

UNIVERSIDAD DE GUAYAQUIL

FACULTAD DE CIENCIAS MATEMATICAS Y FISICAS

**CARRERA DE INGENIERIA EN SISTEMAS
COMPUTACIONALES**

Prototipo de aprendizaje automático con el uso
de redes neuronales artificiales aplicado
al establecimiento de límite de crédito
de cartera cliente

TESIS DE GRADO

Previa a la obtención del Título de:

INGENIERO EN SISTEMAS COMPUTACIONALES

AUTOR: JOSUÉ JONATHAN ORTEGA CONSTANTINE

TUTOR: ING. VICENTE VIZUETA

GUAYAQUIL – ECUADOR

2011

UNIVERSIDAD DE GUAYAQUIL

FACULTAD DE CIENCIAS MATEMATICAS Y FISICAS

**CARRERA DE INGENIERIA EN SISTEMAS
COMPUTACIONALES**

Prototipo de aprendizaje automático con el uso
de redes neuronales artificiales aplicado
al establecimiento de límite de crédito
de cartera cliente

TESIS DE GRADO

Previa a la obtención del Título de:

INGENIERO EN SISTEMAS COMPUTACIONALES

AUTOR: JOSUÉ JONATHAN ORTEGA CONSTANTINE

TUTOR: ING. VIZENTE VIZUETA

GUAYAQUIL – ECUADOR

2011

CERTIFICADO DE ACEPTACIÓN DEL TUTOR

En mi calidad de Tutor del Primer Curso de Fin de Carrera, nombrado por el Departamento de Graduación y la Dirección de la Carrera de Ingeniería en Sistemas Computacionales de la Universidad de Guayaquil,

CERTIFICO:

Que he analizado el Proyecto de Grado presentado por el egresado ORTEGA CONSTANTINE JOSUÉ JONATHAN, como requisito previo para optar por el título de Ingeniero cuyo problema es:

Prototipo de aprendizaje automático con el uso de redes neuronales artificiales aplicado al establecimiento de límite de crédito de cartera cliente, considero aprobado el trabajo en su totalidad.

Presentado por:

Ortega Constantine Josué Jonathan

C.I: 0924361256

Tutor: Ing. Vicente Vizueta

Guayaquil, Septiembre 30 del 2010

Guayaquil, Septiembre 30 del 2010

APROBACION DEL TUTOR

En mi calidad de Tutor del trabajo de investigación, "PROTOTIPO DE APRENDIZAJE AUTOMÁTICO CON EL USO DE REDES NEURONALES ARTIFICIALES APLICADO AL ESTABLECIMIENTO DE LÍMITE DE CRÉDITO DE CARTERA CLIENTE" elaborado por el Sr. JOSUÉ JONATHAN ORTEGA CONSTANTINE, egresado de la Carrera de Ingeniería en Sistemas Computacionales, Facultad de Ciencias Matemáticas y Físicas de la Universidad de Guayaquil, previo a la obtención del Título de Ingeniero en Sistemas, me permito declarar que luego de haber orientado, estudiado y revisado, la Apruebo en todas sus partes.

Atentamente

.....

Ing. Vicente Vizueta

TUTOR

DEDICATORIA

El siguiente trabajo se lo dedico a Dios fuente de la admirable inspiración hacia su naturaleza, a mis padres sustento e instrumentos divinos para mi formación, a mi querida hija Melody la cual incentiva mis ganas de superación y a su madre el amor de mi vida que siempre está en mi mente y corazón.

AGRADECIMIENTO

El principal agradecimiento es al Creador de todo lo conocido y desconocido, Dios, su fortaleza en mi debilidad produce una resoluble capacidad de entender y maravillarse por el conocimiento, y a mis padres que con su apoyo incondicional alentaron cada día mi vida estudiantil e indicaron los pasos a seguir por esta senda llena de confusión y locura.

TRIBUNAL DE GRADO

Ing. Frenando Abad Montero
DECANO DE LA FACULTAD
CIENCIAS MATEMATICAS Y FISICAS

Ing. Juan Chanabá Alcócer
DIRECTOR

ING. GEORGE SOLEDISPA

ING. ANGEL PLAZA

ING. VICENTE VIZUETA
TUTOR

DR. JOSÉ JÚPITER WILES
SECRETARIO

UNIVERSIDAD DE GUAYAQUIL

FACULTAD DE CIENCIAS MATEMATICAS Y FISICAS CARRERA DE INGENIERIA EN SISTEMAS COMPUTACIONALES

Prototipo de aprendizaje automático con el uso de redes neuronales artificiales aplicado al establecimiento de límite de crédito de cartera cliente

Autor: Ortega Constantine Josué Jonathan

Tutor: Ing. Vicente Vizuela

RESUMEN

El presente trabajo trata el comportamiento neuronal artificial biológico implementado en un prototipo informático para simular su arquitectura y desarrollo. Mediante la arquitectura multicapa de redes neuronales artificiales se intenta modelar el esquema Feedforward que utilizando el algoritmo de Backpropagation y mejorando ciertas variables dependiente del rendimiento del mismo se puede lograr entrenar a una determinada red recibiendo varios patrones de datos preliminares y encontrando el modelo matemático que rige dichos valores. Para la comprobación y evaluación del algoritmo de aprendizaje utilizado, el modelo neuronal recibirá un estado de cartera de cliente, con valores de límites de crédito anteriores, y encontrará el modelo que adapte esos patrones y así poder establecer el límite para los próximos datos a validar.

UNIVERSIDAD DE GUAYAQUIL
FACULTAD DE CIENCIAS MATEMATICAS Y FISICAS
CARRERA DE INGENIERIA EN SISTEMAS COMPUTACIONALES

Prototipo de aprendizaje automático con el uso de
redes neuronales artificiales aplicado al
establecimiento de límite de
crédito de cartera cliente

Proyecto de trabajo de grado que se presenta como requisito para optar por el título de
INGENIERO EN SISTEMAS COMPUTACIONALES.

Autor: Ortega Constantine Josué Jonathan

C.I.: 0924361256

Tutor: Ing. Vicente Vizueta

Guayaquil, Septiembre 30 del 2010

INDICE GENERAL

Certificado de aceptación del tutor	ii
Aprobación del tutor	iii
Dedicatoria	iv
Agradecimiento.....	v
Resumen	vii
Generíndice general	ix
Índices de cuadros	xii
Índices de gráficos	xiii
Resumen	xiv
Abstract	xv
Introducción	i
Capítulo I.....	1
El problema	1
Establecimiento del límite de crédito para clientes de cartera.....	1
Ubicación del problema en un contexto	1
Situación conflicto nudos críticos.....	2
Causas y consecuencias del problema	2
Delimitación del problema.....	3
Formulación del problema.....	4
Objetivos.....	6
Alcance del prototipo	7
Justificación e importancia.....	7
Capítulo II.....	9
Marco teórico	9

Antecedentes del estudio	9
Fundamentación teórica	9
Relación cerebro – computador convencional.....	11
Funcionamiento neuronal biológico	13
Potencial de acción y estructura de una neurona.....	14
Proceso de generación del potencial de acción.	16
Nociones generales:	19
Redes neuronales artificiales	21
Principales ventajas de las redes neuronales artificiales.....	23
Historia de las redes neuronales artificiales	24
Arquitecturas de redes neuronales.....	27
Estructura y funcionamiento de un rna	30
Aprendizaje de una red neuronal artificial	33
Modelos de redes neuronales actuales	33
Perceptron simple	34
Algoritmo de aprendizaje para el perceptron simple	41
Redes adaline (adaptive linear neuron).....	43
Redes madaline (multiple adaline).....	44
Regla least mean squares.....	46
Perceptron multicapa.....	48
Redes de hopfield	52
Red neuronal feedforward.....	52
Metodologías de construcción de una red neuronal	55
Normalización de patrones de entrenamiento.	57
Comportamiento general de aprendizaje.....	61
Criterios de procesamiento para determinar el aprendizaje de una red neuronal.....	63
Asociación de los patrones para el entrenamiento de una red neuronal	64

Consideraciones generales para establecer la mejor configuracion inicial de entrenamiento	65
Soluciones a problemas comunes en el entrenamiento de una red neuronal con algoritmo backpropagation	69
Preguntas a contestarse.....	70
Variables de investigacion.....	71
Definiciones conceptuales	71
Capitulo III	72
Metodología	72
Diseño de la investigación.....	72
Modalidad de investigacion.....	72
Tipo de investigación.....	72
Población y muestra	73
Evaluaciones de la muestras:	73
Configuración de entrenamiento	74
Recursos utilizados para la elaboración del prototipo.....	75
RECURSOS UTILIZADOS PARA LAS PRUEBAS DE RENDIMIENTO	75
Diseño del prototipo.....	77
Algoritmo de arquitectura – dinámica - aprendizaje	77
Glosario de términos	95
Capitulo IV	99
Conclusiones y recomendaciones.....	99
Conclusiones	99
Recomendaciones.....	102
Bibliografias.....	104
Anexos i	106
Pruebas realizadas	106
Anexos ii	110
Interfaz grafica	110

Grafica de error de aprendizaje.....	111
Interfaz de usuario.....	112
Pantallas del prototipo de desarrollo.....	112
Anexos iii.....	120
Manual tecnico melody neural network.....	120
modelo entidad - relacion.....	125
diccionario DE DATOS.....	126

INDICES DE CUADROS

Cuadro 1. Cerebro vs ordenador convencional.....	11
Cuadro 2. Recursos de hardware utilizados vs tiempo de respuesta	69

INDICES DE GRAFICOS

Grafico 1. Parte de una neurona biológica	13
Grafico 2. Potencial de acción	16
Grafico 3. Red neuronal de una sola capa.....	24
Grafico 4. Red neuronal multicapa	25
Grafico 5. Redes neuronales recurrentes	26
Grafico 6. Esquema neuronal artificial de un perceptron.....	27
Grafico 7. Representación de las funciones or y and en un perceptron simple	34
Grafico 8. Representación de las funciones xor en un perceptron	35
Grafico 9. Decisión lineal del problema xor	36
Grafico 10. Representación de una neurona adaline	38
Grafico 11. Representación de una red neuronal adaline.....	39
Grafico 12. Representación de una red neuronal madaline.....	40
Grafico 13. Representación de la superficie de error	42
Grafico 14. Representación del descenso por la gradiente de error	42
Grafico 15. Superficie de error con respecto a los pesos sinápticos.....	44
Grafico 16. Modelo elemental de una neurona de red feedforward	48
Grafico 17. Diagrama de flujo para la implementación de un prototipo neuronal	51
Grafico 18. Función sigmoide utilizada para la activación de un neurona artificial	53
Grafico 19. Función sigmoide utilizando un umbral de 1	54
Grafico 20. Función sigmoide utilizando un umbral de 3	55
Grafico 21. Situación de convergencia lenta del algoritmo de aprendizaje backpropagation	58

UNIVERSIDAD DE GUAYAQUIL
FACULTAD DE CIENCIAS MATEMATICAS Y FISICAS

CARRERA DE INGENIERIA EN SISTEMAS COMPUTACIONALES

PROTOTIPO DE APRENDIZAJE AUTOMÁTICO CON EL USO DE REDES
NEURONALES ARTIFICIALES APLICADO AL ESTABLECIMIENTO DE LÍMITE
DE CRÉDITO DE CARTERA CLIENTE

Autor: Ortega Constantine Josué Jonathan

Tutor: Ing. Vicente Vizueta

RESUMEN

En la actualidad el encontrar nuevos modelos de aprendizaje para los ordenadores se ha centrado en un reto para los investigadores en el campo de la informática, es allí donde se revisan nuevos conceptos y paradigmas de la inteligencia artificial, específicamente el estudio del funcionamiento de las redes neuronales artificiales. La utilización de los distintos algoritmos que existen dentro de este campo hace que la investigación sea el comienzo de un futuro prometedor y lleno de expectativas. El trabajo se centra en el estudio de uno de los algoritmos más potentes de la actualidad “Backpropagation” y su utilización en una serie de datos preliminares para encontrar el modelo óptimo de procesamiento de los mismos.

UNIVERSIDAD DE GUAYAQUIL
FACULTAD DE CIENCIAS MATEMATICAS Y FISICAS

CARRERA DE INGENIERIA EN SISTEMAS COMPUTACIONALES

PROTOTIPO DE APRENDIZAJE AUTOMÁTICO CON EL USO DE REDES
NEURONALES ARTIFICIALES APLICADO AL ESTABLECIMIENTO DE LÍMITE
DE CRÉDITO DE CARTERA CLIENTE

Autor: Ortega Constantine Josué Jonathan

Tutor: Ing. Vicente Vizuela

ABSTRACT

At the present time to find new models of learning for computers has focused on a challenge for researchers in the field of informatics that is where we review new concepts or paradigms of artificial intelligence, specifically in the study of the functioning of artificial neural networks. The use of different algorithms that exist within this field makes that research is the beginning of a promising future, full of expectations. . The work focuses on the study of one of the most powerful algorithms today, “Backpropagation” and its use in a preliminary series of patterns to find the optimal model of processing that data.

INTRODUCCION

En la actualidad nos encontramos con el paradigma común de desarrollo de software; que parte de un problema específico, un algoritmo de solución y un mantenimiento de código fuente posterior al desarrollo para poder reflejar cambios de requerimientos por parte de los usuarios del producto, esto es, un problema donde intervienen recursos tales, como: económico, tiempo, capital humano que de una u otra manera minimizan la relación costo-beneficio. Uno de los caminos que la tecnología prepara en el futuro, es la implementación de programas que puedan aprender en base a la experiencia, y formulen sus propios patrones de decisiones, ahora imaginemos un mundo lleno de maquinas inteligentes, capaces de tomar decisiones como el resultado de la experiencia en el diario vivir, podríamos tener tal vez una paradoja donde *“programaríamos a un ordenador para que se programe”*. Es un problema que encierra el mundo virtual en que vivimos, que nos insta a soñar con ordenadores que incrementen su procesamiento, mejoren en la toma de decisiones sin supervisión y la capacidad de predecir el estado futuro a partir de premisas preliminares.

Una de las ramas de la informática que se especializa en el estudio de la inteligencia y razonamiento del hombre orientado a un ordenador, es la computación artificial, que

desde que Alan Turing propuso su máquina de Turing¹, hemos visto como las ciencias computacionales avanzan a converger en esa idea.

Si bien es cierto, la inteligencia artificial no experimentan mucho crecimiento comparado con crecimiento exponencial del hardware, seguido del software, es sin duda el que unirá esos dos componentes importantes del área de informática para simular de manera casi perfecta al comportamiento y pensamiento humano. Prácticamente es una de las ramas más compleja que existe, debido a que estudia la forma en que se podrán utilizar los paradigmas de vida del hombre y llevarlo hacia una maquina.

Este trabajo encierra un estudio acerca del funcionamiento neuronal biológico aplicado a un ordenador y sus distintos modelos que han sido elaborados en el tiempo, y tomar uno de aquellos para elaborar un prototipo de aprendizaje aplicado al establecimiento de límites de crédito de los clientes de una empresa. Estos modelos conocidos como Redes Neuronales Artificiales (RNA) son un nuevo paradigma que se caracterizan por el procesamiento en paralelo, memoria distribuida y su adaptabilidad², basado en el funcionamiento estructurado de pequeñas neuronas que realizan tareas simples, pero que, al estar correctamente interconectas son capaces de generar conocimiento en base a patrones dados.

¹ *On computable numbers, with an application to the Entscheidungsproblem* Sociedad Matemática de Londres en 1936.

² Ramón Hilera José, Redes Neuronales Artificiales. Fundamentos, Modelo y Aplicaciones

Para la explicación del modelo a utilizar se evaluará uno de los temas importantes dentro del manejo de la cartera de clientes, lo cual es una aproximación real del límite de crédito asignado a dichos clientes.

El avance que han tenidos las redes neuronales artificiales para elaborar soluciones en base a patrones, son un camino a tomar para hacer aprender a un sistema en base a la experiencia que tenga este sobre el tema. Si bien es cierto es un campo aun no muy explotado dentro de la inteligencia artificial, pero forma uno de los primeros pasos para contribuir en la elaboración de ordenadores inteligentes que puedan pensar como humanos.

Estudios elaborados por algunos investigadores conocidos dentro de la rama de ciencias de la computación, tales como: Warren McCulloch, Walter Pitts, David Rumelhart, John McCarthy, Marvin Minsky, Seymour Papert entre los principales, han hecho posible la elaboración de modelos de neuronas artificiales que han sido propuestos para el estudio del pensamiento artificial y nueva forma de ver el mundo virtual de los ordenadores.

CAPITULO I

EL PROBLEMA

ESTABLECIMIENTO DEL LÍMITE DE CREDITO PARA CLIENTES DE CARTERA

UBICACIÓN DEL PROBLEMA EN UN CONTEXTO

Dentro de muchas empresas dedicadas a brindar servicios a sus clientes encontramos unas de las políticas utilizadas para un eficaz procesamiento de transacciones, es el establecimiento de un límite de crédito que cada cliente tendrá dentro de los procesos de intercambio que tenga con determinada organización.

El proceso de la asignación de crédito muchas veces depende de una determinada entidad crediticia³, pero de igual forma se presentan variaciones con respecto al tiempo, podremos fijar políticas que pueden ir cambiando de acuerdo al comportamiento que tengas dichos clientes al realizar sus transacciones tales como débitos o créditos que mantengan con la empresa.

Muchas veces el análisis de una cartera de cliente es inconsistente con respecto al comportamiento individual, esto se da por la falta de procesos inteligentes que establezcan un modelo a predecir para futuras determinaciones de crédito.

³ http://www.aaep.org.ar/espa/anales/pdf_00/salloum_vigier.pdf

Frecuentemente la productividad de un negocio se ve reflejado por esa mala toma de decisiones.

SITUACIÓN CONFLICTO NUDOS CRÍTICOS

Algunas preguntas dentro de los procesos de determinación de crédito general son: ¿Es el límite de crédito real para el cliente?, ¿Permite mejorar o limitar la rentabilidad del negocio?, ¿Cómo se comportan después de un crédito asignado?, son preguntas que están casi presente dentro una determinada empresa. Son una cantidad de variables de entrada tales como la efectivización durante un periodo determinado o tal vez las moras en los pagos parciales, que conllevan a conflictos con las políticas actuales de dicha organización.

CAUSAS Y CONSECUENCIAS DEL PROBLEMA

Una de las causas a estos conflictos se debe a la falta de conocimiento de herramientas en el mercado de análisis de datos por parte de los gerentes o dueños del negocio produciendo una desventaja frente a sus competidores directos e indirectos.

Imaginemos una entidad de préstamos que fija un determinado crédito al cliente basándose en las políticas actuales de dicha organización, sin tomar en cuenta el comportamiento histórico o actual de esos clientes, versus los pagos a tiempo de los préstamos, podríamos tener una desventaja debido a la dispareja balanza de los prestamistas.

Pero ciertas empresas utilizando la tecnología, emplea un conjunto de modelos inteligentes para el análisis; podrán predecir el comportamiento de sus clientes en base a premisas preliminares de información, por ejemplo el concepto de Data Mining.⁴

Sin duda alguna una entidad que no cuente con las herramientas necesarias en el análisis de su información se verá en un nivel de desventaja con las que si invierten en mejorar sus procedimientos de análisis.

Pero en el medio actual, muchos programadores aun desconocen de ciertos algoritmos utilizados en países desarrollados para la implementación de sistemas expertos, sistemas de predicciones, y algunos de estos basado en concepto de redes neuronales, que conlleva a disponer de pocas herramientas propias desarrolladas en nuestro país.

DELIMITACIÓN DEL PROBLEMA

El problema definido para nuestro estudio es: ¿Cómo establecer un límite de crédito basado en la experiencia?. Para resolver esta pregunta debemos tomar una serie de variables de entrada para el análisis con su respectiva salida, que serán usadas para la simulación del aprendizaje y así poder determinar el futuro comportamiento a datos no entrenados.

⁴Data Mining Program. University of Central Florida Departamento de Estadística. <http://dms.stat.ucf.edu/>

Tomaremos aquellas variables de entrada y salida, ejecutaremos el algoritmo de entrenamiento utilizado, verificaremos su aprendizaje por medio de su porcentaje de error y evaluaremos su rendimiento con nuevos valores de entrada.

FORMULACIÓN DEL PROBLEMA

Debemos encontrar el modelo matemático que se aproxime a la determinación del límite de crédito con la ayuda de movimientos de cartera preliminares de clientes. Para resolución y comprobación del mismo es necesario tomar un grupo de variables de entrada que están basados en un modelo de estudio gerencial de administración de cartera por parte de la empresa donde brindo mis servicios.⁵

La empresa que se dedica a la importación de repuestos para tracto camiones llamada Freno Preciso LTDA, facilitó una tabla de datos histórica con cierta modificación en los nombres de clientes, pero que muestran el movimiento de los mismo en un periodo determinado.

Los parámetros que ellos utilizan para el establecimiento del límite de crédito, están determinados por siete variables de movimientos, con un modelo y estudio matemático exclusivo de la misma organización que no se detalla en el presente documento por políticas de privacidad, pero que al final de la investigación será encontrado al momento de entrenar y hacer aprender a nuestra RNA.

⁵ Muestra tomada de una cartera a un rango de fecha determinados de Freno Preciso LTDA. Julio del 2010

Las variables son detalladas a continuación:

1. Total de pagos en efectivo del cliente en un tiempo determinado.
2. Valor total de facturas generadas.
3. Valor total de notas de ventas generadas.
4. Valor total de cheques girados a fecha para los pagos.
5. Cheques protestados.
6. Valor de total de notas de crédito.
7. Valor total de notas de débito.

Las variables anteriores formarán parte como *variables de entrada* que utilizaremos para el entrenamiento de la red.

La *variable de salida* que comprobará el éxito del aprendizaje de la red, es la propia aproximación del límite de crédito.

El entrenamiento de nuestra RNA constará de 1094 registros o patrones que alimentará a la misma para su aprendizaje.

El estudio se basa en definir las mejores variables de aprendizaje y arquitectura neuronal para minimizar el porcentaje de error y tiempo de respuesta.

OBJETIVOS

El objetivo general será el de simular un prototipo de aprendizaje utilizando un modelo neuronal artificial con el algoritmo backpropagation, que para su comprobación recibirá datos en base a movimientos de cartera de clientes y podrá encontrar el modelo matemático que rige a esos datos preliminares y así determinar la configuración indicada para que un nuevo registro se adapte a los mismos.

Después de determinar el objetivo general es necesario alcanzarlo mediante los siguientes objetivos específicos detallados a continuación.

- 1.- La arquitectura y modelo creado permitirá modificar el valor de los pesos y caminos asociados a cada procesador neuronal o neuronas artificiales. (Etapa de Entrenamiento)
- 2.- Después de la ejecución de la red esta mostrará los porcentajes de error de desempeño basado en los errores promedio por cada etapa para sus futuras evaluaciones. (Proceso de Gestión de errores)
- 3.- Visualizará la carga de procesamiento requerida para la etapa de aprendizaje. (Tiempo Computacional).
- 4.- Permitirá comparar la complejidad computacional en la ejecución del algoritmo a utilizar, esto en base al cambio de distintos parámetros de aprendizaje realizado en varias pruebas.

5.- Mostrará el límite de crédito asociado a cada cliente después de su ejecución.

ALCANCE DEL PROTOTIPO

- ❖ Diagramación estructural de la red a utilizar, neuronas y enlaces.
- ❖ Visualización de la curva de error del proceso de entrenamiento.
- ❖ Tiempo computacional del prototipo.
- ❖ Evaluación de los datos preliminares con el modelo obtenido y su respectivo error.
- ❖ Evaluación para patrones no vistos en el entrenamiento.

JUSTIFICACIÓN E IMPORTANCIA

El rápido incremento de procesos que mejoran los análisis de datos, los avances en las investigaciones en el campo de la ingeniería e inteligencia computacional, el aumento del rendimiento de distintas arquitecturas, los estudios basado en la neurociencia y el performance de los distintos modelos matemáticos basado en arquitectura neuro-artificiales, son las razones y causas que conlleva a la realización de este trabajo, donde se espera obtener la comprensión de uno de los algoritmos más utilizados para las predicciones, el análisis de datos conocido como backpropagation.

Lo que deseo encontrar mediante este estudio es tener la idea del cómo utilizar pequeños grupos de redes neuronales y utilizarlos en complejos procesos. Las distintas hipótesis

generadas y utilizadas en este trabajo podrán alimentar una serie de conocimientos adquiridos con la finalidad de rediseñar el modelo estudiado.

Si pensamos a gran escala, la importancia de prototipos de aprendizaje en nuestro medio tendremos la idea de subsistemas inteligentes que compartan su información para procesos cada vez más grande y desarrollos de problemas cada vez más complejos.

Otra de las justificaciones a la elaboración del presente documento y desarrollo, es la de servir como un marco de trabajo para futuras investigaciones dentro del campo a estudiar.

Finalmente servirá de base profesional a mi carrera los conceptos aprendidos e implementados, para futuras mejoras y adaptaciones a procesos similares, de los cuales poder alcanzar unos de mis objetivos a largo plazo, como el de desarrollar programación adaptiva y auto organizada para los procesos de implementación de aplicaciones.

CAPÍTULO II

MARCO TEÓRICO

ANTECEDENTES DEL ESTUDIO

Dentro del estudio de procesos inteligentes en el análisis de datos y reconocimientos de patrones, encontramos los modelos de redes neuronales artificiales. Investigaciones preliminares desde la comprensión del funcionamiento de las neuronas en el cerebro por el científico español Santiago Ramón y Cajal, hasta la definición del algoritmo backpropagation por David Rumelhart, Geoffrey Hinton y Ronald Williams en 1986, son los puntales para el presente estudio. Basados en el funcionamiento del sistema neuronal biológico que utilizando sus conceptos pretende elaborar una arquitectura artificial que represente a la misma.

FUNDAMENTACIÓN TEÓRICA

Existen múltiples definiciones sobre el concepto de redes neuronales artificiales, algunos mencionamos a continuación:

*“Una nueva forma de computación, inspirada en los modelos biológicos”*⁶

*“Un modelo matemático compuesto por un gran número de elementos procesales organizados en niveles”*⁷

⁶ Ramón Hilera José, Redes Neuronales Artificiales. Fundamentos, Modelo y Aplicaciones pág. 9

*“Son redes interconectadas masivamente en paralelo de elementos simples y con organización jerárquicamente, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso.”*⁸

*“Una RNA es una colección de procesadores elementales (neuronas artificiales), conectadas a otras neuronas o bien a entradas externas y con una salida que permite propagar las señales por múltiples caminos.”*⁹

Las redes neuronales son una estructura de computación distribuida basada en la composición del sistema nervioso de los seres humanos. La arquitectura de una red neuronal está conformada por pequeños procesadores elementales llamadas neurona, interconexiones, y poseen algoritmos para ajustar sus pesos para alcanzar su rendimiento esperado.

Sus propiedades importantes son:

- ❖ Conjunto de unidades elementales, las cuales poseen una baja capacidad de procesamiento.
- ❖ Una compleja estructura de conexiones usando enlaces ponderados.
- ❖ Parámetros libres que serán ajustados para satisfacer los requerimientos de desempeño.

⁷ Ramón Hilera José, Redes Neuronales Artificiales. Fundamentos, Modelo y Aplicaciones pág. 9

⁸ Teuvo Kohonen, Self-Organization and Associative Memory Springer, second edition (1998)

⁹ Raquel Flores López y José Miguel Fernández. Redes Neuronales Artificiales. *Fundamentos teóricos y aplicaciones prácticas*. (2008) Pág. 16

❖ Alto grado de paralelismo.

Una de las características importantes de las redes neuronales es su capacidad de aprendizaje a partir de patrones de entrada, y la capacidad de reajustar los pesos de su modelo para optimizar su desempeño, también conocidos como entrenamiento de la red.

Lo importante de este nuevo paradigma es la esquematización que tiene y su orientación a simular la estructura biológica animal, utilizando el funcionamiento de procesos neuronales del sistema nervioso.

RELACIÓN CEREBRO – COMPUTADOR CONVENCIONAL

Por tratar de simular al cerebro humano, el computador toma ciertas características, que aunque no igualan al funcionamiento biológico del cerebro, son muy eficientes a la hora de resolver problemas. La arquitectura de un ordenador sigue las detalladas por Von Neumann,¹⁰ compuesto por un microprocesador que ejecuta series de instrucciones complejas de forma fiable, mientras que el cerebro humano está formado por millones de unidades elementales (procesador) llamadas neuronas interconectadas entre sí, que realizan funciones simples.

Una de las notables diferencias es el aprendizaje de ambos esquemas; las neuronas aprenden basado en la experiencia y estímulos que recibe del entorno, siguiendo una

¹⁰ *First Draft of a Report on the EDVAC (Electronic Discrete Variable Automatic Computer (30 de junio de 1945)*

arquitectura masivamente en paralelo, y el ordenador necesita ser programado y su procesamiento básicamente en serie.¹¹

CUADRO NO. 1
CEREBRO VS ORDENADOR CONVENCIONAL

Características	Cerebro Humano	Computador
Velocidad de Proceso	Entre 10^{-3} y 10^{-2} segundos.	Entre 10^{-8} y 10^{-9} segundos.
Procesamiento	Paralelo	Secuencial (Serie)
Número de procesadores	Entre 10^{11} y 10^{14}	Pocos
Conexiones	10000 por neurona	Pocas
Almacenamiento	Distribuido	Direcciones fijas
Tolerancia a Fallos	Amplia	Poca o nula
Tipo de control de proceso	Auto organizado	Centralizado
Consumo de energía para ejecutar una operación por segundo	10^{-16} Julio	10^{-6} Julio

Elaboración: Redes Neuronales y Sistemas Difusos

Fuente: Martin del Brío y Sanz Molina (2001)

¹¹ Redes Neuronales Artificiales. *Fundamentos teóricos y aplicaciones prácticas.* (2008) Raquel Flores López y José Miguel Fernández. Pág. 11 y 12

FUNCIONAMIENTO NEURONAL BIOLÓGICO

Llevando la explicación desde el punto de vista biológico, la idea de estructurar un esquema de redes neuronales artificiales, comienza con el estudio del funcionamiento neuronal biológico del sistema nervioso animal, más de cien mil millones de neuronas¹² interconectadas entre sí con la capacidad de procesar y generar información.

En 1889 en el congreso de la sociedad anatómica alemana en Berlín, Santiago Ramón y Cajal,¹³ médico y premio nobel español pudo explicar mediante sus estudios investigativos el esquema y funcionamiento del sistema neuronal, pequeñas unidades independientes y debidamente interconectadas entre sí colaboraban para el proceso de transmisión unidireccional de los impulsos nerviosos que trabajan a manera de un concepto denominado potencial de acción.

El intercambio de información viene dado por la transmisión del potencial de acción entre neuronas. *“Los elementos básicos de un sistema neuronal biológico son las neuronas, que se agrupan en conjuntos compuestos por millones de ellas organizadas en capas, constituyendo un sistema con funcionalidad propia.”*¹⁴

¹² Robert W. Williams and Karl Herrup Originally published in The Annual Review of Neuroscience 11:423–453 (1988). Last revised Sept 28, 2001. http://www.nervenet.org/papers/NUMBER_REV_1988.html.

¹³ López-Muñoz, F; Boya, J., Alamo, C. (16 October 2006). «Neuron theory, the cornerstone of neuroscience, on the centenary of the Nobel Prize award to Santiago Ramón y Cajal

¹⁴ Redes Neuronales Artificiales. *Fundamentos teóricos y aplicaciones prácticas*. (2008) Raquel Flores López y José Miguel Fernández. Pág. 16

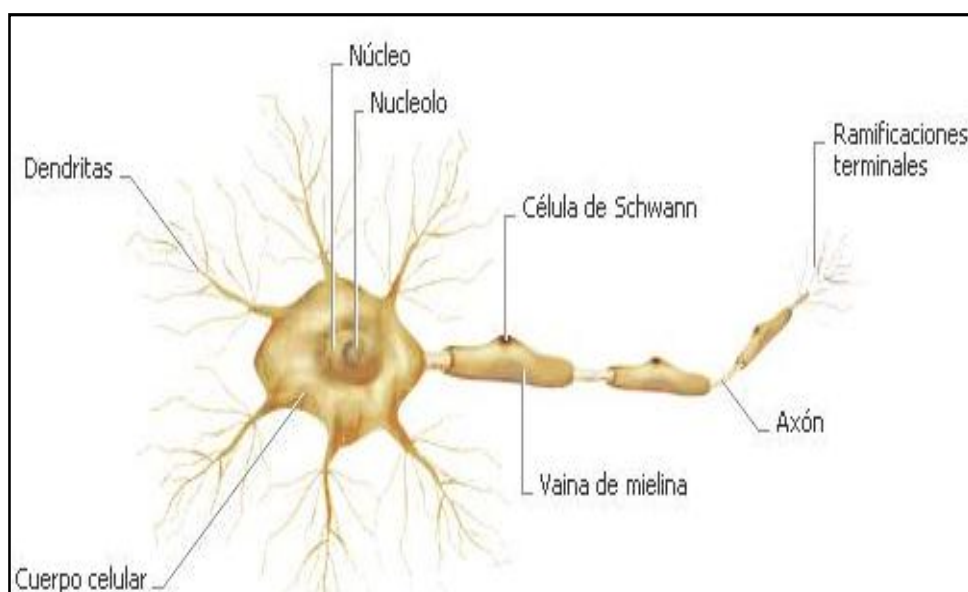
POTENCIAL DE ACCIÓN Y ESTRUCTURA DE UNA NEURONA

Conocido como impulso eléctrico, es la onda eléctrica que viaja por lo largo de la neurona, y es el encargado de producir el proceso de transmisión de información de una neurona a otra.

Para la transmisión del potencial de acción intervienen todos los elementos neuronales de conforman una neurona. Véase gráfico 1, que son los encargados de participar en el proceso de transmisión de información.

GRAFICO 1

PARTES DE UNA NEURONA BIOLÓGICA



Elaboración: Facultad de PSICOLOGÍA. IES Santa Eulalia

Fuente: <http://psi-anjen.blogia.com/2007/111201-neuronas-y-sus-tipos.php>, Tomado el Mayo 1 del 2010.

Las *neuronas* forma la parte del sistema nervioso como células encargadas de generar y receptor los estímulos nerviosos para la generación de información, que es compartida en complejas redes neuronales. Cada neurona está compuesta de un *cuerpo celular* que envuelve a un núcleo también llamado soma celular.

Dentro del *núcleo neuronal* está el *nucléolo* considerado una estructura macromolecular, debido a que no posee membrana y su función principal es la de producir y ensamblar los componentes ribosómicos para almacenar proteínas.

El núcleo suele estar ubicado en el centro de la neurona y ser muy visible especialmente en las neuronas pequeñas, suelen tener poros nucleares encargados de la recepción de proteínas para enviarlas al nucléolo.

Para la recepción de los elementos químicos cada neurona está formada por ramificaciones cortas llamadas *dendritas* unidas directamente al soma que muchas veces posee un contorno irregular desarrollando espinas.

Unas de las principales partes de la neurona es su *axón*, debido que su función es la de transmitir el potencial eléctrico a través de la neurona y esta a su vez generar un proceso llamado sinapsis, tal como a continuación lo explicamos.

El axón forma las *células de schwann* que son células gliales periféricas que acompañan a la neurona dentro de su crecimiento y desarrollo, recubren al axón formando una vaina aislante de *mielina*.

La generación del potencial de acción es recursiva, es decir se efectúa varias veces dependiendo de un proceso preliminar. Cuando el impulso nervioso llega a la membrana presináptica después de viajar por el axón, este activa a las vesículas presinápticas que contienen a los neurotransmisores, que son los encargados de viajar a la grieta sináptica hacia la siguiente neurona. Los neurotransmisores son receptados en el botón dendrítico y son los encargados de abrir el camino para que los iones de sodio puedan ingresar, y se produzca una nueva polarización en la siguiente neurona y se produzca el potencial de acción de esta neurona.¹⁵

PROCESO DE GENERACIÓN DEL POTENCIAL DE ACCIÓN.

En el proceso de generación del potencial de acción, tenemos iones de sodio, potasio y calcio actuando conjuntamente, y son los encargados de polarizar o despolarizar a la neurona. Los iones de calcio se encuentran al exterior de la membrana plasmática al final del axón e ingresan por el canal de calcio dependiente de voltaje para ayudar a que las vesículas presinápticas liberen a los neurotransmisores.

Después que los neurotransmisores son liberados estos pueden llegar a ser captados por los canales iónicos receptores de la neurona postsináptica y estos a su vez abiertos para permitir la entrada de un flujo de sodio debido al potencial de equilibrio, este proceso despolariza a la neurona siguiente y genera un potencial graduado a forma de corriente

¹⁵ Fidel Ramón y Jesús Hernández-Falcón División de Posgrado e Investigación y Departamento Fisiología Facultad de Medicina. UNAM. <http://www.facmed.unam.mx/historia/Propiedades1.html>

electro tónica perdiendo intensidad con la distancia. Es aquí donde comienza el viaje del potencial de acción en la neurona adyacente, la interacción de los iones de sodio que ingresan más los que se encuentran al interior de la neurona y junto a los de potasio pueden alterar la carga neta del potencial de membrana.

Una vez que la neurona es despolarizada hasta el potencial umbral los canales de sodio y potasio dependientes de voltaje se abren permitiendo el ingreso de iones de sodio y salida de iones de potasio alterando la polarización de la neurona. Esto efecto produce un nuevo potencial de acción que viaja por el axón hasta el siguiente nodo de ranvier donde están ubicados nuevos canales de sodio y potasio que produce un salto del impulso y un nuevo potencial de acción. Una vez que el potencial eléctrico de la membrana se acerca al potencial de equilibrio para el sodio, el canal de sodio dependiente de voltaje se cierra y el canal de potasio entra en mayor actividad hiperpolarizando a la membrana y preparándola para un nuevo potencial de acción.

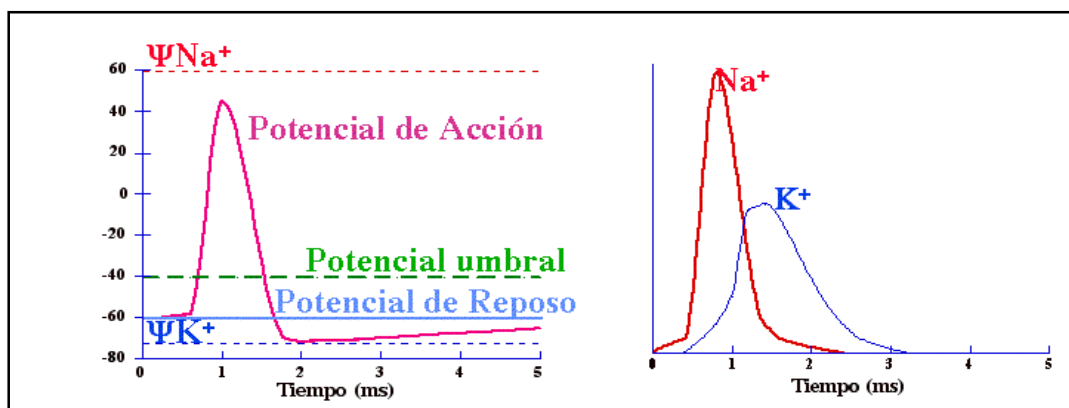
Es así como el potencial de acción viaja a través de la neurona de salto en salto hasta llegar a la terminal del axón permitiendo la apertura de los canales de calcio dependiente de voltaje para el ingreso de los iones de calcio al interior de la neurona. Se repite el proceso después del ingreso del calcio a la neurona. Para que la neurona restablezca la concentración de sodio y potasio, las bombas de sodio y potasio que toman iones de sodio y potasio para expulsarlo o ingresarlos hacia la neurona y establecer el estado de equilibrio o reposo preliminar. Este proceso puede suceder miles de veces por segundo.

El estado de la membrana de una neurona en estado de reposo es normalmente negativo en la zona interior, los canales de sodio y potasio permanecen cerrados y el potencial de la membrana es de -70Mv .¹⁶

Los canales de sodio están inactivos o cerrados cuando el potencial eléctrico de la membrana es de -90Mv , cuando el potencial varía entre -70Mv y -50Mv este canal es activado permitiendo la entrada de sodio, y de esta manera el exterior queda cargado negativamente y el interior queda cargado positivamente. Véase gráfico 2.

GRAFICO 2

POTENCIAL DE ACCIÓN



Elaboración: Departamento de Bioquímica de la Universidad de Zaragoza. Milagros Mediana.

Fuente: http://www.unizar.es/departamentos/bioquimica_biologia/docencia/ELFISICABIOL/PM/PM.gif. Mayo 1/2010

¹⁶ Gillian Pocock, Elsevier España Fisiológica Humana. La base de la medicina (2005) pág. 78 – 79.

Los canales de potasio están inactivos cuando el potencial eléctrico de la membrana es de -90Mv, cuando el potencial de la neurona se acerca a 0 Mv este canal se abre lentamente permitiendo la salida de los iones de potasio y así el interior quedará cargado negativamente y el exterior positivamente. (Etapa de polarización de la membrana)¹⁷

La bomba de sodio y potasio es la que se encargará de volver a estado de reposo a la neurona, expulsando 2 iones de potasio y extrayendo 3 iones de sodio de la neurona. Por lo consiguiente el exterior es más rico en Na⁺ y Cl⁻ que el interior de K⁺.

NOCIONES GENERALES:

1. El aprendizaje en los sistemas biológicos está basados en redes muy complejas de neuronas interconectadas.
2. Las neuronas reciben el 10% de señales electromagnéticas provenientes del exterior y el casi el 90 % de otras neuronas, a través de las sinapsis de las dendritas.
3. Si la acumulación de estímulos recibidos supera un cierto umbral, la neurona se dispara. Es decir puede contribuir al paso de la información a la siguiente neurona.
4. El cerebro humano está compuesto por 10^{11} neuronas con 10^4 conexiones individuales.

¹⁷ Gillian Pocock, Elsevier España Fisiológica Humana. La base de la medicina (2005) pág. 79 – 80.

5. Comparados con los ordenadores actuales, las neuronas son lentas para activar y desactivarse, utilizan 10^{-3} segundos.
6. Las conexiones sinápticas pueden potenciar o debilitar la señal recibida.
7. Las conexiones sinápticas son dinámicas. Con el desarrollo del aprendizaje algunas conexiones se potencian, otras se debilitan.
8. Los seres humanos realizan las tareas de percepción mucho mejor que la de un ordenador, por ejemplo un niño de 6 años puede reconocer visualmente a su madre.
9. La inteligencia de las neuronas radica en las interconexiones que cada una tienen con otras, su alto grado de paralelismo es la clave.
10. Aunque se trata de simular el funcionamiento y estructura neuronal biológica, no se refleja de manera completa en el modelo computacional.
11. Las RNA se presentan como modelos matemáticos con distintos algoritmos de aprendizaje y entrenamiento.

REDES NEURONALES ARTIFICIALES

Después del estudio precedente acerca del funcionamiento de una red neuronal biológica, y teniendo en cuenta que el sistema neuronal es el encargado de procesar la información que obtenemos del entorno y emitir respuestas de acuerdo a esa percepción, se han elaborado varios modelos matemáticos que puedan representar un esquema y funcionamiento aproximado al de las neuronas.

Las redes neuronales artificiales son la representación artificial de las propiedades del sistema neuronal biológico, diseñada con mecanismos artificiales para tratar de acercar a los ordenadores a dar una respuesta aproximada a las del cerebro.

La estructura de una red neuronal está basada en una serie de nodos representados por las neuronas y un número mayor de enlaces basado en el esquema de grafos dirigidos.

Una RNA está compuesta por unidades elementales llamadas neuronas, que realizan la función de procesar la información que reciben, y así emitir una respuesta, su función es realizada por los cálculos de la suma ponderada de sus entradas por el valor representativos de los pesos de sus conexiones para luego mediante una función de activación modificar este valor para procesar una salida.

La inteligencia de la red reside en las conexiones que cada neurona tiene, por la representación de pesos o valores en cada uno de los enlaces, representando las dendritas del cerebro.

La etapa de aprendizaje denominado también aprendizaje automático es la que se realiza la modificación de los pesos de cada conexión de acuerdo al algoritmo de aprendizaje utilizado. Existen dos tipos de algoritmos de aprendizajes:

Algoritmos Supervisados, es el aprendizaje a la que se expone una RNA teniendo en cuenta la función salida – respuesta esperada $f(s, r)$, es decir, se analiza la salida que emite RNA y se compara con la respuesta esperada, para una restructuración de su pesos con la finalidad de alcanzar el mínimo de error.

Los datos de entrada y salida que son utilizados dentro de los algoritmos supervisados son presentados en formas de pares de objetos comúnmente llamados vectores. La salida esperada pueden ser valores numéricos o datos clasificados.

La función de los algoritmos supervisados es la de crear un modelo capaz de predecir el valor correspondiente a cualquier objeto de entrada válida después de haber sido sometido a una serie de ejemplos preliminares. Para esto debe generalizar sus resultados.

Algoritmos no Supervisados, es aquel que no se le indica una respuesta, pero es capaz de utilizar conceptos relacionados, acerca de su tema, utilizando el concepto de memoria asociativa.

Su diferencia con los algoritmos supervisados es que no hay un conocimiento a priori, tratando a los datos de entrada como un conjunto de variables aleatorias, siendo construido un modelo de densidad para el conjunto de datos. Su comprensión depende


tanto explícitamente como implícitamente de una distribución de probabilidad sobre un conjunto de entrada.


Se establece un grado de similitud entre miembros de las mismas agrupaciones de datos.


PRINCIPALES VENTAJAS DE LAS REDES NEURONALES ARTIFICIALES


- ✓ *Disminución de agotamiento.*- Las neuronas no sufren agotamiento físico como los seres humanos por fatiga.
- ✓ *Aprendizaje.*- Tienen la capacidad de aprender o comparar un patrón específico con las entradas que recibe, es capaz de encontrar un modelo que ajuste los datos.
- ✓ *Tolerancia a fallos.*- Debido a su estructura, pueden seguir operando si uno de sus conexiones falla.
- ✓ *Paralelismo Masivo.*- El procesamiento de información es paralelo, debido a la multiplicidad de sus conexiones.
- ✓ *Auto organización.*- De acuerdo a los pesos o ponderaciones de cada neurona, esta almacena su representación de una información procesada.
- ✓ *Fácil Implementación.*- Su estructura contiene procesos simples de implementar con modelos matemáticos existentes.
- ✓ *Procesamiento Automático.*- Es implementada funciones que son las encargadas de trabajar en conjunto para el entrenamiento y aprendizaje de información.

HISTORIA DE LAS REDES NEURONALES ARTIFICIALES

-  1943. Warren McCulloch y Walter Pitts publicaban el artículo “A logical Calculus of ideas Immanent in Nervous Activity”. La colaboración de un neurobiólogo y un matemático genera un modelo abstracto de neurona en el que la probabilidad de que una neurona se activase dependía de la señal de entrada y de la sinapsis de conexión.

-  1949. Donald Hebb Publica el libro “The organization of the Behavior”, donde se describe cómo pueden aprender las neuronas. Si la neurona de entrada y la neurona de salida estaban ambas activadas entonces se reforzaba la conexión sináptica de ambas.

-  1951. Marvin Minsky y Dean Edmons fabrican con tubos, motores y dispositivos mecánicos una máquina capaz de aprender. Para ello se basaron en las ideas de MacCulloch y Pitts. Aunque rudimentaria, esta máquina fue fascinante en su tiempo.

-  1956. Organizada por Minsky, John McCarthy, Nathaniel Rochester y Claude Shannon se celebró la primera conferencia sobre Inteligencia Artificial.

- ✚ 1959. Frank Rosenblatt desarrolla su concepto de perceptron. Un sistema que permitía interpretar patrones tanto abstractos como geométricos. El perceptron supone un gran avance en la IA y el inicio de las RNA.

- ✚ 1962. Marcian Hoff. Desarrolla un modelo llamado ADALINE. Este modelo de RNA es capaz de clasificar los datos en espacios separables linealmente. La red ADALINE era extremadamente similar al perceptron y pronto se generalizó estos resultados al modelo de Rosenblatt.

- ✚ 1969. Marvin Minsky y Seymour Papert publican el libro llamado “perceptrons” en el que presentan el principal problema del perceptron, el famoso problema del XOR o el Or exclusivo. El XOR es un operador lógico en el que la conclusión es verdadera si es verdad uno de los supuestos, pero es falsa en caso de que ambos sean falsos o verdaderos. La solución de este problema no es linealmente separable y el perceptron no podía superarlo. Además en este artículo se plantea “el problema de la asignación de créditos”. Una red de perceptrones compuesta por varias capas podría solucionar el XOR, sin embargo eso no es posible porque no se sabe cuánto tienen que modificarse los pesos de la capa intermedia. Este libro desmoralizó a todos los investigadores y provocó un descenso del interés en las RNA.

✚ En la época posterior al libro de Minsky y Papert el interés generalizado por las RNA había desaparecido, sin embargo algunos investigadores decidieron continuar con las investigaciones. En este periodo aparecen los modelos de Anderson, Kohonen, Grossberg, Hopfield.

✚ 1986. Rumelhart, McClelland y el PDP (Rumelhart, McClelland & PDP, 1986)
Estos dos investigadores fundaron el PDP (Parallel Distributed Processing) un grupo dedicado al estudio del conocimiento. De este grupo se editó el libro “Parallel Distributed Processing: Explorations in the Microstructures of Cognition”. Este libro supuso una revolución dentro de las RNA, en él se exponía el modelo Back-Propagation que resolvía el problema de la asignación de créditos propuesto por Minsky. Después de este libro se produjo una explosión en las RNA apareciendo modelos, técnicas, campos de aplicación y fusiones híbridas de modelos.

El interés en el tema ha durado hasta la actualidad donde las RNA son una herramienta matemática – computacional importante para el desarrollo de problemas con mayor grado de complejidad.

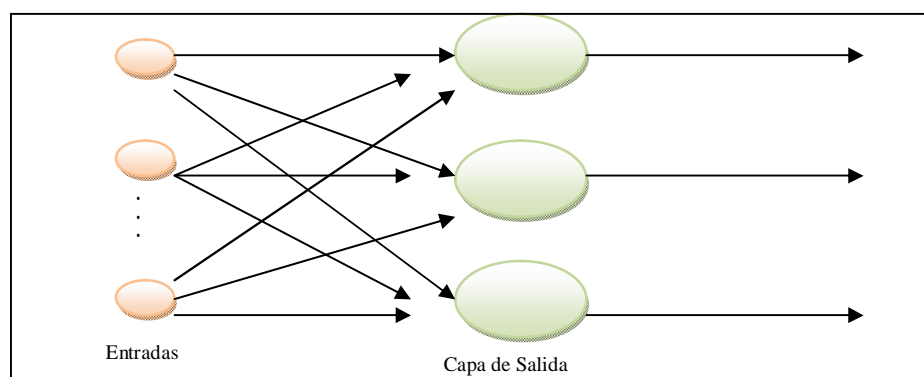
ARQUITECTURAS DE REDES NEURONALES

Existen dos tipos de topologías definidas por su diseño y estructura utilizadas para representar una red neuronal:

Redes de una capa, son aquellas formadas por una sola capa de neuronas que reciben varias entradas y emiten una salida, su gran limitación es que solo son usadas para resolver problemas linealmente separables, por ejemplo la función AND y OR. Véase gráfico 3.

GRAFICO 3

RED NEURONAL DE UNA SOLA CAPA



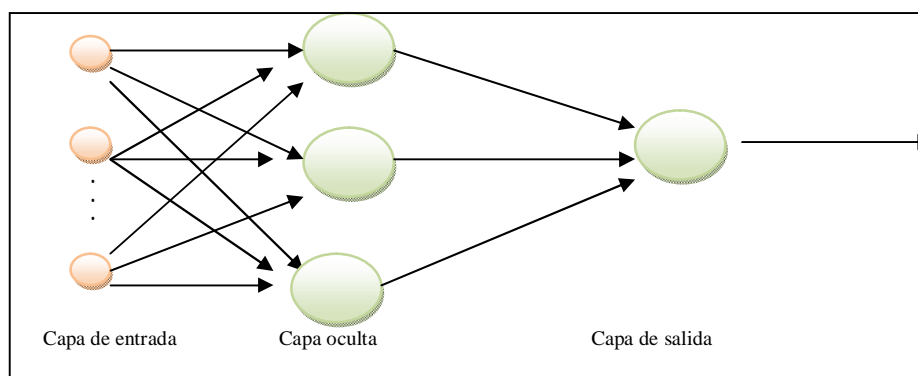
Elaboración: Propia del autor del presente trabajo.

Fuente: Basado en el concepto de arquitecturas neuronales de una sola capa. Pág. Actual. Mayo 25 del 2010

Las *redes multicapas*, son la solución a la limitación anterior, formadas por una o varias capas ocultas, una capa de entrada y una de salida; que colaboran para la resolución de problemas no lineales. Véase gráfico 4.

GRAFICO 4

RED NEURONAL MULTICAPA



Elaboración: Propia del autor del presente trabajo.

Fuente: Basado en el concepto de arquitecturas multicapa. Pág. Actual. Mayo 25 del 2010

Dentro de otras clasificaciones definimos aquellas que se diferencian por sus conexiones entre sí, de las cuales mencionamos:

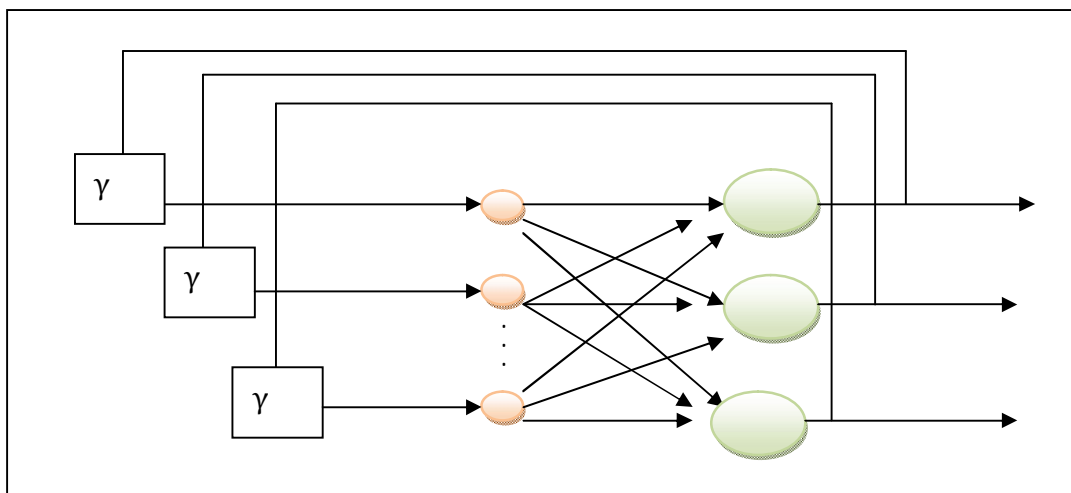
Redes recurrentes, que son las que pueden tener más de un ciclo de activación, y viene dadas por la retroalimentación de las señales que provienen de otras neuronas o de la misma neurona. Véase gráfico 5.

Redes no recurrentes, la transferencia o propagación de señales solo se realiza en un solo sentido hacia adelante (Feedforward), es decir no existe retroalimentación.

Además podemos mencionar que por la cantidad de interconexiones que existen entre neuronas, se pueden separar a dos grandes grupos, que son las redes neuronales totalmente conectadas hacia adelante o hacia atrás (Feedback), donde cada neurona de la capa n está debidamente conectada con todas las neuronas de la capa $n+1$. Las otras redes neuronales que siguen en este grupo son las parcialmente conectadas, donde no existe la interconexión total entre neuronas.

GRAFICO 5

RED NEURONAL RECURRENTES



Elaboración: Propia del autor del presente trabajo.

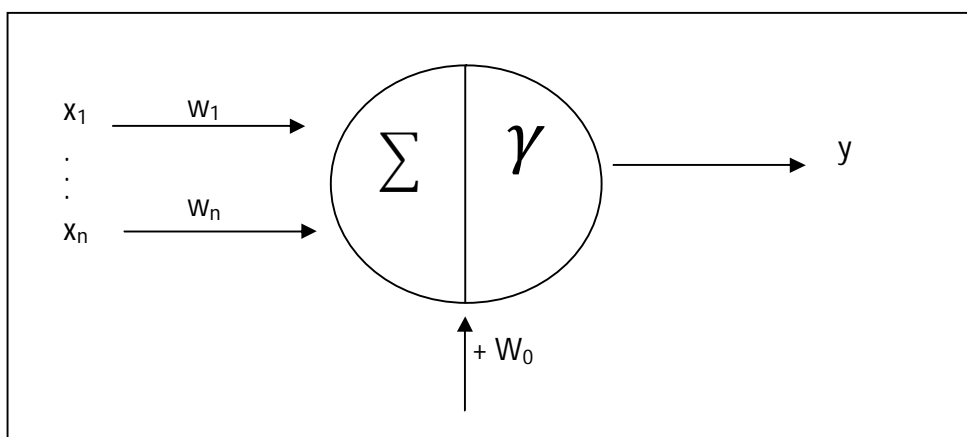
Fuente: Basado en el concepto de arquitecturas de neuronas recurrentes. Pág. Actual. Mayo 25 del 2010

ESTRUCTURA Y FUNCIONAMIENTO DE UN RNA

En 1943 el neurólogo estadounidense Warren Sturgis McCulloch y el matemático Walter Pitts, concibieron un modelo simple matemático de una neurona artificial en su trabajo “*A logical Calculus of ideas Immanent in Nervous Activity*”¹⁸ como se muestra en el gráfico 6.

GRAFICO 6

EZQUEMA NEURONAL ARTIFICIAL DE UN PERCEPTRON



Elaboración: Jorge Fierro. Advanced Tech Computing Group. Universidad Técnica de Loja

Fuente: advancedtech.wordpress.com <http://advancedtech.files.wordpress.com/2008/08/amagen3.jpg>

Como vemos el modelo en la gráfica 6. Una RNA está compuesta por un vector de entradas $x = (x_1, \dots, x_n)$, w_0 es el umbral de acción o activación, las entradas están enlazada con la neurona a través de un vector de pesos $w = (w_1, \dots, w_n)$, por ultimo tenemos una salida (y).

¹⁸ 1943. *Bulletin of Mathematical Biophysics* 5: 115-133, por The Society for Mathematical Biology

El proceso consiste en generar una salida a partir de la función de activación (γ), a la sumatoria ponderada de entre el vector de entrada x y el vector de pesos w correspondiente más un sesgo w_0 , como la fórmula a continuación:

Fórmula 1:

$$y = \gamma \sum_{i=1}^n x_i w_i + w_0$$

La función de activación (γ) propuesta por McCulloch y Pitt posee una salida binaria conocida como la función la función signo representada así:

$$\text{sgn}(z) = \begin{cases} 1 & z \geq 0 \\ -1 & z < 0 \end{cases}$$

Rango de salidas = $\{-1, 1\}$

Algunas otras utilizadas son la función lineal escalón unitario o también las funciones no lineales sigmoidal:

$$\text{Escalón unitario} \quad U(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

Rango de salidas = $\{0, 1\}$

Sigmoide o logística $s(z) = \frac{1}{1+e^{-z}}$

Rango de salidas = { 0 , 1 }

Tangente hiperbólica $s(z) = \tanh z$

Rango de salidas = { -1 , 1 }

Esta función que puede ser lineal o no lineal tiene el papel de normalizar la salida cuando el umbral de entrada se supera.

Las funciones lineales no se usan mucho ya que proporciona muy poca potencia de cálculo, para este caso se usan las funciones como la sigmoide o tangente hiperbólica que hace que la red no se comporte como una función lineal.

Cada nodo o neurona se conecta a otras unidades a través de arcos dirigidos, representando la conexión axón – dendrita.

El último factor que define a una red neuronal es el tipo de entrenamiento que se realiza, lo cual tiene por objeto modificar el valor de los pesos de interconexión, de manera que la red se comporte de una manera determinada.

APRENDIZAJE DE UNA RED NEURONAL ARTIFICIAL

Después de contar con la arquitectura escogida, es necesario definir el tipo de entrenamiento que se aplicará a una RNA para que aquella red sea capaz de aprender, y esto en base a la modificación o regularización de las conexiones o pesos. Estos pesos cambian de acuerdo al proceso iterativo de cada patrón de entrenamiento (llámese patrón de entrenamiento al conjunto de datos en función de entradas y salidas), donde se comparan la salida deseada por aquel patrón y la salida real de la red actual.

Las conexiones de un RNA, se pueden ilustrar de una forma similar a las de las sinapsis biológicas, donde se recodifica las conexiones de una neurona a otra, es decir se pueden crear, quitar o modificar conexiones, matemáticamente expresado con valores nulos, diferentes al actual, o simplemente el valor actual del peso sináptico.

MODELOS DE REDES NEURONALES ACTUALES

Hay varios modelos de redes elaboradas a base de los tipos de algoritmos de aprendizajes mencionados anteriormente.

Dentro de los modelos con arquitectura monocapa se encuentra el perceptron simple como ya se mencionó fue el primer modelo de RNA en aparecer, y las redes ADALINE.

Los principales modelos que contienen más de una capa son: El perceptron multicapa, Redes backpropagation, redes de Feedforward, redes MADALINE que lo veremos más adelante en este trabajo.

En los modelos recurrentes tenemos la red de Hopfield, la máquina de Boltzmann.

PERCEPTRON SIMPLE

El primer modelo RNA propuesto por el informático Frank Rosenblatt de Cornell Aeronautical Labs en 1958, 14 después de los trabajos de McCulloch y Pitts (véase pág 30). Es una red formada por una sola capa de neurona que recibe señales de entrada y emite una salida.

Este tipo de RNA representa resolución de modelos lineales donde la salida de una neurona es una función lineal de sus entradas. Si bien es cierto existen problemas complejos de solución con funciones no lineales difícil de aplicar a una arquitectura de perceptron simple, son estos los modelos primeros en aparecer y forman la base para las actuales arquitecturas neuronales.¹⁹

El perceptron puede utilizarse tanto como clasificador, como para representar funciones de dos estados, pues su neurona de salida es del tipo MacCulloch-Pitts.

Rosenblatt basados en los trabajos preliminares de Pitts y McCulloch (1943), detalló su trabajo en cinco puntos:

1. La actividad de la neurona es un proceso binario.
2. Para excitar a una neurona, se debe excitar previamente el valor umbral de la misma.

¹⁹ Raúl Pino Diez, Alberto Gómez Gómez, Nicolás de Abajo Martínez, Sistemas Expertos, Redes Neuronales y Computación evolutiva. Universidad de Oviedo 2001. Pág. 49

3. El retardo sináptico es el único significativo.
4. Las sinapsis inhibitoras impiden el disparo de la neurona.
5. La estructura de la red que se realice con estas neuronas no cambiaran en el tiempo.

Pa representar el esquema de un perceptron simple: la neurona recibe un vector de valores de entrada x y emite un salida out (y), representado por:

Fórmula 2:

$$g(x) = w_0 + FL(w, x)$$

Fórmula 3:

$$g(x) = w_0 + \sum_{i=1}^n w_i x_i$$

La función $FL(w, x)$ es una función lineal donde x son los datos de entrada y w los pesos sinápticos relacionados con dos neuronas artificiales, que se le aplica un valor umbral w_0 cuya entrada es igual a -1, para una representación de división lineal del plano R^n . El perceptron está constituido por un conjunto de entradas que reciben los patrones de entrada a reconocer o clasificar y una neurona de salida que se ocupa en clasificar a los patrones de entradas en dos clases, como vemos a continuación:

$$out(x) = \begin{cases} 1 & g(x) > 0 \\ 0 & \text{caso contrario} \end{cases}$$

Para nuestro caso la función de transferencia es la función signo, la cual permite reforzar o inhibir el estado de la neurona de acuerdo a un valor umbral o valor de activación, así:

$$f(x) = \begin{cases} 1 & \text{si } W_1X_1 + W_2X_2 + \dots + W_nX_n \geq W_0 \\ 0 & \text{si } W_1X_1 + W_2X_2 + \dots + W_nX_n < W_0 \end{cases}$$

El aprendizaje del perceptron simple viene dado por la determinación y modificación de los pesos sinápticos y umbral, mediante un proceso adaptivo con valores iniciales aleatorios que se van modificando iterativamente cuando la salida de la red no coincide con la salida deseada. De esta forma la regla de aprendizaje del perceptron simple se define como:

Fórmula 4:

$$W_{ij}(t+1) = w_{ij}(t) + \Delta W_{ij}(t) \quad t = 1, 2, 3, \dots$$

Fórmula 5:

$$\Delta W_{ij}(t) = \eta (t_j - Y_j) X_i$$

Donde,

t = Patrón actual de entrenamiento

t_j = Salida Esperada del patrón actual

η = Factor de Aprendizaje

Y_j = Salida actual de la red.

X_i = Entrada actual de la red.

La fórmula 4 y 5 mencionadas anteriormente nos indica que la variación del peso W_{ij} es proporcional al producto del error de la (salida deseada - la salida esperada) por la neurona de entrada de la red. El factor de aprendizaje o constante de proporcionalidad es un parámetro positivo debido a que cuanto mayor es más se modifica el peso sináptico y viceversa, es decir que determina la proporcionalidad de aprendizaje, al ser pequeño la red aprenderá poco a poco.

Esta regla de aprendizaje es un método de detención y corrección de errores, al modificar los pesos cuando la red se equivoca en su salida. Por ejemplo cuando tenemos un patrón que no pertenece a la clase de salida se corrige el valor del peso sináptico reforzándolo o

debilitándolo con un valor proporcional al valor de entrada para que se acerque a la misma, por el contrario los pesos permanecen estables cuando el valor de salida de red sea igual al de la salida esperada.

El valor umbral también se modifica de acuerdo a la regla de aprendizaje detallada en la página anterior, teniendo en cuenta que su entrada siempre es -1. De esta forma:

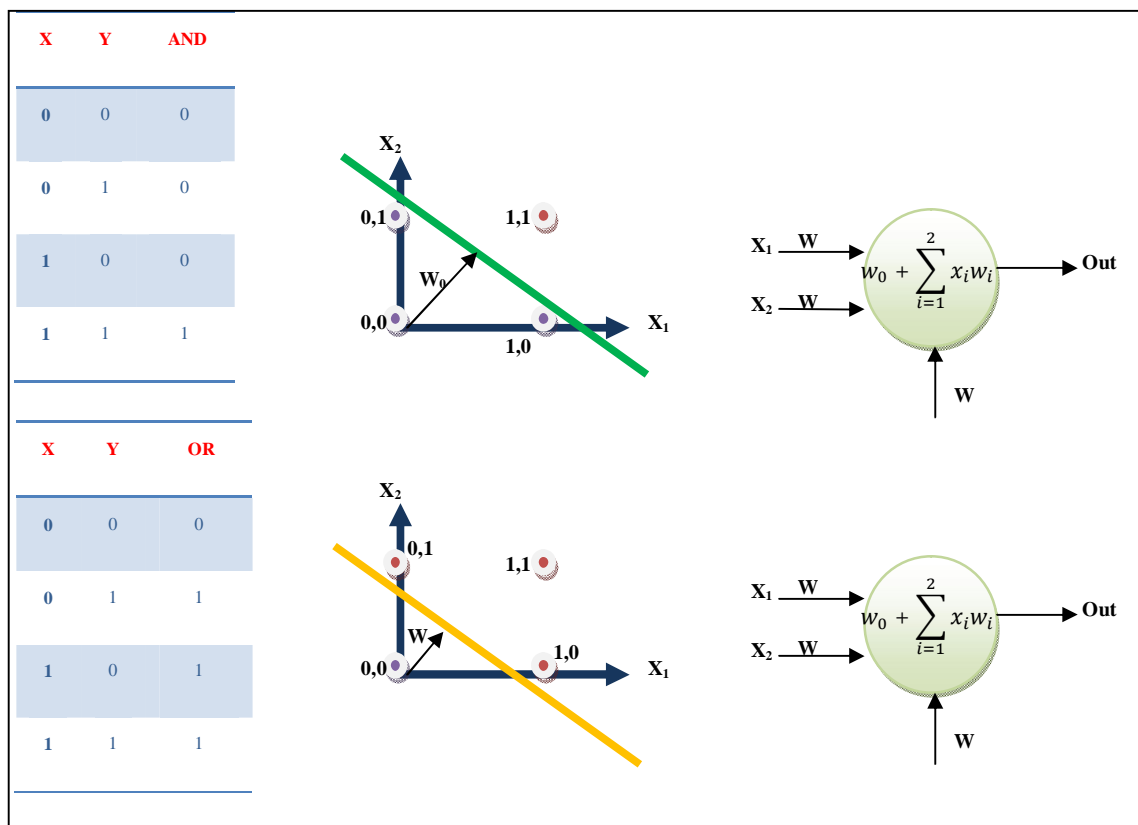
$$W_1X_1 + W_2X_2 + \dots + W_nX_n \geq W_0 \Leftrightarrow W_1X_1 + W_2X_2 + \dots + W_nX_n + W_0 \geq 0$$

$$W_0 = W_{n+1}X_{n+1} \quad ; X_{n+1} = -1$$

Establecemos un problema con sus respectivas entradas o datos y tratamos de representar sus soluciones en un hiperplano, lo cual se deberá calcular una función lineal que separe espacialmente ese distinto conjunto de soluciones. Por ejemplo; las funciones AND y OR son linealmente separables, su representación la vemos en la gráfico 7:

GRAFICO 7

REPRESENTACIÓN DE LAS FUNCIONES OR Y AND EN UN PERCEPTRON SIMPLE



Elaboración: Propia del autor del presente trabajo.

Fuente: Representación de la solución de la función Or y And en un perceptron simple. Pág. Anterior. Mayo 30 del 2010.

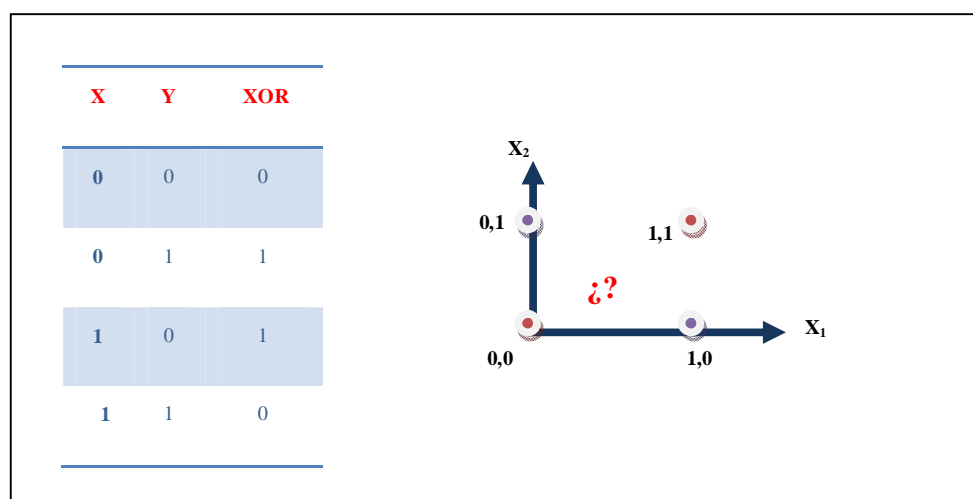
* Círculos rojos representa a 1, círculos lilas representan un 0.

Como podemos visualizar en el gráfico 7, ambas funciones se las puede separar linealmente. El valor umbral o W_0 es el que lleva la función lejos del origen de acuerdo a los conjuntos de soluciones en el hiperplano R^n .

Su limitante fue la de resolver solo problemas linealmente separables, hecho que después de analizarlo Minsky y Papert 1969 decidieron sacar su libro perceptron, donde detallaban todas sus limitantes y la incapacidad de resolver el problema del OR exclusivo o XOR. Véase gráfico 8.

GRAFICO 8

REPRESENTACIÓN DE LA FUNCIÓN XOR EN UN PERCEPTRON



Elaboración: Propia del autor del presente trabajo.

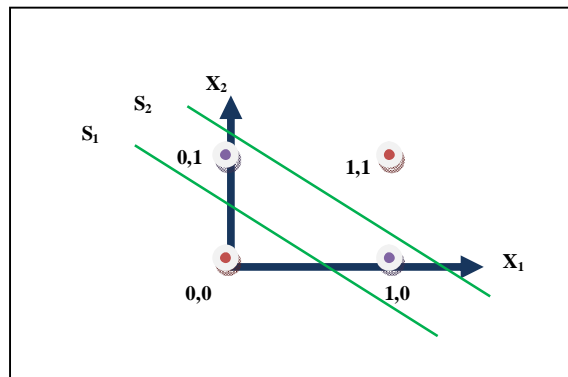
Fuente: Basado en la no solución de la función Or exclusivo en un perceptron simple. Pág. Actual. Mayo 30 del 2010

* Círculos rojos representa a 1, círculos lilas representan un 0.

Como notamos en la figura 9, una neurona de tipo perceptron intenta dar una solución lineal formada por 2 rectas S_1 y S_2 , en el plano. Consideremos una neurona que se tenga la función de dividir el plano resultante por la primera solución S_1 dependiendo de las respuestas de las dos primeras neuronas X_1 y X_2 . Es así como se originarían los perceptrones de varias capas.

GRAFICO 9

DECISIÓN LINEAL DEL PROBLEMA XOR



Elaboración: Propia del autor del presente trabajo.

Fuente: Tomado de la forma lineal de separación única del problema XOR en un hiperplano. Mayo 30 del 2010

ALGORITMO DE APRENDIZAJE PARA EL PERCEPTRON SIMPLE

Existen varios algoritmos de enteramientos simples para perceptrones capaces de encontrar el patrón adecuado para cualquier problema que sea separable linealmente.

Uno de los algoritmos de entrenamiento es la función de activación umbral.

A continuación el algoritmo de aprendizaje y entrenamiento:

Entrada: Un conjunto de entrenamiento, que recibe como entrada valores correspondientes a un punto de R^n y un factor de aprendizaje η , donde η determina el ritmo de aprendizaje, la cual no puede ser ni muy pequeña puesto que implica un aprendizaje lento, ni demasiado grande que podría conducir a oscilaciones, por lo general es una constante positiva tal que $0 < \eta < 1$.

A continuación el algoritmo del perceptron simple:

1. Inicialización de los pesos w_i aleatoriamente.
2. Hasta que se cumpla una determinada condición para su terminación:

2.1 Para cada ejemplo del conjunto de entrenamiento:

- Se presenta un vector \bar{x} .
- El perceptron calcula la salida $out(x) = w_0 + FL$
- Se actualizan los pesos w_i . Para cada peso hacer

$$w_i = w_i + \eta (\delta - out) x_i \quad (\delta = \text{salida esperada})$$

3. Retorna el perceptron entrenado \bar{w} .

Por ejemplo si en cada iteración la salida esperada menos la salida real es positiva así que $(\delta > out(x))$, los pesos w_i correspondiente a cada x_i aumentaran para aproximar a la salida esperada y viceversa. Si la salida esperada es igual a la salida real entonces w_i no cambia.

Después de finitas iteraciones el algoritmo converge a un vector de pesos \bar{w} que clasifica correctamente la salida siempre y cuando estos sean linealmente separable y η se ha suficientemente pequeño.

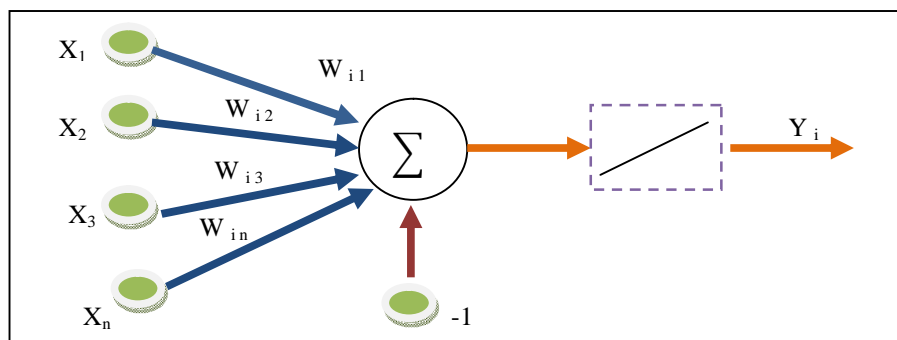
REDES ADALINE (ADAPTIVE LINEAR NEURON)

Fueron desarrolladas por Bernie Widrow en 1961 en la Universidad de Stanford, tomando parecido al funcionamiento del perceptron simple pero con una neurona de repuesta lineal cuya entradas pueden ser continua.

Una de las diferencias notables de las redes ADALINE está en la regla de aprendizaje “WIDROW-HOFF” también conocida como regla LMS (Least Mean Squares)²⁰, donde la modificación de los pesos es continuo y proporcional al error cometido por la neurona.

GRAFICO 10

REPRESENTACIÓN DE UNA NEURONA ADELIN



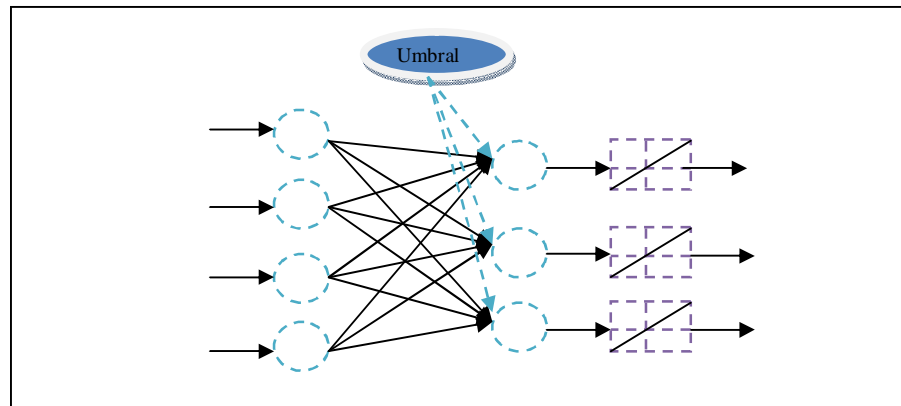
Elaboración: Bonifacio Martin del Brio y Alfredo Sanz Molina

Fuente: Gráfico adaptado. Redes Neuronales y sistemas Difusos. 2da. Edición ampliada. Pág. 55

²⁰ Regla media mínimos cuadrados, véase página 46

GRÁFICO 11

REPRESENTACIÓN DE UNA RED NEURONAL ADELIN



Elaboración: José Miguel Fernández, Raquel Flores López.

Fuente: Gráfico Adaptado. Redes Neuronales. Pág. 49

REDES MADALINE (MULTIPLE ADALINE)

También desarrolladas por WIDROW (1961), son una representación múltiple de redes Adaline conectadas entre sí (Véase gráfico 12), utilizando la regla “WIDROW-HOFF”.

La salida de estas redes está representada por:

Fórmulas 6:

$$Y_j = \sum_{i=0}^N W_{ij} X_i$$

$$W_{ij}(t + 1) = W_{ij}(t) + \Delta W_{ij}(t)$$

$$\Delta W_{ij}(t) = \alpha (t_j - y_j) X_i$$

O bien;

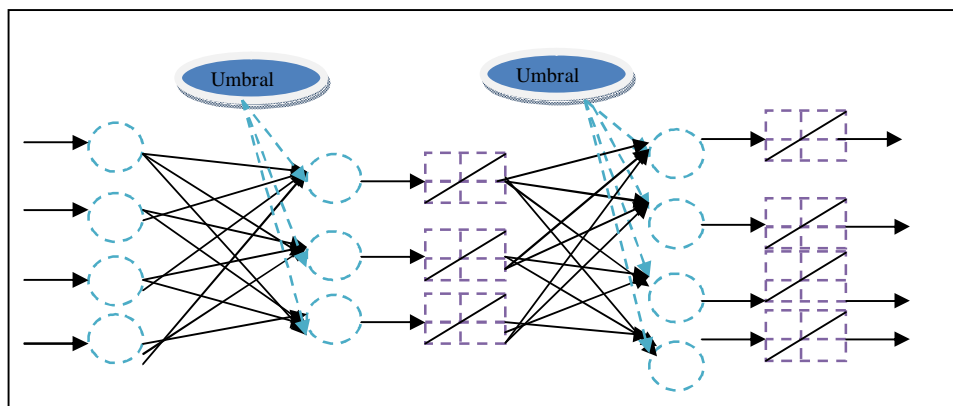
Fórmula 7:

$$\Delta W_{ij} = \alpha \sum_{u=1}^p (t_j^u - y_j^u) X_i^u$$

La función de error es cuadrática respecto a los pesos y esto se debe a la linealidad de la neurona de salida, dando como resultado una superficie en forma de paraboloide con un mínimo global, sin tener en cuenta los pesos iniciales de la red.

GRÁFICO 12

REPRESENTACIÓN DE UNA RED NEURONAL MADELINE



Elaboración: José Miguel Fernández, Raquel Flores López.

Fuente: Gráfico Adaptado. Redes Neuronales. Pág. 49

REGLA LEAST MEAN SQUARES

También conocida como la regla delta, trata de encontrar una matriz de pesos W con valores óptimos para la resolución del problema, enfocándose en el concepto de mínimos cuadrados, mediante múltiples iteraciones.

La regla consiste mediante una función de error poder medir el rendimiento actual de la red, dependiendo de los valores sinápticos de la red y establecer el proceso de optimización que modifique los pesos para alcanzar el punto óptimo de la red.

Uno de los métodos más utilizados es el descenso por el gradiente, donde por cada patrón que reciba la red determinará una función de Error $E(W)$ en base al error actual, dado cada patrón se deberá iterativamente encontrar el mínimo local, para poder obtener un mínimo global óptimo, que será el objetivo del proceso de aprendizaje de la red.

Para determinar lo anterior se empieza determinando una configuración de pesos iniciales, y para $t = 0$ se calcula el sentido de la máxima variación de la función $E(W)$ para $W = 0$ que es definido por su gradiente en $W(0)$. El sentido de la máxima variación (máxima gradiente) apuntará a la hipersuperficie de E (gráfico 13 – 14), después de esto se modifican los pesos siguiendo el sentido contrario al que indica el gradiente llegando a un mínimo local después del mismo proceso iterativo, es decir :

Fórmula 8:

$$W(t + 1) = W(t) - \eta \nabla E(W)$$

GRÁFICO 13

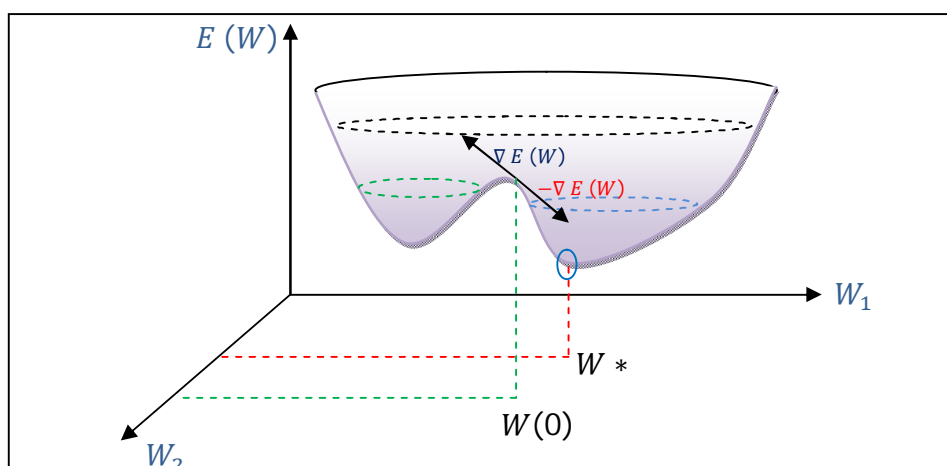
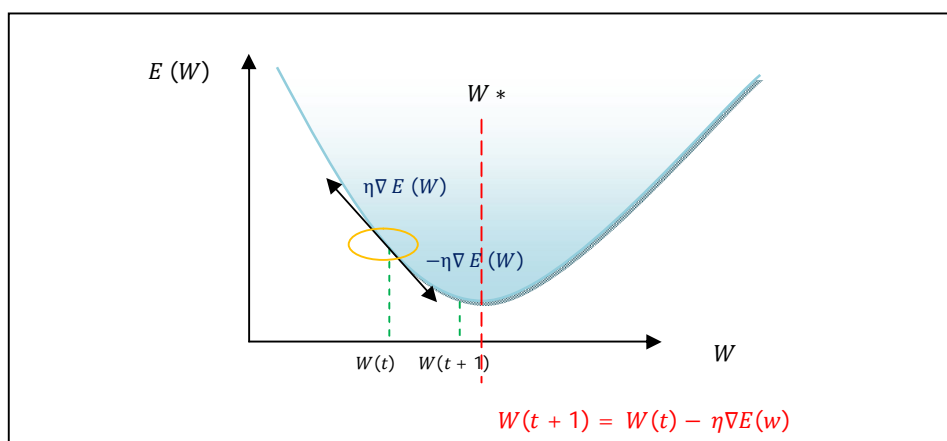
REPRESENTACIÓN DE LA SUPERFICIE DE ERROR $E(W)$ 

GRÁFICO 14

REPRESENTACIÓN DEL DECENSO POR LA GRADIENTE DE ERROR HACIA UN MÍNIMO LOCAL



Elaboración: Propia del autor del presente trabajo.

Fuente: Gráfico del descenso por la gradiente de error sin superficie. Adaptado al gráfico 13 pág. Anterior.

PERCEPTRON MULTICAPA

La gran limitante del perceptron simple mencionada anteriormente conllevaron con la ayuda del modelo previo al diseño de una RNA con n capas las cuales, interactuaban entre si para realizar el proceso de desencadenar varios perceptrones juntos y representar un modelo de solución más próximo al conjunto de problemas no lineales.

El perceptron multicapa está compuesto de varias capas ocultas, la cuales sus entradas son las salidas de las unidades de la capa anterior.

Recordamos que un red neuronal está en contacto con su entorno a través de sensores y emite una respuesta mediante actuadores, el diseño de este tipo de redes acercan a la realidad del procesamiento de información del cerebro, las capas ocultas representan la estructura funcional de cálculo y transformación de información.

La idea general de este tipo de perceptron es combinar las diferentes unidades y utilizar una función de activación no lineal para aumentar la cantidad de representaciones de funciones en un hiperplano.

Se construye un perceptron utilizando como función de activación una función derivable para su representación, algunas de las más utilizadas son la función sigmoide dado por:

Fórmula 9:

$$\gamma = \frac{1}{1 + e^{-g(x)}}$$

Para la etapa de entrenamiento se trata de minimizar el error medio cuadrático y se determina como la regla delta:

Fórmula 10:

$$E(\bar{w}) = \frac{1}{2} \sum (t_p - o_p)^2$$

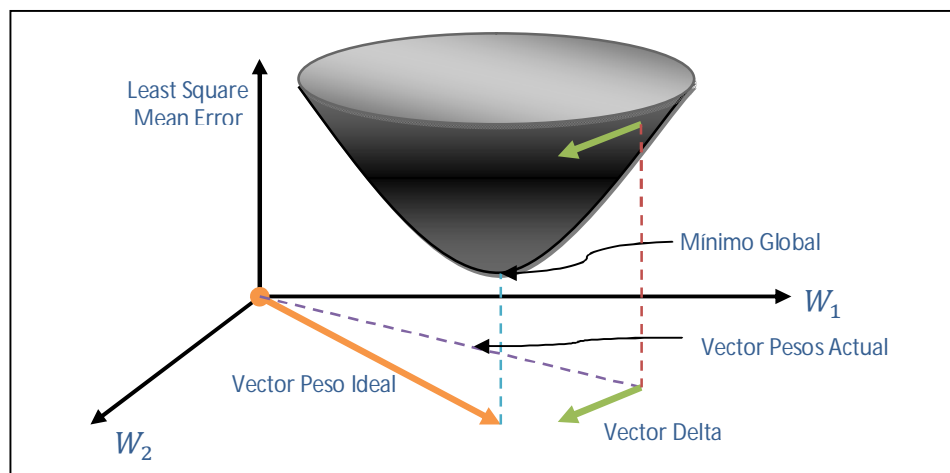
o_p = respuesta de la red para el patron p

t_p = respuesta esperada para el patron p

$o_p = \gamma(g(x))$;

GRÁFICO 15

SUPERFICIE DE ERROR CON RESPECTOS A LOS VECTOR DE PESOS O SINAPSIS.



Elaboración: Facultad de Ingeniería Eléctrica de la Universidad Tecnológica de Pereira

Fuente: <http://www.640kbits.es/?p=11>.

Teniendo la gradiente del error en un punto de la superficie se obtendrá la dirección de mayor crecimiento de la función, como el objetivo de la regla de aprendizaje es minimizar el error, deberá tomarse la dirección negativa de dicho gradiente para aproximarse al mínimo global de la superficie del error medio cuadrático.

Denominamos error medio cuadrático:

Fórmula 11:

$$\hat{E} = \frac{1}{p} \sum_{p=1}^p E_p^2$$

Donde $E = t_p - O_p$, que indica la sumatoria de las diferencias al cuadrado entre lo real y lo proyectado

En una superficie diferenciable la dirección del máximo crecimiento viene dado por el valor del gradiente $\nabla E(\bar{w})$. El valor negativo de ∇ (Fórmula 12b) proporciona la dirección de máximo descenso hacia el mínimo de la superficie.

Se comienza con un valor de pesos aleatorios y modificando n veces (Fórmula 12b) en desplazamientos pequeños en la dirección contraria al gradiente así:

Fórmula 12:

$$a) \quad \bar{w} = \bar{w} + \Delta \bar{w}.$$

$$b) \quad \Delta \bar{w} = -\eta \nabla E(\bar{w})$$

$$\text{gradiente: c) } \nabla E(\bar{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

El vector de las derivadas parciales de E con respecto a cada w_i , representa la gradiente (Fórmula 12c).

En general que da expresado así:

Fórmula 13:

$$\nabla E(\bar{w}) = \frac{\partial}{\partial \bar{w}} \frac{1}{2} \sum_p (t_p - O_p)^2$$

$$\text{si: } \phi = (t_p - O_p)^2 \quad ; \quad \beta = \sum_i w_i x_i$$

$$\frac{\partial E}{\partial \beta} = \frac{\partial E}{\partial \phi} \frac{\partial \phi}{\partial \beta} = \frac{1}{2} (2) \phi \frac{\partial \phi}{\partial \beta} = \phi \frac{\partial \phi}{\partial \beta}$$

$$\frac{\partial E}{\partial w_i} = \sum_p (t_p - O_p) O_p (1 - O_p) X_p$$

Los pesos para las neuronas de salidas se actualizaran de la siguiente forma de acuerdo a su patrón (Fórmula 14):

Fórmula 14:

$$w_i = w_i + \eta (t_p - O_p) O_p (1 - O_p) X_p + \alpha \Delta w_i ;$$

$$\phi = (t_p - O_p) O_p (1 - O_p) \text{ es el error de salida}$$

Comúnmente $\alpha = \text{valores cercanos a } 1$ para evitar mínimos locales

Para las capas ocultas,

$$\phi_j = O_j (1 - O_j) \sum_{\# \text{ in } a \text{ } j} w_i \phi_j$$

REDES DE HOPFIELD

Durante la década de los 80' apareció este tipo de modelos y aportó mucho a los avances en el campo de las RNA, estableciendo un nuevo modelo de arquitectura, dinámica y entrenamiento parecidos a ciertos modelos de Física estadísticas, utilizando diversos modelos matemáticos.

RED NEURONAL FEEDFORWARD

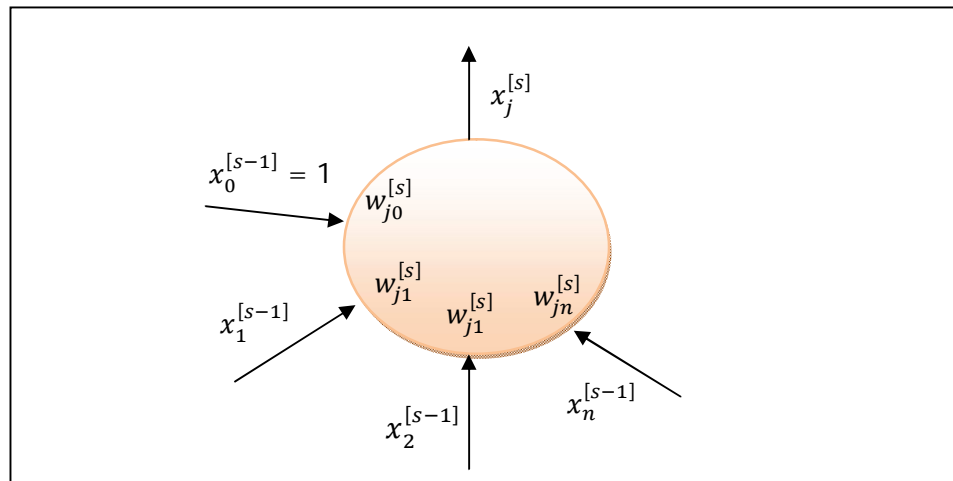
En informática significa alimentación hacia adelante. Un sistema dinámico presenta retroalimentación (Feedforward) si la salida de un elemento del sistema incluye en parte de la entrada aplicada a ese elemento. Las redes neuronales feedwoard son una de las mas estudiadas a nivel científico y en campos de aplicación, son redes multicapas y recurrentes. Se distinguen por:

- ✓ Las neuronas o nodos constituyen elementos básicos de procesamiento.
- ✓ La arquitectura de la red descrita por las conexiones ponderadas entre los nodos.
- ✓ Para encontrar los parámetros de la red se utiliza el algoritmo de entrenamiento.

Una RNA de feedwoard está compuesta por lo menos en una capa de entrada y salida con un número determinado de nodos, teniendo la posibilidad de tener una capa oculta en medio de las dos capas señaladas.

GRÁFICO 16

MODELO ELEMENTAL DE UNA NEURONA DE RED FEEDFORWARD



Elaboración: Paolo Massino Buscema Roma

Fuente: Artificial Neuronal Network. Febrero 22, 2007. Patente No. US 2007/0043452 A1

Las redes Feedforward actuales presentan un comportamiento considerable mejor que las estructuras Feedback, ya que requieren menor capacidad de memoria y llegan más rápidamente a una solución una vez entrenada.

En la figura se muestra el modelo elemental de una neurona artificial, donde:

$[s]$ = Representa el numero de capas, para $s=1$ capa de entrada, incrementa su valor hasta la capa de salida s .

$X_j^{[s]}$ = Es la variable de salida de j_n nodos de una capa $[s]$.

$X_i^{[s-1]}$ = Son las i_n entradas de la capa $[s-1]$ a la capa s

$X_0^{[s-1]}$ = Es usado como el factor umbral, es una entrada a la capa $[s]$, generalmente es un valor fijo de 1.

$W_{ji}^{[s]}$ = Son los pesos de las conexiones entre los nodos i_n de la capa $[s-1]$ y los j_n nodos de la capa s

n = número de entradas del nodo.

En cada nodo de la red opera de una transformación no lineal a una transformación lineal de cada entrada (Fórmula 15).

Fórmula 15:

$$X_j^{[s]} = F(L(W_{ij}^{[s]}, X_i^{[s-1]}))$$

Donde F es una función no lineal, como por ejemplo la función sigmoide, y L es una función lineal representada por el sumatorio de las entradas de cada nodo multiplicado por sus respectivos pesos así (Fórmula 16):

Fórmula 16:

$$X_j^{[s]} = \gamma \sum_{i=0}^n W_{ij}^{[s]} X_i^{[s-1]}$$

METODOLOGÍAS DE CONSTRUCCIÓN DE UNA RED NEURONAL

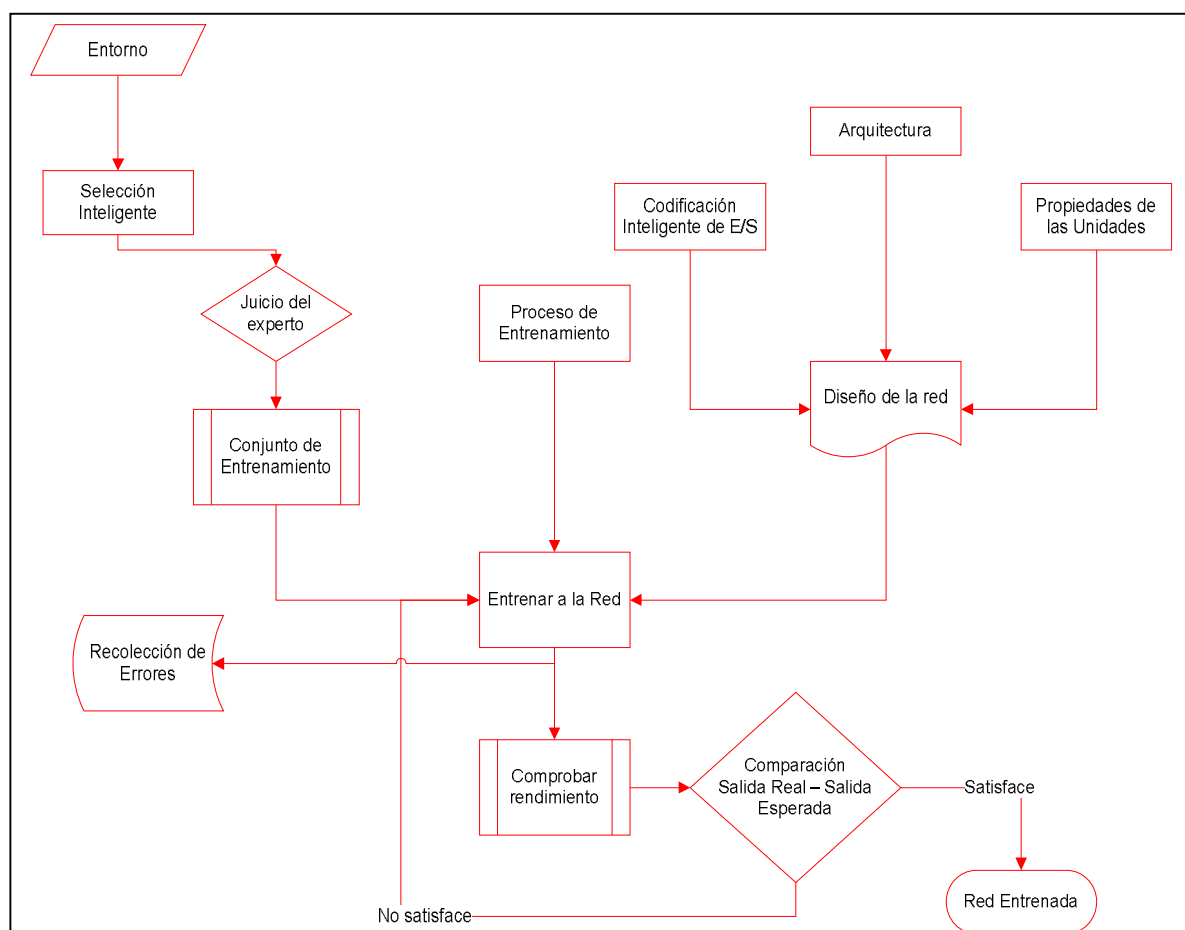
Se deberá crear una arquitectura del modelo a utilizar indicando las propiedades de cada unidad y los procesos que dichas unidades van a realizar, por ejemplo la actualización de los pesos, el efecto que tendrá las funciones de activación, el factor de aprendizaje, todo lo que enmarca al diseño de la red.

El proceso de aprendizaje que dicho modelo seguirá, vendrá dado en función de la actividad para la cual va a ser entrenado, haciendo una analogía con el funcionamiento del ordenador: En el proceso de percepción de un objeto determinado, por ejemplo un automóvil, son muchas neuronas las que intervienen, una de las cuales fue entrenada para distinguir un círculo de las demás figuras geométricas y poder asimilar a una rueda, esto, si lo vemos a pequeñas escala. Ahora podemos imaginar las distintas unidades especializadas con procesos simples e individuales que colaboraran para el proceso de percepción de un automóvil.

Lo más importante del modelo está en su entrenamiento, es aquí donde muchas veces intervienen el juicio experto que ayudará a validar si la respuesta es la correcta. Una metodología de construcción de una red neuronal como se muestra a continuación.

GRÁFICO 17

DIAGRAMA DE FLUJO PARA LA IMPLEMENTACION DE UN PROTOTIPO NEURONAL



Elaboración: Daniel Borrajo y Fernando Fernández

Fuente: Universidad Carlos III Madrid. Grupo de Investigación en Planificación y Aprendizaje Automático.

<http://www.plg.inf.uc3m.es/docwe>

NORMALIZACIÓN DE PATRONES DE ENTRENAMIENTO.

Después de contar con una arquitectura definida para la RNA, se deberá normalizar los datos para que el aprendizaje se aproxime rápidamente al óptimo o deseado, y esto seguidamente de haber definido una buena selección de pesos, umbrales, momento y factor de aprendizaje inicial.

La normalización de datos consiste en redefinir los datos de entrada y salida para nuestro modelo creado; esto es minimizar el rango en el que se sitúen los mismos para que el tiempo que se encargue de encontrar sus representaciones sea el mínimo.

Si utilizamos una determinada función común no lineal para la activación de una neurona sea cual sea, vendrá delimitada por dos asíntotas que la función se acercará por sus dos extremos con valores infinitamente grande o pequeño, como vemos en el gráfico 18.

Mientras la función trata de acercarse a las asíntotas correspondientes el proceso de aprendizaje retrasa su convergencia a los valores cercanos a 0 o 1 respectivamente, es por este motivo que se utiliza un esquema de normalización para las entradas y las salidas de los respectivos patrones de entrenamiento, tratando de que el rango que se encuentre sea lo más estrecho posible. Por ejemplo (Rango de 0,1 – 0,9 en la función sigmoide).

GRÁFICO 18

FUNCIÓN SIGMOIDE UTILIZADA PARA LA ACTIVACIÓN DE UNA NEURONA ARTIFICIAL

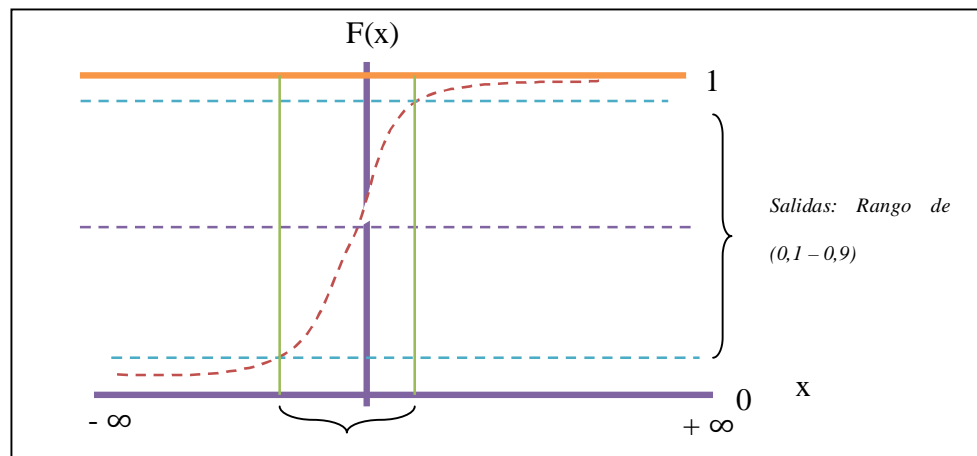
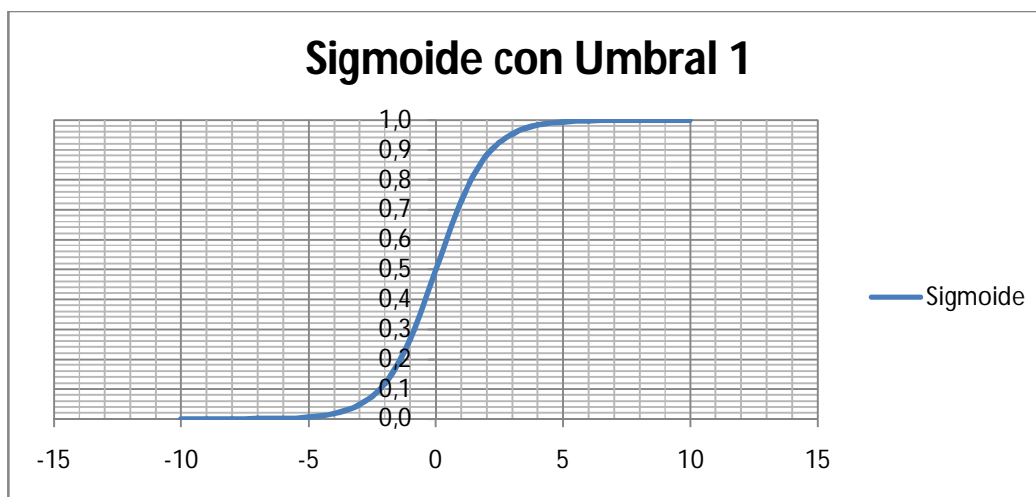


GRÁFICO 19

FUNCIÓN SIGMOIDE UTILIZANDO UN UMBRAL DE 1



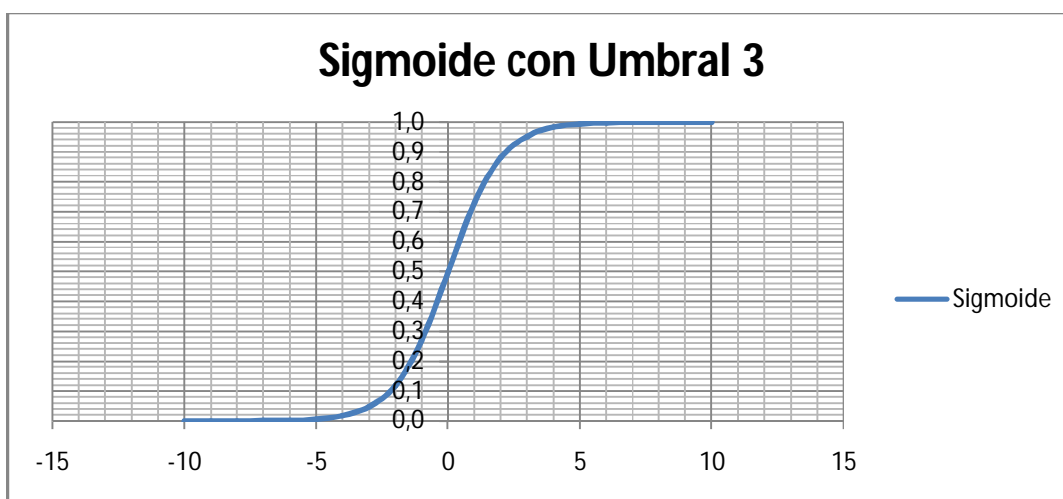
Elaboración: Elaborado por el autor del presente trabajo.

Fuente: Gráfico adaptado a la función sigmoide con valores de entrada de rango $\{-10,10\}$, generando salidas de rango $\{0,0,1,0\}$

Visualizamos en el gráfico 19, que, para que la salida tome un rango de 0 a 1 las entradas debe estar en el un rango cercano a 0. Para la gráfico 19 notamos que las salidas comienzan a ser asintóticos a partir de valores -5 y 5, aunque el rango más optimo es el que se acerque a 0.

GRÁFICO 20

FUNCIÓN SIGMOIDE UTILIZANDO UN UMBRAL DE 3



Elaboración: Elaborado por el autor del presente trabajo.

Fuente: Gráfico adaptado a la función sigmoide con valores de entrada de rango $\{-10,10\}$, generando salidas de rango $\{0,0, 1,0\}$

Los métodos más utilizados para la normalización de datos son:

Método Máximo:

Encuentra el máximo valor a normalizar del vector $\text{valor}(i)$, y se representa mediante la fórmula:

Fórmula 17:

$$\text{Valor Normalizado} = \frac{\text{Valor}(i)}{\text{Máximo}}$$

Método Mínimo - Máximo:

Encuentra el máximo y mínimo valor a normalizar del vector valor (i), y se representa mediante la fórmula:

Fórmula 18:

$$\text{Valor Normalizado} = \frac{\text{Valor}(i) - \text{Mínimo}}{\text{Máximo} - \text{Mínimo}}$$

Los datos normalizados cae en el rango de 0 a 1, sea cual sea los rangos originales, teniendo en cuenta que siempre habrá un valor 0 y un valor 1.

COMPORTAMIENTO GENERAL DE APRENDIZAJE

Dentro del aprendizaje que se lleva a cabo dentro de un proceso neuronal artificial que recibe una serie de entradas y genera salida existe un comportamiento similar al de el sistema neuronal, donde;

“El aprendizaje puede definirse por el que una red neuronal crea, modifica o destruye sus conexiones o pesos en respuesta a su información de entrada”.²¹

En el sistema neuronal artificial este proceso pasa por las modificaciones de los pesos pasa a tener distintos valores. La red ha aprendido cuando los valores de los pesos permanecen estables es decir;

²¹ Raquel Flores López y José Miguel Fernández. Redes Neuronales Artificiales. *Fundamentos teóricos y aplicaciones prácticas*. (2008) Pág. 31

$$\frac{\partial w_{ij}}{\partial t} = 0$$

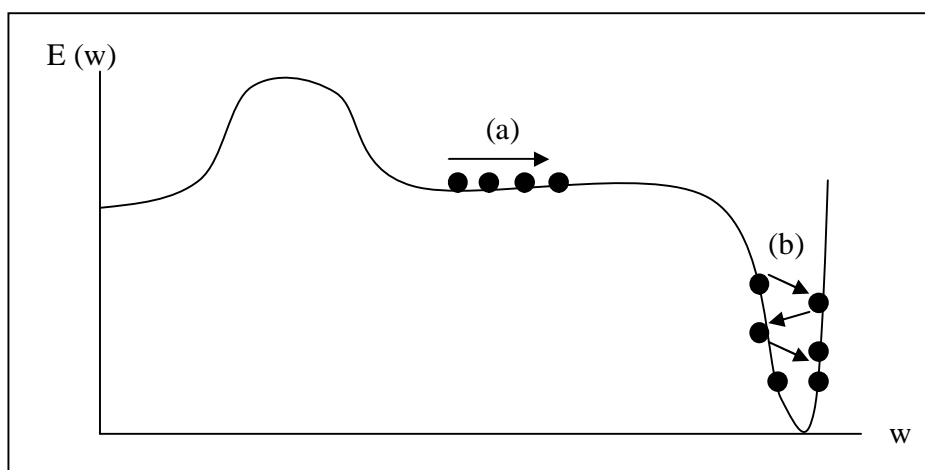
Algo importante que debe considerarse para el aprendizaje, es conocer cuáles son los criterios que se siguen para cambiar los valores de los pesos o conexiones.

En ocasiones el cómputo de la derivada parcial de la función de error con respecto a los pesos puede reducir de manera mínima dando como resultado un aprendizaje lento. Por lo tanto:

- Si la superficie de error es casi plana a lo largo de una dimensión del vector de pesos, la magnitud de la derivada resulta muy pequeña necesitando un número mayor de iteraciones (véase grafico 20 región (a)).
- Si la superficie de error es altamente curva a lo largo de una dimensión del vector de pesos, la magnitud de la derivada es elevada por lo cual el ajuste de peso es elevado con la posibilidad de sobrepasar el valor óptimo buscado (véase grafico 20 región (a)).

GRÁFICO 21

SITUACIONES DE CONVERGENCIA LENTA DEL ALGORITMO DE APRENDIZAJE BACKPROPAGACIÓN



Elaboración: Raquel Flores López y José Miguel Fernández.

Fuente: Redes Neuronales Artificiales. Fundamentos teóricos y aplicaciones prácticas. (2008) Pág. 73

CRITERIOS DE PROCESAMIENTO PARA DETERMINAR EL APRENDIZAJE DE UNA RED NEURONAL

Uno de los criterios que se sigue es que se determina si la red puede aprender durante su funcionamiento habitual (Aprendizaje ON LINE) o si se necesita la desconexión hasta que el proceso de aprendizaje termine (Aprendizaje OFF LINE).

Cuando el aprendizaje es en línea no se distingue una fase de operación con una fase de entrenamiento, es decir la red siempre estará en ejecución aun cuando se le presente nuevos datos.

El aprendizaje fuera de línea diferencia una fase de entrenamiento de una fase de operación, nuevos datos ingresan a un entrenamiento dando lugar a una configuración ideal de pesos y un grupo de datos de pruebas para las validaciones. Debido a su carácter estático estos sistemas no presentan problemas de estabilidad en su funcionamiento.

ASOCIACION DE LOS PATRONES PARA EL ENTRENAMIENTO DE UNA RED NEURONAL

Llamaré asociación a las distintas formas de presentar las entradas y salidas de un patrón determinado para el entrenamiento y validación de una red.

Una de las formas que existe es la de heteroasociación, donde la red aprende mediante una serie de datos asociados en parejas, de tal forma que cuando se le presente una entrada A_i debe responder con una salida B_i . Como:

$$[(A_1, B_1), (A_2, B_2), (A_3, B_3)]$$

La otra forma es la auto asociativa, donde la red aprende ciertas informaciones A_1, A_2, \dots, A_n de tal forma que cuando se le presente una información de entrada realizará la auto correlación, respondiendo a cada uno de los datos almacenados, el más parecido a la entrada.

Mediante estas dos formas podremos separar a dos grandes modelos de RNA; Las redes heteroasociativas, donde asocian información de entrada con diferentes informaciones de

salida, y las redes autoasociativas, donde asocian una entrada con el ejemplar mas parecido de los almacenados conocidos por las red.

CONSIDERACIONES GENERALES PARA ESTABLECER LA MEJOR CONFIGURACION INICIAL DE ENTRENAMIENTO

Si bien es cierta una arquitectura neuronal sin una configuración inicial no podrá ser posible el aprendizaje, es importante considerar ciertos aspectos para encontrar una buena definición de variables iniciales.

Teniendo en cuenta que la variación de dichas variables puede afectar notoriamente a la aceleración del proceso de aprendizaje y a la evaluación del entrenamiento neuronal.

Uno de los primeros aspectos a considerar es el número de capas que tendrá nuestra arquitectura neuronal. Partiendo de que una red neuronal solo puede constar de una capa de entrada y una de salida, es necesario detenerse a analizar el número de capas oculta que tendrá dicha RNA.

Como vimos en el presente trabajo (gráfico 9) solo es posible definir una solución no lineal a partir de por lo menos una capa de neuronas ocultas. Ahora bien el cómo definir el número de capas. Por mucho tiempo se ha llegado a través de estudios a conclusiones de que una sola capa oculta resulta suficiente para que la red actuara como una aproximador universal de funciones. Esta serie de estudios se inició a partir de la década de los 80 cuando DENKER verificó que toda función booleana podía representarse a

través de una red multicapa con una sola capa oculta. Con el teorema de Kolmogorov, HETCH-NIELSEN (1987 – 1990) demostró con una arquitectura similar al de red multicapa y con una sola capa oculta era suficiente para el desarrollo de problemas no lineales. Y así durante la década del 80 – 90 varios grupos de investigadores propusieron diversos modelos matemáticos que demostraban lo mismo (FUNAHASHI, 1989, HORNYK 1991, KURKOVÁ, 1992, BARRON, 1994).

En general no existe una regla fija que nos indique el número de capas ocultas que tendrá la red que utiliza el algoritmo backpropagation y la decisión de la misma depende de simple intuición del diseñador de la red, sin embargo existen una serie de reglas generales aplicadas por investigadores en el campo:

Regla 1: Cuando aumenta la complejidad del problema a utilizar, el número de capas ocultas también deberían aumentarse.

Regla 2: Si el proceso que se está modelando es separado en dos etapas, entonces nuevas capas ocultas pueden ser adicionadas. Si el proceso no es separable en varias etapas, entonces las capas adicionales solo permiten memorizar y no siempre es una solución general.

Regla 3: El número de patrones de entrenamiento disponibles establecen una cota superior en el número de neuronas en la(s) capa(s) oculta(s). Para calcular esta cota superior primero hay que dividir el número total de neuronas de las capas de entrada y

salida. Dividir el resultado de nuevo por un factor de escala entre 5 y 10. Factores de escala grandes son usados cuando existen datos con ruido. Cuando los datos de entrada tienen muchísimo ruido será necesario incluso factores más elevados (20 o 50), mientras que datos en los que no exista apenas ruido podríamos disminuir el factor hasta un valor de 2. Es muy importante que la(s) capa(s) oculta(s) tengan pocas neuronas ya que demasiadas neuronas podrían llevar a que los patrones de entrada fueran memorizadas de manera que ningún tipo de generalización fuera posible, haciendo que la red fuera inútil con nuevos datos de entrada.²²

Un segundo detalle a considerar es el valor de los pesos y umbrales iniciales, diferentes condiciones pueden dar diferentes resultados, es importante determinar que por ser pesos aleatorios debemos definir un rango en el que se encuentren estos valores. Para encontrar una configuración cercana a la óptima es necesario llevar a cabo múltiples procesos de aprendizajes. Por lo general el valor de los pesos aleatorios debe ser muy pequeño por el manejo de una función de activación cercana a 0.

El número de iteraciones es también importante debido a que él un número mayor de estas supone un mayor tiempo de procesamiento, recordemos que para la red converja en un menor número de iteraciones es necesario definir una configuración inicial mejor.

²² Ivan Martinez Ortiz. Universidad de Complutense de Madrid, trabajo de Doctorado Aprendizaje Automático. Facultad de Informática. http://gaia.fdi.ucm.es/people/pedro/aad/ivan_martinez.pdf

También el tamaño de muestra o número de patrones a entrenar, hace que la red tenga un mejor aprendizaje, una reducida cantidad de patrones minimiza un aprendizaje óptimo. Sin embargo una mayor cantidad de patrones maximiza el tiempo de proceso de la red neuronal.

El valor de la variable de aprendizaje está por lo general ubicado entre 0 y 1, se utiliza valores pequeños para acelerar el proceso de aprendizaje.

El termino momento (β) fue propuesto como una de las mejoras para el algoritmo backpropagation, proporcional al incremento de la iteración anterior. Reduce las posibilidades de inestabilidad que se crea en la variación de los pesos al minimizar la posibilidad de caer en un mínimo local, en general este factor se encarga de acelerar la convergencia de las ponderaciones.

La fórmula (véase la fórmula 12a) para la actualización de los pesos se verá afectada de la siguiente manera (Fórmula 19):

Fórmula 19:

$$w_{t+1} = w_t + \Delta w_t + \beta \Delta w_{t-1}$$

El término momento trata de evitar oscilaciones en la hipersuperficie de error e incrementar la convergencia en regiones con poca pendiente. Existen pocos desarrollos teóricos que determina una mejor selección de esta variable, desde el punto de vista empírico se suelen tomar valores entre 0 y 1 generalmente próximos al extremo superior,

por ejemplo 0,7 o 0,8²³. Se ha verificado que cuanto mayor sea el término momento más reducido debe ser la tasa de aprendizaje η , se establece la siguiente relación entre estas dos variables²⁴:

- η decrece a medida que β aumenta de 0 a 1.
- El uso del término momento acelera el proceso de aprendizaje.
- El uso de una tasa de momento muy elevada no da lugar a inestabilidades siempre que el factor de aprendizaje sea suficientemente pequeño.

SOLUCIONES A PROBLEMAS COMUNES EN EL ENTRENAMIENTO DE UNA RED NEURONAL CON ALGORITMO BACKPROPAGATION

Es importante detectar un fracaso para poder reducir el error. A continuación se presentan los errores típicos que impiden un mejor aprendizaje²⁵:

1. Si la combinación de los pesos ha llegado a un local mínimo de la superficie de error y no varía, la solución es volver a inicializar los pesos aleatorios y comenzar nuevamente.

²³ Raquel Flores López y José Miguel Fernández. Redes Neuronales Artificiales. *Fundamentos teóricos y aplicaciones prácticas*. Pág. 75

²⁴ Raquel Flores López y José Miguel Fernández. Redes Neuronales Artificiales. *Fundamentos teóricos y aplicaciones prácticas*. Pág. 75-76

²⁵ http://www.colinfahey.com/neural_network_with_back_propagation_learning/neural_network_with_back_propagation_learning_es.html

2. El tipo de aprendizaje es demasiado alto (más de 1 es probablemente excesivo) y las actualizaciones siempre rebasan el objetivo, entonces debemos reducir la tasa de aprendizaje.
3. Es demasiado lento el tiempo de aprendizaje, se mejora aumentando el factor de aprendizaje.

PREGUNTAS A CONTESTARSE

¿Cuál es el número óptimo de unidades neuronales para obtener los mejores resultados de aprendizaje?

¿Cuál es el valor óptimo de la tasa de aprendizaje y momento para una mejor convergencia?

¿Determinar la configuración inicial de pesos y umbrales para el desarrollo del problema actual de establecimiento de límite de crédito?

¿Determinar el tiempo requerido para que el aprendizaje obtenga un margen de error deseado?

VARIABLES DE INVESTIGACION

- Curva de error.
- Porcentajes de error por cada patrón de entrenamiento.

DEFINICIONES CONCEPTUALES

Curva de error, llamada así a la función de la sumatoria cuadrática de la variación de los pesos en relación a los números de ciclos de entrenamiento.

El porcentaje de error por cada patrón de entrenamiento después de haber sido efectuado el proceso de aprendizaje.

CAPITULO III

METODOLOGÍA

DISEÑO DE LA INVESTIGACIÓN

MODALIDAD DE INVESTIGACION

La modalidad de investigación elegida es la de un proyecto factible al poder comprobar el rendimiento del proceso de entrenamiento a la que va ser expuesta la arquitectura planteada, mediante una serie de configuraciones y sus resultados obtenidos.

Se podrá verificar el rendimiento de todos las propuesta aquí expuestas, mediante gráficos y cuadros donde se compararan las diferentes variables de investigación planteadas en el capitulo anterior.

TIPO DE INVESTIGACIÓN

Este trabajo utiliza el método investigativo al tener en cuenta lo siguiente:

El método investigativo donde el investigador modifica o interviene directa o indirectamente sobre el objeto de estudio para encontrar las condiciones necesarias. Se diseña experimentos al el fin de reproducir el objeto de estudio.

Dentro de las características de este método encontramos: Aislar el objeto de estudio y sus propiedades, reproducción del mismo en condiciones controladas y modificar las condiciones bajo las cuales tiene el lugar el proceso que se estudia.

Al necesitar una serie de experimentos tales como variar el número de unidades neuronales que necesitamos, establecer los distintos valores de pesos y umbrales, cambiar los factores de aprendizaje y momento, podemos determinar que el presente trabajo utiliza el método experimental.

POBLACIÓN Y MUESTRA

Para la evaluación del presente prototipo se utilizó una muestra de 1094 registros de la cartera de clientes de la empresa donde yo presto mis servicios.

EVALUACIONES DE LA MUESTRAS:

Se tomará de la tabla de cartera los siguientes campos a representar como valores de entrada:

- Valores de Efectivo
- Valores de Cheques al día
- Valores de Depósito
- Valores de Cheques a fecha
- Valores de Cheques Protestados
- Valores de Factura
- Valores de Notas de Ventas

Tenemos que establecer 7 neuronas la cuales van a alimentar la capa de entrada a cada una de las respectivas neuronas.

El valor de salida será el Valor de límite de Crédito actual, que representará la neurona en la capa de salida.

CONFIGURACIÓN DE ENTRENAMIENTO

Arquitectura:

Capa	# Neuronas
1	7
2	3
3	1

Variables de Aprendizajes:

Factor de Aprendizaje	0.30
Factor Momento	0.70
Umbral	-1
Pesos aleatorios	± 0.5
Números de Ciclos	10000

RECURSOS UTILIZADOS PARA LA ELABORACIÓN DEL PROTOTIPO

Memoria Ram:	3 GB
Procesador:	Intel Core i5 2,40 GHz
Sistema Operativo:	Windows 7
Tiempo de Ejecución del Entrenamiento:	33 Segundos

RECURSOS UTILIZADOS PARA LAS PRUEBAS DE RENDIMIENTO

CUADRO NO. 2
RECURSOS DE HARDWARE UTILIZADOS VS TIEMPO DE RESPUESTA

Memoria	Procesador	Tiempo de ejecución (10000 ciclos)
512 MB	Pentium III	11,20 minutos aprox.
512 MB	Pentium IV	8,40 minutos aprox.
1 GB	Pentium IV	5,25 minutos aprox.
1 GB	AMD Athlon Dual Core 1,80 GHz	2 minutos aprox.
3 GB	Intel Core i5 2,40 GHz	33 seg. aprox.

Dentro de los recursos de hardware utilizados para las distintas pruebas tenemos que estos afectan directamente al más importante criterio de rendimiento en la ejecución del algoritmo como es:

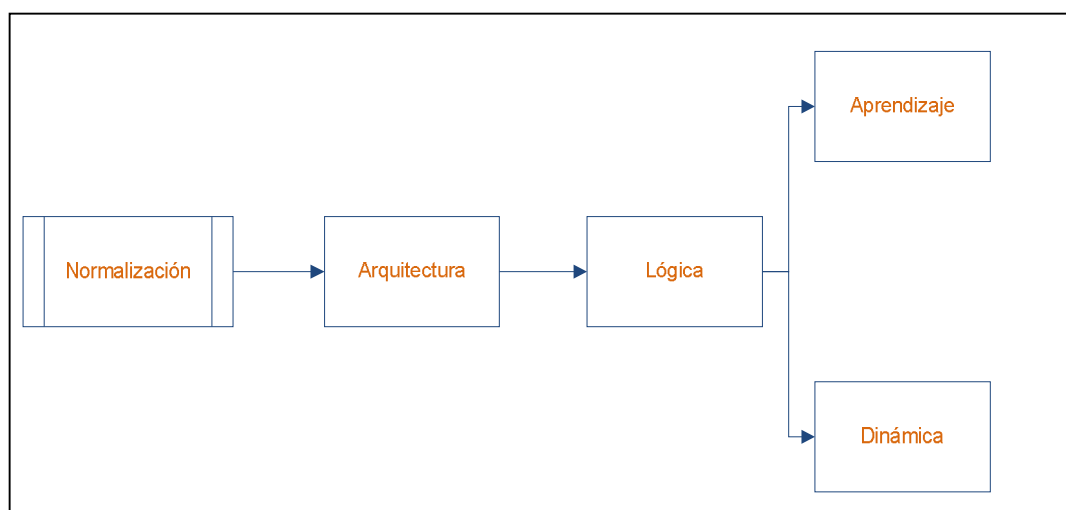
El tiempo de respuesta, que es el tiempo que emplea el algoritmo de entrenamiento para resolver un problema. Como vemos en el cuadro mientras utilizamos procesadores con mayor velocidad de procesamiento y poder de ejecución en paralelo, el tiempo de respuesta minimiza.

Las pruebas realizadas se muestran en el anexo I.

DISEÑO DEL PROTOTIPO

ALGORITMO DE ARQUITECTURA – DINÁMICA - APRENDIZAJE

Para el diseño del prototipo a presentar he elaborado una arquitectura de neuronas agrupadas en niveles de orden conocidas como capas debidamente interconectadas entre sí utilizando una conexión total por cada capa. El diseño lo he dividido en cuatro partes:



Etapas de diseño

La normalización ubicará los patrones de entrada y salida dentro del intervalo manejable para el prototipo.

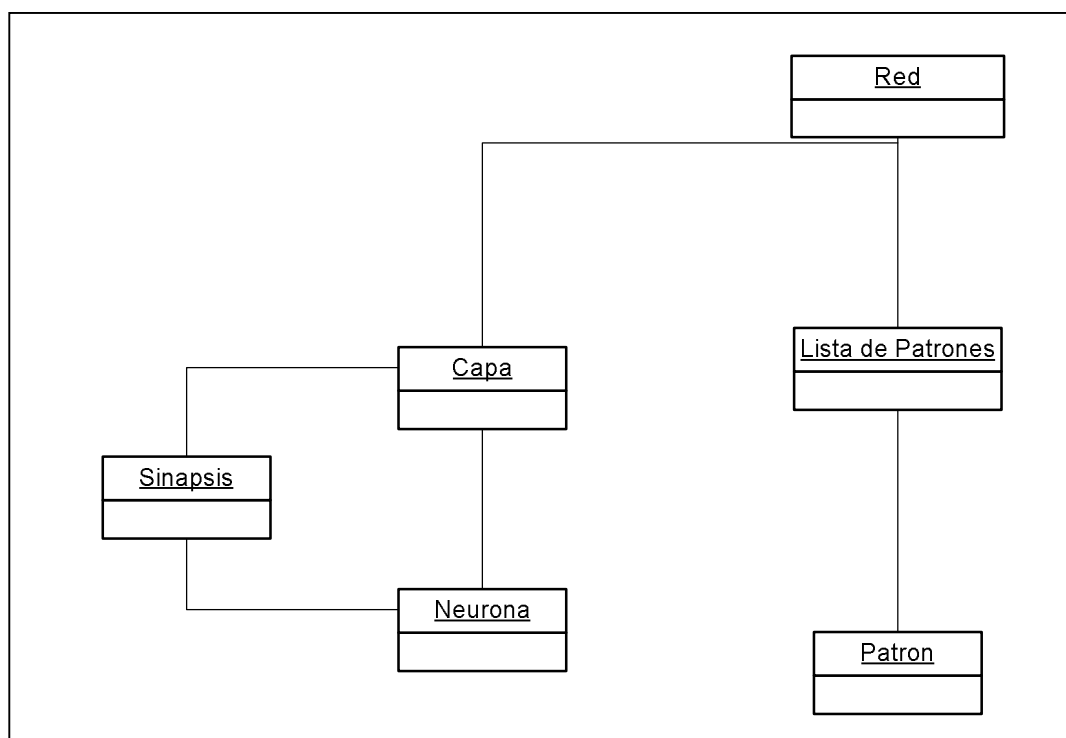
La arquitectura es la encargada de determinar el escenario en el que se desarrollará el modelo.

La lógica vendrá definida por la dinámica: la ejecución y proceso los patrones correspondiente manejando sus determinados ciclos, Y aprendizaje que reestructurará el modelo enfocado a sus conexiones para la convergencia final.

Modelos de Diseño:

Modelo de Objetos:

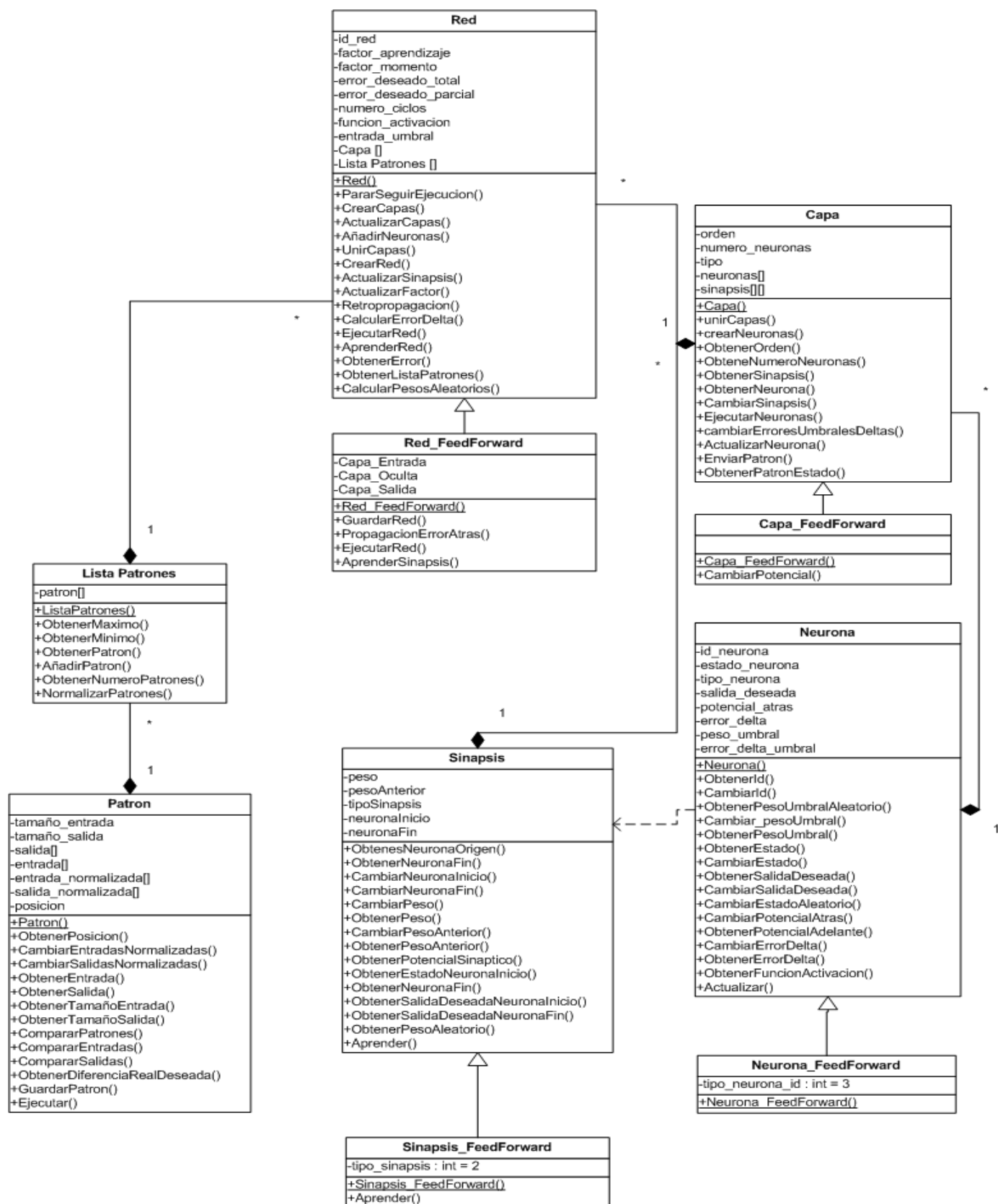
En el modelo de objetos del prototipo podemos definir la estructura del marco de trabajo basado en objetos que determiné para su construcción. Partiendo de un objeto llamado red el cual será el inicio del desarrollo y su determinada composición por parte de un objeto que contendrá los patrones agrupados de manera ordenada llamado Lista de Patrones. A sus ves la red especifica un objeto superior en su nivel llamado Capa, que será el encargado de agrupar a los objetos neuronas con sus sinapsis o pesos correspondientes. El objeto neurona es el que representa la unidad básica de procesamiento, y la cual junto al objeto sinapsis pasarán de diferentes estado en la ejecución y aprendizaje del modelo. En la figura 19 podemos tener de manera genera la descomposición del marco de trabajo de una manera generalizada.



Modelo de Objetos

Modelo de Clases:

El modelo a continuación determina los atributos y relaciones que tendrán cada clase utilizada en el desarrollo. La representación estructural de cada clase viene dependiendo de las necesidades del proyecto, teniendo la posibilidad de permitir la extensibilidad del proyecto dependiendo de la arquitectura a estudiar. Para el prototipo actual el cual he implementado, se adapta fácilmente a una arquitectura de tres capas de neuronas (entrada – oculta - salida) para representar una dinámica hacia adelante o Feedforward y una retropropagación para el aprendizaje o algoritmo Backpropagation.



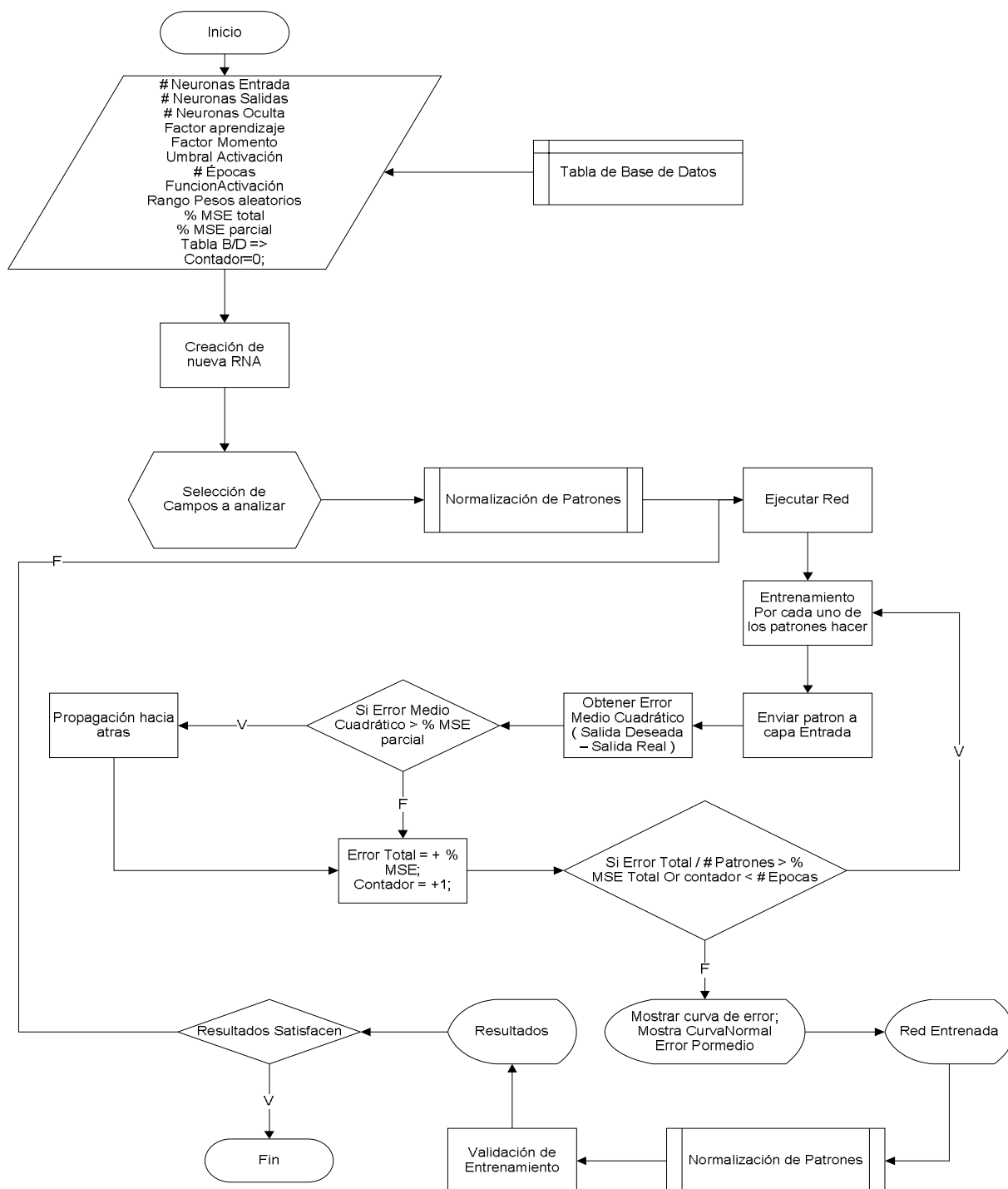


Diagrama de Flujo

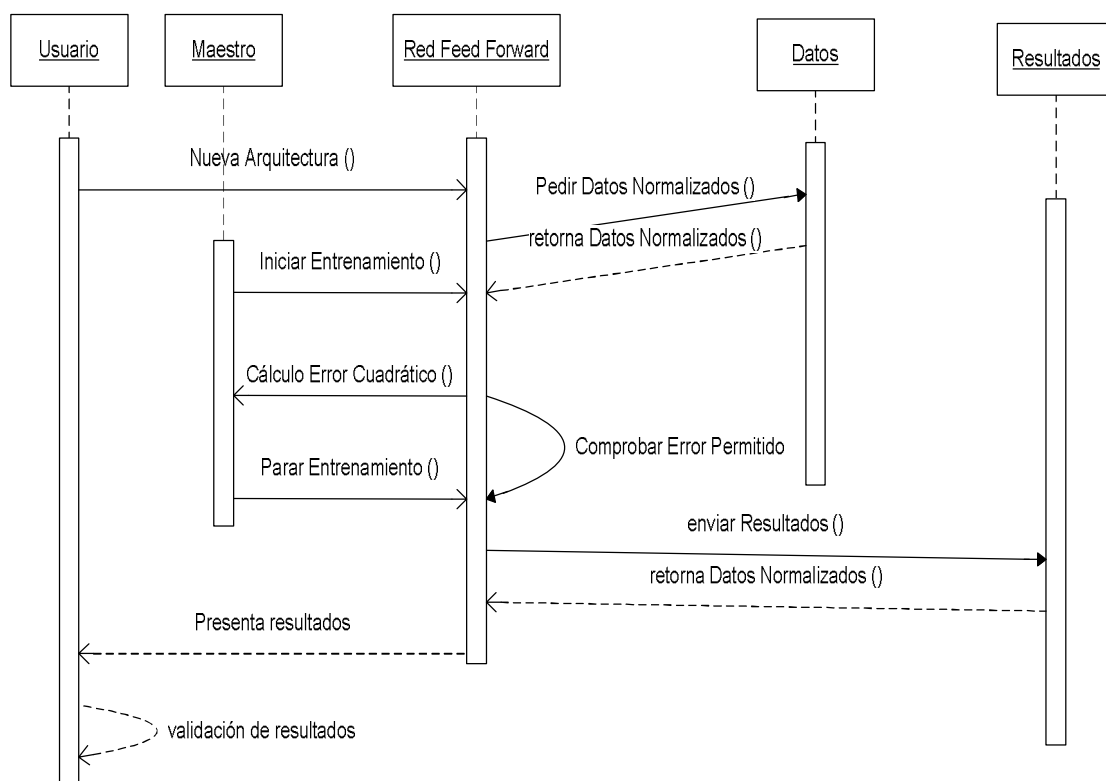


Diagrama de Secuencia

Diseño de la Normalización:

En el diseño de la normalización de datos determinamos algunas clase que serán las responsable del manejo de los diferentes patrones de entrada y salida que manipulará el modelo neuronal.

Estas clases interactúan directamente tanto con la arquitectura y la dinámica de red.

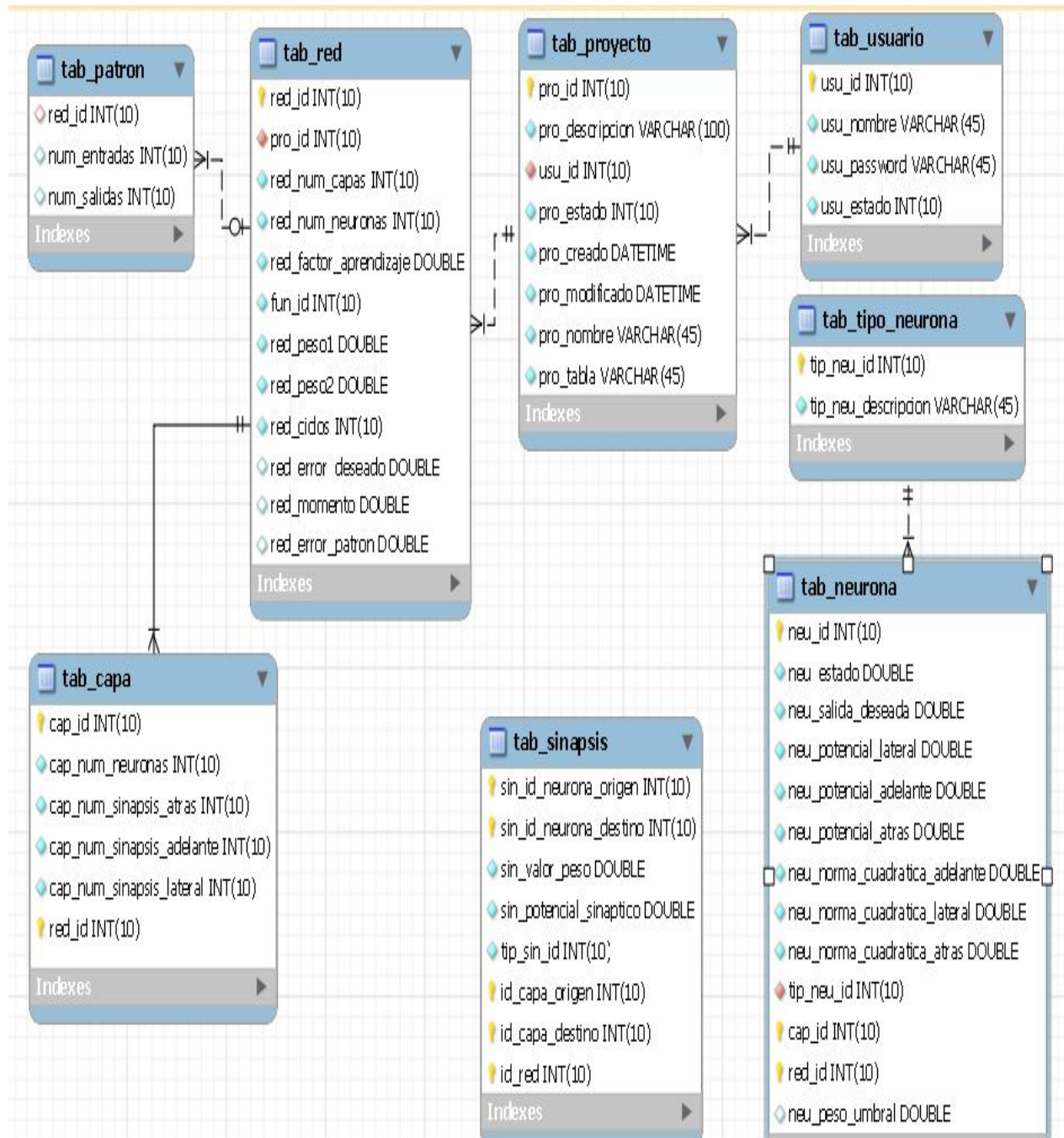
Con la arquitectura al crear el modelo normalizado correspondiente de patrones y su agrupación a la que llamo Lista de Patrones y con la dinámica al envió a su capa correspondiente, sea de entrada o salida.

Diseño de la Arquitectura:

Dentro de la arquitectura es necesario definir los procesos de construcción del modelo, los cuales constará por:

- ✓ Creación de Neuronas
- ✓ Creación de Capas
- ✓ Creación de sinapsis y umbrales

Modelo Entidad - Relación



Explicación del Algoritmo de Desarrollo

Dentro de la definición de los conceptos estudiados en el presente trabajo de investigación pude definir ciertos lineamientos a la hora de desarrollar el algoritmo de entrenamiento o ejecución y el respectivo proceso de aprendizaje. Estos lineamientos basados en los modelos matemáticos estudiados nos pueden ayudar a dar un paso importante en la implementación del código.

Cuando desarrollamos el modelo típico para la construcción de la arquitectura neuronal a utilizar nos damos cuenta que el uso de ciertas clases que podemos heredar para codificar nuestro propio patrón son muy importantes en el momento de representarlas.

Lo primero que se hizo fue la creación de la arquitectura neuronal para ello utilizamos la clase Red que nos está desarrollada para darnos la posibilidad de heredar a una nueva clase que para nuestro prototipo, la cual llamaremos Red_FeedForward, que utiliza un constructor que recibe como parámetros los elementos necesario para definir y crear nuestro nuevo proyecto. En unas cuantas líneas habremos elaborado una arquitectura de tres capas con unión total de neuronas con dirección hacia adelante.


```

public Red_FeedForward(int numNeuronasEntrada,int numNeuronasOcultas,int numNeuronasSalida,double factorAprendizaje,int
ciclos,double peso1,double peso2)
{
    super.setRed();
    capa_entrada= new Capa_FeedForward(numNeuronasEntrada,0);
    capa_oculta= new Capa_FeedForward(numNeuronasOcultas,1);
    capa_salida= new Capa_FeedForward(numNeuronasSalida,2);

    super.setCapa(capa_entrada, 0);
    super.setCapa(capa_oculta, 1);
    super.setCapa(capa_salida, 2);

    super.setPesosAleatoriosNeuronasRed(peso1, peso2);
    super.unirCapas(0, 1, "Total", "Sinapsis_FeedForward",peso1,peso2);
    super.unirCapas(1, 2, "Total", "Sinapsis_FeedForward",peso1,peso2);

    super.setFactoraprendizaje(factorAprendizaje);
    super.setNumeroCiclos(ciclos);

    capa_oculta.setPotencial(capa_entrada);
    capa_salida.setPotencial(capa_oculta);
}

```

Para la ejecución de la red debemos utilizar el método abstracto **ejecutar_red** donde le indicamos a la RNA que envíe las señales hacia adelante y propague la información hasta llegar a las determinadas salidas. Para hacer posible este procedimiento es necesario que el método reciba un patrón a entrenar y una bandera lógica activada que nos indicará que enviamos el patrón a aprender y nos retornará un patrón ejecutado. Los pasos del algoritmo son los siguientes:

*Por cada uno de los patrones de entrenamiento hasta alcanzar condición de parada**
hacer:

1. Enviar un patrón a la capa de entrada y sus respectivas neuronas la cual distribuirá a los estados de neurona inicial con los valores normalizados.
2. Hacer la propagación hacia adelante, es decir calcular los valores de activación de las neuronas superiores, hasta llegar a las neuronas de salida con un valor de salida de red.
3. El patrón procesado es aquel que tiene los respectivos valores de salidas de la ejecución del patrón actual.

```
public Patron ejecutarRed(Patron patron, boolean aprender)
{
    int num_capas=this.getNumCapas();
    // 1.-Enviar el Patron de Entrenamiento n a la capa de Entrada
    super.getCapa(0).enviarPatron(patron,aprender);

    // 2.- Propagacion FeedForward hacia adelante
    for (int xi=1 ;xi < num_capas; xi++)
    {
        super.setPotencialAtras(xi);
        super.getCapa(xi).actualizarNeuronas(super.getFuncionActivacion());
    }

    // 3.- devuelve el patron procesado
    return super.getCapa(this.getNumCapas()-1 ).getEstado();
};
```

```

if (id_capa != 0 )
{
    for(int i=0; i< num_capas_i ;i++)
    {
        potencial=0;
        for (int j=0; j<num_capas_j;j++)
        {
            potencial= potencial + (capas[id_capa-1].getNeurona(j).getEstado() * capas[id_capa-1].getSinapsis(j, i,
            "").getPeso() );
        }
        potencial=potencial + (this.entradaUmbral * capas[id_capa].getNeurona(i).getPesoUmbral());
        capas[id_capa].getNeurona(i).setPotencialAtras(potencial);
    }
}

```

4. Calcular el error cuadrático, esto es la diferencia cuadrática de la diferencia salida de red vs salida deseada
5. Si el error cuadrático calculado por el patrón actual es mayor que el error parcial por patrón deseado, entonces procederemos a la retropropagación para el aprendizaje y reajuste de los pesos y umbrales sinápticos. Caso contrario seguiremos con el paso 1.

```

public void propagacionErrorAtras(Patron salidaRed, Patron salidaDeseada)
{
    double error=0;
    double error_umbral=0;
    int num_capas_i=this.getNumCapas()-1;
    int num_capa_z;
    int num_neuronas_j;

    for(int c=num_capas_i ; c>= 0 ;c--)
    {
        num_neuronas_j=this.getCapa(c).getNumNeuronas();
        for (int j=0; j<num_neuronas_j ;j++)
        {
            error=0;
            if (c==num_capas_i)
            {
                error =salidaDeseada.getValorNormalizadoSalida(j)- salidaRed.getSalida()[j];
                this.getCapa(c).getNeurona(j).setErrorDelta(error);
            }
            else
            {

```

```

        error=0;
        error_umbral=0;
        num_capa_z=this.getCapa(c+1).getNumNeuronas();
        for (int z=0; z < num_capa_z; z++)
        {
            error= error + (this.getCapa(c).getSinapsis(j, z, null).getPeso())*
                (this.getCapa(c+1).getNeurona(z).getErrorDelta());
        }
        this.getCapa(c).getNeurona(j).setErrorDelta(error);
    }
}
}
}

```

6. Realizar el aprendizaje hacia adelante, utilizando la fórmula de aprendizaje estudiada en este trabajo.

```

public void aprenderSinapsis()
{
    int num_capas_a=this.getNumCapas()-1;
    int num_neuronas_b;
    int num_neuronas_c;

    for (int a=0; a<num_capas_a;a++) // por todas las capas
    {
        num_neuronas_b=this.getCapa(a).getNumNeuronas();
        for (int b=0; b<num_neuronas_b;b++)
        {
            num_neuronas_c=this.getCapa(a+1).getNumNeuronas();
            for (int c=0; c< num_neuronas_c;c++)
            {
                this.getCapa(a).getSinapsis(b, c, null).aprender(this.getFactorAprendizaje(),this.getMomento() );
            }
        }
    }
}
}

```

* Condiciones de parada:

- ❖ Que el error total cuadrático por cada patrón sea menor que el error deseado definido para el entrenamiento
- ❖ Que el número de épocas sea igual al número determinador para el entrenamiento actual.

Herramientas Utilizadas

Para el desarrollo del prototipo se ha contado con herramientas libres tanto para el desarrollo como para el estudio del funcionamiento de aplicaciones similares:

El desarrollo del prototipo es completamente en java donde podemos aprovechar los conceptos orientado a objetos y poder representar de una manera eficaz y simple con respecto a la programación estructural. Poder plasmar las distintas entidades que participan para lograr tanto la arquitectura como la lógica funcional en manera de objetos, ayuda al desempeño flexible y a una escalabilidad de mantenimiento y desarrollo horizontal.

Otras de las ventajas muy amplias con respecto a varios lenguajes de programación es la ejecución de su código en la maquina virtual de Java (JVM) haciendo indistinto su funcionamiento en multiplataforma.

La representación de los datos para la utilización dentro del prototipo como para la evaluación de esta, originó el uso de MySQL como el gestor de base de datos, debido a su flexibilidad de mantenimiento, y la fácil compactación con el sistema operativo.

Bibliotecas de JAVA utilizadas:

- JGraph es una biblioteca Java de fuente abierta para la visualización de gráficos. Se rige a los patrones de diseño de la librería swing brindando una familiarización con su fuente.
- JFreechart, su fácil utilización para el desarrollo de gráficos estadísticos hace una de las mejores alternativas al momento de elegir una api para nuestro prototipo. Su uso dentro del proyecto netamente es para el dibujo de la curva de error, la distribución normal de los errores promedio.

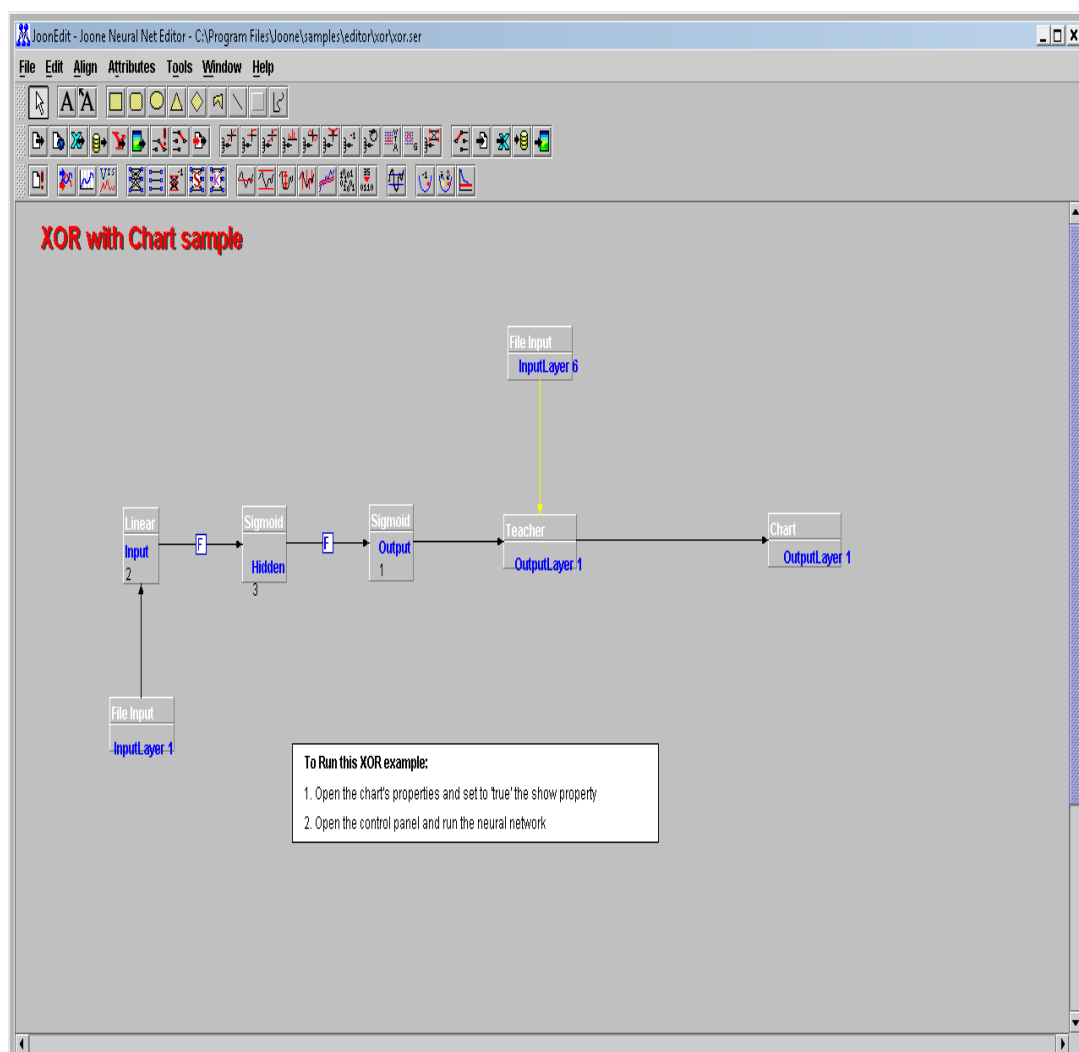
Herramientas estudiadas:

JOONE

Joone (Java Oriented Object Neural Engine)²⁶, es un framework de licencia libre, que junto a Encog y Neuroph más utilizados para la construcción de arquitecturas neuronales artificiales, está desarrollado en Java.

Permite la construcción de arquitecturas tanto visual como a través de código con palabras reservadas propia del aplicativo. Su fuerte está basado en el desarrollo de arquitecturas con aprendizaje backpropagation, aunque se permite la construcción de una amplia gama de sistemas adaptivos.

²⁶Joone versión 2.0.0RC1 desarrollado por Paolo Marrone, <http://sourceforge.net/projects/joone/>, descarga, página original www.joone.org

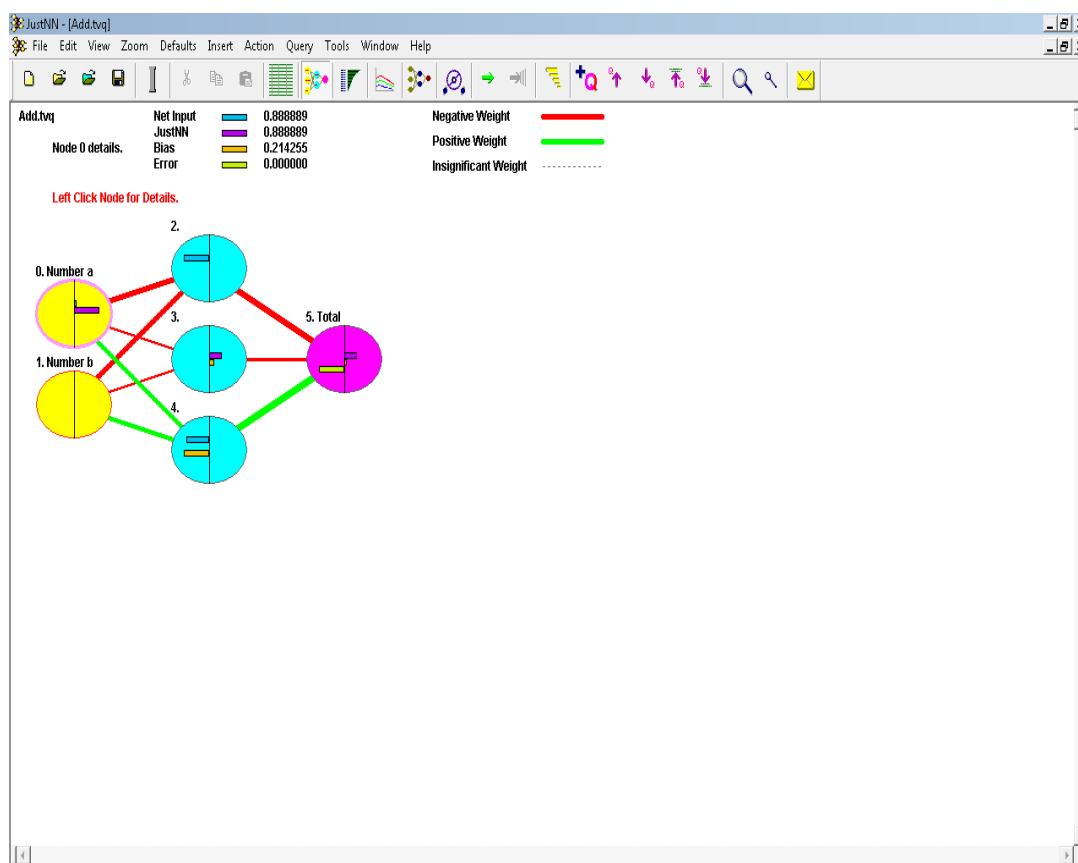


Interfaz de aplicación Joone

JustNN

Free Neural Network Software²⁷, es un sistema de creación de redes neuronales para plataformas Microsoft gratuito, ofrece la posibilidad de creación de arquitecturas utilizando algoritmo de backpropagation, y diseño multicapa.

Existen versiones de pagos tales como EasyNN-plus y SwingNN²⁸



²⁷ JustNN desarrollado por Neural Planner Software Ltd, última versión estable 17 Marzo 2010. <http://www.justnn.com/>

²⁸ JustNN desarrollado por Neural Planner Software Ltd, última versión estable 11 Junio 2010. <http://www.swingnn.com/> <http://www.easynn.com/> última versión estable 3 Agosto 2010

Interfaz de JustNN

Otras Herramientas existentes en el mercado:

Stock Neuro Master²⁹, es otra de las aplicaciones basada en tecnología RNA, que analiza el comportamiento del mercado de valores para la toma de decisiones económicas.

²⁹ <http://www.stockneuromaster.com/> versión 2.0 estable 3 agosto 2010

GLOSARIO DE TÉRMINOS

Aprendizaje Automático.- Es la capacidad que tiene la red de reestructurar sus conexiones neuronales mediante el cambio de los peso, para la minimización de errores en la etapa de aprendizaje.

Aprendizaje Supervisado.- Es aquel que se efectúa sobre un modelo de red neuronal y conoce las respuestas de salida para su comparación y minimización de errores.

Aprendizaje No Supervisado.- Se utiliza las memoria asociativa para el aprendizaje.

Axón.- Es el camino o propagación neuronal por la cual viaja de forma unidireccional el impulso nervioso de una neurona a otra. Su longitud puede alcanzar hasta 1 metro. Los axones están cubiertos de mielina.

Botón dendrítico: Son terminales de las neuronas y son encargados de recibir los estímulos en forma de impulso nervioso.

Células gliales: Son células que brindan soporte a las neuronas, controlando la composición iónica, los niveles de neurotransmisores.

Dendrita.- Son ramas o terminales cortas de una neurona que funcionan como receptores de los impulsos nervioso. Es el lugar donde se reciben los mensajes químicos de otras neuronas.

Excitabilidad: Capacidad de la neurona para producir un cambio en el proceso de polarización.

Espacio Sináptico.- Es el espacio intermedio donde las neuronas se comunican entre sí.

Grieta sináptica: Espacio entre dos neuronas.

Ion: Es una partícula cargada positiva o negativamente a pérdida o ganancia de electrones respectivamente.

Impulsos Nerviosos.- Onda eléctrica, impulso eléctrico o potencial de acción que recorre toda la neurona, que es consecuencia del cambio transitorio de permeabilidad de la membrana plasmática.

Membrana Plasmática.- Es la lamina que engloba a la neurona. Están polarizadas a causa de la diferencia de potencial que se produce en el interno y externo de la misma. (Ion).

Memoria Asociativa.- Es el tipo de memoria que es capaz de relacionar conceptos almacenados y recuperarlos para dar una respuesta en base a sus asociaciones.

Mielina.- Están situadas alrededor de los axones en forma de cubierta y su función principal es la de permitir el paso del impulso nervioso al ser el aislante electroquímico. Mientras más mielinizado esté el axón más rápido se transmitirán los impulsos.

Multicapa: Se la denomina a aquellas redes neuronales compuesta por varias capas de neurona, que conjuntamente trabajan para la el procesamiento y distribución de información para un evento específico.

Nodos de Ranvier.- Se denomina al espacio micrométrico de longitud que la mielina no envuelve al axón. Su función es la de permitir que el impulso nervioso se traslade con mayor velocidad, de manera de saltos de nodo a nodo.

Neurotransmisor.- Viajan de una neurona a otra. Tipo de proteínas que emite la neurona emisora (presináptica) hacia el espacio sináptico y que terminaran de excitar o inhibir el potencial de acción de la neurona receptora (postsináptica).

Presinaptica: Conocido como el lugar antes del proceso de sinapsis.

PostSinaptica: Conocido como el lugar después del proceso de sinapsis.

Polarizar y despolarizar: Aumentar o disminuir el grado iónico de una neurona.

Permeabilidad: Facilidad de penetración molecular.

Pesos Sinápticos.- Es el valor representativo asociado a cada enlace neurona- neurona.

Sinapsis.- Entre la terminación del axón de la neurona emisora y la dendrita de la célula receptora se produce un proceso intercelular denominado sinapsis, donde los impulsos nerviosos son pasados al espacio sináptico a través de neurotransmisores y estos a su vez captados por los receptores de neurotransmisor en la neurona postsináptica.

Soma o Cuerpo Celular.- Es el que contiene el núcleo de la neurona, es el que procesa toda la información que las dendritas receptan de otras células. Contiene material genético en forma de cromosomas.

Sodio NA.- Es uno de los elementos químicos que están presente dentro del proceso neuronal dentro o fuera de la membrana plasmática.

Ribosomas: Son moléculas encargadas de sintetizar proteínas a partir del ADN transcrita en forma de ARN mensajero.

Transmisión Excitadora.- Incrementa la posibilidad de producir un potencial de acción.

Transmisión Inhibidora.- Disminuye la posibilidad de producir un potencial de acción.

Transmisión Moduladora.- Cambia el patrón y/o la frecuencia de la actividad producida por las células involucradas.

Capa oculta: Es llamada a las capas que no están conectadas directamente a la recolección directa de datos o a la salida de la respuesta, sino que son las que colaboran para el procesamiento de información.

CAPITULO IV

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

Al finalizar la elaboración de esta investigación podemos concluir determinando la importancia del estudio neuronal biológico basado en sistemas neuronales agrupados para formar un gran esquema inteligente y adaptivo en los problemas de decisiones. Se pudo comprobar que la elaboración de pensamiento y la inteligencia radican en las conexiones que las neuronas tienen entre sí, tomando como evidencia la representación de ese esquema de red neuronal en un ordenador mediante un prototipo de aprendizaje que buscaba encontrar el modelo matemático que representaba un patrón de datos preliminar.

Después de entender la capacidad de procesamiento que pueden generar grupos de redes neuronales en nuestro cerebro y las acciones que gobiernan al hombre, podemos sacar conclusiones al haber tenido la capacidad de plasmar uno de los algoritmos llamado backpropagation que representaban una arquitectura neuronal simple, pero que podría llegar a convertirse en una de las muchas redes neuronales que enlazaríamos en un futuro. Consiguientemente el proceso de entrenamiento y aprendizaje permitirían el enlace distribuido para poder utilizar funciones abstractas del mundo real.

Dentro de la elaboración del prototipo es importante determinar, que el recurso de hardware es muy sustancial a la hora de obtener el rendimiento del proceso de aprendizaje y entrenamiento, al realizar las pruebas en distintas plataformas de hardware se llegó a la conclusión que la óptima selección de estos recursos mejoran el rendimiento en cuanto a tiempo de respuestas. Se recomienda las distintas prácticas en procesadores de 2 o más núcleos, y aprovechar su procesamiento en paralelo.

El presente trabajo pudo representar a una red neuronal formadas por pequeñas y simples neuronas que interconectadas entre sí lograron generar procesos para la solución de un problema específico, aunque en el avance del estudio se hicieron pruebas con funciones no lineales, como la función XOR, el grado de escalabilidad no se vio afectado, es decir la posibilidad de ingresar una serie de datos preliminares las cuales el prototipo levantará para el análisis respectivo.

Uno de los campos posibles que el presente prototipo puede adaptar es el que tiene que ver con las predicciones según una base de datos preliminares, y es que el estudio se basó en representar el pensamiento humano utilizando como punto de vista el aprendizaje en base a premisas.

Después de la ejecución del proceso de aprendizaje los resultados esperados fueron óptimos para sus aproximaciones reales teniendo como un margen de error de 1,74 % de los datos clasificados.

Es importante recordar que desde que el hombre aparece en un punto determinado en la historia, la representación que le daba a su entorno, venia de apreciaciones comunes que compartía con su semejante, a esto podemos llamar experiencia que informáticamente lo vemos representado en una base de datos. Esos datos utilizados correctamente y ordenadamente pueden llegar a generar información que no solo es usada el presente tiempo, sino que también refleja hacia donde se quiere llegar con datos futuros.

Al tratar de tomar un problema actual, donde debíamos buscar un modelo que explique los cálculos preliminares y que determine los datos futuros pudimos ver que se puede adaptar estos conceptos a los problemas generales, donde la mayoría se comportan como soluciones no lineales.

El presente estudio es el comienzo de una manera propia de poder representar y solucionar problemas del entorno, el avance de los estudios en este campo será una nueva forma de programación basada en la auto representación de esquemas o modelos matemáticos.

Después de crear un núcleo basado en la arquitectura biológica es posible diseñar un modelo complejo basado en la unión de varias redes neuronales, es decir, establecer cual o cuales de ellas realizarían los procesos que hagan depender a las demás del funcionamiento inteligente, todas interactuando para resolver de manera aleatoria procesos separables comúnmente.

Trabajar sobre modelos inteligentes son el comienzo de un futuro prometedor donde la representación de la información será más que una simple organización estructural y se convertirá en una gran distribución de datos sincronizados e interactivos para generación de pensamiento.

RECOMENDACIONES

Es importante señalar que el estudio de este algoritmo y sus usos dentro de la resolución de problemas donde interviene distintos puntos de decisiones enmarca un grado de minimización de procesos y recursos humanos para la resolución de los mismos. La utilización de un modelo neuronal artificial para distintas soluciones del mundo abstracto permite maximizar el grado de eficiencia frente a la intervención humana en la toma de decisiones, que siempre está situado con un fuerte grado de incertidumbre y con las posibilidades de fallo debido a muchas circunstancias, principalmente como cansancio y estado de ánimo.

Se recomienda ver cada problema real con una nueva manera de solucionarlo tal como lo hace el cerebro humano, modularizando un gran problema y definiendo unidades elementales participativas que se encarguen de las soluciones parciales y a su vez validando su participación de error en el sistema general.

Incentivar el uso de nuevos algoritmos para el desarrollo de aplicaciones inteligente, reutilizar los conceptos aquí planteados y revisados para rediseñar modelos adaptivos a

procesos complejos, motivar el espíritu de la investigación neuroartificial y promover los desarrollos de principios de agrupación y predicciones basados en premisas preliminares.

Por último hasta el día de hoy la ciencia trata de encontrar un modelo que ajuste los eventos, el diseño y el tiempo que engloba al universo entero. Una teoría que pueda explicar los fenómenos que ocurren y ocurrirán en un tiempo determinado, sería un gran paso al desarrollo científico, el presente trabajo expande mi manera de pensar y podría hipotéticamente pensar que se podría encontrar el modelo que rigen dichos eventos, diseño y tiempo encontrando las variables principales para su análisis.

BIBLIOGRAFIAS

- [1] W.S. MacCulloch and W. Pitts. A logical Calculus of ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 5: 115-133, 1943
- [2] M. Minsky and S. Papert. Perceptrons. Tecnical report, Cambridge: Mit Press, 1969
- [3] Russell, S. y Norvig, P, Inteligencia Artificial un enfoque moderno, segunda edición, Prentice Hall 2004
- [4] Ramón Hilera José, Redes Neuronales Artificiales. Fundamentos, Modelo y Aplicaciones
- [5] Dr. C. George Boeree, Departamento de Psicología, Universidad de Shippensburg, Psicología General, disponible en <http://www.psicologia-online.com/ebooks/general/neuronas.htm>
- [6] Dr. Diego Andina de la Fuente, Antonio Vega Corona, Departamento de Señales, Sistemas y Radiocomunicaciones, Universidad Politécnica de

Madrid, disponible en

<http://www.gc.ssr.upm.es/inves/neural/ann1/anntutorial.html>

[7] Ing. Alfonso Vallesteros, Universidad de Malaga, Tutorial introducción a las Redes Neuronales, disponible en <http://www.redes-neuronales.netfirms.com/tutorial-redes-neuronales/tutorial-redes.htm>

[8] Xavier Padern, Introduccion a las redes neuronales artificiales, www.redcientifica.com, disponible en <http://www.redcientifica.com/doc/doc199903310003.html>

[9] Alberto Aragon, Silvia Casado y Joaquin Pacheco, Optimización de simulador de flujo de pasajero con redes neuronales, Dpto Economía Aplicada. Universida Burgos. España

[10] Paolo Massino Mussema, Rome IT, Artificial Neural Network Publication Number US 2007/0043452 A1, 2004

ANEXOS I

PRUEBAS REALIZADAS

# Prueba	Entradas	Ocultas	Salidas	Factor Aprendizaje	Factor Momento	Error por Patrón	Error por Época	Ciclos	Función Activación	Correctamente	Incorrectamente	Total	% Error	Tiempo Ejecución	Épocas	Observación
1	7	4	1	0,80	0,70	1,00E-05	1,00E-05	10000	2	974	120	1094	10,97	0:03:12	10000	Prueba AMD 1,6, 1 GB ram
1	7	4	1	0,25	0,80	1,00E-05	1,00E-05	10000	2	985	109	1094	9,96	0:03:20	10000	Prueba AMD 1,6, 1 GB ram
1	7	4	1	0,25	0,80	1,00E-05	1,00E-05	20000	2	978	116	1094	10,60	0:06:47	20000	Prueba AMD 1,6, 1 GB ram
1	7	4	1	0,30	0,70	1,00E-05	1,00E-05	10000	2	979	115	1094	10,51	0:03:14	10000	Prueba AMD 1,6, 1 GB ram
1	7	4	1	0,50	0,80	1,00E-05	1,00E-05	10000	2	970	124	1094	11,33	0:03:03	10000	Prueba AMD 1,6, 1 GB ram
1	7	4	1	0,40	0,70	1,00E-05	1,00E-05	10000	2	993	101	1094	9,23	0:03:19	10000	Prueba AMD 1,6, 1 GB ram
1	7	4	1	0,40	0,70	1,00E-05	1,00E-05	30000	2	990	104	1094	9,51	0:10:30	10000	Prueba AMD 1,6, 1 GB ram
2	7	5	1	0,40	0,70	1,00E-05	1,00E-05	10000	2	989	105	1094	9,60	0:04:01	10000	Prueba AMD 1,6, 1 GB ram
5	7	6	1	0,80	0,70	1,00E-04	1,00E-05	10000	2	1039	55	1094	5,03	0:03:55	10000	Prueba AMD 1,6, 1 GB ram
5	7	6	1	0,25	0,70	1,00E-04	1,00E-05	10000	2	1063	31	1094	2,83	0:03:43	10000	Prueba AMD 1,6, 1 GB ram
5	7	6	1	0,10	0,70	1,00E-04	1,00E-05	10000	2	1048	46	1094	4,20	0:03:49	10000	Prueba AMD 1,6, 1 GB ram
5	7	6	1	0,30	0,70	1,00E-04	1,00E-05	10000	2	1034	60	1094	5,48	0:03:34	10000	Prueba AMD 1,6, 1 GB ram
5	7	6	1	0,25	0,80	1,00E-04	1,00E-05	10000	2	1062	32	1094	2,93	0:03:30	10000	Prueba AMD 1,6, 1 GB ram
5	7	6	1	0,25	0,90	1,00E-04	1,00E-05	10000	2	1064	30	1094	2,74	0:03:29	10000	Prueba AMD 1,6, 1 GB ram
5	7	6	1	0,25	0,90	1,00E-04	1,00E-05	15000	2	1060	34	1094	3,11	0:05:19	10000	Prueba AMD 1,6, 1 GB ram
5	7	6	1	0,20	0,95	1,00E-04	1,00E-05	10000	2	1021	73	1094	6,67	0:03:32	10000	Prueba AMD 1,6, 1 GB ram
5	7	6	1	0,25	0,90	1,00E-04	1,00E-05	10000	2	1049	45	1094	4,11	0:03:29	10000	Prueba AMD 1,6, 1 GB ram
5	7	6	1	0,25	0,75	1,00E-04	1,00E-05	10000	2	1067	27	1094	2,47	0:03:37	10000	Prueba AMD 1,6, 1 GB ram
7	7	7	1	0,25	0,75	1,00E-04	1,00E-05	10000	2	979	115	1094	10,51	0:05:30	10000	Prueba AMD 1,6, 1 GB ram
7	7	7	1	0,40	0,80	1,00E-04	1,00E-05	10000	2	990	104	1094	9,51	0:05:38	10000	Prueba AMD 1,6, 1 GB ram

12	7	6	1	0,25	0,75	1,00E-04	1,00E-05	10000	2	1067	27	1094	2,47	0:03:38	10000	Prueba AMD 1,6, 1 GB ram
18	7	6	1	0,25	0,80	1,00E-04	1,00E-05	10000	2	1054	40	1094	3,66	0:03:30	10000	Prueba AMD 1,6, 1 GB ram
19	7	6	1	0,80	0,70	1,00E-04	1,00E-05	10000	2	1010	84	1094	7,68	0:03:36	10000	Prueba AMD 1,6, 1 GB ram
19	7	6	1	0,25	0,80	1,00E-04	1,00E-05	10000	2	1046	48	1094	4,39	0:03:29	10000	Prueba AMD 1,6, 1 GB ram
21	7	5	1	0,25	0,80	1,00E-04	1,00E-05	10000	2	1065	29	1094	2,65	0:02:54	10000	Prueba AMD 1,6, 1 GB ram
21	7	5	1	0,20	0,85	1,00E-04	1,00E-05	10000	2	1061	33	1094	3,02	0:02:58	10000	Prueba AMD 1,6, 1 GB ram
22	7	6	1	0,25	0,80	1,00E-04	1,00E-05	10000	2	1060	34	1094	3,11	0:03:30	10000	Prueba AMD 1,6, 1 GB ram
22	7	6	1	0,25	0,75	1,00E-04	1,00E-05	10000	2	1060	34	1094	3,11	0:03:27	10000	Prueba AMD 1,6, 1 GB ram
23	7	7	1	0,25	0,75	1,00E-04	1,00E-05	10000	2	1063	31	1094	2,83	0:03:59	10000	Prueba AMD 1,6, 1 GB ram
23	7	7	1	0,10	0,80	1,00E-04	1,00E-05	10000	2	1020	74	1094	6,76	0:04:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,25	0,75	1,00E-04	1,00E-05	10000	2	1072	22	1094	2,01	0:02:04	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,20	0,80	1,00E-04	1,00E-05	10000	2	1068	26	1094	2,38	0:02:08	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,70	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1075	19	1094	1,74	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,40	0,60	1,00E-04	1,00E-05	10000	2	1053	41	1094	3,75	0:02:03	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,35	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05	10000	2	1050	44	1094	4,02	0:02:01	10000	Prueba AMD 1,6, 1 GB ram
24	7	3	1	0,30	0,75	1,00E-04	1,00E-05									

ANEXOS II

INTERFAZ GRAFICA

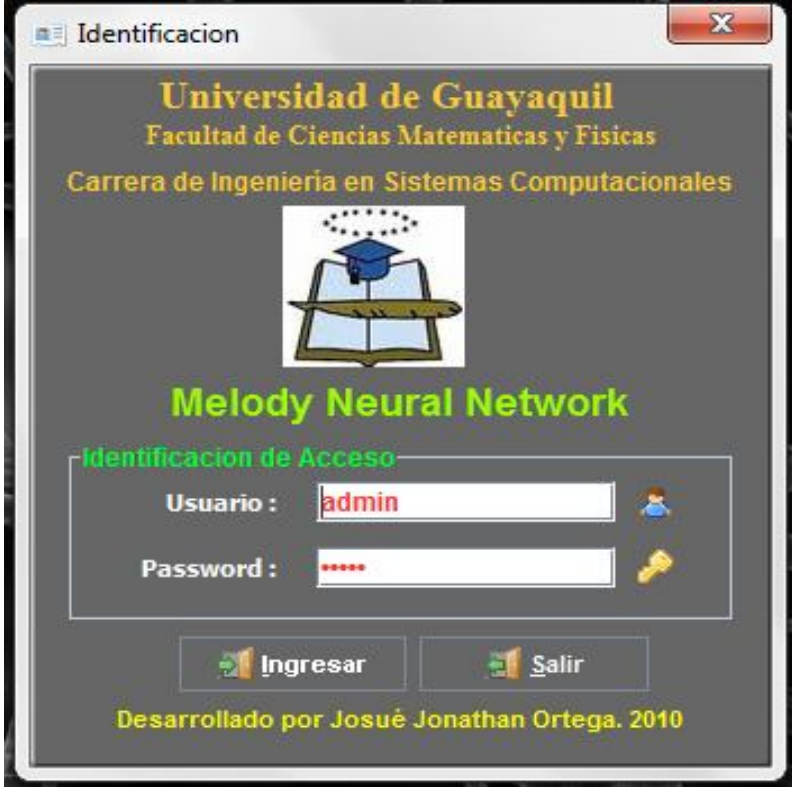
GRAFICA DE ERROR DE APRENDIZAJE



INTERFAZ DE USUARIO

PANTALLAS DEL PROTOTIPO DE DESARROLLO

Pantalla de login: Pantalla de Acceso al prototipo.



The screenshot shows a login window titled "Identificacion". The window has a dark gray background with yellow and green text. At the top, it reads "Universidad de Guayaquil", "Facultad de Ciencias Matematicas y Fisicas", and "Carrera de Ingenieria en Sistemas Computacionales". Below this is a logo of a graduation cap on an open book. The title "Melody Neural Network" is displayed in green. A section titled "Identificacion de Acceso" contains two input fields: "Usuario :" with the text "admin" and a user icon, and "Password :" with red dots and a key icon. At the bottom, there are two buttons: "Ingresar" and "Salir". The footer text reads "Desarrollado por Josué Jonathan Ortega. 2010".

Identificacion

Universidad de Guayaquil
Facultad de Ciencias Matematicas y Fisicas
Carrera de Ingenieria en Sistemas Computacionales

Melody Neural Network

Identificacion de Acceso

Usuario : admin

Password :

Ingresar Salir

Desarrollado por Josué Jonathan Ortega. 2010

Pantalla de creación de nuevo proyecto: Permite crear una nueva arquitectura neuronal con su respectiva configuración inicial.

Proyecto existente # 24

Arquitectura

Tipo de Red:

- ☐ Perceptron Simple
- ☐ Perceptron Multicapa
- ☒ Red FeedForward

Nombre del Proyecto:

Neuronas:

Entrada: # Ocultas: # Salida:

Selección de Datos

cartera_clientes
cartera_clientes_clases
funcion_and
funcion_or
funcion_xor

co...	no...	ciu...	li...	sa...	de...	val...	val...	val...	val...	val...	val...	val...	val...	val...	val...	val...	val...	es...
0...	...	84	1...	0...	0...	0...	0...	0...	0...	0...	0...	0...	0...	0...	0...	0...	0...	0
0...	V...	2	5...	0...	0...	1...	0...	52...	1...	15...	5...	0...	2...	10...	2...	10...	0...	22...
0...	C...	84	1...	0...	0...	3...	0...	0...	0...	37...	0...	0...	0...	37...	0...	0...	0...	0
0...	Q...	2	5...	0...	0...	1...	0...	90...	8...	18...	7...	0...	8...	99...	1...	7...	0...	2...

Dinámica

Funciones Activación:

$$s(z) = \frac{1}{1 + e^{-z}}$$

Pesos Aleatorios entre: entre

Valores de entrenamiento

Factor de Aprendizaje:

Momento:

Valor Umbral:

Ciclos / Repetición:

























% MSE Total:

% MSE por patrón:

Descripción:

Control:

Pantalla de Abrir Proyecto: Permite traer un proyecto existente ya creado con su respectiva arquitectura y parámetros de entrenamiento

Abrir Proyecto						
Id	Nombre	Descripción	Creado	Modificado	Usuario	Ver
0	Función XOR		2010-10-08	2010-10-08	JOSUE ORTEGA	
1	Limite de Credito		2010-10-08	2010-10-08	JOSUE ORTEGA	
2	Cartera Cliente	CHE CHF NCR NDB CHP EFE	2010-10-08	2010-10-08	JOSUE ORTEGA	
3	Funcion AND		2010-10-12	2010-10-12	JOSUE ORTEGA	
4	Funcion OR		2010-10-12	2010-10-12	JOSUE ORTEGA	
5	Prueba Limite Credito		2010-10-12	2010-10-12	JOSUE ORTEGA	
6	Funcion And con Tangente Hiperbólica		2010-10-19	2010-10-19	JOSUE ORTEGA	
7	Prueba		2010-10-21	2010-10-21	JOSUE ORTEGA	
8	Prueba 1		2010-10-23	2010-10-23	JOSUE ORTEGA	
9	Prueba 2		2010-10-23	2010-10-23	JOSUE ORTEGA	
10	Prueba 3		2010-10-23	2010-10-23	JOSUE ORTEGA	
11	Prueba 4		2010-10-23	2010-10-23	JOSUE ORTEGA	
12	Establecimiento de Limite de Credito	Appe=0.25; Mom=0.75; in=7 hi=6 out=1 Error:2,4...	2010-10-23	2010-10-23	JOSUE ORTEGA	
13	Prueba 6		2010-10-23	2010-10-23	JOSUE ORTEGA	
14	Prueba 7		2010-10-24	2010-10-24	JOSUE ORTEGA	
15	Prueba 8		2010-10-27	2010-10-27	JOSUE ORTEGA	
16	Prueba 5		2010-11-21	2010-11-21	JOSUE ORTEGA	
17	Prueba # 17		2011-02-12	2011-02-12	JOSUE ORTEGA	
18	Prueba # 18		2011-02-12	2011-02-12	JOSUE ORTEGA	
19	Prueba # 19		2011-02-12	2011-02-12	JOSUE ORTEGA	
20	Prueba # 20		2011-02-12	2011-02-12	JOSUE ORTEGA	
21	Prueba # 21		2011-02-12	2011-02-12	JOSUE ORTEGA	
22	Prueba # 22		2011-02-12	2011-02-12	JOSUE ORTEGA	
23	Prueba # 23		2011-02-12	2011-02-12	JOSUE ORTEGA	

Pantalla de Selección de patrones de datos: Permite seleccionar los campos de la tabla seleccionada para el aprendizaje y determinar mediante las palabras reservadas IN= Entrada OUT=salida; para cada uno de los campos que pertenecerán a los patrones. El número de entrada y salida debe ser el mismo que el que indica la arquitectura creada.

Tabla de Patrones de Datos



id	x	y	out
1	1	1	0
2	1	0	1
3	0	1	1
4	0	0	0

Entradas (IN) Salidas(OUT) ?

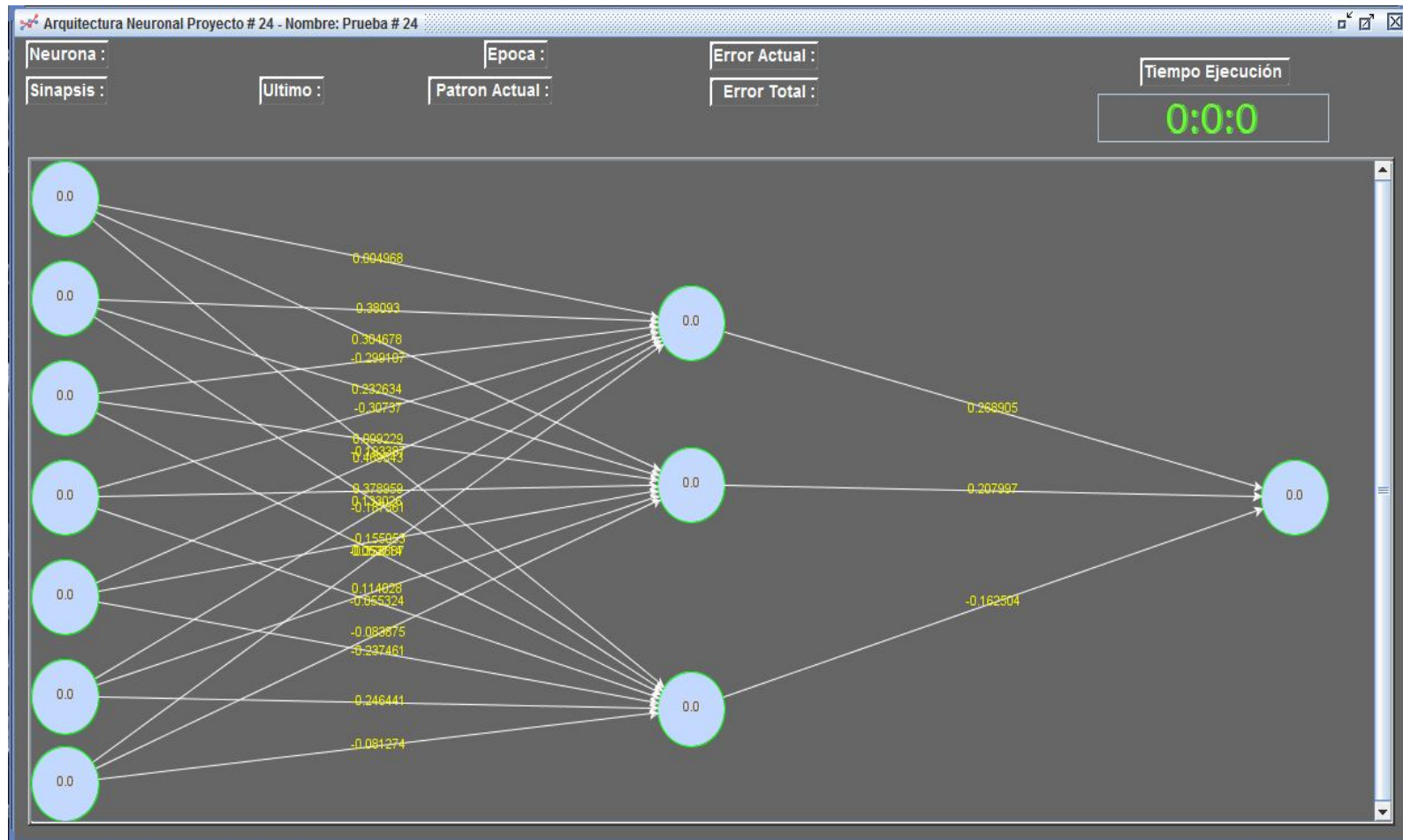
Columna	Tipo Datos	Tipo Columna
id	int	
x	int	IN
y	int	IN
out	int	OUT

Neuronas Entradas : 2
Neuronas Salidas : 1
Total Patrones :

Control

 Guardar  Salir

Pantalla de Dibujo Neuronal: Visualiza la red neuronal distribuida en capa con sus respectivas neuronas y sinapsis representadas por su valor.



Pantalla de Gráfica de Curvas de Error: Muestra el seguimiento del error por medio de la curva de error descendiente.



Pantalla de Resultados: Permite dar seguimiento de los resultados de la red, antes y después de su ejecución, dando la posibilidad del ingreso de un nuevo patrón para su evaluación.

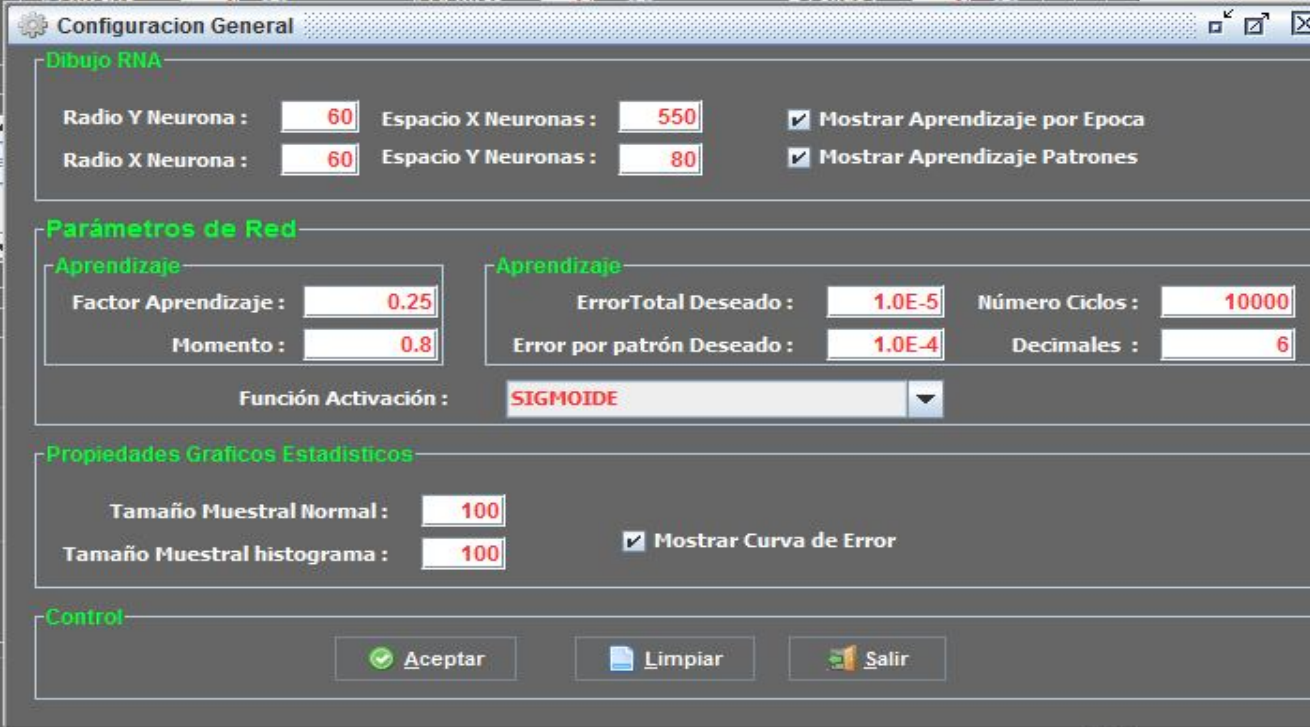
Resultados											
Referencia	Entrada 1	Entrada 2	Entrada 3	Entrada 4	Entrada 5	Entrada 6	Entrada 7	Salida Esperada1	Salida de Red	Valor Normalizado	Clase Definida
Patron # 0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1000.0	12152	0	10000
Patron # 1	1530.48	5258.8	10731.29	5472.49	0.0	2880.82	10650.42	50000.0	21500	0	10000
Patron # 2	378.36	0.0	0.0	0.0	0.0	0.0	378.36	1000.0	12224	0	10000
Patron # 3	1894.18	902.39	8097.04	7194.64	0.0	832.73	9995.82	50000.0	18858	0	10000
Patron # 4	1619.4	4775.33	5654.56	875.78	0.0	1374.23	8139.73	50000.0	16356	0	10000
Patron # 5	0.0	641.0	2765.84	1973.74	0.0	2765.84	0.0	10000.0	13486	0	10000
Patron # 6	0.0	500.0	1471.46	971.46	0.0	2991.3	0.0	10000.0	13029	0	10000
Patron # 7	1023.15	11444.05	11636.45	190.93	0.0	2782.32	11644.16	50000.0	21900	0	10000
Patron # 8	0.0	0.0	2234.0	2234.0	0.0	0.0	2234.12	10000.0	13306	0	10000
Patron # 9	541.4	0.0	941.1	941.1	0.0	1482.5	0.0	10000.0	12685	0	10000
Patron # 10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1000.0	12152	0	10000
Patron # 11	547.43	0.0	0.0	0.0	0.0	0.0	547.43	10000.0	12257	0	10000
Patron # 12	0.0	0.0	934.27	934.27	0.0	1370.77	0.0	10000.0	12638	0	10000
Patron # 13	476.67	26619.6	33261.58	6506.16	0.0	2968.61	33883.06	150000.0	79532	0	50000
Patron # 14	0.0	1264.62	1264.62	0.0	0.0	1276.01	0.0	10000.0	12686	0	10000
Patron # 15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1000.0	12152	0	10000
Patron # 16	331.05	581.32	5874.29	5283.43	905.01	2025.54	4018.29	10000.0	15824	0	10000
Patron # 17	473.84	12180.15	12180.15	0.0	0.0	1837.3	11210.85	50000.0	21740	0	10000
Patron # 18	0.0	600.01	2190.94	1590.93	0.0	2290.27	0.0	10000.0	13207	0	10000
Patron # 19	955.71	1091.38	1091.38	0.0	0.0	1632.15	472.43	10000.0	12806	0	10000
Patron # 20	27.5	24693.98	88817.97	64313.59	0.0	61119.29	34465.12	150000.0	173277	1	150000
Patron # 21	3320.44	3204.44	7470.84	4260.7	0.0	1832.78	0358.42	50000.0	18380	0	10000

Resumen															
Números de Neuronas x Capas				Factores Utilizados		% límite Error utilizados		Patrones Procesados				Recursos Generados			
Codigo Prueba	Entradas	Ocultas	Salidas	Aprendizaje	Momento	Error/Patron	Error/Epoca	# Ciclos/Epoca	Func. Activación	Correctamente	Incorrectamente	# Patrones/Datos	% Error	Tiempo	Epocas utiliza...
24	7	3	1	0.25	0.75	1.0E-4	1.0E-5	10000	2	529	565	1094	51.65	00:01:29	2

Observaciones :

Guardar Resultado	Nueva Consulta	Verificar	Calcular	Salir
-------------------	----------------	-----------	----------	-------

Pantalla de configuración general: Permite determinar una nueva configuración para el modelo



Configuración General

Dibujo RNA

Radio Y Neurona : 60 Espacio X Neuronas : 550 ☒ Mostrar Aprendizaje por Epoca
Radio X Neurona : 60 Espacio Y Neuronas : 80 ☒ Mostrar Aprendizaje Patrones

Parámetros de Red

Aprendizaje

Factor Aprendizaje : 0.25 ErrorTotal Deseado : 1.0E-5 Número Ciclos : 10000
Momento : 0.8 Error por patrón Deseado : 1.0E-4 Decimales : 6

Función Activación : SIGMOIDE

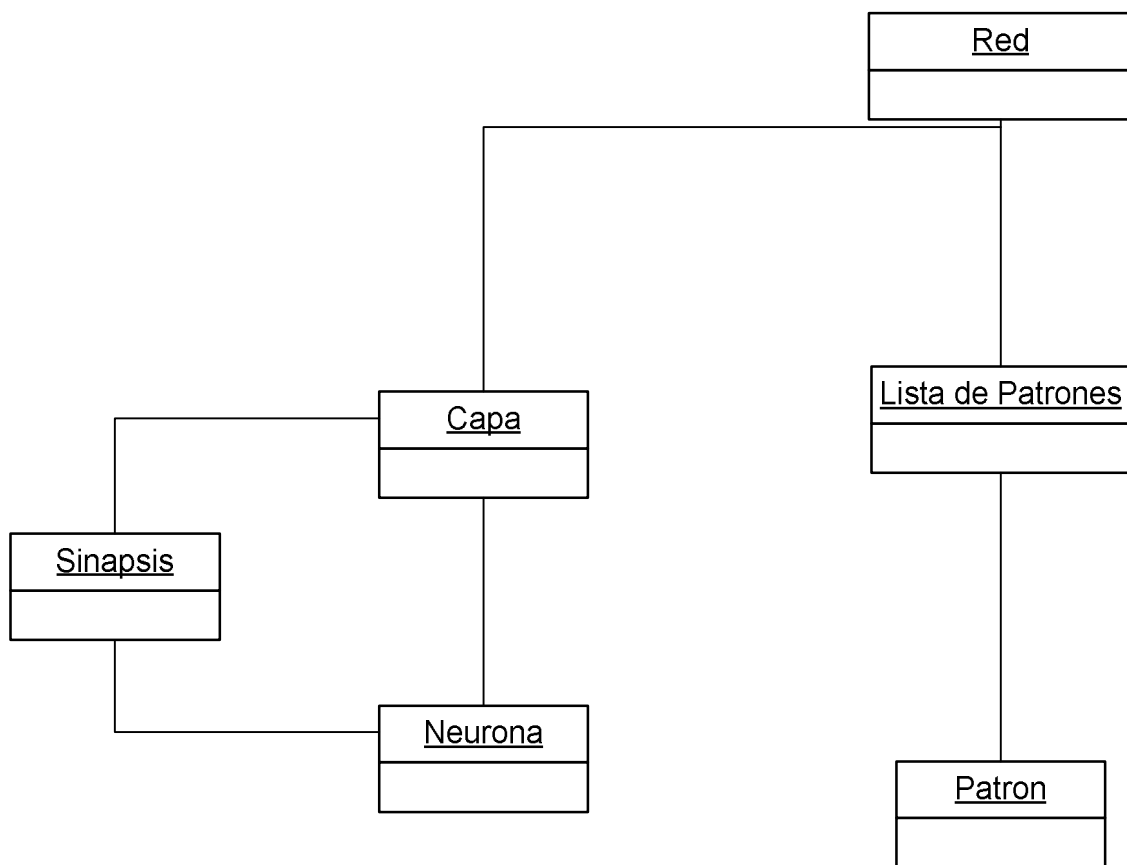
Propiedades Graficos Estadísticos

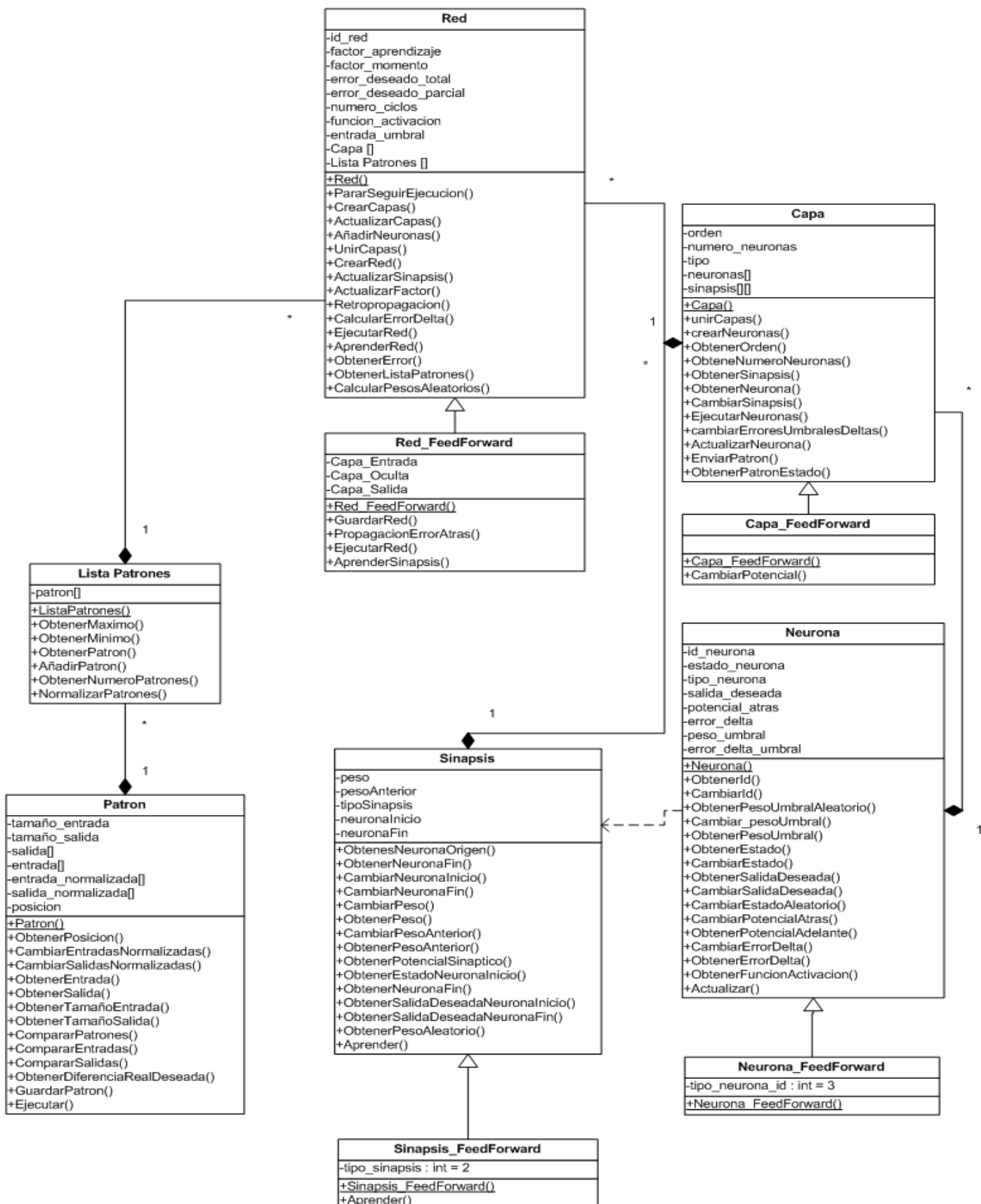
Tamaño Muestral Normal : 100 ☒ Mostrar Curva de Error
Tamaño Muestral histograma : 100

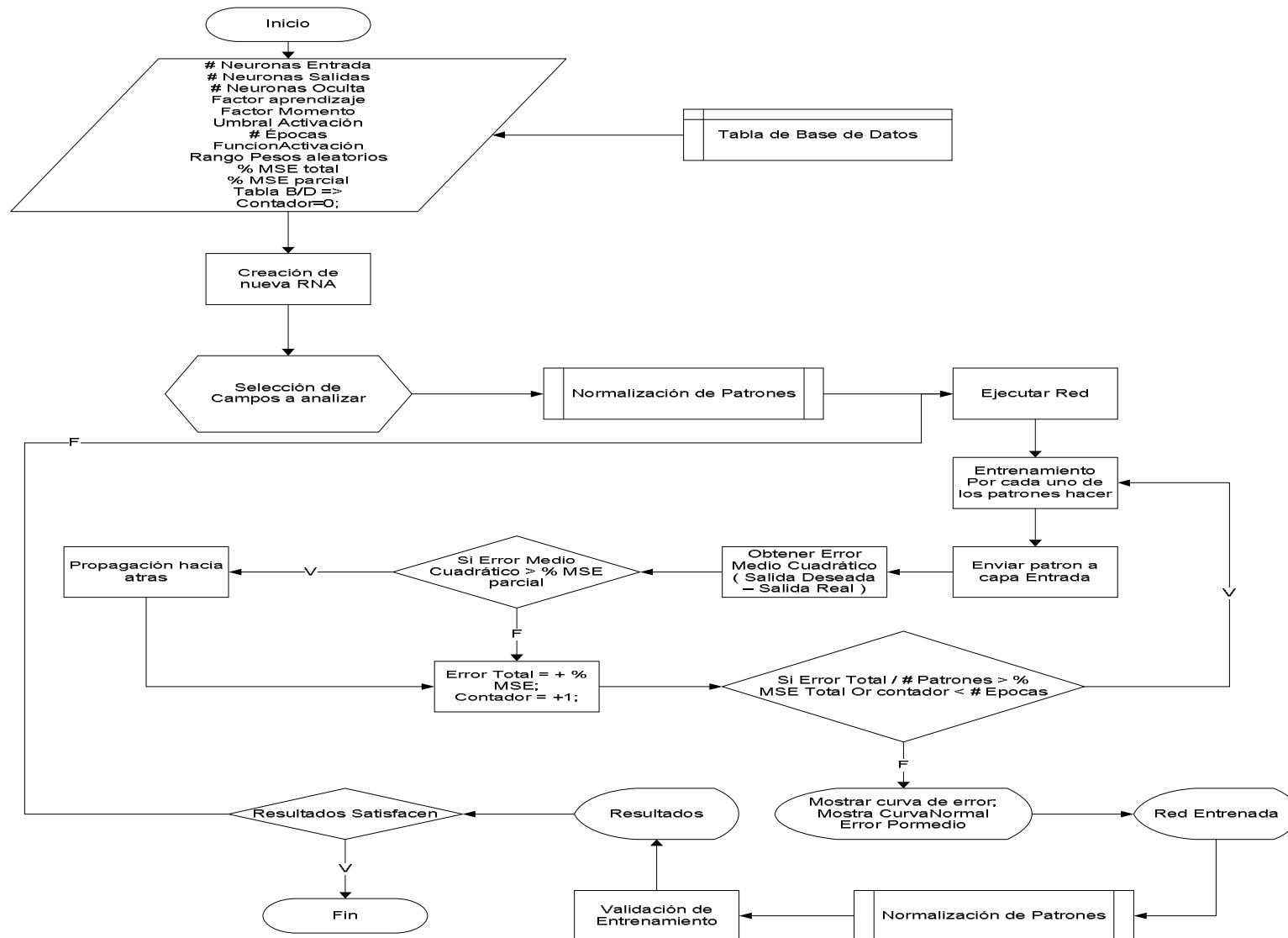
Control

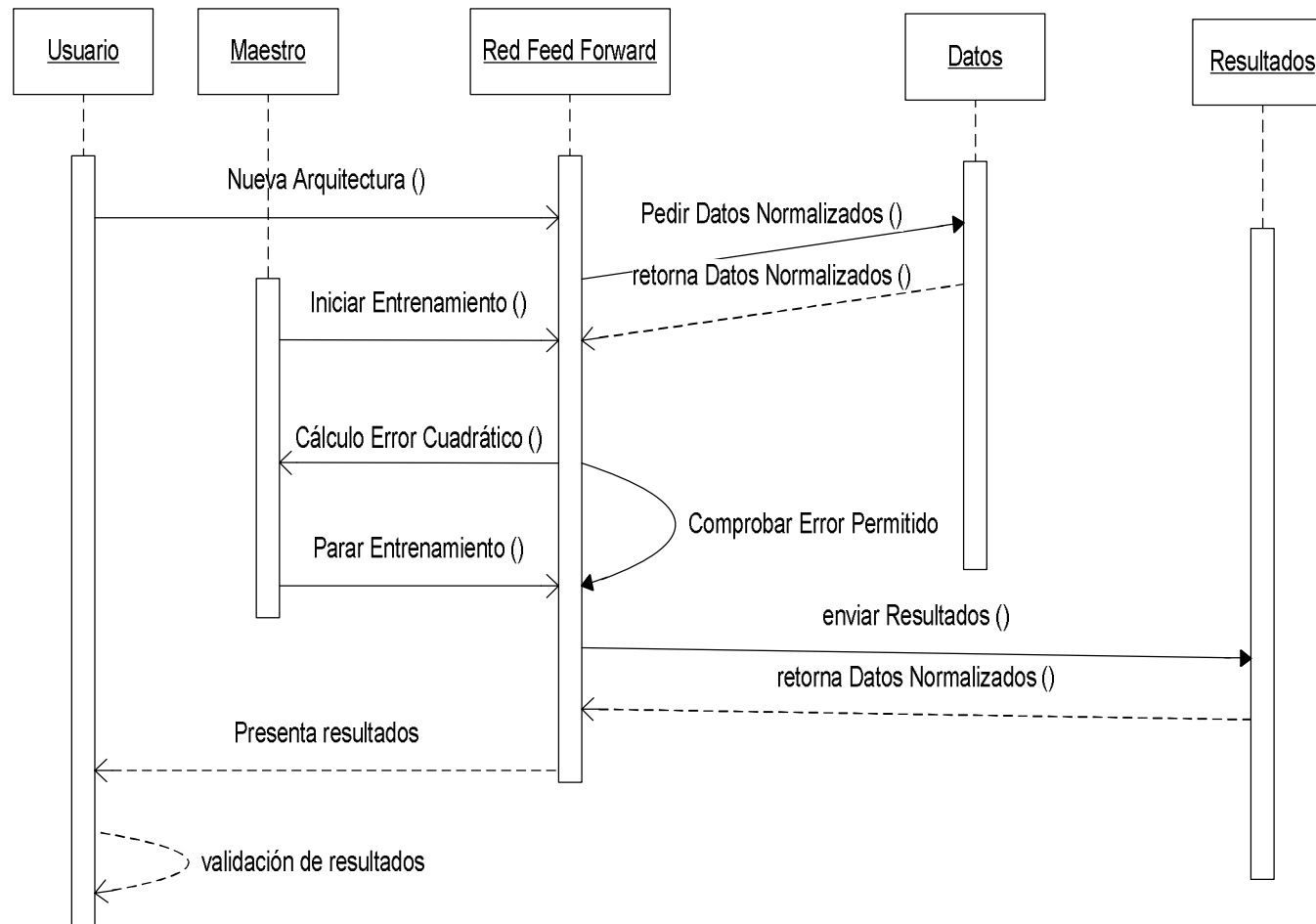
ANEXOS III

MANUAL TECNICO MELODY NEURAL NETWORK

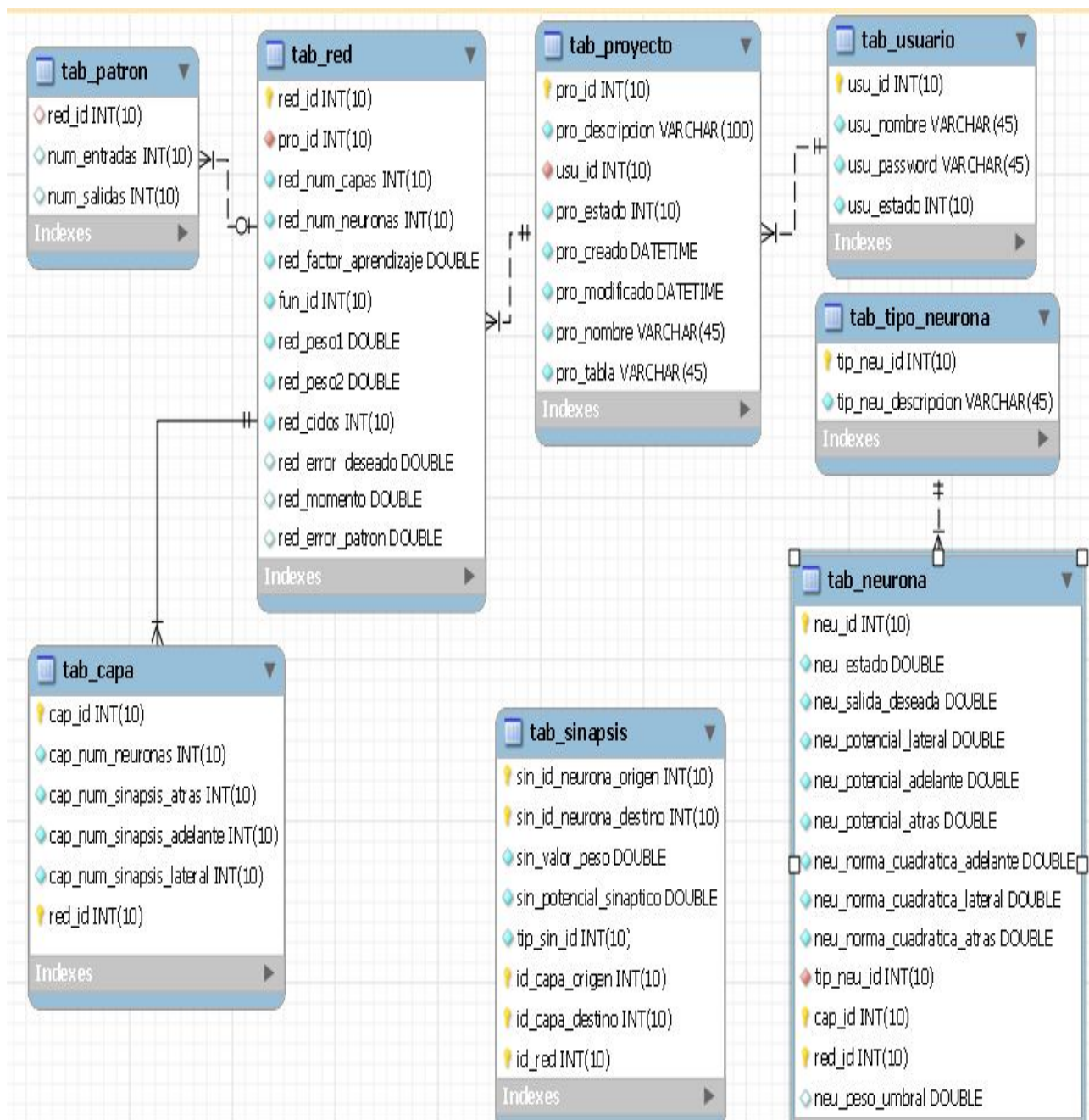
**Modelo de Objetos**





**DIAGRAMA DE SECUENCIA**

MODELO ENTIDAD - RELACION



DICCIONARIO DE DATOS

Tabla: tab_usuario

Descripción: Tabla que contiene los datos de los usuarios permitidos a manipular el prototipo.

Campo	Tipo de Dato	Tamaño	Descripción
usu_id	int	10	Identificador primario secuencial.
usu_nombre	varchar	45	Nombre del usuario.
usu_password	varchar	45	Password del usuario.
usu_estado	int	10	Estado del registro (1=Activo ; 0=Eliminado)

Tabla: tab_proyecto

Descripción: Tabla que contiene todos los datos de cada proyecto.

Campo	Tipo de Dato	Tamaño	Descripción
pro_id	int	10	Identificador primario secuencial
pro_nombre	varchar	45	Nombre del proyecto
pro_descripcion	varchar	100	Descripción del proyecto
usu_id	int	10	Clave referencial a la tabla usuario (usu_id)
pro_creado	int	10	Fecha de creación del proyecto
pro_tabla	varchar	45	Nombre de la tabla de datos para el análisis de aprendizaje.

Tabla: tab_funcion_activacion

Descripción: Tabla que contiene los tipos de funciones de activación.

Campo	Tipo de Dato	Tamaño	Descripción
fun_id	int	10	Identificador primario secuencial
fun_descripcion	varchar	45	Nombre de la función de activación.

Tabla: tab_red

Descripción: Tabla que contiene todos los datos de la red neuronal

Campo	Tipo de Dato	Tamaño	Descripción
red_id	int	10	Identificador primario secuencial
pro_id	int	10	Clave referencial a la tabla tab_proyecto (pro_id)
red_num_capas	int	10	Numero de capas totales que tiene la red
red_num_neuronas	int	10	Numero de neuronas totales que tiene la red
red_factor_aprendizaje	double		Factor usado para la velocidad de aprendizaje de la red
fun_id	int	10	Clave referencial a la tabla tab_funcion_activacion (fun_id)
red_peso1	double		Valor límite inferior usado para el establecimiento de pesos iniciales
red_peso2	double		Valor límite superior usado para el establecimiento de pesos iniciales
red_ciclos	int	10	Numero de ciclos utilizados por la red para el aprendizaje
red_error_deseado	double		Valor de error determinado de la red para terminar su entrenamiento.
red_momento	double		Factor de momento utilizado para la estabilidad del entrenamiento.
red_error_patron	double		Valor de error permitido por cada patrón de la red en el entrenamiento.

Tabla: tab_capa

Descripción: Tabla que contiene todos los datos de cada capa de una red neuronal

Campo	Tipo de Dato	Tamaño	Descripción
cap_id	int	10	Identificador primario. Nivel de capa de la red
red_id	int	10	Identificador Primario. Clave referencial a la tabla tab_red (red_id)
cap_num_neuronas	int	10	Número total de neuronas que contendrá capa de la red.
cap_num_sinapsis_atras	int	10	Número total de enlaces sinápticos hacia atrás de cada capa de la red.
cap_num_sinapsis_adelante	int	10	Numero total de enlaces sinápticos hacia delante de cada capa de la red.
cap_num_sinapsis_lateral	int	10	Número total de enlaces sinápticos laterales de cada capa de la red.

Tabla: tab_neurona

Descripción: Tabla que contiene todos los datos de una neurona de red.

Campo	Tipo de Dato	Tamaño	Descripción
neu_id	int	10	Identificador primario. Nivel de neurona por capa de red.
red_id	int	10	Identificado primario. Clave referencial a la tabla tab_red (red_id)
cap_id	int	10	Identificador primario. Clave referencial a la tabla tab_capa (cap_id)
neu_salida_deseada	double		Valor de la salida deseada que tendrá la neurona actual.
neu_potencial_lateral	double		Valor del potencial sináptico lateral de la neurona actual.
neu_potencial_atras	double		Valor del potencial sináptico atrás de la neurona actual.
neu_norma_cuadratica_a_delante	double		Valor de la norma cuadrática que una neurona tiene hacia adelante, utilizada para la validación del error deseado.
neu_norma_cuadratica_a_tras	double		Valor de la norma cuadrática que una neurona tiene hacia atrás, utilizada para la validación del error deseado.
neu_norma_cuadratica_lateral	double		Valor de la norma cuadrática que una neurona tiene lateralmente, utilizada para la validación del error deseado.
tip_neu_id	int	10	Clave referencial a la tabla tab_tipo_neurona. Indica el tipo de neurona.
neu_peso_umbral	double		El valor umbral que tiene la neurona.

Tabla: tab_tipo_neurona

Descripción: Tabla que contiene el tipo de neurona.

Campo	Tipo de Dato	Tamaño	Descripción
tip_neu_id	int	10	Identificador primario secuencial
tip_neu_descripcion	varchar	45	Nombre del tipo de neurona

Tabla: tab_sinapsis

Descripción: Tabla que contiene todos los datos de la sinapsis de red (Enlaces de 2 neuronas).

Campo	Tipo de Dato	Tamaño	Descripción
id_red	int	10	Identificador primario. Clave referencial a la tabla tab_red (id_red).
tip_sin_id	int	10	Clave referencial a la tabla tab_tipo_sinapsis.
id_capa_origen	int	10	Identificador primario. Indica la capa origen que tiene la sinapsis actual.
id_capa_destino	int	10	Identificador primario. Indica la capa destino que tiene la sinapsis actual.
sin_potencial_sinaptico	double		Valor del potencial sináptico que tiene la sinapsis actual.
sin_valor_peso	double		Valor de peso sináptico que tiene la sinapsis actual
id_neurona_origen	int	10	Identificador primario. Indica el id neurona origen de acuerdo a la capa origen.
id_neurona_destino	int	10	Identificador primario. Indica el id neurona destino de acuerdo a la capa destino.

Tabla: tab_patron

Descripción: Tabla que contiene los patrones que la red utilizará para el entrenamiento

Campo	Tipo de Dato	Tamaño	Descripción
red_id	int	10	Clave referencial a la tabla tab_red (red_id).
num_entradas	int	10	Número de variables de entradas que tendrá la red basado en patrones.
num_salidas	int	10	Número de variables de salidas que tendrá la red basado en patrones.

Tabla: tab_detalle_patron

Descripción: Tabla que contiene los patrones que la red utilizará para el entrenamiento

Campo	Tipo de Dato	Tamaño	Descripción
red_id	int	10	Clave referencial a la tabla tab_red (red_id).
nombre_columna	varchar	50	Nombre de la columna , representa el nombre del campo de la tabla a analizar.
tipo	char	3	Indica el tipo de variable de patrón. IN=Entrada OUT=Salida

Tabla: tab_pruebas

Descripción: Tabla que contiene las pruebas realizadas con el prototipo.

Campo	Tipo de Dato	Tamaño	Descripción
idPrueba	int	11	Clave Primaria. Numero de proyecto o prueba que se evalua.
numEntradas	int	11	Numero de neuronas de entradas que tiene la red evaluada.
numOcultas	int	11	Numero de neuronas de la capa oculta que tiene la red evaluada.
numSalidas	int	11	Numero de neuronas de la capa de salida que tiene la red evaluada.
factorAprendizaje	double		Factor de aprendizaje que tiene la red evaluada.
factorMomento	double		Factor momento que tiene la red evaluada.
errorPorPatron	double		Error por patrón que utilizó la red evaluada.
errorPorEpocas	double		Error por épocas que utilizó la red evaluada.
ciclos	double		Ciclos determinados condicionalmente para la terminación de la prueba
funcionActivacion	double		Numero de función utilizada para el entrenamiento de la red evaluada.
correctos	int		Numero de patrones correctamente clasificados
incorrectos	int		Numero de patrones incorrectamente clasificados.
total	int		Número total de patrones utilizados para la evaluación de la red.
porError	double		Porcentaje del error total de aprendizaje de la evaluación de la red.
tiempoEjecucion	varchar	10	Tiempo de ejecución de la red evaluada.
epocasUtilizadas	int		Numero de épocas o ciclos utilizados en el proceso de entrenamiento.
Observacion	varchar	1000	Observación de la evaluación.

CODIGO FUENTE MELODY NEURAL NETWORK

PAQUETE ARQUITECTURA

CLASE Red

```
package paq_arquitectura;
import paq_logica.cls_variables_globales;
```

/* Es una de las clase principales del proyecto, proporciona metodos para la definición de arquitectura, dinamica y aprendizaje de la red a utilizar
Por tener metodos abstractos no definidos como el:

```
+ constructor
+ ejecutar
+ aprender
```

```
public abstract class Red {
private boolean entrenada=false;
private int id_proyecto;
private int id_red;
private String tabla;
private int numCapas ;
private int numCapasEntrada;
private int numCapasSalidas;
private int numCapasOcultas;
private int numNeuronas;
private int numNeuronasEntrada;
private int numNeuronasSalida;
private int numNeuronasOcultas;
private String tipo_Neurona_Entrada;
private String tipo_Neurona_Salida;
private String tipo_Neurona_Ocultas;
private double[][] capaOrden;
private GestionProyecto secuencia = new GestionProyecto() ;
private double factorAprendizaje;
private double momento;
private double error_deseado_total;
private double error_deseado_patron;
private int numCiclos;
private int funcionActivacion;
```

// bandera que indica si la red ha sido entrenada

// clave primaria de la red

// nombre de la tabla de b/d a entrenar

```

private double peso1;
private double peso2;
private double entradaUmbral;
public boolean parar=false;
public boolean patronesSeteados=false;
private Capa capas[];
private ListaPatrones listaPatrones;
private cls_variables_globales numero= new cls_variables_globales();

```

// el constructor que se debe definir para las redes a utilizar

```

public Red(){};

public void setParar(boolean valor)
{
    this.parar=valor;
}

public boolean getParar()
{
    return this.parar;
}
public void setEntradaUmbral(double umbral)
{
    this.entradaUmbral= umbral;
}
public double getEntradaUmbral()
{
    return this.entradaUmbral;
}
public void setErrorDeseado(double errorDeseado)
{
    this.error_deseado_total=errorDeseado;
}
public double getErrorDeseado()
{
    return this.error_deseado_total;
}
public void setErrorDeseadoPatron(double errorPatron)
{
    this.error_deseado_patron=errorPatron;
}

```

```

// rango inferior de pesos iniciales
// rango superior de pesos iniciales

```

// controla si se asociado los datos de análisis con patrones


```

    }
    public double getErrorDeseadoPatron()
    {
        return this.error_deseado_patron;
    }
    public void setIdProyecto(int id_proyecto)
    {
        this.id_proyecto=id_proyecto;
    }
    public int getIdProyecto()
    {
        return this.id_proyecto;
    }
    public void setTabla(String tabla)
    {
        this.tabla=tabla;
    }
    public String getTabla()
    {
        return this.tabla;
    }
}

```

// crea la capa de entrada con numNeuronas de acuerdo a su tipo de Neurona

```

public void crearCapaEntrada(int numNeuronas, String tipoNeurona)
{
    this.numNeuronasEntrada= numNeuronas;
    this.tipo_Neurona_Entrada=tipoNeurona;
    this.numCapasEntrada= this.numCapasEntrada + 1;           // solo habrá una capa de entrada
}

```

/*Crea la capa de salida con numNeuronas de acuerdo a su tipo de Neurona el numero maximo de capa es la cota el número máximo de capas que habrá en la red .p.e. un perceptron unicapa => 2 capas, capas entonces NumMaximodeCapas = 3 */

```

public void crearCapaSalida(int numNeuronas, String tipoNeurona, int numMaxCapas)
{
    this.numNeuronasSalida= numNeuronas;
    this.tipo_Neurona_Salida=tipoNeurona;
    this.numCapasSalidas=this.numCapasSalidas+1;
}

```

```

// crea una nueva capa oculta con numNeuronas
public void crearCapaOculta(int numNeurona, String tipoNeurona)
{
    this.numNeuronasOculta= numNeuronas;
    this.tipo_Neurona_Oculta=tipoNeurona;
    this.numCapasSalidas= this.numCapasSalidas+1;
}

public void setCapa(Capa capa, int nivel)
{
    this.capas[nivel]= capa;
}

public Capa getCapa(int nivel)
{
    return capas[nivel];
}

public int getId()
{
    return this.id_red;
}

public void setId(int id_red)
{
    this.id_red=id_red;
}

public void setRed()
{
    this.numCapas=0;
    this.numCapasEntrada=0;
    this.numCapasSalidas=0;
    this.numCapasOcultas=0;
    this.numNeuronas=0;
    this.numNeuronasEntrada=0;
    this.numNeuronasSalida=0;
    this.numNeuronasOculta=0;
    this.tipo_Neurona_Entrada="";
    this.tipo_Neurona_Salida="";
    this.tipo_Neurona_Oculta="";
    this.capas= new Capa[3];
}

```

```

    }
    public void setSinapsis(int capaOrigen, int neuronaOrigen, int capaDestino, int neuronaDestino, double peso)
    {
        this.getCapa(capaOrigen).setSinapsis(neuronaOrigen, neuronaDestino, peso);
    }
    public void setPesosUmbrales(int capa, int neurona, double peso)
    {
        this.getCapa(capa).getNeurona(neurona).setPesoUmbral(peso);
    }
    public void unirCapas(int CapaOrigen, int CapaDestino,String tipo_union,String nombreClaseSinapsis, double peso1, double peso2)
    {
        capas[CapaOrigen].unirCapas(capas[CapaDestino], tipo_union, "FeedForward", peso1,peso2);
    }
    public int getNumCapas()
    {
        return this.capas.length;
    }
    public int getnumNeuronas()
    {
        int Neuronas=0;
        for (int i=0; i< this.getNumCapas();i++)
        {
            Neuronas=Neuronas+ this.getCapa(i).getNumNeuronas();
        }
        return Neuronas;
    }

    public void setFactoraprendizaje(double factor)
    {
        this.factorAprendizaje=factor;
    }
    public double getFactorAprendizaje()
    {
        return this.factorAprendizaje;
    }
    public void setMomento(double momento)
    {
        this.momento=momento;
    }
}

```

```
public double getMomento()
{
    return this.momento;
}
public void setNumeroCiclos(int ciclos)
{
    this.numCiclos=ciclos;
}
public int getNumCiclos()
{
    return this.numCiclos;
}
public void setFuncionActivacion(int funcion)
{
    this.funcionActivacion=funcion;
}
public int getFuncionActivacion()
{
    return this.funcionActivacion;
}
public void setPesosAleatoriosNeuronasRed(double minimo, double maximo)
{
    this.setPeso1(minimo);
    this.setPeso2(maximo);
}
private void setPeso1(double peso1)
{
    this.peso1=peso1;
}
private void setPeso2(double peso2)
{
    this.peso2=peso2;
}
public double getPeso1()
{
    return this.peso1;
}
public double getPeso2()
{
    return this.peso2;
}
```

```

}
public void setPotencialAtras(int id_capa)
{
    double potencial;
    int num_capas_i=capas[id_capa].getNumNeuronas();
    int num_capas_j= capas[id_capa-1].getNumNeuronas();

    if (id_capa != 0 ) // menos la capa 1
    {
        for(int i=0; i< num_capas_i ;i++) // POR TODAS LAS NEURONAS DE LA CAPA I
        {
            potencial=0;
            for (int j=0; j<num_capas_j;j++)
            {
                potencial= potencial + (capas[id_capa-1].getNeurona(j).getEstado() * capas[id_capa-1].getSinapsis(j, i, "").getPeso() ) ;
            }
            potencial=potencial + (this.entradaUmbral * capas[id_capa].getNeurona(i).getPesoUmbral()) ; // sumar el valor umbral
            capas[id_capa].getNeurona(i).setPotencialAtras(potencial);
        }
    }
}

public void setListaPatrones(Patron[] patrones)
{
    this.listaPatrones= new ListaPatrones(patrones.length);
    this.listaPatrones.addPatrones(patrones);
    this.listaPatrones.normalizarpatrones();
    patronesSeteados=true;
}

public boolean getPatronesSeteados()
{
    return patronesSeteados;
}

public ListaPatrones getListaPatrones()
{
    return this.listaPatrones;
}

public double error_medio_cuadratico(double salidaDeseada, double salidaRed)
{
    return numero.Redondear(Math.pow((salidaDeseada-salidaRed), 2)/2);
}

```

```

    public void calculaErrorDelta(Capa capa)
    {
        for (int i=0; i<capa.getNumNeuronas();i++)
        {
            capa.getNeurona(i).setErrorDelta(capa.getNeurona(i).getErrorDelta());
        }
    }
    public Patron ejecutarRed(Patron p, boolean aprender)
    {
        return null;
    };
    public void aprender( ListaPatrones patrones){};
}

```

CLASE Red_FeedForward

```

package paq_arquitectura;
import paq_logica.cls_sql;
import java.sql.ResultSet;

public class Red_FeedForward extends Red
{
    private Capa capa_entrada;
    private Capa capa_oculta;
    private Capa capa_salida;
    private ResultSet resultado;
    private cls_sql clase_sql= new cls_sql();

    public Red_FeedForward() {};
    // ***** ARQUITECTURA DE LA RED *****
    public Red_FeedForward(int numNeuronasEntrada,int numNeuronasOculta,int numNeuronasSalida,double factorAprendizaje,int ciclos,double peso1,double
    peso2)
    {
        super.setRed();
        // añadir una neurona para el bias
        capa_entrada= new Capa_FeedForward(numNeuronasEntrada ,0);
        capa_oculta= new Capa_FeedForward(numNeuronasOculta,1);
        capa_salida= new Capa_FeedForward(numNeuronasSalida,2);

        super.setCapa(capa_entrada, 0);
        super.setCapa(capa_oculta, 1);
        super.setCapa(capa_salida, 2);
    }
}

```

```

        super.setPesosAleatoriosNeuronasRed(peso1, peso2);
        super.unirCapas(0, 1, "Total", "Sinapsis_FeedForward", peso1, peso2);
        super.unirCapas(1, 2, "Total", "Sinapsis_FeedForward", peso1, peso2);

        super.setFactoraprendizaje(factorAprendizaje);
        super.setNumeroCiclos(ciclos);

        capa_oculta.setPotencial(capa_entrada);
        capa_salida.setPotencial(capa_oculta);
    }

    public void guardarRed(Red_FeedForward red, int numProyecto )
    {
        String sql;
        boolean exito;
        int num_sinapsis_adelante=0;

        // guardar en la tabla red
        sql= clase_sql.armarQuery("insert_red")+ "(" + red.getId() + "," + numProyecto +
            "," + red.getNumCapas() + "," + red.getnumNeuronas() + "," + red.getFactorAprendizaje() +
            "," + red.getFuncionActivacion() + "," + red.getPeso1() + "," + red.getPeso2() + "," + red.getNumCiclos() + "," + red.getMomento() + "," +
            red.getErrorDeseadoPatron() + "," + red.getErrorDeseado() + ")";
        exito= clase_sql.ejecutar_sql(sql);

        for (int i=0; i < red.getNumCapas(); i++) // recorrer todas las capas (i)
        {

            //guardar en la tabla capa
            if (i!=red.getNumCapas()-1)
                num_sinapsis_adelante=red.getCapa(i).getNumSinapsisAdelante();
            else
                num_sinapsis_adelante=0;

            sql=clase_sql.armarQuery("insert_capa")+ "(" + i + "," + red.getCapa(i).getNumNeuronas()+",0," + num_sinapsis_adelante+",0," + red.getId() + ")";
            exito= clase_sql.ejecutar_sql(sql);

            for (int j=0; j<red.getCapa(i).getNumNeuronas(); j++) // recorrer todas las neuronas (j)
            { // guardar en la tabla Neuronas de la capa i
                sql=clase_sql.armarQuery("insert_neurona") + "(" + red.getCapa(i).getNeurona(j).getId() +
                    "," + red.getCapa(i).getNeurona(j).getEstado() + ",0,0,0,0,0,0," + red.getCapa(i).getNeurona(j).getIdTipoNeurona() +
                    "," + i + "," + red.getId() + "," + red.getCapa(i).getNeurona(j).getPesoUmbral() + ")";

                exito= clase_sql.ejecutar_sql(sql);
            }
        }
    }

```

```

// guardar en la tabla Sinapsis de Neurona hacia adelante
if (red.getCapa(i).getOrden() != red.getNumCapas()-1) // desde la primera capa hasta la capa n-1
{
    for (int z=0; z< red.getCapa(i+1).getNumNeuronas();z++)
    {
        sql= clase_sql.armarQuery("insert_sinapsis")+ "("+ j+ ", "+ z + ", " + red.getCapa(i).getSinapsis(j,z,"total").getPeso()+",0,"+
        red.getCapa(i).getSinapsis(j,z,"total").getTipoSinapsis()+ ", " + i + ", "+ (i+1) + "," + red.getId()+")" ;

        exito= clase_sql.ejecutar_sql(sql);
    }
}
}
}

public void propagacionErrorAtras(Patron salidaRed, Patron salidaDeseada)
{
    double error=0;
    double error_umbral=0;
    int num_capas_i=this.getNumCapas()-1;
    int num_capa_z;
    int num_neuronas_j;

    for(int c=num_capas_i ; c>= 0 ;c--) // Recorrer capas hacia atras
    {
        num_neuronas_j=this.getCapa(c).getNumNeuronas();
        for (int j=0; j<num_neuronas_j ;j++) // recorre las neuronas
        {
            error=0;
            if (c==num_capas_i) // ultima capa
            {
                error =salidaDeseada.getValorNormalizadoSalida(j)- salidaRed.getSalida()[j];
                this.getCapa(c).getNeurona(j).setErrorDelta(error);
            }
            else
            {
                error=0;
                error_umbral=0;
                num_capa_z=this.getCapa(c+1).getNumNeuronas();
                for (int z=0; z < num_capa_z; z++ ) // z maneja las neuronas de la capa superior
                {
                    error= error + (this.getCapa(c).getSinapsis(j, z, null).getPeso()) * (this.getCapa(c+1).getNeurona(z).getErrorDelta());
                }
                this.getCapa(c).getNeurona(j).setErrorDelta(error);
            }
        }
    }
}

```



```

    }
}

// ++++++ DINAMICA A NIVEL DE RED ++++++
/*Toma como parametro un patron de entrada y devuelve otro patron procesado */
@Override

public Patron ejecutarRed(Patron patron, boolean aprender)
{
    int num_capas=this.getNumCapas();
    // 1.-Enviar el Patron de Entrenamiento n a la capa de Entrada
    super.getCapa(0).enviarPatron(patron,aprender);

    // 2.- Propagacion FeedForward hacia adelante
    for (int xi=1 ;xi < num_capas; xi++)
    {
        super.setPotencialAtras(xi);
        // Las Neuronas de la capa siguiente consultan a las de la capa anterior y se actualizan
        super.getCapa(xi).actualizarNeuronas(super.getFuncionActivacion());
    }
    // 3.- devuelve el patron procesado
    return super.getCapa(this.getNumCapas()-1 ).getEstado();
};

// ++++++ APRENDIZAJE A NIVEL DE RED ++++++
/* Define como aprende la red de la lista de patrones*/

public void aprenderSinapsis()
{
    int num_capas_a=this.getNumCapas()-1;
    int num_neuronas_b;
    int num_neuronas_c;

    for (int a=0; a<num_capas_a;a++) // por todas las capas
    {
        num_neuronas_b=this.getCapa(a).getNumNeuronas();
        for (int b=0; b<num_neuronas_b;b++) //por las neuronas de la capa i
        {
            num_neuronas_c=this.getCapa(a+1).getNumNeuronas();
            for (int c=0; c< num_neuronas_c;c++)
            {
                this.getCapa(a).getSinapsis(b, c, null).aprender(this.getFactorAprendizaje(),this.getMomento(),this.getFuncionActivacion() );
            }
        }
    }
}

```

```

    }
}

@Override
public void aprender(ListaPatrones listaPatrones)
{
    for (int i=0; i< listaPatrones.getNumeroPatrones();i++)
    {
        double error_red=0;
        Patron salidaReal= new Patron ();
        Patron salidaDeseada= new Patron();

        salidaReal= this.ejecutarRed(listaPatrones.getPatron(i),true);
        salidaDeseada= listaPatrones.getPatron(i);

        for (int j=this.getNumCapas()-1;j>0;j--)
        {
            for(int z=0; z< this.getCapa(j).getNumNeuronas();z++)
            {
                error_red=error_red + salidaReal.getDiferenciaRealDeseada(salidaDeseada, z);
                this.getCapa(j).getNeurona(z).setErrorDelta(salidaReal.getDiferenciaRealDeseada(salidaDeseada, z));

                if ((this.getCapa(j).getNeurona(z).getErrorDelta()!=0) && j!=0 ) //menos la capa 0
                    this.setSinapsis(j-1, i, i, error_red);
            }
        }
    }
}
}

```

CLASE Capa

// La clase capa la encargada de agrupar y organizar a las neuronas por nivel de procesamiento

```

package paq_arquitectura;

public abstract class Capa
{
    private int orden; // indica el nivel de capa 0= capa de entrada y n=capa_salida
    private int num_neuronas;
    private String tipo;
    private Neurona[] neuronas;
    private Sinapsis[][] sinapsis;
    private Patron patron;
}

```

```

public Capa({});

public Capa(int num_neuronas, int nivel_id, String tipo_neurona)
{
    this.num_neuronas= num_neuronas;
    this.tipo=tipo_neurona;
    this.orden= nivel_id;

    if (tipo_neurona.equals("Perceptron_Simple") )
    {
        neuronas= new Neurona_PerceptronSimple[num_neuronas];
        for(int i=0; i<num_neuronas; i++)
            crearNeurona("Perceptron_Simple",i);
    }

    if (tipo_neurona.equals("FeedForward") )
    {
        neuronas= new Neurona_FeedForward[num_neuronas];

        for(int i=0; i<num_neuronas; i++)
            crearNeurona("FeedForward",i);
    }
}

// une la capa actual a la capa_a_unir con el uso de la clase sinapsis
public void unirCapas(Capa capa_superior,String tipo_union, String tipo_sinapsis,double peso1, double peso2)
{
    if (this.getOrden()!=capa_superior.getOrden()) // si es la misma capa
    {
        if (tipo_union.equals( "Total"))
        {
            sinapsis = new Sinapsis[this.num_neuronas][capa_superior.getNumNeuronas()];

            for(int i=0; i<this.getNumNeuronas();i++) // capa Inicio
            {
                for (int j=0; j< capa_superior.getNumNeuronas();j++) // capa destino
                {
                    sinapsis[i][j]= new Sinapsis_FeedForward(this.getNeurona(i),capa_superior.getNeurona(j),0);
                    sinapsis[i][j].setPesoAleatorio(peso1, peso2);
                    capa_superior.getNeurona(j).setPesoUmbral(capa_superior.getNeurona(j).getPesoUmbralAleatorio(peso1, peso2));
                }
            }
        }
    }
}

```

```

        }
    }
}

public void crearNeurona(String tipo_neurona, int id )
{
    if (tipo_neurona.equals("FeedForward") )
    {
        neuronas[id]=new Neurona_FeedForward(id,0);
    }
    if (tipo_neurona.equals("Perceptron_Simple") )
    {
        neuronas[id]=new Neurona_PerceptronSimple(id,0);
    }
}

// Devuelve el identificador(Orden) de la capa; para la capa de entrada =0
public int getOrden()
{
    return this.orden;
}

public int getNumNeuronas()
{
    return this.num_neuronas;
}

public int getNumSinapsisAdelante()
{
    return sinapsis.length;
}

// devuelve el una neurona de la capa con identificador id_neurona
public Neurona getNeurona(int id_neurona)
{
    return this.neuronas[id_neurona];
}

// devuelve la sinapsis que unela neurona Origen y destino de acuerdo a su identificador de neurona
public Sinapsis getSinapsis(int NeuronaOrigen, int NeuronaDestino,String tipo_union)
{
    return sinapsis[NeuronaOrigen][NeuronaDestino];
}

public void setSinapsis(int neuronaOrigen, int neuronaDestino,double peso)

```

```
{
    this.sinapsis[neuronaOrigen][neuronaDestino].setPeso(peso);
}
```

//Este método llama al método 'Metodo' de todas las neuronas de la capa que deben ser de la clase 'Tipo'.

```
public void ejecutarNeurona(String tipo, String metodo)
{
    if (tipo.equals("Neurona_FeedForward"))
    {
        if(metodo.equals("metodo_especial"))
            for (int i=0;i<this.num_neuronas;i++)
                this.neuronas[i].metodo_especial();
    }
}

public void setErroresUmbralsDelta(double errorDeltaUmbral)
{
    for (int x=0; x<this.getNumNeuronas();x++)
    {
        this.getNeurona(x).setErrorDeltaUmbral(errorDeltaUmbral);
    }
}
```

// este metodo actualiza todas las neuronas de la capa segun se lo define el metodo actualizar de la neurona hija (Sincronismo)

```
public void actualizarNeuronas(int tipo)
{
    int num_neurona=this.num_neuronas;
    for (int s=0; s<num_neurona;s++)
    {
        this.neuronas[s].actualizar(tipo);
    }
}
```

//Este metodo hace que todas las neuronas tengan por estado los valores del patron. Se asigna por orden, es decir la primera neurona recibe el primer valor y asi sucesivamente

```
public void enviarPatron(Patron p , boolean aprender )
{
    int tamañoEntrada= p.getTamañoEntrada();

    for (int i=0;i< tamañoEntrada;i++)
    {
        if (aprender)
            this.neuronas[i].setEstado(p.getValorNormalizadoEntrada(i)); // enviar datos normalizados
        else
            this.neuronas[i].setEstado(p.getValorEntrada(i));
    }
}
```

```

    }
    // Hace que todas las neuronas de la capa tengan por salida deseada los valores de salida que tiene el patron
    public void enviarSalida(Patron p)
    {
        for (int i=0;i<this.num_neuronas;i++ )
        {
            // this.neuronas[i].setSalidaDeseada(p.);
        }
    }

    public Patron getEstado()
    {
        double entradas[]=new double[0];
        double salidas[]= new double[this.num_neuronas];
        for (int i=0;i<this.num_neuronas;i++)
        {
            salidas[i]= this.neuronas[i].getEstado();
        }
        this.patron= new Patron(entradas,salidas);
        return patron;
    }

    public void setPotencial(Capa capa_referencia){};
}

```

CLASE Capa_FeedForward

```

package paq_arquitectura;

public class Capa_FeedForward extends Capa
{

    public Capa_FeedForward(int num_neuronas, int nivel)
    {
        super(num_neuronas,nivel,"FeedForward");
    }

    @Override
    // Potencial atras de neurona i hasta neurona j
    public void setPotencial(Capa capa_referencia)
    {

        double potencial=0;
        double norma=0;
    }
}

```

```

for (int i=0;i< super.getNumNeuronas();i++)
{
    norma=0;
    potencial=0;
    for (int j=0;j<capa_referencia.getNumNeuronas();j++)
    {
        potencial=potencial +( capa_referencia.getNeurona(j).getEstado() * capa_referencia.getSinapsis(j, i, "").getPeso()) ;
        norma= norma + Math.pow( capa_referencia.getNeurona(j).getEstado() * capa_referencia.getSinapsis(j, i, "").getPeso(),2);
    }

    potencial=potencial - (super.getNeurona(i).getPesoUmbral()*-1);
    super.getNeurona(i).setPotencialAtras(potencial);
    super.getNeurona(i).setNormaCuadraticaAtras(norma);
}
}

}

package paq_arquitectura;
import java.util.Random;
import paq_logica.cls_variables_globales;

public abstract class Neurona
{
    private int neu_id;
    private double neu_estado;
    private int neu_tipo_neurona;
    private double neu_salida_deseada;
    private double neu_potencial_lateral;
    private double neu_potencial_adelante;
    private double neu_potencial_atras;
    private double neu_norma_cuadratica_adelante;
    private double neu_norma_cuadratica_lateral;
    private double neu_norma_cuadratica_atras;
    private double error_delta=0;
    private double peso_umbral;
    private double error_delta_umbral=0;
    private cls_variables_globales numero= new paq_logica.cls_variables_globales();

    public Neurona({});

    public Neurona(int id, double peso)
    {
        setId(id);
        setEstado(peso);
    }
}

```

CLASE Neurona

// identificador de la neurona, el sesgo tiene el id=0
// Estado de la neruona
//1=Perceptron Simple,2= Perceptron Multicapa, 3= FeedForward

```

public int getId()
{
    return this.neu_id;
}
public void setId(int id)
{
    this.neu_id=id;
}

public double getPesoUmbralAleatorio(double minimo, double maximo)
{
    java.util.Random estado_aleatorio= new java.util.Random();
    return numero.Redondear(minimo + (estado_aleatorio.nextDouble()*(maximo-minimo)));
}
public void setPesoUmbral(double pesoUmbral)
{
    this.peso_umbral= numero.Redondear(pesoUmbral);
}
public double getPesoUmbral()
{
    return this.peso_umbral;
}
// retorna el id del tipo de neurona
public int getIdTipoNeurona()
{
    return neu_tipo_neurona;
}
// setear el tipo de neurona
public void setTipoNeurona(int id_tipo)
{
    this.neu_tipo_neurona= id_tipo;
}
// devuelve el estado de activacion de la neurona
public double getEstado()
{
    return this.neu_estado;
}
// actualiza el estado de la neurona
public void setEstado(double nuevoEstado)
{
    this.neu_estado= numero.Redondear(nuevoEstado);
}
// devuelve el valor almacenado de la salida deseado de la neurona
public double getSalidaDeseada()
{
    return this.neu_salida_deseada;
}

```



```

}
// actualiza el valor de la salida deseada de la neurona
public void setSalidaDeseada(double nuevaSalidaDeseada)
{
    this.neu_salida_deseada= nuevaSalidaDeseada;
}
// actualiza el estado de la neurona a uno aleatorio comprendido entre un minimo y un maximo, lo set con un double
public void setEstadoAleatorio(double minimo, double maximo)
{
    Random estado_aleatorio= new Random();
    this.neu_estado= minimo + (estado_aleatorio.nextDouble()*((maximo+1)-minimo));
}
// actualiza el estado de la neurona a uno aleatorio comprendido entre un minimo y un maximo, lo set con un int
public void setEstadoAleatorio(int minimo, int maximo)
{
    Random estado_aleatorio= new Random();
    this.neu_estado= minimo + ((int)estado_aleatorio.nextInt()*((maximo+1)-minimo));
}
/****** Potenciales Neuronales ******/
public void setPotencialAdelante(double potencial)
{
    this.neu_potencial_adelante=numero.Redondear(potencial);
}
public void setPotencialLateral(double potencial)
{
    this.neu_potencial_lateral=numero.Redondear(potencial);
}
public void setPotencialAtras(double potencial)
{
    this.neu_potencial_atras=numero.Redondear(potencial);
}
public double getPotencialLateral()
{
    return this.neu_potencial_lateral;
}
public double getPotencialAdelante()
{
    return this.neu_potencial_adelante;
}
public double getPotencialAtras()
{
    return this.neu_potencial_atras;
}
public void setErrorDelta(double error)
{
    this.error_delta= numero.Redondear(error);
}

```

```

}
public double getErrorDelta()
{
    return this.error_delta;
}
public void setErrordeltaUmbral(double error_delta_umbral)
{
    this.error_delta_umbral=numero.Redondear(error_delta_umbral);
}
public double getErrorDeltaUmbral()
{
    return this.error_delta_umbral;
}
public void setNormaCuadraticaAdelante(double norma_cuadratica)
{
    this.neu_norma_cuadratica_adelante=norma_cuadratica;
}
public void setNormaCuadraticaLateral(double norma_cuadratica)
{
    this.neu_norma_cuadratica_lateral=norma_cuadratica;
}
public void setNormaCuadraticaAtras(double norma_cuadratica)
{
    this.neu_norma_cuadratica_atras=norma_cuadratica;
}
public double getNormaCuadraticaAdelante()
{
    return this.neu_norma_cuadratica_adelante;
}
public double getNormaCuadraticaAtras()
{
    return this.neu_norma_cuadratica_atras;
}
public double getNormaCuadraticaLateral()
{
    return this.neu_norma_cuadratica_lateral;
}
public double getFuncionActivacion(int tipo, String direccionPotencial)
{
    switch (tipo) // funcion Signo
    {
        case 0:
            if (direccionPotencial.equals("Atras"))
                return getFuncionSigno(this.getPotencialAtras());
            if (direccionPotencial.equals("Adelante"))
                return getFuncionSigno(this.getPotencialAdelante());
    }
}

```

```

        if (direccionPotencial.equals("Lateral"))
            return getFuncionSigno(this.getPotencialLateral());
        break;
    case 1: // funcion Escalon
        if (direccionPotencial.equals("Atras"))
            return getFuncionEscalon(this.getPotencialAtras());
        if (direccionPotencial.equals("Adelante"))
            return getFuncionEscalon(this.getPotencialAdelante());
        if (direccionPotencial.equals("Lateral"))
            return getFuncionEscalon(this.getPotencialLateral());
        break;
    case 2: // funcion Sigmoide
        if (direccionPotencial.equals("Atras"))
            return getFuncionSigmoide(this.getPotencialAtras());
        if (direccionPotencial.equals("Adelante"))
            return getFuncionSigmoide(this.getPotencialAdelante());
        if (direccionPotencial.equals("Lateral"))
            return getFuncionSigmoide(this.getPotencialLateral());
        break;

    case 3: // funcion Tanh
        if (direccionPotencial.equals("Atras"))
            return getFuncionTahn(this.getPotencialAtras());
        if (direccionPotencial.equals("Adelante"))
            return getFuncionTahn(this.getPotencialAdelante());
        if (direccionPotencial.equals("Lateral"))
            return getFuncionTahn(this.getPotencialLateral());
        break;
    }
    return 0;
}

Public double getFuncionSigno(double potencial)
{
    if (potencial>=0 )
        return 1;
    else
        return -1;
}

public double getFuncionEscalon(double potencial)
{
    if (potencial>=0 )
        return 1;
    else
        return 0;
}

```

```

    public double getFuncionSigmoide(double potencial)
    {
        return 1/(1+Math.exp(-potencial));
    }
    public double getFuncionTahn(double potencial)
    {
        return Math.tanh(potencial);
    }
    public double getDerivadaSigmoide(double potencial)
    {
        return potencial*(1-potencial);
    }

    public double getDerivadaTanh(double potencial)
    {
        return Math.pow((1/ Math.cosh(potencial)),2);
    }
    // Debemos pasar de un estado a otro a la neurona
    public abstract void actualizar(int tipo);
// METODO DE LA DINAMICA DE LA NEURONA
    public abstract void metodo_especial();
}

```

CLASE Neurona_FeedForward

```

package paq_arquitectura;

public class Neurona_FeedForward extends Neurona
{
    private int tipo_neurona_id=3; // id de la Neurona FeedForward

    public Neurona_FeedForward();

    public Neurona_FeedForward(int id, double peso)
    {
        super(id,peso);
        super.setTipoNeurona(tipo_neurona_id);
    }

// redefinir el metodo actualizar ++++++ DINAMICA A NIVEL DE NEURONA
    public void actualizar(int tipo)
    {
        super.setEstado( super.getFuncionActivacion(tipo, "Atras"));
    }
}

```

CLASE Sinapsis

```
package paq_arquitectura;

import paq_logica.cls_variables_globales;

public abstract class Sinapsis
{
    private double peso;
    private double peso_anterior;
    private int tipo_sinapsis;
    private Neurona neurona_inicio;
    private Neurona neurona_fin;
    private cls_variables_globales numero= new paq_logica.cls_variables_globales();

    public Neurona getNeuronaInicio()
    {
        return this.neurona_inicio;
    }
    public Neurona getNeuronaFin()
    {
        return this.neurona_fin;
    }
    public void setNeuronaInicio(Neurona inicio)
    {
        this.neurona_inicio=inicio;
    }

    public void setNeuronaFin(Neurona fin)
    {
        this.neurona_fin=fin;
    }
    public void setTipoSinapsis(int tipo)
    {
        this.tipo_sinapsis=tipo;
    }
    public int getTipoSinapsis()
    {
        return this.tipo_sinapsis;
    }
    public double getPeso()
    {

```

```

    return this.peso;
}
// actualizar el peso de la sinapsis
public void setPeso(double nuevo_peso)
{
    this.peso= numero.Redondear(nuevo_peso);
}
public void setPesoAnterior(double peso_anterior)
{
    this.peso_anterior=numero.Redondear(peso_anterior);
}
public double getPesoAnterior()
{
    return this.peso_anterior;
}
// retorna el potencial Sinaptico: Peso actual de la sinapsis * el estado neurona entrante
public double getPotencialSinaptico()
{
    return this.peso * this.neurona_inicio.getEstado();
}
// devuelve el estado de la neurona que entra a la sinapsis
public double getEstadoNeuronaInicio()
{
    return this.neurona_inicio.getEstado();
}
// devuelve el estado de la neurona que sale a la sinapsis
public double getEstadoNeuronaFin()
{
    return this.neurona_fin.getEstado();
}

// devuelve la salida deseada de la neurona que entra a la sinapsis
public double getSalidaDeseadaNeuronaInicio()
{
    return this.neurona_inicio.getSalidaDeseada();
}
// devuelve la salida deseada de la neurona que sale a la sinapsis
public double getSalidaDeseadaNeuronaFin()
{
    return this.neurona_fin.getSalidaDeseada();
}

public void setPesoAleatorio(double minimo, double maximo)
{
    double f=0;
    java.util.Random estado_aleatorio= new java.util.Random();

```

```

        f=minimo + (estado_aleatorio.nextDouble()*(maximo-minimo));
        setPeso( numero.Redondear(f));
        setPesoAnterior(0);
    }

```

/* Se debe definir para el tipo específico a usar de sinapsis ya que determina el comportamiento en cuanto al aprendizaje. Lo que debe hacer es actualizar el peso de la sinapsis siguiendo la regla de APRENDIZAJE del tipo de red a utilizar */

```

    public abstract void aprender(double factorAprendizaje, double momento,int funcion_activacion );{}

```

CLASE Sinapsis__FeedForward

```

package paq_arquitectura;

public class Sinapsis_FeedForward extends Sinapsis
{
    private int tipo_sinapsis=2;

    public Sinapsis_FeedForward(Neurona inicio, Neurona fin,double peso)
    {
        super.setNeuronalInicio(inicio);
        super.setNeuronaFin(fin);
        super.setPeso(peso);
        super.setTipoSinapsis(tipo_sinapsis);
    }

    // ++++++ APRENDIZAJE A NIVEL DE SINAPSIS ++++++
    public void aprender(double factorAprendizaje, double momento , int funcion_activacion)
    {
        double peso_anterior= this.getPesoAnterior();
        double peso_actual= this.getPeso();
        double peso_siguiete=0;
        double error_delta_neurona_fin=this.getNeuronaFin().getErrorDelta();
        double salida_neurona_inicio=this.getEstadoNeuronalInicio();
        double salida_neurona_fin=this.getEstadoNeuronaFin();
        double derivada_salida_neurona_fin=0;

        if (funcion_activacion==2)
        {
            derivada_salida_neurona_fin=this.getNeuronaFin().getDerivadaSigmoide(salida_neurona_fin);
        }
        if (funcion_activacion==3)
        {
            derivada_salida_neurona_fin=this.getNeuronaFin().getDerivadaTanh(salida_neurona_fin);
        }

        int id_neurona_inicio= this.getNeuronalInicio().getId();

```

```

int id_neurona_fin=this.getNeuronaFin().getId();

    peso_siguiente= peso_actual + (factorAprendizaje * error_delta_neurona_fin * salida_neurona_inicio * derivada_salida_neurona_fin) + (momento *
        (peso_actual-peso_anterior));

    //actualizar el peso umbral
    this.getNeuronaFin().setPesoUmbral(this.getNeuronaFin().getPesoUmbral() + (factorAprendizaje*error_delta_neurona_fin*-
        1*derivada_salida_neurona_fin) );

    this.setPeso(peso_siguiente);
    this.setPesoAnterior(peso_actual);
}
}

```

CLASE ListaPatrones

```

package paq_arquitectura;
import paq_logica.cls_variables_globales;

// La clase actual es la agrupacion de patrones de la red a utilizar, es conveniente para el manejo de gran cantidad de patrones

public class ListaPatrones
{
    private Patron patrones[];
    private double clases[];
    private double temp[];
    private int num_clases_diferentes=0;
    private cls_variables_globales numero= new cls_variables_globales();

    ListaPatrones(int id_red,int num_patrones)
    {
        this.patrones= new Patron[num_patrones];
    }
    ListaPatrones(int num_patrones)
    {
        this.patrones= new Patron[num_patrones];
    }
    public void crear_ordenar_clases()
    {
        temp=new double[patrones.length];
        clases= new double[getNumClasesDistintas()];

        for (int d=0;d<clases.length;d++)
        {
            clases[d]=temp[d];
        }
    }
}

```



```

    }
    ordenar_clases();
}
private void ordenar_clases()
{
    int longitud=clases.length;
    double tmp;

    for(int i=0;i<longitud;i++)
    {
        for (int j=0;j<longitud-1;j++)
        {
            if (clases[j]>clases[j+1])
            {
                tmp=clases[j];
                clases[j]=clases[j+1];
                clases[j+1]=tmp;
            }
        }
    }
}
private boolean existe_clase(double valor)
{
    for (int x=0;x<this.temp.length;x++)
    {
        if (valor== this.temp[x])
            return true;
    }
    return false;
}
private int getNumClasesDistintas()
{
    num_clases_diferentes=1;
    temp[num_clases_diferentes-1]= this.patrones[0].getValorSalida(0); //guarda el primer valor
    for (int i=1;i< this.patrones.length;i++)
    {
        if (!existe_clase(this.patrones[i].getValorSalida(0)))
        {
            num_clases_diferentes=num_clases_diferentes+1;
            temp[num_clases_diferentes-1]= this.patrones[i].getValorSalida(0);
        }
    }
    return num_clases_diferentes;
}

public void addPatrones(Patron[] p)

```

```

{
    this.patrones=p;
}
public double[] getClases()
{
    return this.clases;
}
public double getValorNormalizadoConsulta(double valor)
{
    double valorD;
    valorD= (valor - this.getMINIMO())/(this.getMAXIMO()-this.getMINIMO());

    return numero.Redondear(valorD);
}
public void normalizarpatrones()
{
    int tamEntradas=patrones[0].getTamañoEntrada();
    int tamSalidas= patrones[0].getTamañoSalida();
    double entrada[];
    double salida[];
    double maximo;
    double minimo;
    double valorD;

    for(int z=0; z<patrones.length;z++)
    {
        entrada = new double[tamEntradas];
        for (int x=0;x<tamEntradas;x++)
        {
            valorD= (patrones[z].getValorEntrada(x)- this.getMINIMO())/(this.getMAXIMO()-this.getMINIMO());
            entrada[x]=numero.Redondear(valorD); //Double.valueOf(numero);

        }
        patrones[z].setEntradaNormalizada(entrada);

        salida = new double[tamSalidas];
        for (int x=0;x<tamSalidas;x++)
        {
            valorD= (patrones[z].getValorSalida(x)- this.getMINIMO())/(this.getMAXIMO()-this.getMINIMO());
            salida[x]= numero.Redondear(valorD); //Double.valueOf(numero);
        }
        patrones[z].setSalidaNormalizada(salida);
    }
}

public double getMaxEntrada(int posicion)

```

```

{
    double maximo=patrones[0].getValorEntrada(posicion);
    for (int i=0; i<patrones.length;i++)
    {
        if ( patrones[i].getValorEntrada(posicion)> maximo )
            maximo=patrones[i].getValorEntrada(posicion);
    }
    return maximo;
}

public double getMaxSalida(int posicion)
{
    double maximo=patrones[0].getValorSalida(posicion);
    for (int i=0; i<patrones.length;i++)
    {
        if ( patrones[i].getValorSalida(posicion)> maximo )
            maximo=patrones[i].getValorSalida(posicion);
    }
    return maximo;
}

public double getMinEntrada(int posicion)
{
    double minimo=patrones[0].getValorEntrada(posicion);
    for (int i=0; i<patrones.length;i++)
    {
        if ( patrones[i].getValorEntrada(posicion)< minimo )
            minimo=patrones[i].getValorEntrada(posicion);
    }
    return minimo;
}

public double getMinSalida(int posicion)
{
    double minimo=patrones[0].getValorSalida(posicion);
    for (int i=0; i<patrones.length;i++)
    {
        if ( patrones[i].getValorSalida(posicion)< minimo )
            minimo=patrones[i].getValorSalida(posicion);
    }
    return minimo;
}

public double getMAXIMO()
{
    double maximo=0;
    double temp;
    for (int x=0;x<patrones[0].getTamañoEntrada();x++)

```

```

    {
        temp=this.getMaxEntrada(x);
        if (temp>maximo)
            maximo=temp;
    }
    for (int x=0;x<patrones[0].getTamañoSalida();x++)
    {
        temp=this.getMaxSalida(x);
        if (temp>maximo)
            maximo=temp;
    }
    return maximo;
}

public double getMINIMO()
{
    double minimo=0;
    double temp;
    for (int x=0;x<patrones[0].getTamañoEntrada();x++)
    {
        temp=this.getMinEntrada(x);
        if (temp<minimo)
            minimo=temp;
    }
    for (int x=0;x<patrones[0].getTamañoSalida();x++)
    {
        temp=this.getMinSalida(x);
        if (temp<minimo)
            minimo=temp;
    }
    return minimo;
}

// Devuelve el patrón que esta en la posicion i. Debe estar en el rango 0... numPatrones()-1
public Patron getPatron(int posicion)
{
    return this.patrones[posicion];
}

public int getNumeroPatrones()
{
    return this.patrones.length;
}

public double getClaseDefinida(double valor)
{
    double clase=0;
    double temp1;
    double temp2=this.clases[this.clases.length-1];

```

```

    for(int i=0;i<this.clases.length;i++)
    {
        temp1=Math.abs(valor-clases[i]);
        if (temp1<temp2)
            clase= clases[i];
        temp2=temp1;
    }
    return clase;
}
}

```

CLASE Patron

```

package paq_arquitectura;
import paq_logica.cls_sql;
// Las rna funcionan con patrones, es decir la lista de valores de entrada o de salida que corresponde a los estados de las neuronas

```

```

public class Patron
{
    private int tamaño_entrada;           //Tamaño del array de Estados que tendra el patron.
    private int tamaño_salida;           // tamaño de array salidas deseadas q tendra patron
    private double salida[];
    private double entrada[];
    private double entrada_normalizada[];
    private double salida_normalizada[];
    private int posicion;           // posicion del patron 0,1,2.....N
    private cls_sql sql= new cls_sql();

    // Crea un patron con un array de tamños de entrada y salida, inicializando los valores correspondiente del array
    public Patron(int tamSalida, int tamEntrada, double salida[],double entrada[], int posicion)
    {
        this.tamaño_entrada=tamEntrada;
        this.tamaño_salida=tamSalida;
        this.entrada= new double[tamEntrada];
        this.entrada_normalizada= new double[tamEntrada];
        this.entrada= entrada;
        this.salida = new double[tamSalida];
        this.salida_normalizada= new double[tamSalida];
        this.salida=salida;
        this.posicion=posicion;
    }

    public Patron(){};

    // Crea un patron con un array de tamños de entrada y salida, pero sin inicializar valores (vacio)
    public Patron(int tamSalida, int tamEntrada)
    {

```

```

        this.tamaño_entrada=tamEntrada;
        this.tamaño_salida=tamSalida;
        this.entrada= new double[tamEntrada];
        this.salida = new double[tamSalida];
    }
    public Patron(double entradas[],double salidas[])
    {
        this.tamaño_entrada= entradas.length;
        this.tamaño_salida= salidas.length;
        this.entrada=entradas;
        this.salida=salidas;
    }
    public int getPosicion()
    {
        return this.posicion;
    }
    public void setEntradaNormalizada(double[] datos)
    {
        this.entrada_normalizada=datos;
    }
    public void setSalidaNormalizada(double[] datos)
    {
        this.salida_normalizada =datos;
    }
    // actualiza o cambia Posición a que se refiere.tiene que tener valores entre 0... tamañoEntrada-1
    public void cambiarSalida(double valor, int posicion)
    {
        this.salida[posicion]=valor;
    }
    public void cambiarentrada(double valor, int posicion)
    {
        this.entrada[posicion]=valor;
    }
    // devuelve el valor de entrada de array de acuerdo a su posicion
    public double getValorEntrada(int posicion)
    {
        return this.entrada[posicion];
    }

    public double getValorNormalizadoEntrada(int posicion)
    {
        return this.entrada_normalizada[posicion];
    }
    public double getValorNormalizadoSalida(int posicion)
    {
        return this.salida_normalizada[posicion];
    }

```

```

    }
    public double getValorSalida(int posicion)
    {
        return this.salida[posicion];
    }
    public int getTamañoEntrada(Patron p)
    {
        return p.tamaño_entrada;
    }
    public int getTamañoEntrada()
    {
        return this.tamaño_entrada;
    }
    public int getTamañoSalida()
    {
        return this.tamaño_salida;
    }
    public int getTamañoSalida(Patron p)
    {
        return p.tamaño_salida;
    }
    // comprar el patron actual con otro patron en sus entradas y salidas
    public boolean compararPatrones(Patron p)
    {
        if (compararEntradas(p) && compararSalidas(p) )
            return true;
        else
            return false;
    }

    public boolean compararEntradas(Patron p)
    {
        for (int i=0; i< this.tamaño_entrada-1; i++)
        {
            if (this.entrada[i] != p.getValorEntrada(i))
                return false;
            }
        return true;
    }
    public boolean compararSalidas(Patron p)
    {
        for (int i=0; i< this.tamaño_salida-1; i++)
        {
            if (this.salida[i] != p.getValorSalida(i))
                return false;
            }
        }
    }

```

```

    return true;
}

public double[] getSalida()
{
    return this.salida;
}

public double[] getEntrada()
{
    return this.entrada;
}

public double getDiferenciaRealDeseada(Patron salidaDeseada,int posicion )
{
    return salidaDeseada.getSalida()[posicion] - this.salida[posicion] ;
}

/* Compara el estado del patron actual con la salida deseada del patron p. Se utilizara cuando se desea revisar que un patron q ha salido de una computacion de una red es igual a otro que tenemos como objetivo.*/
public boolean compararEntradasSalidas(Patron p)
{
    for (int i=0; i< this.tamaño_entrada-1; i++)
    {
        if (this.entrada[i] != p.getValorSalida(i))
            return false;
    }
    return true;
}

public Patron ejecutar(Patron p)
{
    return p;
}

```


HERRAMIENTAS UTILIZADAS

El desarrollo del prototipo es completamente en java donde podemos aprovechar los conceptos orientado a objetos y poder representar de una manera eficaz y simple con respecto a la programación estructural. Poder plasmar las distintas entidades que participan para lograr tanto la arquitectura como la lógica funcional en manera de objetos, ayuda al desempeño flexible y a una escalabilidad de mantenimiento y desarrollo horizontal.

APIs utilizadas

- **JGraph** es una biblioteca Java de fuente abierta para la visualización de gráficos. Se rige a los patrones de diseño de la librería swing brindando una familiarización con su fuente.
- **JFreechart**, su fácil utilización para el desarrollo de gráficos estadísticos hace una de las mejores alternativas al momento de elegir un api para nuestro prototipo. Su uso dentro del proyecto es para el dibujo de la curva de error, la distribución normal de los errores promedio.

Requerimientos mínimos

Memoria Ram:	512 MB
Procesador:	Dual Core 1.6
Sistema Operativo:	Windows, Linux, Unix, Solaris
Java Runtime Environment 6	
Mysql Server 5.0	

Instalación

- 1.- Importar a la base de datos los archivos: redesneuronales.sql y tablas.sql
- 2.- Copiar la carpeta MelodyNeuronalNetwork a cualquier ubicación de directorio.
- 3.- Ejecutar el Red Neuronal.jar