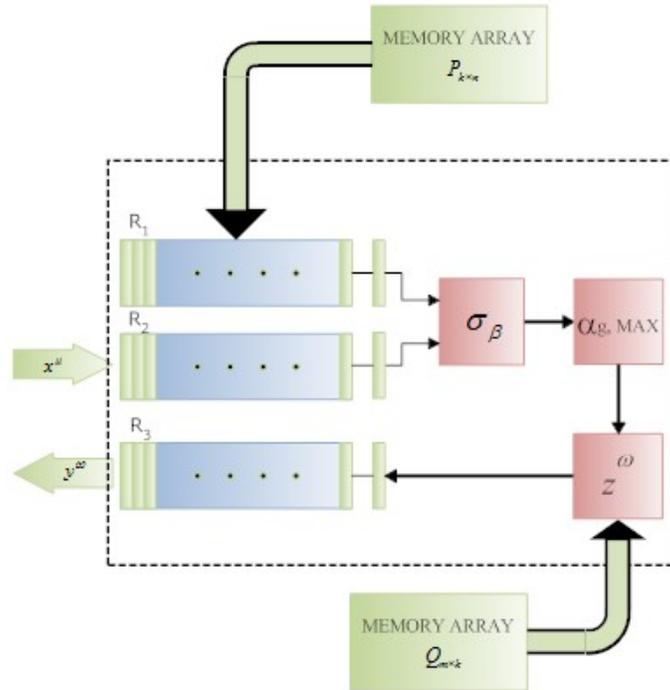


Figura 4.7 Arquitectura Propuesta para la Fase de Recuperación.

La conclusión general es que, la arquitectura propuesta para la fase de recuperación, permite recuperar n veces más rápido que el modelo propuesto en [18], siempre que sea posible instanciar n unidades Beta operando en paralelo.

4.4 Factibilidad de implementación

Un algoritmo muestra su eficiencia al proporcionar una o varias soluciones correctas acerca de un problema. Una medida de la eficiencia de un algoritmo es el tiempo de cómputo empleado (complejidad temporal) para resolver el problema, al aplicar valores de entrada con un tamaño específico. Otra medida empleada en valorar la eficiencia de un algoritmo es la cantidad de memoria utilizada (complejidad espacial) cuando los valores de entrada presentan un tamaño determinado [81].

Un **circuito Booleano** se describe como una conexión de compuertas, cuyos valores presentes en sus entradas se propagan hacia las salidas, realizando el cómputo de acuerdo con las funciones contenidas en las compuertas. De manera similar, las redes neuronales Alfa-Beta sin pesos realizan el cómputo de los valores de entrada, propagando los resultados en cada etapa hacia la salida. Basado en el modelo descrito por Borodin [82], se presenta a continuación las definiciones formales que sirven para sustentar la descripción, operación y factibilidad de implementación de una red neuronal Alfa-Beta sin pesos.

Definición. Un circuito Booleano es una gráfica acíclica dirigida finita. Cada una de los vértices v contenidos en el circuito Booleano son de un tipo τ , donde:

$$\tau(v) \in \{\text{entradas}\} \cup B_0 \cup B_1 \cup B_2 \cup B_3$$

El **grado de incidencia de entrada o de salida** es el número de terminales de entrada o de salida de un vértice. Un vértice v con $\tau(v) \in B_i$, presenta un grado de incidencia de entrada igual a i . Un vértice v con $\tau(v) = \text{entradas}$ presenta un grado de incidencia de entrada igual a 0 y se le denomina **entrada**. Las entradas de κ se denotan por el conjunto ordenado de vértices (x_1, x_2, \dots, x_n) . Un vértice con grado de incidencia de salida igual a 0 se le conoce como **salida**. Las salidas de κ se denotan por el conjunto ordenado de vértices (y_1, y_2, \dots, y_m) .

Empleando la definición anterior, para cada red neuronal Alfa-Beta sin pesos se realiza el cómputo de las entradas con base en la siguiente definición:

Definición. Una red neuronal Alfa-Beta sin pesos κ con entradas (x_1, x_2, \dots, x_n) y salidas (y_1, y_2, \dots, y_m) realiza el cómputo de una función $f: \{0,1\}^n \rightarrow \{0,1\}^m$ y la forma en que lleva a cabo dicha tarea se describe a continuación. A la entrada x_i , donde $1 \leq i \leq n$, se le asigna un valor $\nu(x_i)$ de $\{0,1\}$, que representa el i -ésimo bit del argumento de la función. En cada neurona se asigna un valor $\nu(v) \in \{0,1\}$ que es el resultado de aplicar $\tau(v)$ al valor o los valores de las conexiones que se encuentran a la entrada de la neurona v . Al realizar la evaluación sobre la red neuronal, la función contiene el valor $(\nu(y_1), \nu(y_2), \dots, \nu(y_m))$, cuya salida y_j , donde $1 \leq j \leq m$, contribuye al j -ésimo bit de salida.

La definición proporciona una red neuronal que contiene un número fijo de terminales de entrada. Pero al utilizar la definición anterior, se limita a la red neuronal en su funcionamiento. Para superar esta restricción, se puede emplear un método que permita describir a las redes neuronales sin pesos y ajustar el número de terminales y nodos para un proceso en particular. En este trabajo de tesis se emplea el método de codificación estándar, el cual se describe a continuación.

Definición. La **codificación estándar** para una red neuronal Alfa-Beta sin pesos κ es una cadena presente en $\{0,1\}^*$, agrupada en **secuencias cuádruples** (v, η, e_1, e_2) para el caso de la operación α y **quíntuples** (v, η, e_1, e_2, e_3) para el caso de la operación β , acompañadas de secuencias $\langle\langle x_1, x_2, \dots, x_n \rangle\rangle$ y $\langle\langle y_1, y_2, \dots, y_m \rangle\rangle$. Se incluye en la codificación una numeración entera única y arbitraria de cada neurona presente en κ . Cada una de las secuencias cuádruples y quintuples hacen referencia a una neurona η con número v , y sus conexiones hacia otras neuronas se describen de la siguiente forma:

$$\eta \in \{entrada\} \cup B_0 \cup B_2 \cup B_3$$

Las entradas a la neurona v se enumeran de acuerdo a la operación a realizar (α ó β), el número v de la i -ésima entrada (donde $1 \leq i \leq n$) se define por x_i , mientras que la j -ésima salida (donde $1 \leq j \leq m$) se define por y_j .

Con la definición precedente, se indica que las redes neuronales Alfa-Beta sin pesos son objetos que se pueden generar y manipular de manera fácil y compacta. El ejemplo mostrado en la figura 4.8 servirá para indicar la forma en que se realiza la codificación estándar con un nivel de abstracción alto.

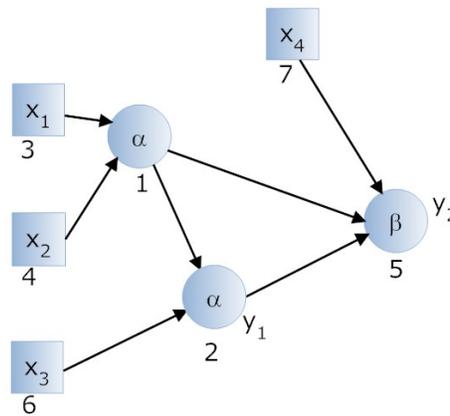


Figura 4.8. Ejemplo de una red neuronal Alfa-Beta sin pesos κ codificada

$$\langle \langle (3, entrada, -, -), (4, entrada, -, -), (6, entrada, -, -), (7, entrada, -, -) \\ (1, \alpha, 3, 4), (2, \alpha, 1, 6), (5, \beta, 7, 1, 2), \langle \langle 3, 4, 6, 7 \rangle \rangle, \langle \langle 2, 5 \rangle \rangle \rangle \rangle$$

La codificación obtenida en el ejemplo anterior se puede convertir a distintos tipos de descripción que permitan su implementación en software y/o hardware.

4.4.1 Recursos presentes en una red neuronal Alfa-Beta sin pesos

La figura 4.9 muestra gráficamente los recursos presentes en las redes neuronales Alfa-Beta sin pesos, capaces de cuantificarse y que permiten la medición de la complejidad. A continuación se presentan las definiciones que sirven para tal propósito.

Definición. Sea κ una red neuronal Alfa-Beta sin pesos. El **tamaño** de κ , $tmo(\kappa)$, es el número de neuronas presentes en κ .

El tamaño proporciona una medida de la cantidad de cómputo a realizar. Al considerar especificaciones de cómputo paralelo, el tamaño se relaciona con el número de operaciones.

Definición. Sea κ una red neuronal Alfa-Beta sin pesos. La **profundidad** de κ , $pdf(\kappa)$, es la longitud de la ruta más larga que comienza con una neurona de entrada y concluye con una neurona de salida presentes en κ .

La profundidad indica una medida del tiempo requerido para que la red neuronal realice el cómputo. Al considerar especificaciones de cómputo paralelo, la profundidad se relaciona directamente con el tiempo, ya que se puede observar a todas las neuronas “activándose” en paralelo tan pronto como las entradas están disponibles.

Definición. Sea κ una red neuronal Alfa-Beta sin pesos. El **ancho** de κ , $aco(\kappa)$, esta dado por

$$\max_{0 < i < \text{gro}} |\{v \mid 0 < l(v) \leq i \text{ y existe conexión entre la neurona } v \text{ y la } w, l(w) > i \}|$$

Donde $l(w)$, la profundidad de w , es la longitud del recorrido mas largo desde cualquier entrada a la neurona w .

El ancho es un recurso poco común. De manera intuitiva, el ancho corresponde al máximo número de neuronas presentes en un nivel.

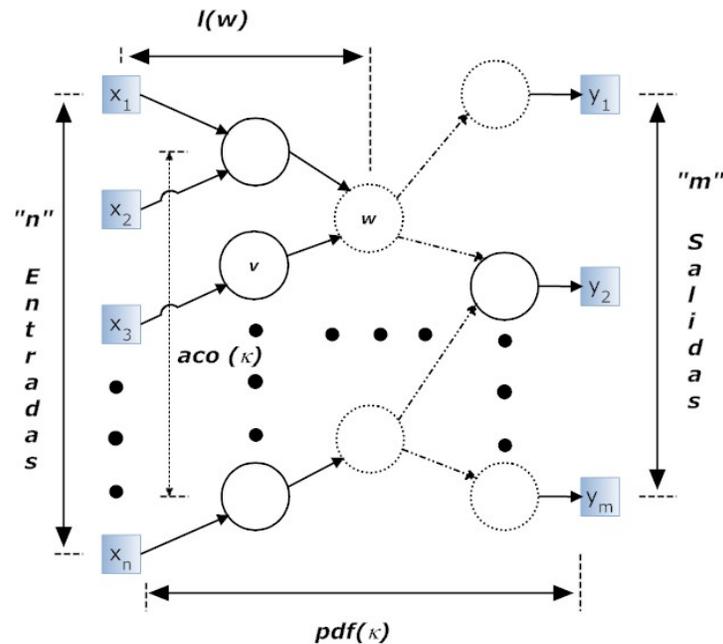


Figura 4.9. Arquitectura de una red neuronal Alfa-Beta sin pesos de tamaño $tmo(\kappa)$

Por lo anterior, se considera que una red neuronal Alfa-Beta sin pesos con n entradas y m salidas es un objeto finito que realiza el cómputo de una función a partir de cadenas binarias de longitud n a cadenas binarias de longitud m . Esto implica generar diferentes

redes neuronales Alfa-Beta sin pesos para entradas de longitud distinta. Con objeto de evitar lo anterior y poder aplicar las redes neuronales Alfa-Beta sin pesos a funciones con cadenas binarias de longitud arbitraria, la sección siguiente describe la manera de resolver dicha situación.

4.4.2 Familias de redes neuronales uniformes Alfa-Beta sin pesos

Considérese una red neuronal Alfa-Beta sin pesos con n entradas y m salidas, y que la longitud de m es una función (posiblemente constante) exclusiva de la longitud de n . Esto sugiere que solo se empleen aquellos casos de funciones donde la longitud de $f(x)$ es la misma para todas las entradas x de n -bits (a dicha longitud se le designa como $m(n)$). Bajo estas circunstancias, se puede representar a la función $f_\kappa : \{0,1\}^* \rightarrow \{0,1\}^*$ mediante el empleo de secuencias infinitas de redes neuronales $\{\kappa_n\}$, donde cada red neuronal κ_n realiza el cálculo de f con el número de entradas de longitud n restringido. Dicha secuencia recibe el nombre de familia de redes neuronales Alfa-Beta sin pesos.

Definición. Una familia de redes neuronales Alfa-Beta sin pesos $\{\kappa_n\}$ es un conjunto de redes neuronales Alfa-Beta sin pesos, donde cada κ_n realiza el cómputo de la función $f^n : \{0,1\}^n \rightarrow \{0,1\}^{m(n)}$. La función calculada por $\{\kappa_n\}$ y que se denota por f_κ es la función:

$$f_\kappa : \{0,1\}^* \rightarrow \{0,1\}^*$$

definida por $f_\kappa(x) \equiv f^{l(x)}(x)$.

Existe el caso especial donde la salida siempre presenta una longitud unitaria, y resulta de particular importancia para la definición de las clases de lenguaje.

Definición. Sea $\{\kappa_n\}$ una familia de redes neuronales Alfa-Beta sin pesos que genera el cómputo de la función $f_\kappa : \{0,1\}^* \rightarrow \{0,1\}$. El lenguaje L_κ , aceptado por $\{\kappa_n\}$, es el conjunto $L_\kappa = \{x \in \{0,1\}^* \mid f_\kappa(x) = 1\}$.

Al ubicarse en este punto, surge la pregunta de cómo describir una colección de redes neuronales Alfa-Beta sin pesos infinita. Al no contar con restricciones, se obtiene una familia de redes neuronales Alfa-Beta sin pesos no uniforme. Esta familia presenta la capacidad de calcular funciones no computables. Como ejemplo se puede considerar a $\{\kappa_n\}$, donde la red neuronal κ_n presenta un número n de entradas, donde todas ellas son incógnitas, y donde una neurona v de salida muestra una función constante. Dicha neurona se define como una función constante 1 si la n -ésima máquina de Turing se detiene en su propia descripción de programa, y se define como una función 0 en otro caso. De esta forma, la familia de redes neuronales Alfa-Beta sin pesos $\{\kappa_n\}$ realiza el cómputo de una función f_κ no decidible, y puede ser utilizada para resolver el problema de detención.

No existe un método para describir a los miembros de una familia de redes neuronales Alfa-Beta sin pesos no uniforme. Para resolver el problema, se puede proporcionar un algoritmo que genere a cada uno de los miembros de la familia. Esto es, se define a una familia de redes neuronales Alfa-Beta sin pesos mediante un programa, por ejemplo utilizando algún lenguaje de descripción de hardware, que tome n entradas y produzca una salida codificada $\tilde{\kappa}_n$ para el n -ésimo miembro. Al llevar a cabo lo anterior, se discretiza a la familia de redes neuronales Alfa-Beta sin pesos no uniforme y se describe en términos de un objeto finito.

Definición. Una familia de redes neuronales Alfa-Beta sin pesos $\{\kappa_n\}$ se describe como **uniforme** en un espacio logarítmico si la transformación $1^n \rightarrow \tilde{\kappa}_n$ se puede calcular como una máquina de Turing determinística en un espacio $O(\lg(n + \text{tamaño}(\kappa_n)))$.

Se puede observar que la complejidad de producir la descripción de κ_n se describe en términos del tamaño de la red neuronal resultante, en lugar de emplear el método convencional de expresar lo anterior con base en la longitud de la entrada. La codificación unaria de n asegura que, en el caso de las redes neuronales de tamaño polinomial, la complejidad también sea logarítmica en términos de la longitud de la entrada. Los conceptos de familia de redes neuronales Alfa-Beta sin pesos y de la uniformidad se emplean para sustentar la definición de la clase de complejidad paralela NC.

4.4.3 La Clase de Complejidad NC

Una red neuronal Alfa-Beta sin pesos puede aceptarse como un modelo de cómputo paralelo, ya que muestra su relación directa con el modelo de los circuitos Booleanos a través de las definiciones mencionadas previamente.

La pregunta básica realizada en el caso presentado aquí es: ¿Qué problemas se pueden resolver significativamente más rápido si se emplean varias familias de redes neuronales Alfa-Beta sin pesos en lugar de emplear una sola familia?

Nicholas Pippenger expresa de manera formal en [83], para el caso del cómputo paralelo, la clase de problemas que se pueden resolver a una gran velocidad empleando de forma viable cómputo paralelo con una determinada cantidad de hardware. La clase NC ha prevalecido de manera independiente a un determinado modelo de cómputo. De manera análoga para las redes neuronales Alfa-Beta sin pesos, tenemos la siguiente definición:

Definición. La Clase NC es el conjunto de todos los lenguajes L , tales que L es aceptado por una familia de circuitos Booleanos uniforme en espacio logarítmico que posee simultáneamente una profundidad $(\log n)^{O(1)}$ y tamaño $n^{O(1)}$.

Por lo previamente discutido, la familia de redes neuronales Alfa-Beta sin pesos $\{\kappa_n\}$ satisface la definición anterior y, por lo tanto, su complejidad.

CAPÍTULO 5

Resultados

Este capítulo es de vital importancia en el presente trabajo de tesis, puesto que no solo se ilustran los conceptos descritos en los capítulos anteriores, sino que además, se expone la eficacia de la red neuronal Alfa-Beta sin pesos, para recuperación y clasificación de patrones.

En la sección 5.1 se describen las principales características de cada una de las bases de datos que fueron usadas durante la fase experimental, así como las capacidades del equipo de cómputo utilizado.

En la sección 5.2, se presentan los resultados obtenidos en términos del índice de recuperación.

Asimismo, en la sección 5.3 se muestra el desempeño alcanzado, en términos del índice de clasificación.

5.1 Bases de datos y equipo utilizado

Para la realización de esta fase, fueron usados dos conjuntos de datos; tomados del repositorio de bases de datos de la Universidad de California en Irvine. En cada una de las bases de datos utilizadas, cada renglón representa un patrón. La primera columna contiene un identificador, que indica a que clase pertenece cada uno de estos. Las columnas subsecuentes, contienen información sobre las características de dichos patrones.

5.1.1 Iris Plants Database

Esta es tal vez la base de datos más conocida que se encuentra en la literatura del reconocimiento de patrones. El conjunto de datos contiene 3 clases de 50 instancias cada una, donde cada clase hace referencia a un tipo de planta de iris (Iris Setosa, Iris Versicolor e Iris Virginica). Cada patrón que representa una planta de Iris tiene 4 atributos mas uno que representa la clase. En esta base de datos los rasgos que representan el largo y ancho de los pétalos están altamente correlacionados.

Tabla 5.1. Características del primer conjunto de datos usado

Rasgos	Descripción
1	Clase: 1: Iris Setosa, 2:Iris Versicolor, 3:Iris Virginica
2	Largo del sépalo en centímetros
3	Ancho del sépalo en centímetros
4	Largo del pétalo en centímetros
5	Ancho del pétalo en centímetros

5.1.2 Contraceptive Method Choice Database (CMC)

El segundo conjunto de datos utilizado consiste en un subconjunto de la Encuesta Nacional de Control Anticonceptivo en Indonesia. Cada uno de los patrones (instancias), representa las características de una mujer casada, no embarazada al momento de la entrevista. Esta base de datos fue utilizada para predecir el uso de métodos anticonceptivos femeninos, tomando en cuenta sus características socio-económicas y demográficas. Los datos se agrupan en 3 clases: la clase 1 para quien no usa algún método con 629 patrones, la clase 2 para las mujeres que tienen un método a largo plazo con 333 patrones y la clase 3 para aquellas que llevan un método a corto plazo con 511 patrones, en total 1473 patrones.

Tabla 5.2. Características del segundo conjunto de datos usado

Rasgos	Descripción
1	Clase
2	Edad
3	Grado educativo
4	Grado educativo de la pareja
5	Número de niños ya concebidos
6	Religión
7	Trabajo
8	Ocupación de la pareja
9	Índice del estándar de vida
10	Riesgo de vida

5.1.3 Equipo de cómputo utilizado

Los experimentos se realizaron en una computadora personal (PC), con un procesador Intel Core2 Duo a 2.13 GHz, 1024 MBytes de memoria RAM y 78.4 GBytes de espacio en disco duro. Se usó el sistema operativo Windows XP Profesional de Microsoft y se utilizó un software diseñado e implementado en Borland C++ Builder 6.0, ex-profeso para este fin.

5.2 Recuperación de patrones

Los aspectos de interés que se tomaron en cuenta para realizar los experimentos son los mismos que se han registrado en la literatura concerniente a la recuperación de patrones, donde el punto de interés radica esencialmente en el porcentaje de recuperación correcta.

5.2.1 Recuperación del conjunto fundamental completo

La primera estimación del desempeño del modelo de red neuronal Alfa-Beta sin pesos se llevó a cabo aprendiendo y recuperando el conjunto fundamental completo. El proceso al cual se sometió dicho modelo es el siguiente: aprender el primer patrón y tratar de recuperarlo, aprender el segundo patrón e intentar recuperar los dos aprendidos, del mismo modo se continúa hasta haber aprendido la totalidad del conjunto fundamental.

Particularmente, para el primer conjunto de datos se realizaron 150 intentos de recuperación por experimento, mientras que para el segundo conjunto de datos se realizaron 1473 intentos de recuperación por experimento.

Para obtener una estimación confiable del índice de recuperación alcanzado con ambos conjuntos de datos, dicho procedimiento se llevó a cabo 50 veces para cada conjunto de prueba, seleccionando cada vez de manera aleatoria el orden de los patrones del conjunto fundamental.

Tabla 5.3. *Número de intentos de recuperación*

Iris	CMC
7500	73650

Al finalizar todo este proceso de aprendizaje y recuperación, se tienen un total de 7500 intentos de recuperación de patrones para el primer conjunto fundamental y 73650 intentos de recuperación para el segundo conjunto fundamental.

Los resultados expuestos en la Tabla 5.4, se obtienen del promedio de los intentos de recuperación, para cada conjunto de prueba.

Tabla 5.4. *Índice de Recuperación (conjunto completo)*

	Iris	CMC
ADAM	9 %	8 %
CAINN	100 %	100 %

Evidentemente, el desempeño del modelo propuesto en el presente trabajo de tesis, es claramente superior al alcanzado por el modelo ADAM. La representación gráfica de lo anteriormente dicho, se muestra tanto en la Figura 5.1 como en la Figura 6.1; poniendo de manifiesto la eficacia del modelo de red neuronal Alfa-Beta sin pesos para recuperar patrones.

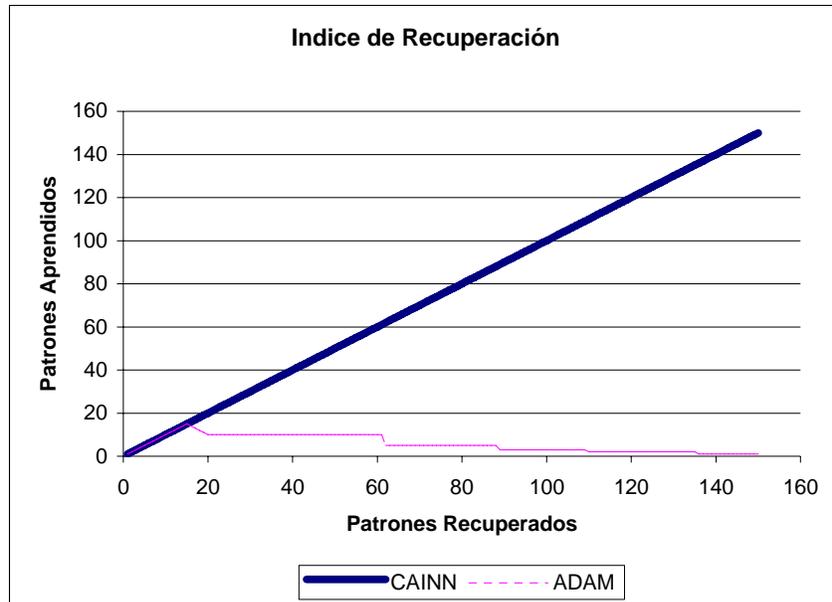


Figura 5.1 Índice de recuperación para el primer conjunto de datos.

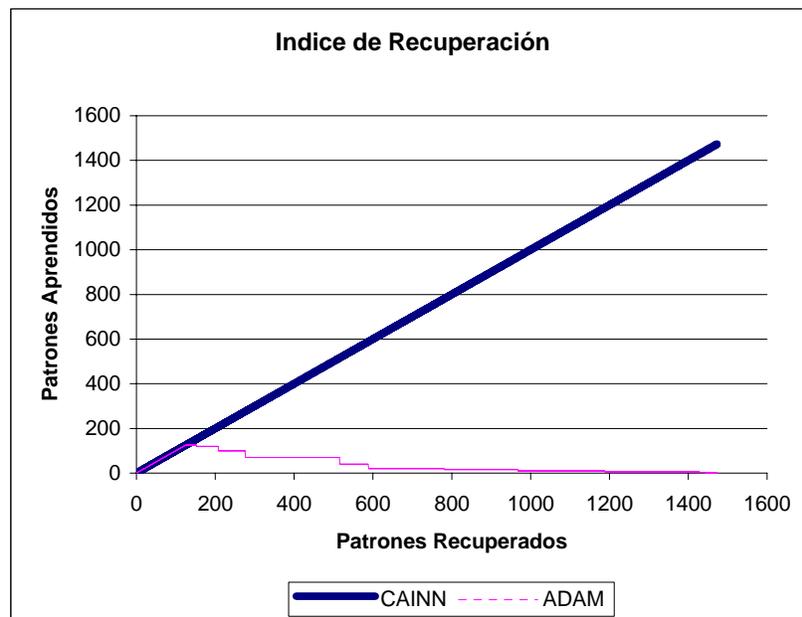


Figura 5.2 Índice de recuperación para el segundo conjunto de datos.

Se considera recuperación correcta cuando se recuperan todos los bits que conforman un patrón. De lo contrario, se considera recuperación no correcta.

Cabe mencionar que, para ambos conjuntos de datos, la red neuronal Alfa-Beta sin pesos, alcanza mejor desempeño en términos del índice de recuperación.

Asimismo, se puede observar que el modelo ADAM sufre de problemas de saturación temprana. Esto provoca que durante la fase de recuperación existan condiciones de ambigüedad en los patrones involucrados, haciendo imposible su recuperación.

5.2.2 Recuperación por particiones del conjunto fundamental

La segunda estimación del desempeño del modelo de red neuronal Alfa-Beta sin pesos se llevó a cabo aprendiendo el 70% del conjunto fundamental y recuperando 30 % restante.

Para obtener una estimación confiable del índice de recuperación alcanzado con ambos conjuntos de datos, dicho procedimiento se llevó a cabo 50 veces para cada conjunto de prueba, seleccionando cada vez de manera aleatoria los patrones involucrados en cada partición.

Tabla 5.5. Índice de Recuperación (por particiones 70 / 30)

	Iris	CMC
ADAM	17 %	12 %
CAINN	74 %	72 %

Los resultados expuestos en la Tabla 5.5, se obtienen del promedio de los intentos de recuperación, para cada conjunto de prueba.

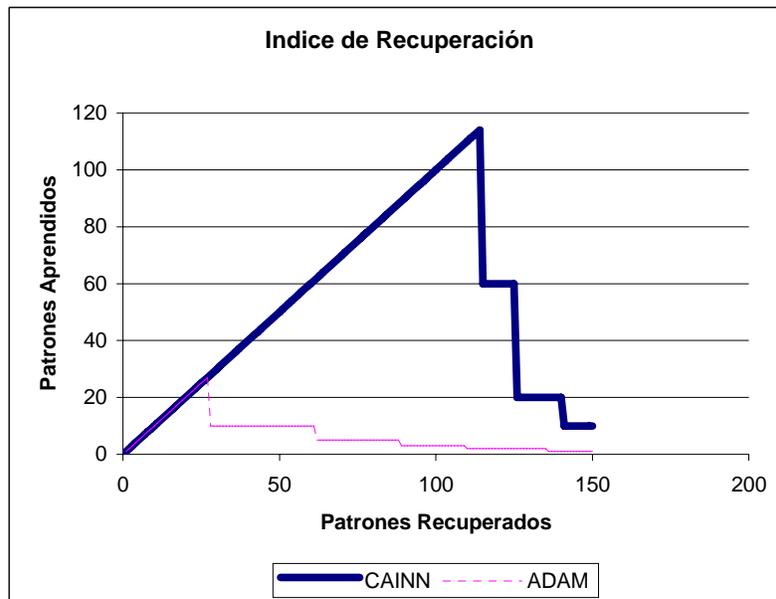


Figura 5.3 Índice de recuperación para el primer conjunto de datos. (70 / 30)

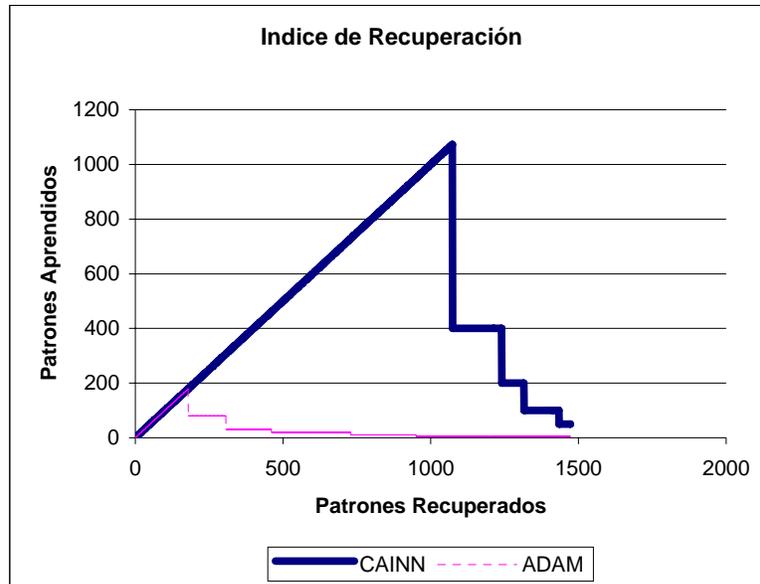


Figura 5.4 Índice de recuperación para el segundo conjunto de datos. (70 / 30)

5.3 Clasificación de patrones

Los aspectos de interés que se tomaron en cuenta para realizar los experimentos son los mismos que se han registrado en la literatura concerniente a la clasificación de patrones, donde el punto de interés radica esencialmente en el porcentaje de clasificación correcta.

5.3.1 Clasificación del conjunto fundamental completo

Para la realización de esta fase, fueron tomadas las mismas bases de datos. Cada uno de los experimentos se llevó a cabo de la siguiente manera: Primeramente, se tomaron de manera aleatoria el mismo número de patrones de entrada para cada clase; esto permite la obtención de un conjunto de prueba balanceado. Posteriormente, se tomó dicho conjunto de prueba para su clasificación.

Para obtener una estimación confiable del índice de clasificación alcanzado con ambos conjuntos de datos, dicho procedimiento se llevó a cabo 50 veces para cada conjunto de prueba.

Los resultados expuestos en la Tabla 5.6, se obtienen del promedio de los intentos de clasificación, para cada conjunto de prueba.

Tabla 5.6. Índice de Clasificación

	Iris	CMC
ADAM	70 %	65 %
CAINN	100 %	100 %

Cabe mencionar que, si bien, los índices de clasificación del modelo ADAM mejoraron con respecto a los resultados alcanzados durante el proceso de recuperación, el modelo propuesto en el presente trabajo de tesis nuevamente supera al modelo ADAM, al clasificar correctamente la totalidad del conjunto fundamental.

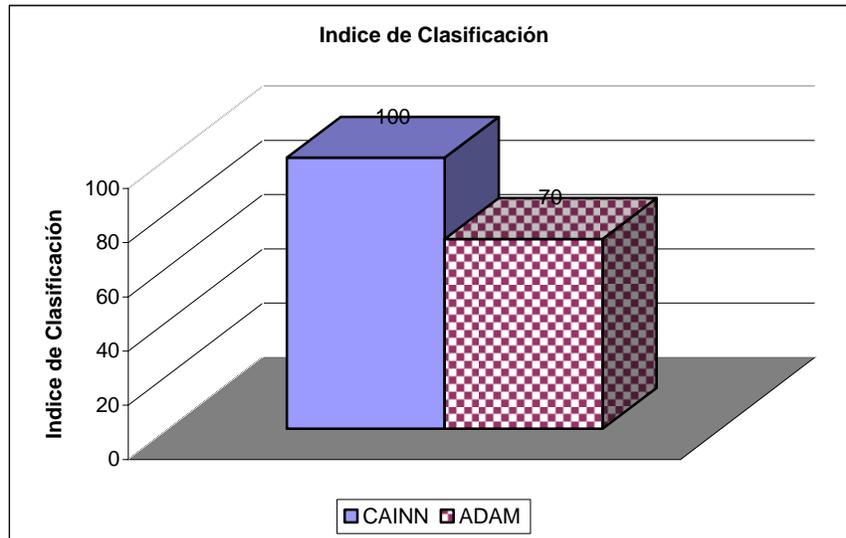


Figura 5.5 Índice de clasificación para el primer conjunto de datos.

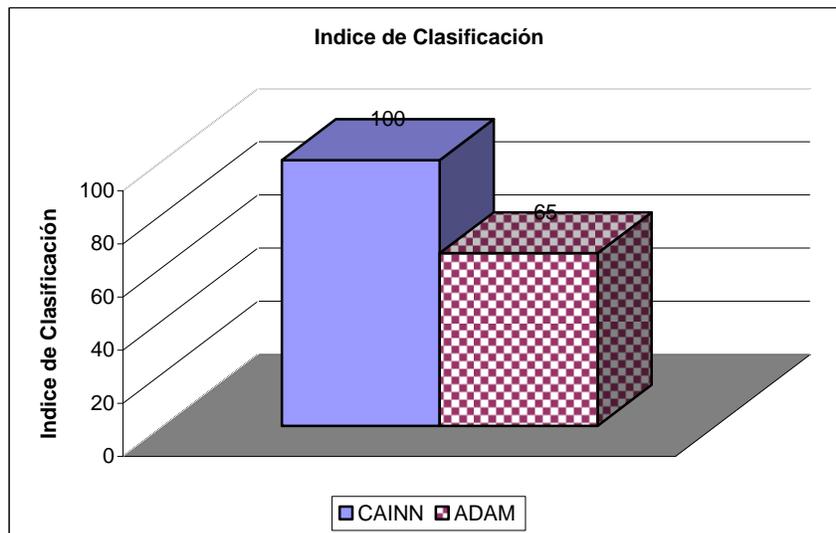


Figura 5.6 Índice de clasificación para el segundo conjunto de datos.

La representación gráfica de lo anteriormente dicho, se muestra tanto en la Figura 9 como en la Figura 10; poniendo de manifiesto la eficacia del modelo de red neuronal Alfa-Beta sin pesos para clasificar patrones.

5.3.2 Clasificación por particiones del conjunto fundamental

La cuarta estimación del desempeño del modelo de red neuronal Alfa-Beta sin pesos se llevó a cabo aprendiendo el 70% del conjunto fundamental y clasificando el 30 % restante.

Para obtener una estimación confiable del índice de clasificación alcanzado con ambos conjuntos de datos, dicho procedimiento se llevó a cabo 50 veces para cada conjunto de prueba, seleccionando cada vez de manera aleatoria los patrones involucrados en cada partición.

Tabla 5.7. Índice de Clasificación (por particiones 70 / 30)

	Iris	CMC
ADAM	58 %	42 %
CAINN	89 %	81 %

Los resultados expuestos en la Tabla 5.7, se obtienen del promedio de los intentos de clasificación, para cada conjunto de prueba.

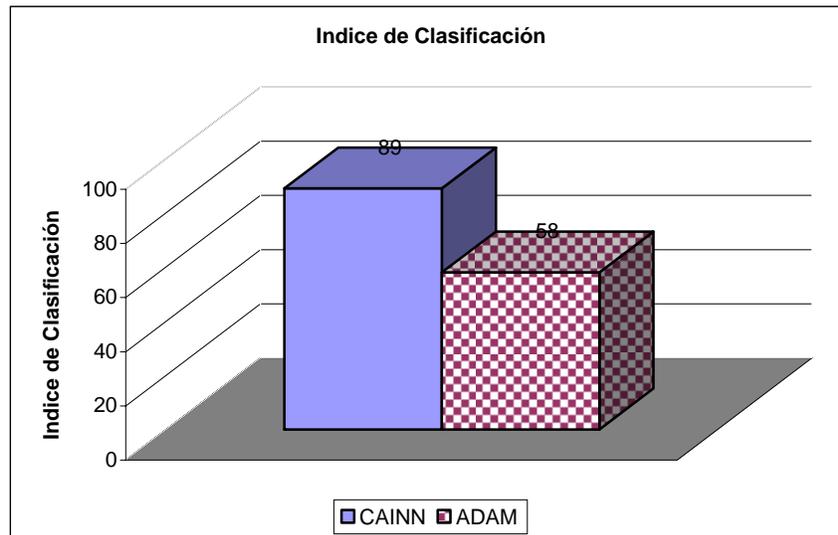


Figura 5.7 Índice de clasificación para el primer conjunto de datos. (70 / 30)

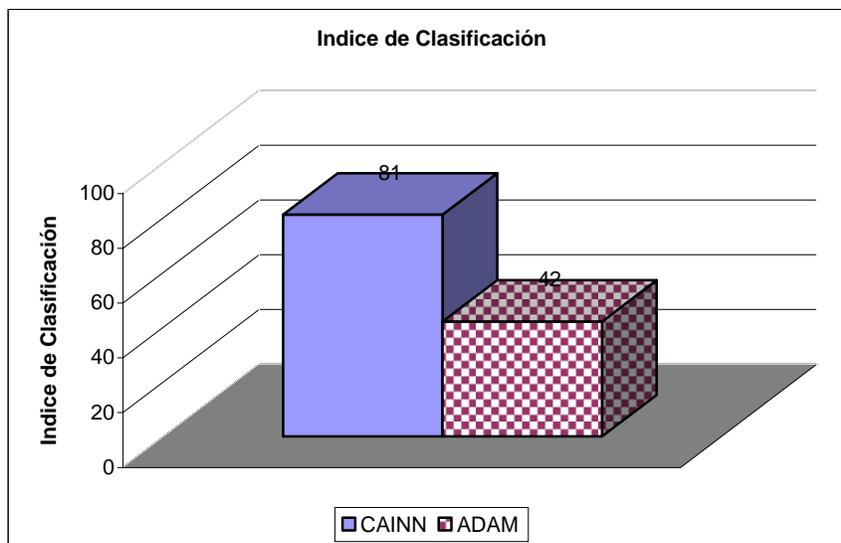


Figura 5.8 Índice de clasificación para el segundo conjunto de datos. (70 / 30)

Los resultados anteriores se pueden comparar con aquellos reportados en la literatura científica. La tabla 5.8 incluye los resultados mencionados arriba, así como los reportados en [106] para Iris, y en [107] y [108] para Iris y CMC.

Tabla 5.8. Comparación de Índices de Clasificación

	Iris	CMC
ADAM	58 %	42 %
CAINN	89 %	81 %
RAMP [106]	97 %	-
STEP [106]	77 %	-
SVM [106] [107]	96 %	54 %
KNN [106] [107]	96 %	48 %
MLP [106] [107]	95 %	53 %
RM [107]	97 %	55 %
C4.5 [108]	94 %	63 %
C4.5+m [108]	93 %	66 %

Con respecto a la base de datos de *Iris Plant*, es claro que CAINN supera ampliamente a ADAM en cuanto al índice de clasificación, superando también a la red neuronal STEP, aunque ante otros clasificadores presenta tan buen desempeño (RAMP, SVM, *k*NN, MLP, RM, C4.5 y C4.5+m), aunque se aproxime a sus resultados. Esta desventaja mostrada con respecto al último conjunto de clasificadores es resarcida en la base de datos CMC, en donde supera a todos los clasificadores incluidos en la tabla por un amplio margen (15% de diferencia con C4.5+m, el clasificador cuyos resultados más se aproximan a los de CAINN).

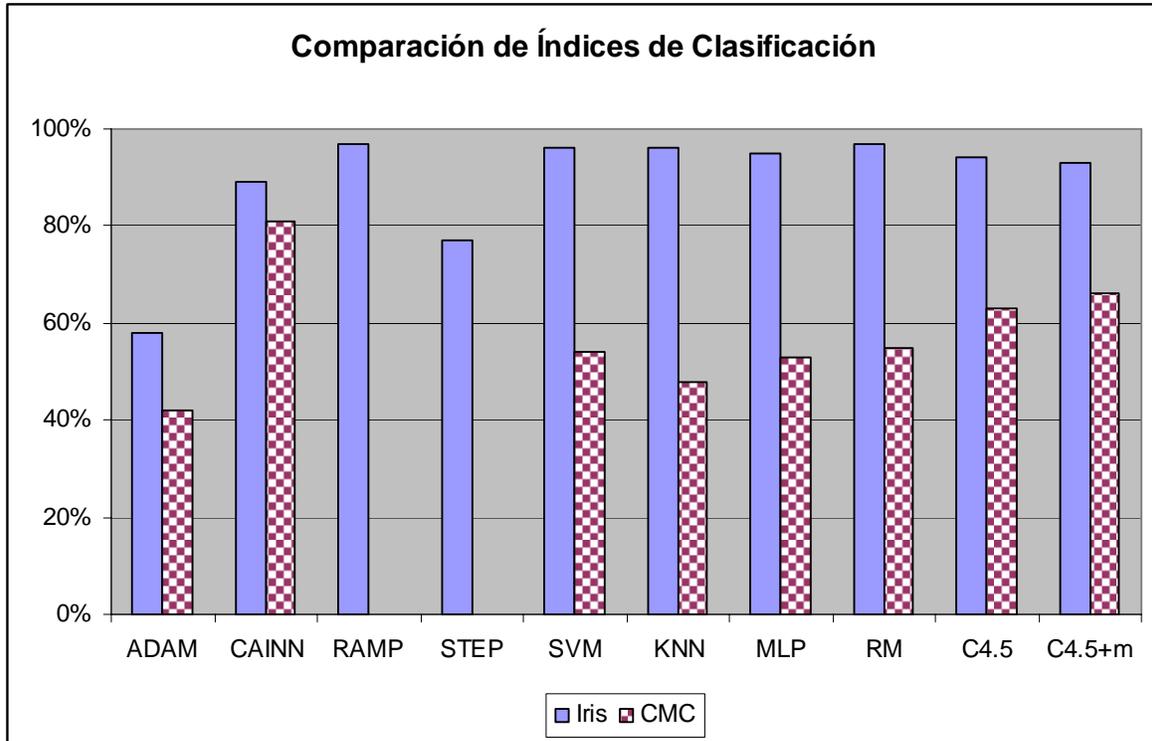


Figura 5.9 Comparación del Índice de Clasificación con otros clasificadores

Este comportamiento no tiene nada de extraño, pues el Teorema de *No Free Lunch* [109-111] indica que cuando un algoritmo (de clasificación) es muy bueno con cierta familia de problemas, debe ser no tan bueno con otras.

Asimismo, cabe mencionar que todos los algoritmos presentes en la tabla 5.8 que muestran un mejor desempeño que CAINN con respecto a la base de datos *Iris Plant*, tienen también una mayor complejidad, varios de ellos siendo incluso iterativos. Así pues, a pesar de ofrecer un índice de clasificación menor que dichos clasificadores (aunque por un margen de alrededor del 5 %), CAINN ofrece las ventajas de ser *one-shot* y relativamente más simple.

CAPÍTULO 6

Conclusiones, aportaciones y trabajo futuro

En este capítulo se presentan las conclusiones derivadas de los resultados obtenidos en el proceso de este trabajo de tesis, y se incluyen las aportaciones teóricas y técnico-científicas obtenidas como resultado de la actividad de investigación sobre el tema. Además, se proponen algunos de los trabajos que se podrían realizar con objeto de continuar con las ideas propuestas aquí, dando la pauta a futuros investigadores sobre los puntos no cubiertos, y que pudieran ser afrontados en otros trabajos de investigación.

6.1 Conclusiones

1. Se presenta el algoritmo de la nueva red neuronal Alfa-Beta sin pesos denominado CAINN (Computing Artificial Intelligent Neural Network), en ambas fases: aprendizaje y recuperación de patrones.
2. Se realiza el diseño e implementación en FPGAs de las operaciones Alfa, Beta, Alfa Generalizada, Sigma-Alfa y Sigma-Beta. Estos diseños sirven de elementos constitutivos de la arquitectura hardware para CAINN.
3. Se establecen formalmente las condiciones necesarias y suficientes de equivalencia entre las redes neuronales sin pesos y los circuitos booleanos, y un estudio de la factibilidad de implementación de CAINN.
4. La Red Neuronal Alfa-Beta sin pesos permite recuperar de forma correcta el conjunto fundamental completo siempre que se presente a la entrada de cada memoria patrones sin alteraciones.
5. El desempeño mostrado por el modelo propuesto, al realizarse los experimentos y estudios comparativos con las bases de datos Iris Plants y CMC, es claramente superior al alcanzado por el modelo ADAM y otros modelos.
6. Se ha empleado la lógica reconfigurable para la integración en hardware de las operaciones Alfa y Beta del modelo CAINN.

6.2 Aportaciones

Las tareas de investigación relacionadas con las redes neuronales Alfa-Beta sin pesos ofrecen diversas contribuciones, las cuales se describen brevemente a continuación:

6.2.1 Aportaciones teóricas

1. Como aporte teórico original en esta tesis, se crea la definición de tres nuevas operaciones: α_g , σ_α y σ_β que sirven de base para el diseño e implementación del producto principal: el modelo CAINN.
2. Se establecen los algoritmos de ambas fases para CAINN: aprendizaje y recuperación de patrones.
3. Se establecen formalmente las condiciones necesarias y suficientes de equivalencia entre las redes neuronales sin pesos y los circuitos booleanos, y un estudio de la factibilidad de implementación de CAINN.

6.2.2 Aportaciones técnico-científicas

1. Se propone la arquitectura que mantiene la red neuronal Alfa-Beta sin pesos, implementada en un dispositivo de arreglo de compuertas programables en campo (FPGA).
2. Las bases de datos Iris plant y CMC fueron empleadas para la revisión y comparación de la operación del modelo de red neuronal propuesta con el modelo ADAM, ofreciendo resultados que exceden varios ordenes de magnitud a su predecesor.
3. Se presenta un modelo que realiza tareas de recuperación, al ser aplicado en las bases de datos Iris plant y CMC. Aunque el propósito inicial de la red neuronal fue el de recuperar patrones que fueron previamente aprendidos, se observa que el comportamiento de la red neuronal al momento de clasificar es aceptable, por lo que puede también emplearse en procesos donde la clasificación es requerida.

6.3 Trabajo futuro

1. Modificar los operadores que dan lugar a las redes neuronales Alfa-Beta sin pesos, para que soporten patrones que presenten alteraciones (ruido).

2. Probar de manera más extensiva y exhaustiva el algoritmo propuesto, mediante el empleo de otras bases de datos, y comparar los resultados obtenidos con la red neuronal Alfa-Beta sin pesos y otros clasificadores de patrones.
3. Con base en el nuevo modelo de redes neuronales Alfa-Beta sin pesos presentado en esta tesis, generar trabajos de estudio e implementación de diversas aplicaciones, entre las cuales se pueden citar aquellas relacionadas con predicción medio-ambiental, medicina deportiva, entre otros.
4. Generar mediante herramientas de software un generador de núcleos de redes neuronales sin pesos, el cuál presente entradas que permitan proporcionar los parámetros necesarios para que la red neuronal Alfa-Beta sin pesos pueda ser dimensionada y reutilizada. Esto a su vez sirve como la plataforma para generar los bloques de construcción básicos de redes neuronales de mayor tamaño, las cuales contendrán en principio bloques Alfa-Beta, pero que puede emplearse para poder realizar modificaciones y/o adecuaciones de otros tipos de redes neuronales que utilicen el mismo principio de reproducibilidad. Esto representa realizar núcleos con propiedad intelectual.

Apéndice A

Simbología

M	memoria asociativa
x	vector columna que corresponde a un patrón de entrada
y	vector columna que corresponde a un patrón de salida
(x, y)	asociación de un patrón de entrada con uno de salida
(x^k, y^k)	asociación de la <i>k</i> -ésima pareja de patrones
{(x^μ, y^μ) μ = 1, 2, ..., p}	conjunto fundamental
A	conjunto al que pertenecen los vectores x y y
B	conjunto al que pertenecen las entradas de la matriz M
p	número de parejas del conjunto fundamental
n	dimensión de los patrones de entrada
m	dimensión de los patrones de salida
∀	cuantificador universal
∈	pertenencia de un elemento a un conjunto
∃	cuantificador existencial
y^μ_k	<i>k</i> -ésima componente del vector columna y^μ
×	producto cruz (entre conjuntos)
m_{ij}	<i>ij</i> -ésima componente de la matriz M
Δm_{ij}	incremento en <i>m_{ij}</i>
·	producto usual entre vectores o matrices
∨	operador máximo
(x^μ)^t	Transpuesto del vector x^μ
δ_{ij}	delta de Kronecker (afecta a los índices <i>i</i> y <i>j</i>)
I	matriz identidad
x̃	versión alterada del patrón fundamental x
W	memoria asociativa morfológica <i>min</i>
∇	producto máximo
Δ	producto mínimo
∧	operador mínimo
α y β	operadores en que se basan las memorias Alfa-Beta
α(x, y)	operación binaria <i>α</i> con argumentos <i>x</i> y <i>y</i>
β(x, y)	operación binaria <i>β</i> con argumentos <i>x</i> y <i>y</i>
∇_α	operador <i>α max</i>
∇_β	operador <i>β max</i>
Δ_α	operador <i>α min</i>
Δ_β	operador <i>β min</i>
⊗	símbolo que representa a las dos operaciones ∇_α y Δ_α
V	memorias asociativas Alfa-Beta tipo <i>max</i>
Λ	memorias asociativas Alfa-Beta tipo <i>min</i>
Z⁺	conjunto de los números enteros positivos
h^k	<i>k</i> -ésimo vector <i>one-hot</i>

κ	Red neuronal Alfa-Beta sin pesos
v	Vértice contenido en una gráfica acíclica dirigida finita
$\tau(v)$	Operación a realizar por parte de la neurona v
B_i	Grado de incidencia de entrada en una neurona
(x_1, x_2, \dots, x_m)	Conjunto ordenado de vértices de entrada
(y_1, y_2, \dots, y_m)	Conjunto ordenado de vértices de salida
(v, η, e_1, e_2)	Secuencia empleada en la codificación estándar de la operación α
(v, η, e_1, e_2, e_3)	Secuencia empleada en la codificación estándar de la operación β
$tmo(\kappa)$	Tamaño de la red neuronal Alfa-Beta sin pesos κ
$pdf(\kappa)$	Profundidad de la red neuronal Alfa-Beta sin pesos κ
$aco(\kappa)$	Ancho de la red neuronal Alfa-Beta sin pesos κ
$\{\kappa_n\}$	Familia de redes neuronales alfa-Bata sin pesos
L_κ	Lenguaje aceptado por $\{\kappa_n\}$

Apéndice B
Virtex-II Pro and Virtex-II Pro X
Platform FPGAs

Apéndice C
Spartan-3 FPGA Family

Referencias

1. Enquist, M. & Ghirlanda, S. (2005). *Neural Networks and Animal Behaviour*. (1st edition) Princeton: Princeton University Press.
2. Boahen, K. (2006). Neuromorphic Microchips. *Scientific American - Secrets of the Senses: How the brain deciphers the world around us. Special Edition*, 16,3, 20-27.
3. Braspenning, P. J., Thuijsman, F. & Weijters, A. J. (1995). *Artificial Neural Networks: An Introduction to ANN Theory and Practice*. (vols. 931/1995) Utrecht: Springer Berlin / Heidelberg.
4. McCulloch, W. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 115-133.
5. Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65, 6, 346-408.
6. Kohonen, T. (1989). *Self-Organization and Associative Memory*. Berlin: Springer-Verlag.
7. Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences of the USA* (pp. 2554-2558).
8. Ritter, G. X. & Sussner, P. (1996). An Introduction to Morphological Neural Networks. In *13th International Conference on Pattern Recognition* (pp. 709-717).
9. Bledsoe, W.W. & Browning, I. (1959). Pattern Recognition and Reading by machine. In *Proc. Eastern Joint Computer Conference* (pp. 225-232).
10. Morciniec, M. & Rohwer, R. (1995). *The n-tuple Classifier: Too Good to Ignore*. Birmingham, UK.
11. Austin, James (1998). RAM based neural networks, a short history. In *RAM-Based Neural Networks*, (1st. ed., pp. 3-17). World Scientific Publishing Co. Pte. Ltd.
12. Figueroa, J., Vazquez, A. & Vargas, E. (1992). 3-Dimensional Pattern Recognition using Weightless Neural Network (Aleksander's model). In *International Joint Conference on Neural Networks* (pp. 380-385). IEEE
13. Ullmann, J. R. (1969). Experiments with the n-tuple Method of Pattern Recognition. *Computers, IEEE Transactions on*, C-18, 12, 1135-1137.

14. Argüelles, A. J. *et. al.* (2005). Pattern recognition and classification using weightless neural networks and Steinbuch Lernmatrix. In SPIE Optics and Photonics 2005 : SPIE, pp. (59160)P1-P8.
15. Bishop, J. M. & Mitchell, R. J. (1994). Auto-associative memory using n-tuple techniques. *Intelligent Systems Engineering*, 3, 4, 222-229.
16. Austin, J. (1988). *Grey scale n-tuple processing*. Berlin: Springer-Verlag.
17. Hassoun, M. H. (1993). *Associative Neural Memories: Theory and Implementation*. Oxford University Press, USA.
18. Yáñez-Márquez, C. (2002). *Memorias Asociativas basadas en Relaciones de Orden y Operadores Binarios*. Tesis de Doctorado, Centro de Investigación en Computación, Mexico City.
19. Acevedo, M. E. (2006). *Memorias Asociativas Bidireccionales Alfa-Beta*. Tesis de Doctorado, Centro de Investigación en Computación, Mexico City.
20. Acevedo, M. E., Yáñez, C. & López, I. (2006). A New Model of BAM: Alpha-Beta Bidirectional Associative Memories. In *Lecture Notes in Computer Science*, Berlin: Springer Verlag, pp. 286-295.
21. Acevedo, M. E., Yáñez C. & López, I. (2006). Alpha-Beta Bidirectional Associative Memories Based Translator. *IJCSNS International Journal of Computer Science and Network Security*, 6, 5A, pp. 190-194.
22. Yáñez-Márquez, C., Cruz-Meza, M.E., Sánchez-Garfias, F.A. & López-Yáñez, I. (2007). *Using Alpha-Beta Associative Memories to Learn and Recall RGB Images*, *Lecture Notes in Computer Science*, LNCS 4493, Springer-Verlag Berlin Heidelberg, pp. 828-833. ISBN: 978-3-540-72394-3.
23. Acevedo-Mosqueda, M.E., Yáñez-Márquez, C. & López-Yáñez, I. (2007). *Alpha-Beta Bidirectional Associative Memories: Theory and Applications*, *Neural Processing Letters*, Vol. 26, No. 1, August 2007, Springer-Verlag Berlin Heidelberg, pp. 1-40. ISSN: 1370-4621.
24. Yáñez, C., Sanchez, L. P. & López, I. (2006). Alpha-Beta Associative Memories for Gray Level Patterns. In J. Wang, Z. Yi, J. M. Zurada, B.-L. Lu, & H. Yin (Eds.) *Advances in Neural Networks - ISNN 2006*, Chengdu: Springer Berlin / Heidelberg, pp. 818-823.
25. Guccione, S. A. & Gonzalez, M. J. (1994). Neural network implementation using reconfigurable architectures. In *Selected papers from the Oxford 1993 international workshop on field programmable logic and applications on More FPGAs*, Oxford : Abingdon EE&CS Books, pp. 443-451.

26. Porrmann, M. *et. al.* (2002). Implementation of artificial neural networks on a reconfigurable hardware accelerator. In Proceedings.10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, pp. 243-250.
27. Restrepo, H. F. *et. al.* (2000). A networked FPGA-based hardware implementation of a neural network application. In IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 337-338.
28. Zhu, J., Milne, G. J. & Gunther, B. K. (1999). Towards an FPGA based reconfigurable computing environment for neural network implementations. In 9th International Conference on Artificial Neural Networks (Conf.Publ.No.470), pp. 661-666.
29. Hodge, V. & Austin, J. (2000). An Evaluation of Standard Retrieval Algorithms and a Weightless Neural Approach, IEEE-INNS-ENNS International Joint Conference on Neural Networks, pp. 591-596.
30. Ritter, G. X., Sussner, P. & Diaz de Leon, J. L. (1998). Morphological associative memories. *IEEE Transactions on Neural Networks*, 9, pp. 281-293.
31. Hebb, D. O. (1949). *Organization of Behaviour*. Wiley.
32. Aleksander, I. & Stonham, T. J. (1979). Guide to pattern recognition using random-access memories. *Computers and Digital Techniques*, 2, 29-40.
33. Austin, J. (1986). *The Design and Application of Associative Memories for Scene Analysis*, YCST 87/06, University of York, UK.
34. Bledsoe, W. W. (1952). Some Aspects of Covering Theory. *Proceedings of the American Mathematical Society*, 3(5), 804-812.
35. Aleksander, I., W.V.Thomas & P.A.Bowden (1984). WISARD, a Radical Step Forward in Image Recognition. *Sensor Review*, 4, 3, 120-124.
36. Ludermir, T. B. *et. al.* (1999). Weightless Neural Models: A Review of Current and Past Works. *Neural Computing Surveys*, 2, 41-61.
37. Rohwer, R. & Lamb, A. (1993). An exploration of the effect of super large n-tuple on single layer RAMnets. In Proceedings of the Weightless Neural Networks Workshop 1993, pp. 33-37.
38. Tarling, R. & Rohwer, R. (1993). Efficient use of training data in the n-tuple recognition method, *Electronics Letters*, vol. 29, no. 24, pp. 2093-2094.
39. Ullmann, J. R. & Kidd, P. A. (1969). Recognition experiments for typed numeral from the envelopes in the mail. *Pattern Recognition*, 1, pp. 273-289.
40. Rohwer, R. & Morciniec, M. (1995). A theoretical and experimental account of n-tuple classifier performance. *Neural Computing*, 8, pp. 629-642.

41. Al-Alawi, R. (2003). FPGA Implementation of a Pyramidal Weightless Neural Networks Learning System. *International journal of Neural Systems*, 13, 4, pp. 225-237.
42. Aleksander, I. (1989). Canonical neural nets based on logic nodes. In First IEE International Conference on Artificial Neural Networks, London : IEE, pp. 110-114.
43. Kan, W. K. & Aleksander, I. (1987). A probabilistic logic neuron network for associative learning. In Proceedings of the IEEE First International Conference on Neural Networks (ICNN87), pp. 541-548.
44. Myers, C. E. & Aleksander, I. (1989). Output functions for probabilistic logic nodes. In First IEE International Conference on Artificial Neural Networks, pp. 310-314.
45. Gorse, D. & Taylor, J. G. (1988). On the equivalence properties of noisy neural and probabilistic RAM nets. *Physics Letters A*, 131, 6, pp. 326-332.
46. Kohonen, T. (1987). *Content-Addressable Memories*, Berlin: Springer-Verlag.
47. Kohonen, T. (1995). *Self-Organizing Maps*. (3rd. ed.) (vols. 30) Springer.
48. Yáñez Márquez, C. & Díaz-de-León Santiago, J.L. (2003). Memorias Asociativas Basadas en Relaciones de Orden y Operaciones Binarias, *Computación y Sistemas*, Vol. 6, No. 4, México, pp. 300-311. ISSN 1405-5546.
49. Simpson, P. K. (1990). *Artificial Neural Systems*. New York: Pergamon Press.
50. Steinbuch, K. V. (1961). Die Lernmatrix. *Kybernetik*, 1, 1, pp. 36-45.
51. Steinbuch, K. V. & Helmab, F. (1961). Nichtdigitale lernmatrizan als perzeptoren. *Kybernetik*, 1, 3, pp. 117-124.
52. Kohonen, T. (1972). Correlation matrix memories. *IEEE Transaccions on computers*, 4,21, pp. 353-359.
53. Papadomanolakis, K. et al. (2002). A Fast Johnson-Mobius Encoding Scheme for Fault Secure Binary Counters. In Proceedings of Design, Automation and Test in Europe (DATE'02), pp. 1-7.
54. Willshaw, D. & Buneman, O. (1969). Non-holographic associative memory. *Nature*, 222, pp. 960-962.
55. Anderson, J. A. (1972). A simple neural network generating an interactive memory. *Mathematical Bioscience*, 14, pp. 197-220.
56. Anderson, J. A. & Rosenfeld, E. (1989). *Neurocomputing: Foundations of Research*. Cambridge: The MIT Press.

57. Abu-Mostafa, Y. & Jacques, J. (1985). Information capacity of the Hopfield model. *IEEE Transactions on Information Theory*, 31, 4, pp. 461-464.
58. Yáñez, C. & Diaz de Leon, J. L. (2001). *Memorias Morfológicas Autoasociativas*. (Rep. No. IT-58). Mexico: CIC-IPN.
59. Yáñez, C. & Diaz de Leon, J. L. (2001). *Memorias Morfológicas Heteroasociativas*. (Rep. No. IT-57). Mexico: CIC-IPN.
60. Omondi, A. R. & Rajapakse, J. C. (2006). *FPGA Implementations of Neural Networks*. (1st. ed.) Dordrecht, The Netherlands: Springer Verlag.
61. Smith, M. (1997). *Application-Specific Integrated Circuits*. (1st. ed.) Addison-Wesley Professional.
62. Maxfield, C. (2004). *The Design Warrior's Guide to FPGAs*. Newnes Elsevier.
63. Actel (2005). *Programming antifuse devices (Application note)*.
64. Austin, J. (1987). ADAM: A Distributed Associative Memory for Scene Analysis. In Proceedings of First International Conference on Neural Networks, San Diego, CA. M. Caudhill & C. Butler (Eds.), pp. 285-295.
65. Rosen, K. H. (1999). *Discrete Mathematics and Its Applications*. McGraw-Hill Science/Engineering/Math.
66. Guccione, S. A. (2001). Reconfigurable computing at Xilinx. In Proceedings. Euromicro Symposium on Digital Systems Design, pp. 102.
67. Cox, C. E. & Blanz, W. E. (1992). GANGLION-a fast field-programmable gate array implementation of a connectionist classifier. *IEEE Journal of Solid-State Circuits*, 27, 3, pp. 288-299.
68. Boukerche, A. & Notare, M. S. M. (2001). Applications of Neural Networks to Mobile Communication System. In A. Y. Zomaya, F. Ercal, & S. Olariu (Eds.) *Solutions to Parallel and Distributed Computing Problems. Lessons from Biological Sciences*, (1st. ed.). John Wiley & Sons, Inc., pp. 255-268.
69. Chang-Min, K. *et. al.* (2003). FPGA implementation of ICA algorithm for blind signal separation and adaptive noise canceling. *IEEE Transactions on Neural Networks*, 14, 5, pp. 1038-1046.
70. Aldape, M. (2007). *Implementación de los modelos Alfa-Beta con lógica reconfigurable*. Tesis de Maestría, Centro de Investigación en Computación, México.
71. Aldape, M., Yáñez, C. & Arguelles, A. J. (2007). Optimized Associative Memories for Feature Selection. In Pattern Recognition and Image Analysis, Girona, Spain : Springer. Joan Marti et al (Eds.), pp. 435-442.

72. Ferrer, D. *et al.* (2004). NeuroFPGA-implementing artificial neural networks on programmable logic devices. In Proceedings on Design, Automation and Test in Europe, Paris : ACM SIGDA. G. Gielen & J. Figueras (Eds.), pp. 218-223.
73. Mehrtash, N. *et al.* (2003). Synaptic plasticity in spiking neural networks (SP/sup 2/INN): a system approach. *IEEE Transactions on Neural Networks*, 14, 5, pp. 980-992.
74. Hikawa, H. (1995). Implementation of simplified multilayer neural networks with on-chip learning. In Proceedings., IEEE International Conference on Neural Networks, pp. 1633-1637.
75. Hikawa, H. (2003). A digital hardware pulse-mode neuron with piecewise linear activation function. *IEEE Transactions on Neural Networks*, 14, 5, pp. 1028-1037.
76. Denby, B. *et al.* (2003). Fast triggering in high-energy physics experiments using hardware neural networks. *IEEE Transactions on Neural Networks*, 14, 5, pp. 1010-1027.
77. Anguita, D., Boni, A. & Ridella, S. (2003). A digital architecture for support vector machines: theory, algorithm, and FPGA implementation. *IEEE Transactions on Neural Networks*, 14, 5, pp. 993-1009.
78. Vollmer, H. (1999). *Introduction to Circuit Complexity*. (1st. ed.) Würzburg: Springer Verlag.
79. Román, I., López, I. & Yañez, C. (2006). A New Classifier Based on Associative Memories. In 15th International Conference on Computing (CIC'06), IEEE Computer Society. A. Gelbukh (Ed.), pp. 55-59.
80. Román, I., López, I. & Yañez, C. (2007). Perfect Recall on the Lernmatrix. Springer-Verlag Berlin Heidelberg. D. Liu (Ed.), pp. 835-841.
81. Wegener, I. (2005). *Complexity Theory: Exploring the Limits of Efficient Algorithms*. (1st. ed.) Dortmund: Springer-Verlag.
82. Borodin, A. (1977). On relating time and space to size and depth. *SIAM Journal on Computing*, 6, 4, pp. 733-744.
83. Pippenger, N. (1979). On simultaneous resource bounds. In 20th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, pp. 307-311.
84. Widrow, Bernard, & Hoff, Marcian, E. (1960) Adaptive Switching Circuits, 1960 IRE WESCON Convention Record, New York: IRE, pp. 96-104.
85. Rumelhart, D.E., Hinton, G.E. & Williams, R.J. (1986). Learning internal representations by error propagation, in D. Rumelhart and J. McClelland, editors.

- Parallel Data Processing, Vol.1, Chapter 8, the M.I.T. Press, Cambridge, MA, pp. 318-362.
86. Nunnari, G., Nucifora, A.F.M., and Randieri, C. (1998). *The application of neural techniques to the modelling of time-series of atmospheric pollution data*, Ecological Modelling 111, pp. 187–205.
 87. Ruiz JCS, Mayora OAI, Torres JJ & Ruiz LGS (1995). Short-term ozone forecasting by artificial neural networks. Elsevier Science, Advances in Engineering Software, vol. 23, pp. 143-149.
 88. Yáñez-Márquez, C., Felipe-Riverón, E.M., López-Yáñez, I. & Flores-Carapia, R. (2006). *A Novel Approach to Automatic Color Matching*, Lecture Notes in Computer Science, LNCS 4225, Springer-Verlag Berlin Heidelberg, pp. 529-538. ISSN: 0302-9743.
 89. Aleksander, I. & Morton, H. (1991). General neural unit: retrieval performance. *IEE Electronics Letters*, 27, pp. 1776-1778.
 90. J. Lucy. (1991). Perfect auto-associators using RAM-type nodes. *IEE Electronics Letters*, 27, pp. 799-800.
 91. E. V. Simões, L. F. Uebel, and D. A. C. Barone. (1996). Hardware implementation of RAM neural networks. *Pattern Recognition Letters*, 17(4), pp. 421-429.
 92. Jorgensen, T.M. & Linneberg, C. (1999). Boosting the performance of weightless neural networks by using a post-processing transformation of the output scores, *Proceedings of the International Joint Conference on Neural Networks 2*, pp. 812-816.
 93. Howells, G., Fairhurst, M.C. & Rahman, F. (2000). An exploration of a new paradigm for weightless RAM-based neural networks, *Connection Science*, vol. 12, no. 1, pp. 65-90.
 94. Yee, P. & Coghill, G. (2004). Weightless neural networks: A comparison between the discriminator and the deterministic adaptive RAM network, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3214, pp. 319-328.
 95. Al-Alawi, R. (2007). Performance evaluation of fuzzy single layer weightless neural network, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 15, no. 3, pp. 381-393.
 96. Aldridge, R.V. & Tattersall, G.D. (1995). Nonlinear filtering using weightless adaptive neural nets, *Proceedings of SPIE - The International Society for Optical Engineering* 2424, pp. 501-508.

97. de Lima Pereira Castro Jr.,Joel (1995). Acquiring semantic power in a 'weightless' neural network, *IEEE International Conference on Neural Networks - Conference Proceedings* 1, pp. 674-679.
98. York, T.A. & Duggan, P.M. (1997). Processing of tomographic data using weightless neural networks, *Journal of Intelligent Systems*, vol. 7, no. 3-4, pp. 349-365.
99. Lauria, S. & Mitchell, R.J. (1998). Weightless Neural Nets for face recognition: A comparison, *Neural Networks for Signal Processing - Proceedings of the IEEE Workshop*, pp. 539-546.
100. Seneviratne, L.D., Visuwan, P. & Althoefer, K. (1999). Weightless neural network based monitoring of screw fastenings, *IEEE International Conference on Intelligent Robots and Systems* 1, pp. 561-566.
101. Martins, W. & César Da Silva, J. (2001). Weightless neural networks on ranking of financial client solvency, *Proceedings of the International Joint Conference on Neural Networks* 3, pp. 1838-1843.
102. Yong, S., Lai, W.K. & Goghil, G. (2004). Weightless neural networks for typing biometrics authentication, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3214, pp. 284-293.
103. Keat, M.C.W., Abdullah, R., Salam, R.A. & Latif, A.A. (2004). Weightless neural network array for protein classification, *Lecture Notes in Computer Science* 3320, pp. 168-171.
104. Román-Godínez, I. & Yáñez-Márquez, C. (2007). *Complete Recall on Alpha-Beta Heteroassociative Memory*, Lecture Notes in Computer Science (ISI Proceedings), LNCS 4827, Springer-Verlag Berlin Heidelberg, pp. 193-202. ISBN: 978-3-540-76630-8.
105. Román-Godínez, I., López-Yáñez, I., & Yáñez-Márquez, C. (2007). *Perfect Recall on the Lernmatrix*, Lecture Notes in Computer Science (ISI Proceedings), LNCS 4492, Springer-Verlag Berlin Heidelberg, pp. 835-841. ISBN: 978-3-540-72392-9.
106. Tran, Q.-., Toh, K.-., Srinivasan, D., Wong, K.-. & Low, S.Q.-. (2005). An empirical comparison of nine pattern classifiers, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 35, no. 5, pp. 1079-1091.
107. Abdullah, M.R.B., Toh, K.-. & Srinivasan, D. (2006). A framework for empirical classifiers comparison, 2006 1st IEEE Conference on Industrial Electronics and Applications, art. no. 4026002.
108. Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets, *Journal of Machine Learning Research*, vol. 7, pp. 1-30.

109. Wolpert, D.H., Macready, W.G. (1997). No free lunch theorems for search. IEEE Transactions on Evolutionary Computation, 1 (1), pp. 83-98.
110. Wolpert, D.H., et al. (1997). No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1 (1), pp. 67-82.
111. Sewell, M.: No Free Lunch Theorems. (2006) Available at: <http://www.no-free-lunch.org/>
112. Monteiro da Silva, A. and Sussner, P. (2007): "Some Theoretical Aspects and Experimental Results on Feedforward Morphological Neural Networks", in Proceedings of the International Symposium on Mathematical Morphology , pp. 51 - 52, Rio de Janeiro, Brazil, October 2007

Trabajos publicados

Artículos

Aldape-Pérez Mario, Yáñez-Márquez Cornelio, and Argüelles-Cruz Amadeo José (2007). FPGA Implementation of Alfa-Beta Associative Memories. In Magno Congreso Internacional de Computación, Mexico City, Mexico. Amadeo Argüelles et al (Eds), pp. 27-36.

Argüelles Cruz Amadeo J., Aldape Pérez Mario, López Yáñez Itzamá., Yáñez Márquez Cornelio (2007) Aplicación de las redes neuronales Alfa-Beta en el sistema de monitoreo atmosférico de la Ciudad de México. In V Conferencia Científica Internacional de Medio Ambiente Siglo XXI (MAS XXI), Las Villas, Cuba.

Aldape-Pérez Mario, Yáñez-Márquez Cornelio, and Argüelles-Cruz Amadeo José. (2007). Optimized Associative Memories for Feature Selection. In Pattern Recognition and Image Analysis. Girona, Spain : Springer. Joan Marti et al (Eds.), pp. 435-442.

Argüelles Cruz Amadeo J., Diaz de Leon Juan Luis, Yáñez M. Cornelio, Camacho N. Oscar (2005). Pattern Recognition and classification using Weightless Neural Networks (WNN) and Steinbuch Lernmatrix. In Mathematical Methods in Pattern and Image Analysis: San Diego, Cal. USA. Proceedings of SPIE, Vol. 5916. Jaakko T. Astola, Ioan Tabus, Junior Barrera. pp. 59160P-1 – 59160P-8

Libros editados

Víctor Manuel Silva García, Amadeo José Argüelles Cruz, Luis Octavio López Leyva (Editores) (2007). Tópicos Selectos en Ciencias de la Computación. Colección CIDETEC Instituto Politécnico Nacional. Mexico.

Amadeo José Argüelles Cruz, José Luis Oropeza Rodríguez, Oscar Camacho Nieto, Osvaldo Espinosa Sosa (Eds.) (2007) Research in Computer Science. Special issue: Computer Engineering. Vol 30. Instituto Politécnico Nacional, Centro de Investigación en Computación. México ISSN 1870-4069

Medios Electrónicos

Amadeo Argüelles Cruz et al (Eds.) (2007) Mágnum Conference on Computing 2007 Instituto Politécnico Nacional, Centro de Investigación en Computación. Disco compacto ISBN: 978-970-36-0430-2