Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingenieria en Ciencias y Sistemas

IMPLEMENTACIÓN DE SISTEMA HÍBRIDO INTELIGENTE FORMADO POR RED NEURONAL TIPO ART Y ALGORITMO GENÉTICO, ORIENTADO A ROBÓTICA Y ANÁLISIS CONTRA RED NEURONAL TIPO ART1

José Antonio Prera Ivara

Asesorado por el Ingeniero Herman Igor Véliz Linares

Guatemala, junio de 2011

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

IMPLEMENTACIÓN DE SISTEMA HÍBRIDO INTELIGENTE FORMADO POR RED NEURONAL TIPO ART Y ALGORITMO GENÉTICO, ORIENTADO A ROBÓTICA Y ANÁLISIS CONTRA RED NEURONAL TIPO ART1

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA
DE LA FACULTAD DE INGENIERÍA
POR

JOSÉ ANTONIO PRERA IVARA
ASESORADO POR EL ING. HERMAN IGOR VÉLIZ LINARES

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, JUNIO DE 2011

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	Ing. Alfredo Enrique Beber Aceituno
VOCAL II	Ing. Pedro Antonio Aguilar Polanco
VOCAL III	Ing. Miguel Ángel Dávila Calderón
VOCAL IV	Br. Juan Carlos Molina Jiménez
VOCAL V	Br. Mario Maldonado Muralles
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Pedro Pablo Hernández
EXAMINADOR	Ing. Óscar Paz Campos
EXAMINADOR	Ing. César Fernández
SECRETARIA	Inga. Marcia Ivónne Véliz Vargas

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

IMPLEMENTACIÓN DE SISTEMA HÍBRIDO INTELIGENTE FORMADO POR RED NEURONAL TIPO ART Y ALGORITMO GENÉTICO, ORIENTADO A ROBÓTICA Y ANÁLISIS CONTRA RED NEURONAL TIPO ART1

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha noviembre de 2010.

Jose Antonio Prera Ivara

Guatemala, de febrero de 2011

Ingeniero
Marlon Pérez Turk
Director de Escuela
Escuela de ingeniería en ciencias y sistemas
Facultad de Ingeniería, USAC

Señor director:

Atentamente me dirijo a usted para informarle que he tenido a bien asesorar el trabajo de tesis: **"IMPLEMENTACION DE SISTEMA HIBRIDO INTELIGENTE FORMADO POR RED NEURONAL TIPO ART Y ALGORITMO GENETICO ORIENTADO A ROBOTICA Y ANALISIS CONTRA RED NEURONAL TIPO ART1"**, realizado por el estudiante **José Antonio Prera Ivara**, quien se identifica con carné **No. 2003-12586;** previo a optar el título de Ingeniero en Ciencias y Sistemas.

Al respecto quiero indicarle que luego de efectuadas las revisiones y correcciones del caso, encuentro satisfactorio el trabajo por lo que procedo a aprobarlo y remitirlo a usted para su trámite correspondiente.

Atentamente,

Ing. Herman Igor Véliz Linares
COLEGIADO No. 4836

Ing. Herman Igor Véliz Linares
Asesor

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 09 de Marzo de 2011

Ingeniero
Marlon Antonio Pérez Turk
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **JOSÈ ANTONIO PRERA IVARA,** carné **2003-12586,** titulado: **"IMPLEMENTACIÒN DE SISTEMA HIBRIDO INTELIGENTE FORMADO POR RED NEURONAL TIPO ART Y ALGORITMO GENETICO ORIENTADO A ROBOTICA Y ANALISIS CONTRA RED NEURONAL TIPO ART1",** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA EN CIENCIAS Y SISTEMAS
COORDINACION TESIS

Licenciada Susana Lemus Guzmán
Psicóloga

Guatemala, abril 5 de 2011

Señores
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Unidad de Ligüística

Señores:
Por este medio hago constar que realicé las correcciones, en cuanto a redacción, del trabajo de graduación del estudiante **José Antonio Prera Ivara, carnet 2003-12586,** quien abordó el tema: **Implementación de sistema híbrido inteligente formado por red neuronal tipo ART y algoritmo genético, orientado a robótica y análisis contra red neuronal tipo ART1,** como cursante de la carrera de Ingeniería en Ciencias y Sistemas.

Atentamente,

Licenciada Susana Lemus Guzmán
Psicóloga
Colegiada número 1879

Licenciada Susana Lemus Guzmán
Psicóloga
Colegiada 1879

ESCUELA DE CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE CIENCIAS Y SISTEMAS
TEL: 24767644

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación titulado ***"IMPLEMENTACIÓN DE SISTEMA HÍBRIDO INTELIGENTE FORMADO POR RED NEURONAL TIPO ART Y ALGORITMO GENÉTICO, ORIENTADO A ROBÓTICA Y ANÁLISIS CONTRA RED NEURONAL TIPO ART1",*** *presentado por el estudiante JOSÉ ANTONIO PRERA IVARA, aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Ing. Marlon Antonio Pèrez Turk
Director, Escuela de Ingenierìa Ciencias y Sistemas

Guatemala, 27 de junio 2011

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 216 .2010

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al trabajo de graduación titulado: **IMPLEMENTACIÓN DE SISTEMA HÍBRIDO INTELIGENTE FORMADO POR RED NEURONAL TIPO ART Y ALGORITMO GENÉTICO, ORIENTADO A ROBÓTICA Y ANÁLISIS CONTRA RED NEURONAL TIPO ART1,** presentado por el estudiante universitario **José Antonio Prera Ivara,** autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Murphy Olympo Paiz Recinos
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, 29 de junio de 2011.

/gdech

ACTO QUE DEDICO A:

Dios Por ser el eje fundamental en mi vida.

Mi abuela Vidalina Castillo, por apoyarme siempre, compartir su sabiduría y encaminar mi vida hacia el bien.

Mi tía Marleni Castillo, por creer e invertir en mí, con gratitud y cariño, por ser un gran ejemplo en todas las áreas de la vida.

Todos aquéllos que no solamente discuten sobre la inteligencia artificial sino también la crean.

AGRADECIMIENTOS A:

Dios	Por haberme permitido llegar a este momento proveyéndome de inteligencia y salud.
Mis abuelos	Justo, Elba y Vila gracias por su vida, amor y cariño.
Mis padres	Gladys, Enry y Tono por su apoyo y moral y económico, durante distintas etapas.
Mis hermanos	Henry, Cris, por su amistad y soporte incondicional.
Mis tíos	Delma (q.e.p.d.), Luis, Mary, Aury, Chepa, Chani, Marleny, Edgar y Chepe por darme siempre apoyo y cariño.
Mis primos	Luis (q.e.p.d.), Ale (q.e.p.d.), Dina, Mayra, Raquelita, Heber y al resto de la familia que por espacio no puedo mencionar pero los considero mis hermanos.

Mi asesor	Ing. Herman Véliz por brindarme su valioso tiempo, sus consejos y el apoyo recibido durante el desarrollo del trabajo.
Mis compañeros y amigos	Álvaro, Julián, Saimon, Rodolfo, Josué Chávez solo por mencionar algunos nombres de todos aquellos con quienes he compartido momentos importantes a lo largo de mi vida y de quienes he aprendido mucho.
La Universidad de San Carlos de Guatemala	Porque en sus aulas he crecido intelectualmente.
El pueblo de Guatemala	Libre al viento tu hermosa bandera, a vencer o a morir llamará.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
Df	Diferencia
GB	*Gigabyte*
GHz	Gigahercio
MB	*Megabyte*
Mts/s	Metros por segundo
mV	Mili Voltios
Sig.	Nivel de significancia
Exp.	Número de experimento
%cpu	Porcentaje de actividad de Unidad central de proceso
Ms	Tiempo en milisegundos

GLOSARIO

AG	Algoritmo Genético.
ANOVA	Colección de modelos estadísticos y sus procedimientos asociados, en los que la varianza está particionada en ciertos componentes, debido a diferentes variables explicativas.
Aptitud	Capacidad y buena disposición para ejercer o desempeñar una determinada tarea, función, empleo.
ART	*Adaptive Resonance Theory*, teoría de la resonancia adaptativa.
Atractor	Es el conjunto al que un sistema evoluciona después de un tiempo suficientemente largo.
Autogeneración	Proceso en el que tiene lugar una generación o producción de seres o entes a partir solamente de elementos contenidos en el ente generador, sin intervención de un elemento externo.
Autopercepción	Capacidad de determinados sistemas para reconocer cambios dentro de ellos mismos.

Autorreferencia	Fenómeno que ocurre en el lenguaje natural o formal, consistente en una oración o fórmula referente en forma directa a sí misma, a través de algunas oraciones o fórmulas intermedias, o por medio de ciertas codificaciones.
Bioinformática	Aplicación de tecnología de computadores a la gestión y análisis de datos biológicos.
Byte	Unidad básica de almacenamiento de información.
Conexión excitatoria	Impulso transmitido por una neurona, excita la activación de otra con la que está conectada.
Convergencia	Propiedad que poseen algunas sucesiones numéricas de tender a un límite.
CPU *activity*	Cantidad de tiempo que una unidad central ocupa para el procesamiento de instrucciones de un programa de computadora.
ECG	Electrocardiograma, examen que registra la actividad eléctrica del corazón.

Estadística no paramétrica	Rama de la estadística que estudia las pruebas y modelos estadísticos cuya distribución subyacente no se ajusta a los llamados criterios paramétricos.
Evolución	Desarrollo gradual, crecimiento o avance de las cosas u organismos.
Gen computacional	Unidad básica de población.
heap	Pila de memoria, en ella se almacenan los datos introducidos a un sistema y es primordial para el rendimiento de programas.
J2EE	Java 2 *Standar Edition*. Plataforma de programación, parte de la Plataforma Java, para desarrollar y ejecutar *software* de aplicaciones.
JDK	*Software* que provee herramientas para generar aplicaciones en lenguaje java.
JRE	*Java Runtime Enviroment*. Conjunto de utilidades que permite la ejecución de aplicaciones escritas en java.
Layout	*Software* que calcula automáticamente la posición de los objetos.

Modelo matemático	Tipo de modelo científico que emplea algún formulismo matemático para expresar proposiciones sustantivas de hechos, variables, parámetros, entidades y relaciones entre variables y/o entidades u operaciones, para estudiar comportamientos de sistemas complejos, ante situaciones difíciles de observar en la realidad.
Mutación biológica	Alteración o cambio en la información genética (genotipo) de un ser vivo y que, por lo tanto, va a producir un cambio de características, se presenta súbita y espontáneamente y se puede transmitir o heredar.
Patrón	Conjunto de reglas que pueden ser usadas para crear o generar entidades.
Pesos sinápticos	Valores numéricos, se expresan en términos numéricos sencillos (generalmente números enteros o fraccionarios negativos o positivos) con los que “se ponderan” las señales que reciben a través de la sinapsis.

Prueba U de *Mann-Whitney*	Prueba no paramétrica aplicada a dos muestras independientes.
Prueba de *Kruskal-Walis*	Método no paramétrico para probar si un grupo de datos proviene de la misma población. Intuitivamente, es idéntico al ANOVA con los datos reemplazados por categorías. Es una extensión de la prueba de la U de Mann-Whitney para tres o más grupos.
Red ART1	Red neuronal capaz de permitir para su procesamiento únicamente entradas binarias.
Red ART2	Red neuronal capaz de permitir entradas analógicas para su procesamiento.
Retroalimentación	Proceso por el que cierta proporción de la señal de salida de un sistema, se redirige de nuevo a la entrada.
RNA	Ácido ribonucleico, ácido nucleíco formado por una cadena de ribonucleótidos.
Sampler	Mostrador, nos visualiza distintos parámetros del sistema analizado.

Separatriz	Divide el espacio muestral en dos regiones distintas.
Sistema ART1	Desarrollado para este trabajo, consiste en un simulador de red neuronal tipo ART1.
Sistema ART1+AG	Desarrollado para este trabajo que consiste en un simulador de red neuronal tipo ART1 e integra un sistema de algoritmos genéticos, como medio de entrenamiento.
Submodelo	Un modelo que surge a partir de otro como especialización del mismo.
Topología	Forma en que están interconectados los distintos elementos de una red.
XOR	Puerta lógica, realiza la función booleana A'B+AB'.

RESUMEN

Con el advenimiento de la computadora como herramienta principal de procesamiento, se inició la explosión de nuevas formas para abordar problemas que requieren de algún tipo de inteligencia.

En este trabajo se abordan distintos aspectos relacionados con la inteligencia artificial. Entre ellos las teorías que soportan esta rama de la ciencia de la computación, como lo son la teoría de la computabilidad, complejidad computacional, caos y se aborda también una teoría relativamente nueva en computación como lo es la autopoiesis, que surge en biología pero como se demuestra en el trabajo es aplicable a sistemas de inteligencia artificial.

Otro punto abordado es un recorrido histórico por el desarrollo de las redes neuronales desde sus inicios hasta nuestros días para luego revisar distintas propuestas que se han desarrollado en este campo, enfocándonos específicamente en las redes tipo ART, sobre su funcionamiento, su estructura y sus variantes.

Luego se aborda lo relativo a algoritmos genéticos, sus conceptos fundamentales, estructura y aplicaciones. Asimismo se estudian los sistemas inteligentes híbridos, su clasificación y sus aplicaciones, enfatizando en la forma de mezclar redes neuronales con algún otro sistema de inteligencia artificial. Se incursiona también en el desarrollo del sistema ART1+AG en todas sus fases: análisis, diseño, implementación, para luego experimentar con dicho sistema como cerebro de un robot en una aplicación industrial a escala.

Para finalizar, se realizan análisis de rendimiento tanto del sistema ART1 como al sistema ART1+AG para determinar la viabilidad del sistema híbrido, realizando análisis estadístico de forma independiente y comparativa, de modo que se puedan determinar las ventajas de cada uno, para futuras aplicaciones.

OBJETIVOS

General

Desarrollar un sistema funcional que simule el comportamiento de una red neuronal tipo ART1 integrando en su funcionamiento un algoritmo genético.

Específicos

1. Determinar la viabilidad del sistema, analizando estadísticamente su rendimiento, contra un sistema ART1.

2. Recopilar, describir y relacionar conceptos de redes neuronales en especial del tipo ART.

3. Recopilar, describir y relacionar conceptos sobre algoritmos genéticos y sistemas híbridos de inteligencia artificial.

4. Analizar el concepto biológico de autopoiesis y determinar su validez para el sistema a desarrollar.

5. Estructurar un sistema híbrido ART1+AG y conocer sus aplicaciones reales.

6. Determinar la factibilidad de incluir un algoritmo genético como agente entrenador de una red neuronal.

INTRODUCCIÓN

A partir del siglo XX se han desarrollado distintos modelos que intentan resolver, por medio de un computador, problemas que hasta entonces podían ser abordados únicamente por seres humanos y para los cuales se requiere algún tipo de inteligencia. Entre ellos, surgieron dos modelos importantes, el primero relativo a las redes neuronales, que tuvo un auge importante desde principios de los años sesenta hasta los noventa, tiempo en el cual se disminuyó la investigación de este campo, resurgiendo en los últimos años debido al auge de la robótica y la búsqueda de medios para controlar estos sistemas. El segundo modelo surge en los años setenta de la mano de John Henry Holland, trata sobre algoritmos genéticos, ambos presentan una madurez suficiente para estudiarlos y experimentar con ellos.

En el presente trabajo se estudia una forma de mezclar los dos modelos, aprovechando las ventajas de cada uno, abordando desde otro punto de vista la generación de soluciones para problemas que necesitan un alto grado de inteligencia, tomando en cuenta el carácter caótico de las mismas y su grado de complejidad. Además se analiza por qué se dice que el sistema desarrollado tiende a ser autopoiético y cómo, este concepto acuñado en biología, también tiende a explicar fenómenos en la ciencia de la computación y en general puede ser aplicado a distintas disciplinas.

La forma de abordar los problemas no se describe únicamente de manera teórica, también se implementa una aplicación que combina el simulador ART1 con un sistema de algoritmo genético capaz de entrenarlo para luego generar datos y analizar su rendimiento, utilizando pruebas estadísticas sobre las cuales se afirma la viabilidad del sistema, se busca resaltar las ventajas de combinar ambos modelos para la resolución de problemas.

1. TEORÍAS QUE SOPORTAN LA INVESTIGACIÓN

Al analizar un sistema computacional, el cual debe ser capaz de imitar o automatizar decisiones que hasta el momento, son tomadas únicamente por seres humanos, nos damos cuenta que éstas pueden no tener una solución exacta ante una dificultad o bien muchas soluciones requerir una gran cantidad de datos, que crean experiencia como criterio de decisión. Esto nos hace pensar que no podemos encontrar un algoritmo que describa de manera completa la solución a dichos problemas, por lo tanto debemos involucrar la teoría de la computabilidad.

A estos problemas únicamente podemos darle una solución cercana, dicha solución muchas veces se encuentra únicamente después de varias iteraciones del algoritmo propuesto, por lo tanto se debe involucrar la teoría de la complejidad computacional para tratar de optimizar las decisiones. Estudiando esos problemas podremos observar que al generar una nueva iteración de la propuesta de solución, ésta es distinta y se comporta de manera caótica, no tienen todos los casos un comportamiento predecible.

1.1. Teoría de la complejidad computacional

La teoría de la complejidad computacional es parte de la teoría de la computación y estudia de manera teórica, los recursos necesarios para la implementación de un algoritmo, para resolver determinado problema.

Uno de los recursos más comunes se refiere al tiempo (cuántos pasos toma resolver el problema) y espacio (cuánta memoria es ocupada), otros recursos pueden ser considerados como el número de procesadores necesarios para resolver el problema en paralelo.

Es importante abordar la teoría de la complejidad computacional, debido a que es necesario reconocer qué tan complejo puede resultar resolver un determinado problema al utilizar un sistema de inteligencia artificial, para ello debemos conocer que la teoría de la complejidad divide los problemas en dos clases. La clase P en la cual se agrupan aquellos problemas que tienen como máximo una complejidad o coste computacional polinómico. Y la clase NP que agrupa problemas los cuales tienen un coste computacional no polinómico y sin poseer solución algorítmica, que por lo tanto una máquina no puede resolver en un tiempo razonable.

Otro aspecto importante sobre la teoría de la complejidad y la clasificación de problemas, es el relacionado con la toma de decisiones ya que la mayoría de problemas reclaman una respuesta positiva o negativa.

Los problemas que necesitan inteligencia artificial para resolverse tienden a ser de la clase NP o sub-categorías pues el tiempo computacional requerido para hacerlo es muy grande, debido a que éstos tienden a ser no deterministas o bien a tener soluciones en tiempos prolongados. Un ejemplo es la aplicación de un algoritmo para jugar ajedrez en el cual la meta es poner en jaque al rey; una máquina debe evaluar sus movimientos y las del contrincante, esto genera gran coste computacional pudiendo observar que existen al menos 10 elevado a la 120 distintas partidas de ajedrez que se pueden jugar, aunado a un carácter irreductible del problema, nos lleva a encontrar formas de afrontarlo y reducir el tiempo de respuesta.

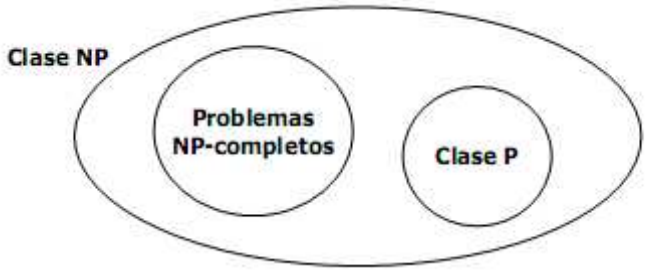
Lo anterior puede abordarse mediante una red neuronal o bien implementar un algoritmo genético que pueda darnos un resultado correcto con menor complejidad. Tiene un coste y está ligado directamente a la efectividad del problema, debido a la naturaleza probabilista y no determinista de las soluciones de inteligencia artificial.

Es importante demostrar que un problema es de tipo NP-completo lo cual significa que es muy complejo y que básicamente es imposible encontrar un algoritmo que nos muestre una solución óptima del mismo, de modo que no se desperdicie tiempo o recursos en esto y se centre en encontrar soluciones aproximadas.

Otro aspecto importante aplicable al proyecto, es el hecho de reducir a problemas de decisiones y observar la complejidad en tiempo y espacio al momento de aplicar soluciones deterministas y sus resultados; así como medir la complejidad entre una red neuronal y un sistema híbrido inteligente compuesto por la red neuronal y un algoritmo genético se puede disminuir la complejidad y observar si el coste en efectividad no es significativo en la solución, es posible al igual encontrar problemas de optimización aunque todo problema de este tipo puede reducirse a cuestiones de decisión.

Uno de los problemas más grandes que se tiene en el campo de teoría de la complejidad es probar que $\mathrm{NP} = \mathrm{P}$, esto significaría que para todos los problemas NP existen algoritmos deterministas que los resuelven en tiempo polinómico pero todavía no ha sido descubierto, aunque que improbable pues está sin verificar, así como tampoco se ha comprobado de manera formal que $\mathrm{NP} \neq \mathrm{P}$ ver figura 1.

Figura 1**. Posible mapa de los problemas NP**



Fuente: http://en.wikipedia.org/wiki/P_versus_NP_problem

Las definiciones hechas anteriormente son válidas para algoritmos deterministas dado que los no deterministas hasta el momento, son imposibles de resolver en la realidad, resolviendo problemas de alta complejidad en tiempos polinómicos, explorando todo el espacio de soluciones en periodos muy cortos.

1.2. Teoría de la computabilidad

Es una rama de la ciencia de la computación, estudia qué problemas pueden tener una solución computacional usando distintos modelos, analiza los problemas de decisión que pueden ser resueltos con un algoritmo equivalente a una máquina de Turing. La teoría de la computabilidad también es un área de la matemática que trata el concepto de procedimiento efectivo, mismo que puede llevarse a cabo siguiendo normas específicas.

De manera formal la teoría de la computabilidad se define como una función $\mathrm{N}^{\mathrm{n}} \rightarrow \mathrm{N}$ se dice ser *computable*-T, o *computable-Turing*, si existe una máquina de Turing que la calcula.

La teoría de la computabilidad es otra área de la ciencia de la computación, difiere de la teoría de la complejidad en que ésta se encarga de expresar los problemas como algoritmos sin tener en cuenta los recursos necesarios para ello; existen muchos problemas indecidibles que no pueden resolverse mediante un algoritmo aunque se tengan espacio y tiempo ilimitado. Esta teoría es importante en el área de inteligencia artificial ya que para muchos de los problemas que pueden resolverse con ella, es difícil encontrar un algoritmo o simplemente son problemas indecidibles a los cuales no puede dársele una solución exacta y hasta hoy día se buscan aproximaciones acertadas como solución.

La máquina de Turing demostró que existen problemas que no pueden resolverse puntualmente con ningún tipo de computador, o no pueden tener una solución exacta, para ello se introdujo el concepto de inteligencia artificial en la cual se trata de dar soluciones aproximadas en las que aparezcan características humanas para dar solución a los problemas tales como generalizar, percibir, comprender, y otras.

En los problemas propuestos a resolverse por el sistema híbrido se debe comprobar la computabilidad de los mismos y esperar respuestas correctas que contengan un alto grado de exactitud.

1.3. Teoría del caos

Es la rama de la física y la matemática que trata ciertos tipos de comportamiento impredecible de los sistemas dinámicos que aparentemente presentan un comportamiento errático o aleatorio, a pesar de que dicho sistema tiene límites y no presenta variables aleatorias.

El caos se refiere a una interconexión subyacente que se manifiesta en acontecimientos aparentemente aleatorios. Es algo contrario a la noción del determinismo para el cual cada evento o acción es resultado concreto de acontecimientos o acciones pasadas, pudiéndose predecir algún comportamiento específico.

Un sistema dinámico puede clasificarse en estable, inestable o caótico; el estable, a lo largo del tiempo tiende a un punto (atractor), un sistema inestable escapa de los atractores y uno caótico manifiesta ambos comportamientos.

Un sistema es caótico cuando reúne las siguientes propiedades:

- Ser sensible a las condiciones iníciales, de modo que al momento de alterar éstas tengamos un resultado distinto aunque esperado dentro de los limites y un resultado es el mismo cuando las condiciones iníciales son similares.

  Un ejemplo de ello es el llamado efecto mariposa en el cual un cambio muy pequeño como el aleteo de una mariposa puede desencadenar una serie de eventos dentro de la atmósfera de modo que se desencadenan en un tornado, de manera formal podemos definir esta propiedad de la siguiente manera. Se dice que una función $f: J \to J$ tiene dependencia de las condiciones iníciales si existe un $\delta > 0$ tal que para todo $x \epsilon J$ y cualquier vecindad de N de x existe un $y \epsilon N$ y un $n \geq 0$ tal que $|f^n(x) - f^n(y)| > \delta$. Donde $f^n(x)$ es igual a $f(f(\ldots f(x)))$ n veces.

- Existe transitividad topológica llamada también mezclado, implica que órbitas alejadas pueden llegar a aproximarse como consecuencia de la peculiar estructura de los atractores caóticos.

De manera formal se dice que una función $f: J \to J$ es topológicamente transitiva si para cualquier par de conjuntos abiertos $U \, y \, V \subset J$ existe un $k > 0$ tal que $f^k(U) \cap V \neq 0$.

- Sus órbitas periódicas deben formar un conjunto denso en una región compacta del espacio físico de modo que, para cualquier condición inicial de v_0 existe otra condición inicial z_0 que ésta a una distancia arbitraria cercana y además periódica.

El caos tiene tres propiedades generales las cuales exploraremos de manera rápida para tener una mayor comprensión de este tema.

A. Ubicuidad, se refiere a la presencia del caos en un número elevado de sistemas, propiedad que se puede observar también en sistemas con perturbación periódica. Lo anterior indica que encontramos caos en diversos lugares como la atmósfera, los embotellamientos de tránsito, los sistemas biológicos con comportamiento no lineal como ocurre el crecimiento de algunos ecosistemas y la reacción de membranas neuronales al ser perturbadas periódicamente por impulsos eléctricos.

B. Universalidad, nos indica que el comportamiento caótico corresponde a normas universales y perfectamente determinadas, de modo que sistemas diferentes se comportan de maneras idénticas en el punto de transición del orden al caos. La universalidad es cualitativa, cuantitativa, estructural, numérica, afecta la forma e implica aparición de nuevos parámetros universales. El caos es como una criatura dormida en la profundidad de un sistema ordenado pero que cuando el sistema alcanza un valor crítico el monstro dormido saca su filosa lengua.

C. “Estructura fractal, es consecuencia del infinito plegamiento sobre sí mismo del espacio de fases y se manifiesta tanto en los atractores extraños como en los de algunas barreras separatrices del dominio de atracción del caos”[1]. Esta interesante propiedad implica una estructura escalar de desdoblamiento, generación y autogeneración. Los fractales tienen la propiedad de autosimilitud lo que nos indica que existe una repetición de detalles en escalas descendentes; por lo tanto, un fractal presenta la misma apariencia independientemente del grado de ampliación con que se vea, las partes recuerdan, parecen o reproducen el objeto entero o partes del mismo.

La autorreferencia en la cual la forma de generar un fractal se realiza mediante algún tipo de algoritmo recurrente, normalmente determinista. La última propiedad de los fractales es la dimensión fraccionaria que representa el medio de ponderar cualidades que de otra manera carecerían de una definición clara como el grado de discontinuidad o irregularidad de un objeto.

La teoría del caos nos presenta la descripción de problemas que no pueden ser tratados por medio de algoritmos secuenciales, sino más bien están sujetos a formas distintas para encontrar soluciones, en este caso nos permite describir problemas que no son abordados por la teoría computabilidad de ni por la de la complejidad.

[1] CAOS, ORDEN Y DESORDEN, En el sistema monetario y financiero internacional, José Jesús Borjón Nieto,2007

Presenta problemas que pueden ser tratados con inteligencia artificial en la cual observamos cómo aquel se manifiesta cuando las conexiones entre los elementos cambian con el tiempo, a consecuencia de movimiento al azar o por otras causas en las que aun pequeños cambios, pueden alterar los resultados.

Es importante conocer la teoría del caos y de qué forma afecta no solamente sistemas orgánicos como el sistema nervioso que influye en la vida de las personas, sino también existe caos a nuestro alrededor y al crear inteligencia artificial como se pretende ya sea en sistemas individuales de redes neuronales o algoritmos genéticos, o bien en sistemas híbridos.

Un sistema inteligente debe poder procesar información de tal manera que existan cambios dentro de él que no afecten significativamente el comportamiento general, posiblemente cambiará el resultado pero el sistema continúa, algo que no se puede decir acerca de un sistema que sigue un algoritmo definido el cual, al encontrar una entrada inesperada puede detenerse por completo, además de reconocer que el mundo que nos rodea tiende al caos por el principio de entropía.

Un ejemplo lo encontramos en la predicción de variables económicas en las que aunque se pueda encontrar un punto de convergencia, pequeñas fluctuaciones cambian totalmente el resultado esperado además de que en cualquier momento, algunos indicadores varían inesperadamente, provocando una falla.

Por lo anterior, en sistemas de inteligencia artificial que buscan encontrar soluciones a problemas complejos los cuales en algunos casos no son computables, es importante tomar en cuenta los componentes caóticos.

1.4. Computabilidad, complejidad, caos y sistemas de inteligencia artificial

Se han abordado las distintas teorías de manera separada, analizando cada una de forma rápida y observamos cómo estas teorías dan un panorama completo de lo que busca la inteligencia artificial. En este caso la implementación y análisis de comparación entre un sistema híbrido inteligente y una red neuronal individual. Sabemos que tanto la computabilidad como la complejidad computacional son áreas fundamentales que sostienen la ciencia de la computación y nos indican qué problemas pueden ser resueltos por las máquinas actuales. Esto es importante ya que es el único medio de que disponemos al respecto y cualquier solución que se desee plantear, debe echar mano de este tipo de equipo.

El escollo fundamental se ve cuando surgen problemas para los cuales no se pueden plantear soluciones razonables (en tiempo y espacio) y acudimos a soluciones alternativas que deben poderse implementar en las máquinas actuales.

Para ello es imprescindible contar con un algoritmo que incluya complejidad logarítmica para poder ser implantado; no puede ignorarse que la mayoría de esos problemas tiene un carácter caótico lo que aumenta la probabilidad de dificultar de encontrar un algoritmo que sea capaz de dar solución exacta al problema, lo que indica que el mismo no es computable. Otros problemas pueden tener una algoritmo exacto pero su espacio de soluciones es demasiado amplio, de tipo factorial por ejemplo, y no pueden ser resueltos en tiempos adecuados por los computadores por lo que al momento de buscar soluciones alternativas, pueden generar caos.

El carácter fractal de los sistemas caóticos puede representar en una sección el todo, y al hablar de redes neuronales tomamos en cuenta que por el momento no conocemos con exactitud el comportamiento biológico de dichas redes, por eso se busca imitar con redes artificiales o tomar cierta sección del comportamiento, lo que puede suponer al poseer un comportamiento caótico, que se tiene representada buena parte del todo, en los sistemas artificiales.

Uno de los aspectos primordiales de este trabajo es realizar un análisis de comparación entre un sistema de red neuronal simple y uno híbrido inteligente, lo que reclama demostrar inicialmente que las propuestas de ambos sistemas son computables, de modo que existe un algoritmo capaz de ser implementado en los ordenadores actuales. Dicha solución puede ser generalizada a los problemas propuestos debiendo poseer una complejidad computacional de orden logarítmico capaz de ser tratada por un ordenador actual.

Por lo tanto, debe observarse cómo la hibridación de un sistema de inteligencia artificial puede o no mejorar los tiempos de respuesta o la complejidad total del sistema al momento de resolver un problema, tomando en cuenta el carácter caótico de los mismos y proponiendo aplicaciones que en la práctica aporten soluciones con aproximación cercana a las deseadas.

De este modo comprendemos que la teoría de la complejidad y computabilidad sostiene la posible implementación del sistema y en principio, nos indica un posible rasgo del problema que necesite “inteligencia” para poder ser resuelto, de modo que si un problema no es computable o tiene una complejidad perteneciente a NP-completo, es posible que necesite de aquella para solucionarlo.

Se puede agregar también que otro rasgo de un problema a ser resuelto con “inteligencia” es que debe poseer comportamiento caótico y no ser resuelto por un algoritmo determinista, podemos agregar que aun los mismos sistemas inteligentes que solucionan problemas difíciles, tienden a ser caóticos.

1.5. Autopoiesis

“Es la capacidad universal de todo sistema para producir estados propios bien diferenciados, que son en su estabilidad los que crear su estructura interna y que pueden enlazar operaciones propias del sistema mediante la auto-organización”[2].

Un sistema con cualidades de autopoiesis tiene la característica de estar abierto al entrono en cuanto a intercambio de material, pero cerrado en cuanto a sus operaciones propias, este tipo de sistemas puede tener lazos operativos entre sus mismas unidades operacionales, lo que nos indica una propiedad de auto-referencia.

El sistema autopoiético se autogenera de otra forma, no podría operar si sus elementos fluctuaran caóticamente. Consta de un complejo conjunto de operaciones realizadas en clausura operativa y autorreferenciadas que son percibidas de forma cognitiva, la interacción con el medio es asegurada a través del auto-contacto y de la auto-percepción, esto sucede por ejemplo en el cerebro percibe cambios ocasionados en distintos circuitos neuronales.

[2] De Máquinas y Seres Vivos: Una teoría sobre la organización biológica Varela, Francisco J.; & Maturana, Humberto R. (1973)

La clausura operacional indica que la misma es regulada exclusivamente por un código específico. Pero el sistema sigue abierto estructuralmente al entorno, en cuanto a mecanismos o sensores que permiten traducir impulsos externos a eventos internos, el sistema debe tener adaptación al entorno como condición primordial de su existencia.

Según lo anterior, observamos que una red neuronal de tipo ART tiende a ser un sistema autopoiético ya que cumple con las condiciones propuestas pues se autogenera, tiene clausura operacional y se comunica hacia afuera, además puede adaptarse a otro sistema externo. El sistema híbrido compuesto por la red ART y algoritmos genéticos, no pierde capacidad de autopoiesis debido a que mantiene sus características y aumenta la capacidad de autogeneración ya que los pesos se autogenerarán, a partir de pesos anteriores.

2. REDES NEURONALES

Las redes neuronales se clasifican en neurológicas y artificiales.

Las redes neuronales biológicas forman el sistema nervioso propiciando en el ser humano la capacidad de obtener información, procesarla y dar una respuesta, además tienen una característica importante que consiste en lograr aprender. De manera análoga, las redes neuronales artificiales tratan de imitar las funciones del modelo biológico adaptándolo, de tal forma que se han conseguido aproximaciones lejanas al funcionamiento real, pero que pueden ayudar a resolver problemas complejos, con el paso del tiempo se han convertido en un paradigma de programación.

Existen diferentes definiciones para redes neuronales artificiales, entre ellas podemos mencionar que son:

A. "Una nueva forma de computación inspirada en modelos biológicos"[3];

B. Un modelo matemático compuesto de un gran número de elementos procesales, organizados en niveles;

C. Un sistema de computación compuesto por un gran número de elementos simples (neuronas) éstos poseen interconexión que procesa información por medio de estados dinámicos, como respuesta a entradas externas;

[3] Las Redes Neuronales Artificiales, fundamentos teóricos y aplicaciones prácticas, Raquel flores, Miguel Fernández 2008

D. Redes interconectadas masivamente, en paralelo de objetos simples, por lo general adaptativos, con una organización jerárquica que interactúan con el mundo real de la misma forma que los sistemas biológicos.

2.1. Historia de las redes neuronales

Las redes neuronales artificiales nacen como intento por imitar la naturaleza y surgen con el desarrollo de la computación en el siglo XX, con el propósito de diseñar sistemas que imitan un comportamiento inteligente realizando tareas que hasta el momento eran exclusivas de un ser vivo, a continuación veremos algunos de los momentos más importantes en el desarrollo de las redes neuronales artificiales.

Fueron los pensadores de la antigüedad los primeros en tratar de explicar el funcionamiento del cerebro y la lógica de los pensamientos, en este apartado nos referimos directamente a Platón y Aristóteles en los años 427-347 a.C. En el año 100 a.C. Herón, de Egipto, es el primero en recopilar información sobre autómatas, construyendo un autómata hidráulico.

Descartes (1569-1650) se considera precursor filosófico de la cibernética de los años 40 ya que se compara su psicologismo con la noción de procesamiento de la información, trató de explicar la percepción afirmando que es un proceso que ocurre en el interior del organismo humano, otros filósofos empiristas del siglo XVII ayudaron también a sentar las bases de la inteligencia artificial y redes neuronales.

Alan Turing (1912 - 1954) fue de los primeros en analizar el cerebro y compararlo con la computación, en 1950 publica el libro *Computing Machinery and Inteligence* con el cual se puede decir que nace la disciplina de la Inteligencia Artificial.

Para el año 1943 Warren McCulloch, neurofisiólogo, y Walter Pitts, matemático, definen formalmente la neurona como una máquina binaria de varias entradas y una salida, siendo éste el primer modelo neuronal moderno.

Donald Hebb en el año 1949, escribió su libro La organización del comportamiento, en él define dos conceptos muy importantes que han sido pilares fundamentales en el campo del las redes neuronales:

A. El aprendizaje se localiza en las conexiones de las neuronas (sinapsis);

B. La información es representada en el cerebro por un conjunto de neuronas activas o inactivas.

Karl S. Lashley (1890-1958) en sus investigaciones concluye que la información en el cerebro es guardada de forma distribuida y que éste puede reconfigurarse de tal manera que si una parte resulta dañada, otras pueden ocuparla.

En 1956 nace el término inteligencia artificial, atribuido a John McCarthy, Marvin Minsky y Claude Shannon en la Conferencia de Darmouth, en este congreso debido al gran entusiasmo generado, se hacen previsiones que jamás se cumplieron, decayendo el estudio de las redes neuronales.

Durante 1959 surge la primera aplicación de redes neuronales en sistemas reales con el nacimiento de la Teoría de la Adaptación Neuronal, el *Adaline -Adaptative Linear Neuron-* y el *Madaline* -Múltiple *adaline*- que se utilizó para eliminar el eco en las líneas telefónicas.

Frank Rosenblatt entre 1957 y 1962 desarrolla el perceptrón que es la red neuronal con mayor antigüedad, misma que fue utilizada para reconocimiento de patrones además de poder generalizar. Presentaba diferentes problemas, entre ellos que no podía clasificar objetos no separables linealmente y que el aprendizaje del preceptrón convergía hacia un estado finito, esto provocó declive en la investigación.

Para 1967 Stephen Grossberg estructura la red Avalancha basándose en sus conocimientos de neurofisiología, esta red resolvía actividades tales como reconocimiento del habla y aprendizaje de brazos de un robot. En el año 1977 Grossber hace un aporte importante ya que desarrolla la arquitectura ART, Teoría de Resonancia Adaptativa, la cual difiere de las otras previamente inventadas, esta red es capaz de desarrollar memoria a largo y corto plazo.

En el año de 1969 el estudio de las redes neuronales estaba desvirtuado y aparece un libro que muchos pensaron seria el final de las redes neuronales. El libro se titulaba "*Perceptrons: An introducction to computational Geometry*" el cual había sido escrito por Marvin Minsky y Seymour Paper. En este libro daba un análisis detallado del perceptrón tanto como sus capacidades y limitaciones.

En el año 1969 el estudio de las redes neuronales estaba desvirtuado, con el aparecimiento de un libro que para muchos representó el final de las redes neuronales.

Se titula *Perceptrons: An introducction to computational Geometry* escrito por Marvin Minsky y Seymour Paper. En él se hizo un análisis detallado tanto del perceptrón como de sus capacidades y limitaciones.

James Anderson en 1977 desarrolla el Asociador Lineal, un modelo basado en el principio de elementos integrales lineales formados por neuronas que sumaban sus entradas, este modelo se cimenta en que las conexiones entre neuronas son reforzadas cada vez que se activan, una mejora de esta red es *Brain State in Box –BSB-.*

En 1974 Paul Werbos desarrolla el algoritmo de aprendizaje de propagación hacia atrás, *Backpropagation,* estudio que concluyó hasta 1985 aclarando por completo su significado. Esta red es un perceptrón multicapa con diferentes funciones de activación en las neuronas artificiales y con una regla de aprendizaje más confiable y segura. Actualmente *backpropagation* es la arquitectura de red neuronal más utilizada y reconocida en aplicaciones.

Durante 1980 el profesor Kunihiko Fukushima propone el modelo *neocognitron,* una red jerárquica multicapa que puede reconocer patrones visuales.

El responsable del renacimiento de las redes neuronales es John Hopfield, físico estadounidense, al publicar el libro Computación neuronal de decisiones en problemas de optimización. En esta publicación Hopfield proponía nuevos sistemas de procesamiento en paralelo, que superaban las limitaciones del perceptrón.

En 1986 Hinton y Sejnowski diseñan la máquina de Boltzman que contiene redes adaptativas dotadas de unidades ocultas, capaces de encontrar una configuración estable para unidades activas o inactivas, utilizando el procedimiento de Montecarlo. Cabe resaltar que esta neurona son verdaderas herramientas análogas y no se limitan a respuestas digitales, todo o nada. Para el año 1987 Kosko desarrolla la red neuronal Memoria Asociativa Bidireccional –BAM-, se diferencia de la red de Hopfield en que en ésta dos objetos o patrones distintos pueden ser almacenados o asociados entre sí, por medio de memoria hatero-asociativa. De esta manera se puede simular la habilidad humana de establecer asociaciones mentales.

Actualmente son muchos los trabajos que se publican sobre redes neuronales y cada vez más son los avances en este campo alrededor del mundo, de tal modo que las redes neuronales han alcanzado una etapa de madurez aceptable y son utilizadas en diferentes aplicaciones como en sistemas de reconocimiento de voz y de imágenes, de apoyo a la medicina, análisis de datos y otras.

2.2. La neurona biológica

Es importante en este punto hacer un acercamiento a la neurona biológica y describir a grandes rasgos su comportamiento ya que muchos de los modelos neuronales hacen referencia a ciertas características que posee esta, aun que en ningún caso se ha logrado acercarse a un comportamiento exacto, sino mas bien todos han sido aproximaciones que logran imitar solo algunas características.

La neurona biológica está integrada por muchos componentes de los cuales haremos referencia a un número reducido que parece importante y aplicable al tema de redes neuronales.

La neurona típica perteneciente al sistema nervioso central, pose una membrana la cual es permeable iónicamente lo cual mantiene una diferencia de potencial entre el fluido que se encuentra en el interior y exterior de la neurona de hecho por diferentes procesos que se tienen dentro de la neurona, se produce una diferencia de potencial 70 a 100 (mV) siendo el más negativo el fluido intracelular a este potencial se le denomina potencial de reposo de la célula lo cual produce la energía necesaria para que la neurona realice sus diferentes procesos.

Como se puede observar en la Figura 2, la neurona posee varias conexiones denominadas dendritas que reciben estímulos provenientes de un axón de otra neurona, estas dendritas poseen quimiorreceptores los cuales son capaces de reaccionar, gracias a los neurotransmisores enviados por la vesícula sináptica.

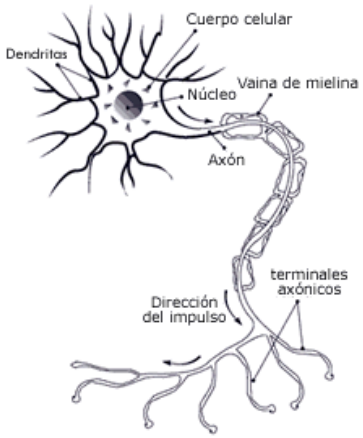
Observamos también el axón, prolongación de la célula en forma de hilo, por medio del cual viaja el impulso nervioso de forma unidireccional y establece conexión con otra neurona por medio de conexiones terminales.

El axón se encuentra recubierto por una vaina de mielina, ésta tiene un efecto aislante que permite la transmisión de impulsos nerviosos a distintas partes del cuerpo; la vaina de mielina es interrumpida, en algunos puntos, por los nodos de Ranvier de aproximadamente un micrómetro de longitud tienen como función trasladar el impulso nervioso con mayor rapidez, dando saltos y con menores posibilidades de error ya que en sí, las fibras nerviosas son malos

conductores y la transmisión se efectúa de manera secuencial, cuando un nodo se despolariza estimula la misma acción en el siguiente nodo y así sucesivamente.

Una vez que el potencial de acción ha pasado por cierto punto, éste no puede ser excitado por un milisegundo, tiempo que tarda en volver a su potencial de reposo lo que limita la frecuencia de transmisión a unos 1000mts/s por segundo.

Figura 2. **Neurona biológica**



Fuente: http://drateresita.blogspot.com/2010/05/guia-para-los-examenes-finales.html

2.2.1. La sinapsis

Es el proceso de comunicación entre neuronas, ocurre como resultado de la liberación de sustancias llamadas neurotransmisoras por parte de la célula presináptica la cual se deposita en un espacio intermedio, espacio sináptico, entre la neurona transmisora y la receptora o postsináptica.

El neurotransmisor puede tener diferentes efectos entre los cuales está la excitación, misma que incrementa la posibilidad de producir un potencial de acción, la inhibición que reduce la posibilidad de producirlo y la modulación la cual cambia el patrón y/o frecuencia de la actividad producida por las células involucradas.

2.3. Analogía neurona artificial y biológica

Fueron McCulloch y Pitts los primeros en tratar al cerebro como un organismo computacional en 1943, al formular su teoría basada en cinco suposiciones que son:

A. La actividad de una neurona es un proceso todo-nada (digital);

B. Es preciso que un número fijo de sinapsis (>1) sean excitadas dentro de un período de adición latente para que se excite una neurona;

C. El único retraso significativo dentro del sistema nervioso es el retardo sináptico;

D. La actividad de cualquier sinapsis inhibitoria, impide por completo la excitación de la neurona en este momento;

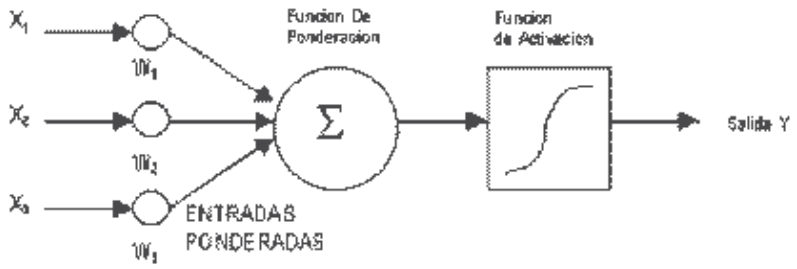
E. La estructura de la red de interconexiones no cambia con el transcurso del tiempo[4].

[4] www.tesis.ufm.edu.gt/pdf/3828.pdf

Con esta teoría establecían un primer acercamiento para modelar el comportamiento del sistema nervioso humano y explicar cómo conceptos relativamente sencillos, dan al cerebro grandes capacidades y aunque el modelo no es preciso, ayudó al desarrollo de las redes neuronales.

Podemos decir entonces que la neurona consta de una serie de entradas Xi equivalentes a las dendritas, éstas son estimuladas por pesos Wi los que representan la forma en que los impulsos entrantes son evaluados por la función de ponderación dando el nivel de potencia de la neurona, éste a su vez será evaluado por la función de activación que da salida a la unidad de proceso como se muestra en la Figura 3.

Figura 3. **Modelo de neurona artificial *McCulloch* y *Pitts***



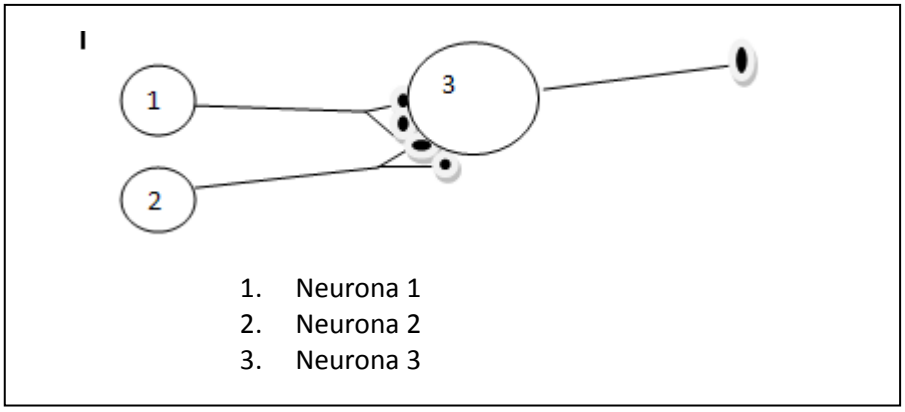
Fuente: http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node12.html

Los valores Xi pueden ser enteros, reales o binarios (McCulloch y Pitts), equivalen a señales que envían otras neuronas a través de las dendritas. Los Wi equivalen al mensaje enviado entre neuronas, es el proceso químico que da lugar a la inhibición o excitación en la sinapsis, e induce a la neurona a cambiar su comportamiento.

La función de ponderación se encarga de agrupar todas las señales provenientes de las dendritas (Entradas Xi) convirtiéndose en una sola que suele ser el total de las entradas y pesos sinápticos. La de activación transforma la salida de la función de ponderación a un dominio en el que trabajan las salidas, ésta esta es única y puede llegar a diferentes neuronas.

Un ejemplo respecto al funcionamiento de las neuronas, basado en el modelo de MacCulloch y Pitts, es el siguiente: sabiendo que las neuronas en este modelo son binarias, están activas o no activas, podemos definir la siguiente notación Ni(t) en la cual la i-esima neurona hace un disparo en el momento t por lo tanto ¬Ni(t) significa que la neurona no ha disparado en ese instante. Utilizando la lógica proposicional podemos describir expresiones proposicionales sencillas como la disyunción N3(t)=N1(t-1) v N2(t-1) .

Figura 4**. Comportamientos de una neurona McCulloch y Pitts**



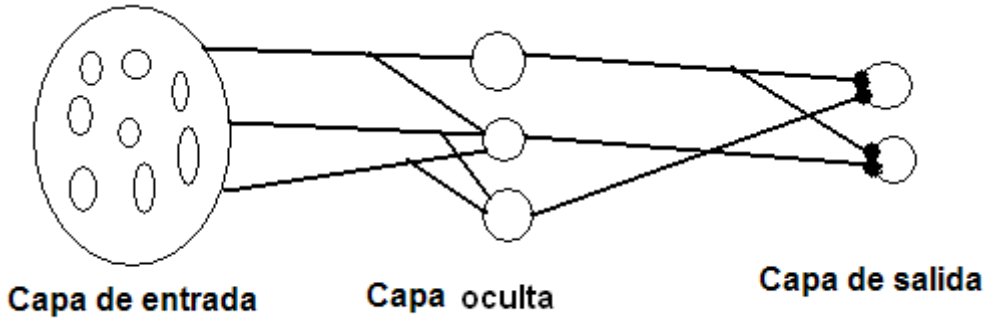
Fuente: elaboración propia.

2.4. Estructura de redes neuronales artificiales

A simple vista el funcionamiento de una neurona parece sencillo, al tener unidas un conjunto de ellas se logra gran poder de procesamiento; de este punto en adelante, a las neuronas artificiales les llamará unidades de procesamiento, no se tomarán como modelos reales del sistema nervioso humano sino como un conjunto de elementos computacionales, considerando este punto se verá la estructura de las redes neuronales.

Una unidad de procesamiento por sí sola no posee capacidad de procesamiento, para obtener provecho de un sistema como éste necesitamos agruparlas en capas; encontramos tres tipos de capas, las de entrada, reciben información del medio, las de salida envían información hacia el medio y las capas ocultas son responsables de procesar información y comunicarla a las dos primeras. La mayoría de autores no recomiendan procesar información desde las capas de entrada o salida, pero en caso necesario se puede hacer.

Figura 5. **Entrada procesamiento y salida de red neuronal**



Fuente: elaboración propia.

Existen diferentes formas de conectar las capas, entre ellas se mencionan:

A. Unión todos con todos. Busca fusionar las unidades de procesamiento de una capa con las unidades de otra capa, este tipo de unión es utilizada por la red de Hopfield.

B. Unión lineal. Adhiere la unidad de procesamiento de una capa con otra de distinta capa; este método es menos usado que el anterior, se utiliza en redes de aprendizaje competitivo.

C. Unión predeterminada. Surge con las redes que son capaces de agregar o quitar neuronas o conexiones, pudiendo de esta manera, conectarse con una o más neuronas de una capa distinta o de la misma.

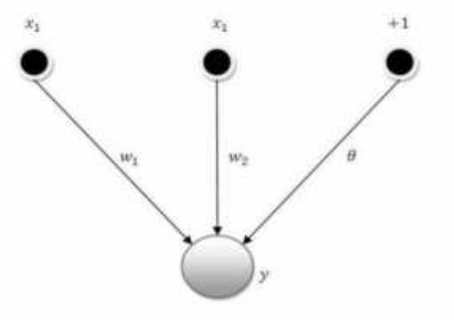
2.5. Clasificación de redes neuronales artificiales

2.5.1. Por su topología

Una red neuronal puede clasificarse por su topología o estructura, distinguiendo características como el número y tipo de capas, si éstas son visibles u ocultas, de entrada o salida y su dirección de propagación. Pueden mencionarse las siguientes:

A. Redes *Feed-Forward*. En este tipo de redes los datos fluyen desde la entrada a las unidades de salida, un ejemplo claro de este tipo de redes son el Perceptrón y el Adaline.

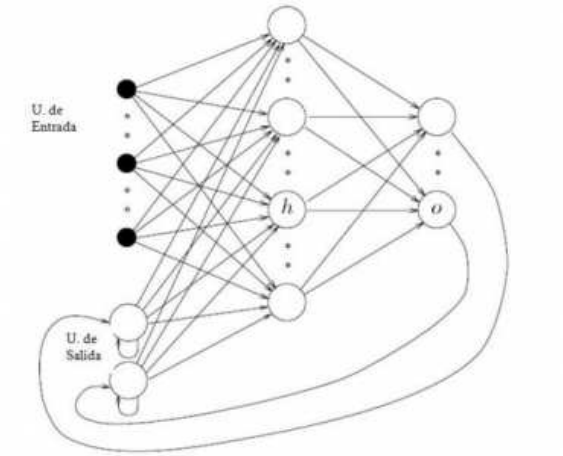
Figura 6. **Red *feed-forward***



Fuente: http://advancedtech.wordpress.com/2007/08/31/topologias-de-redes-neuronales.

B. Redes Recurrentes. Son capaces de evolucionar ya que poseen conexiones de retroalimentación las cuales proporcionan un comportamiento dinámico, entre ellas se ubica la red de Hopfield.

Figura 7. **Red recurrente**



Fuente: http://advancedtech.wordpress.com/2007/08/31/topologias-de-redes-neuronales.

2.5.2. Por su forma de aprendizaje

Las redes neuronales tienen como fin tratar problemas que reclaman cierto grado de análisis, para ello necesitan manejar datos que son memoria volátil los cuales fluyen por la red y en muchas ocasiones no es necesario almacenar. Por otro lado la red necesita almacenar de la mejor forma dichos datos o de procesarlos, para ello utiliza los distintos pesos sinápticos y reglas que los modifican, de modo que puedan converger o no hacia soluciones óptimas.

2.5.2.1. Aprendizaje supervisado

En este tipo de aprendizaje se muestran a la red los patrones y la salida deseada, pudiendo incluir una fórmula matemática con la cual se minimice, para ello es necesario un conjunto de datos de entrada cuya respuesta se conoce.

De esa manera los pesos sinápticos pueden converger hacia un estado en el que la red genera los resultados esperados cuando se le presente un patrón de entrada similar a los del entrenamiento. Ejemplos claros son *Adaline*, el perceptrón multicapa y la memoria asociativa, entre otros.

2.5.2.2. Aprendizaje no supervisado

Este tipo de aprendizaje no necesita que se le den las salidas específicas, las clasifica según su similitud, de acuerdo al grado de parecido que se desee dar a las mismas. De modo que la red va midiendo el grado de parecido que tiene el patrón que llega y compara con los que tiene almacenado, categorizando dichos patrones en distintos grupos, de acuerdo a su algoritmo de aprendizaje.

Del aprendizaje no supervisado se puede desprender el aprendizaje por componentes principales, en éste la red trata de agrupar los comunes en los diferentes patrones, para lograrlo algunas unidades de procesamiento cooperan en la representación del patrón de entrada.

En el aprendizaje competitivo las unidades de procesamiento compiten entre sí, para representar de la mejor manera un patrón de entrada. Este aprendizaje consiste en reforzar las conexiones de las unidades ganadoras y debilitar las otras, de modo que la red tienda hacia las conexiones ganadoras.

El aprendizaje reforzado es parecido al no supervisado pero en éste no se muestra el set completo de datos de salida sino únicamente se le indica a la red si son correctos o no, se basa en la premisa de premiar los resultados correctos y castigar los incorrectos.

Ejemplos de aprendizaje no supervisado tenemos en la red de Hopfield y la máquina de Bolzman, entre otras.

2.6. Tipos principales de redes neuronales

Esta sección incluye ejemplos y un modelo básico de funcionamiento de algunas redes que se consideran importantes en el estudio de las redes neuronales, para mayor comprensión de su funcionamiento.

2.6.1. Perceptrón simple

Es uno de los primeros intentos de modelar sistemas neuronales, fue desarrollado a finales de la década de los 50, se basa en reglas de aprendizaje de Hebb y los trabajos de McCulloch y Pitts, es importante mencionar que el perceptrón tiene distintas limitaciones que en algún momento influyeron para que se detuviera el avance en la investigación respetiva, aunque sigue como modelo pionero en este campo.

El perceptrón funciona de manera que los datos fluyen de la capa de entrada a la de salida, teniendo una capa de sensores que reciben los patrones a reconocer y clasificar; además de una unidad de procesamiento que se ocupa de clasificar las salidas en activo o no activo (1 o 0), respectivamente.

El perceptrón simple tiene la limitación que únicamente puede representar funciones linealmente separables; aprende de manera supervisada, por lo que es importante tener los valores de salida de ciertas tramas de datos al momento de entrenarlo, las tramas esperadas se comparan con las obtenidas y se divide el error entre los pesos, cambiándolos para conseguir salidas parecidas a las esperadas.

El algoritmo de aprendizaje del perceptrón se representa de la siguiente manera.

1. Inicializar los pesos sinápticos con números aleatorios en el intervalo [1,-1];
2. Calcular la k-esima iteración;

$$y(k) = \operatorname{sgn}\left(\sum_{j=1}^{n+1} w_j x_j(k) \right)$$

3. Corregir los pesos sinápticos;
4. Si no han cambiado los pesos sinápticos en las últimas iteraciones parar; (la red se ha estabilizado), de lo contrario volver al paso 2;

2.6.2. Perceptrón multicapa

Como se mencionó anteriormente el perceptrón tiene una gran limitante, sólo puede tratar funciones linealmente separables, por lo tanto no logra representar funciones como XOR. Lo que se resolvió añadiendo capas ocultas. Eso permite establecer regiones de decisión más complejas que las de dos semiplanos lo cual hace el perceptrón simple, con ello también se logra que el perceptrón pueda admitir valores reales. Por lo anterior, se asegura que el perceptrón multicapa es un modelador universal de funciones.

Entre las limitaciones más importantes del perceptrón multicapa se menciona que si la red no se entrena lo suficiente o se hace de manera equivocada, las salidas no serán las esperadas.

La existencia de mínimos locales en la función de error dificulta su entrenamiento ya que una vez alcanzado el mínimo local, el entrenamiento se detiene aunque no se haya logrado la convergencia esperada.

El perceptrón multicapa es utilizado en problemas de asociación de patrones, segmentación de imágenes y compresión de datos; cabe mencionar que un modelo como este no se escogió para ser parte del sistema híbrido por sus limitaciones y su forma de aprendizaje ya que como vemos, en algunos momentos puede no converger de manera satisfactoria y necesitamos un modelo de tipo de aprendizaje no supervisado.

2.6.3. Red de Hopfield

Otro de los modelos pioneros en redes neuronales es la red de Hopfield, es de tipo unicapa y ha influido en el desarrollo de redes posteriores, es no lineal y autoasociativa (aprende a reconstruir los patrones de entrada que memorizó durante su entrenamiento); igual que la anterior está basada en modelos de McCulloch y Pitt.

El principal aporte de Hopfield es conseguir que la red sea recurrente y al mismo tiempo estable, de manera que puede almacenar información y recuperarla aunque esté deteriorada, por su naturaleza autoasociativa.

Entre las características más importantes destaca que tiene conexiones bipolares con pesos simétricos. Todas las unidades son de entrada y salida. Puede tomar valores bipolares [-1,1] y la función de activación es su signo. Cada unidad de procesamiento tiene un estado interno u_i denotado v_i y un estado externo.

La actualización y relación entre dichos valores está dada por las siguientes ecuaciones:

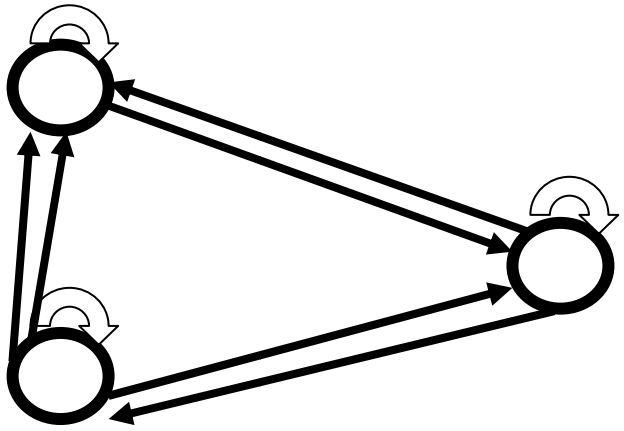
$$u_i(t+1) = \sum_{j=1}^{n} W_{i,j} v_j(t) + I_i$$

$$v_i(t+1) = f(u_i) = \begin{cases} 1 & \text{si } u_i > 0 \\ 0 & \text{si } u_i \leq 0 \end{cases}$$

Donde es una constante externa de entrada a la unidad de procesamiento i y f() es la función de transferencia entre los estados internos y externos.

El trabajo de esta red se da por medio de funciones de energía, a cada estado se le atribuye cierta cantidad de energía, el sistema evoluciona tratando de disminuirla por medio de procesos de relajación hasta alcanzar los mínimos. En la figura 8 se muestra el funcionamiento de una red de Hopfield de tres unidades de procesamiento.

Figura 8. **Red de Hopfield**



Fuente: elaboración propia.

Utiliza el aprendizaje de Hebb asociando los pesos de las sinapsis al estado de los elementos de procesamiento pre y pos sináptica. El algoritmo de aprendizaje para la red de Hopfield estaría dado de la siguiente forma.

1. Para cada patrón de la lista X;
2. Se establece el patrón de entrada de X en los sensores, la capa de entrada;
3. Se hace que las neuronas de la capa de salida se actualicen sus estados a los valores de la capa de entrada;
4. Solicitar que aprendan todas las sinapsis usando las sinapsis laterales;
5. Hacer los pesos de las sinapsis nulos.

2.6.4. Mapas de Kohonen

A partir de un proceso iterativo de comparación de un conjunto de datos y cambios para aproximarse a los mismos, crean un modelo con ellos que ayuda a agruparlos por criterios de similitud.

Son redes con aprendizaje no supervisado; por lo general cuentan con dos capas, una capa de sensores y otra de salida la que realiza los cálculos necesarios.

Un mapa de Kohonen está formado por un conjunto de vectores n-dimensionales distribuidos en diferentes dimensiones, habitualmente dos para

cada vector; se define un vecindario, cada vector puede tener seis u ocho vecinos dependiendo de la retícula rectangular o hexagonal, respectivamente.

El aprendizaje de los mapas de Kohonen se efectúa de forma no supervisada en una subcategoría llamada por competencia, en ella se entrena una unidad de procesamiento a la vez y en las que son adyacentes, a diferencia del perceptrón multicapa, cada zona del espacio de entrada está concentrada y no distribuida. Un algoritmo para entrenar este tipo de redes puede ser el siguiente:

1. Se inicializan pesos en forma aleatoria;
2. Se presenta un vector de entrenamiento;
3. La neurona más cercana al vector es la única que se activa;
4. Solo se modifican.

Existen distintos tipos de redes neuronales y todas son herramientas poderosas que permiten tener en cuenta un nuevo paradigma en cuanto al procesamiento de datos, de modo que se puedan automatizar procesos que con técnicas clásicas no se lograrían.

Gracias a estas herramientas puede emularse de alguna manera el comportamiento del sistema de razonamiento humano y obtener grandes resultados en procesamiento de datos, por la velocidad de los computadores que superan con creces a los sistemas biológicos.

Aquí simplemente se colocaron algunas generalidades en cuanto a las redes neuronales de modo que pueda servir como introducción a los siguientes temas y la implementación del sistema hibrido de inteligencia artificial.

3. REDES NEURONALES TIPO ART

En 1986 Carpenter y Grossberg proponen el modelo conocido como Red de Resonancia Adaptativa –ART- (*Adaptive Resonance Theory*). Este modelo trata de imitar una característica particular de la memoria humana, accede a nuevos aprendizajes sin olvidar lo aprendido previamente. Por ejemplo, asimilar un tipo distinto de operación matemática como la multiplicación, no necesitamos aprender de nuevo operaciones de suma o resta, ésta es una de las principales diferencias que existe entre este modelo y otros estudiados con anterioridad. Otro tipo de modelo necesita ser reentrenado totalmente al momento de captar una nueva funcionalidad. A esto, los autores le denominaron el problema de elasticidad y plasticidad que trata de resolver este modelo.

3.1. Teoría de la resonancia adaptativa

El modelo ART trata de resolver el problema de la plasticidad que se refiere a la capacidad de aprender nuevos patrones y la elasticidad hacer que la red retenga los aprendidos previamente.

Un ejemplo es crear un modelo de red que ayude a clasificar distintos tipos de imágenes con figuras geométricas de tal manera que se pueda enseñar al modelo, a través de entrenamiento, a reconocer un triángulo y un cuadrado y si se deseara que aprendiese a reconocer un pentágono, simplemente se le entrenará con la figura en cuestión, obteniendo como resultado la identificación tanto de las figuras aprendidas previamente como la última.

Esto no sucede con otro tipo de modelos los cuales requieren que se entrene la red de nuevo, enseñándole tanto las figuras previas como las nuevas, o se obtendrían resultados inesperados.

Trata de resolver el problema añadiendo una capa de retroalimentación entre las unidades de procesamiento de la capa de salida y las de la capa de entrada, ayudando de este modo a aprender nueva información, sin destruir la almacenada.

La ART recibe su nombre gracias a la interacción que sucede entre el aprendizaje y el recuerdo dentro de la red. En física la resonancia se produce cuando una vibración de pequeña amplitud y con una frecuencia adecuada, da lugar a oscilaciones ya sea en sistemas eléctricos o mecánicos; de manera análoga en una red neuronal ART la información oscila entre capas, fluyendo de arriba abajo y de abajo hacia arriba, ambas señales se refuerzan con la realimentación.

Se puede alcanzar un estado resonante de dos formas, si la red ha aprendido a reconocer un vector de entrada o nuevos datos y al no encontrarlos los almacenara por primera vez. De esta manera la red puede responder a datos aprendidos anteriormente y a nuevos datos.

Hay que hacer notar que existen dos tipos de memoria, la memoria a corto plazo que se encuentra dada por la actividad de la información entre neuronas y la memoria a largo plazo que se modifica por medio de los pesos sinápticos.

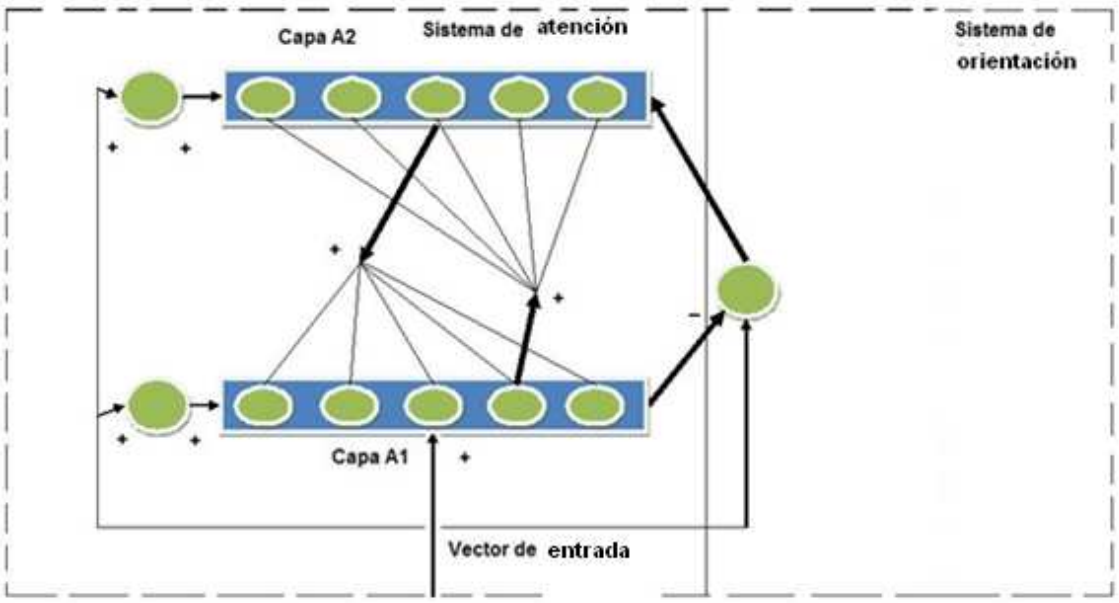
3.2. Características principales de redes ART

Una característica importante del modelo es que se basa en modelos competitivos en los cuales mediante una información de entrada, solamente una de las unidades de procesamiento de la red se activa, se le denomina neurona vencedora, alcanzando un valor de respuesta máximo al competir.

Una o más “neuronas reales” pueden representar una unidad de procesamiento bajo este modelo. En la figura 2.1 se muestran las características principales de una red ART, en ella observamos un sistema de atención en el cual se encuentran dos capas representantes de memoria a corto plazo ya que existen por asociación a una sola aplicación del vector de entrada, los pesos asociados ascendentes y descendentes entre A1 y A2 conforman la memoria a largo plazo, codifican información que forma parte de la red a lo largo del tiempo.

Los nodos de ambas capas están totalmente interconectados con los nodos de la siguiente capa, en la figura 9 observamos que existen símbolos + éstos denotan una conexión excitatoria y los símbolos - una inhibitoria.

Figura 9. **Funcionamiento de red tipo ART**



Fuente: elaboración propia.

Existen dos tipos de redes ART básicas estas son ART1 y ART2, su funcionamiento esencial es similar, las mayores diferencias se encuentran en que la ART1 admite únicamente entradas binarias y deben ser elementos del conjunto {1,0}. Las ART2 además de aceptar valores binarios acepta valores analógicos en los vectores de entada y presenta diferencias de arquitectura compleja.

3.3. Historia y evolución de redes ART

En las últimas dos décadas las redes neuronales han sido redefinidas y utilizadas ampliamente por investigadores e ingenieros, alrededor del mundo, una de las más utilizadas y que ha evolucionado de mejor manera es la red ART, su capacidad de manejar plasticidad y elasticidad han dado a este modelo la virtud de aprender autónomamente en ambientes reales. Han derivado de esta red diferentes modelos como los que se describen a continuación.

3.3.1. *Fuzzy* ART

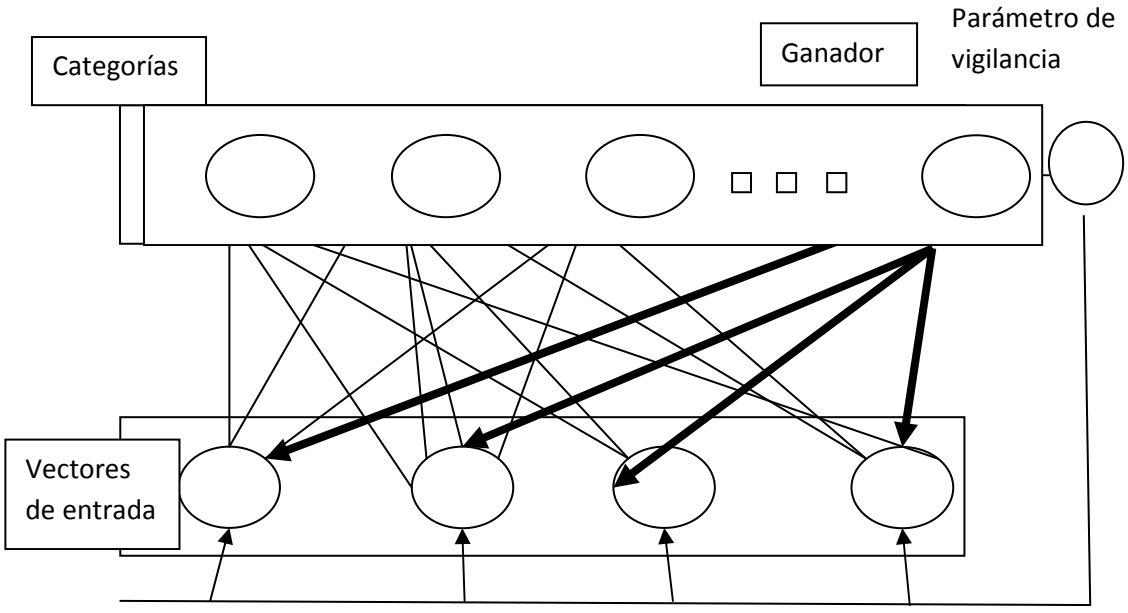
Este tipo de redes utiliza lógica difusa para el reconocimiento de patrones de la red ART. La lógica difusa ha sido empleada exitosamente en muchas aplicaciones de control de procesos automáticos lo que hace pensar que incorporar distintos procesos de lógica difusa dentro de una red ART, provee diferentes ventajas como la disminución de costo y tiempo de procesamiento y la habilidad de reconocer escalas de grises o entradas analógicas.

Tabla I. **Características de redes *Fuzzy* ART**

Elige la categoría ganadora como el mayor valor en la función de elección
Juzga la categoría de resonancia o reinicia por medio de la función de coincidencia
La resonancia ocurre si el valor de coincidencia es mayor a la función de vigilancia. A continuación los pesos de la categoría ganadora son actualizados de acuerdo a los vectores de entrada. De otra manera se reinicia. Entonces una nueva categoría es generada de acuerdo al vector de entrada
Continúa el ciclo anterior para cada vector de entrada

Fuente: elaboración propia.

Figura 10. **Comportamiento de *Fuzzy* ART**



Fuente: elaboración propia.

Tabla II. **Ventajas y limitaciones de *Fuzzy* ART**

Ventajas	**Limitaciones**
Aprendizaje adaptativo	Se basa únicamente en una similitud y un umbral
Crear nuevas clases	Es válida para problemas no supervisados
Actualiza clases aprendidas	Necesita parámetros de vigilancia
Aprendizaje estable predecible matemáticamente	Puede crear muchas categorías
No necesita reentrenamiento	Depende del orden de presentación de los patrones

Fuente: elaboración propia.

3.3.2. ART-EMAP

Adaptive Resonance Theory with spatial and temporal evidence integration for dynamic predictive mapping (ART-EMAP) es un tipo de red neuronal que acumula evidencia temporal y espacial para reconocer objetivos y clases de patrones en ambientes ruidosos o ambiguos, esta red integra evidencia espacial distribuida a lo largo de diferentes categorías que se desean clasificar.

Cuando una decisión de criterio determina que el patrón es ambiguo, una entrada del mismo tipo desconocido, es vista. Entonces una serie de evidencias de múltiples entradas desarrollan un criterio de decisión satisfactorio, ART-EMAP mejorando la exactitud y ampliación del dominio de una *Fuzzy* ARTMAP.

Tabla III. **Características de red ART-EMAP**

Si ninguna clase cumple con el criterio de decisión se deja la salida indecisa y se piden más patrones
Incorpora acumulación de evidencias espaciales
Contiene tres capas
Emplea activación distribuida
Aumenta la efectividad y dominio de un Fuzzy ART

Fuente: Elaboración propia.

Tabla IV. **Otras versiones de redes ART**

ARTMAP-IC
LAMPART
ARAM
CASCADE ARTMAP
HART
GAUSSIAN ARTMAP
ELLIPSOID ARTMAP
HIPERSPHERE ARTMAP
PROBART
TD-ART

Fuente: elaboración propia.

Tabla V. **Aplicaciones de redes ART**

Reconocimiento visual de objetos
Reconocimiento de imágenes y texturas
Procesamiento ECG
Clasificación de patrones médicos
Control borroso y predicción económica

Fuente: elaboración propia.

4. COMPUTACIÓN EVOLUTIVA

Es una técnica surgida como analogía a organismos biológicos y su base genético molecular, para ordenar por medio de un algoritmo (serie de pasos organizados que describen los procesos a seguir para solucionar un problema específico). Esta técnica nació en la década de los 70's de la mano de John Henry Holland, en términos generales los algoritmos toman una población inicial haciéndola evolucionar de manera aleatoria, conservando los especímenes más fuertes de manera similar a la evolución biológica.

Una definición de algoritmo genético es propuesta por John Koza: "Es un algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos individuales con respecto al tiempo, usando operaciones modeladas de acuerdo al principio darwiniano de reproducción y supervivencia del más apto, y tras haberse presentado de forma natural una serie de operaciones genéticas entre las que destaca, la recombinación sexual.

Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud fija que se ajusta al modelo de las cadenas de cromosomas, y se les asocia con cierta función matemática que refleja su aptitud.

Darwin postuló algunos conceptos importantes como la selección natural y la supervivencia de los más fuertes, al observar el comportamiento y desarrollo de diferentes especies a lo largo del tiempo y cómo generación con generación dichos organismo cambian, estos organismos compiten por alimento, agua o refugio incluso ante otros de la misma especie, los que tienen mayor probabilidad de generar más descendientes también tienen mayor probabilidad de transmitir sus genes.

Por el contrario, los individuos “débiles” producirán menor descendencia. Al tomar este concepto e imitarlo, podemos llegar a resolver distintos problemas del mundo real en poblaciones del sistema al cual queremos dar solución, asignándole un valor a cada población, dejando los mejores especímenes y desechando a los débiles de modo que los nuevos se acerquen a una solución óptima del problema.

Debemos tomar en cuenta que implementar una solución de este tipo implica aplicar “fuerza bruta” de manera que para cada elemento de población, sus respectivos genes tendrán una mutación aleatoria que debe ser revisada y analizada por una función clasificatoria de acuerdo a su nivel de efectividad, entonces se tendrán que probar distintas soluciones y de manea automatizada, comprobar su efectividad para aceptarla o no hasta que los individuos se acerquen a soluciones óptimas.

4.1. Ventajas de la computación evolutiva

Dentro de las ventajas más importantes que representa la utilización de algoritmos genéticos se encuentra el no tener que conocer detalles sobre el problema a resolver, así un informático puede solucionar problemas de química, física u otra ciencia conociendo únicamente los genes o partes específicas de él.

Las técnicas de computación tradicionales, funcionan de manera secuencial analizando una solución a la vez, mientras que los algoritmos genéticos presentan la ventaja de trabajar con distintas soluciones que convergen en una solución optima.

Un problema de las soluciones tradicionales es que éstas tienden a perder pertinencia a lo largo del tiempo y por lo general tiene que rehacerse por completo el sistema para encontrar una nueva solución mientras los algoritmos genéticos pueden evolucionar de manera que cambian con el tiempo.

Existen en muchas funciones o problemas que contienen máximos o mínimos locales los cuales pueden provocar falsas soluciones con métodos tradicionales, al utilizar la computación evolutiva podemos encontrar una que se acerque de mejor manera a la solución óptima.

Utilizan operadores probabilísticos en lugar de operadores determinísticos con lo que se obtiene mayor flexibilidad y por lo tanto, más probabilidad de encontrar la solución.

4.2. Desventajas y limitaciones de la computación evolutiva

Dependiendo de los parámetros que se utilicen para evaluar el sistema pueden tardar mucho en converger o no pueden hacerlo. Llegando a converger prematuramente al encontrar problemas dentro de los parámetros establecidos.

La complejidad en cuestiones como el diseño de la función aptitud o criterios de mutación, necesita cierta pericia para obtener resultados deseables.

Una desventaja que con el tiempo podría convertirse en un inconveniente menor es el costo computacional de almacenamiento y cálculo, ya que se emplea espacio físico para almacenar cada elemento de población y una capacidad para procesar el algoritmo respectivo.

Otro inconveniente muy importante se refiere al hecho que dentro de la naturaleza podemos encontrar diversas especies con diferencias sutiles, en muchas ocasiones el algoritmo converge a una única solución.

Empíricamente se ha comprobado que los algoritmos genéticos convergen hacia soluciones aceptables, su campo de aplicación se enfoca en funciones no derivables sin poseer una solución especifica, y aunque existan soluciones especificas para algún tipo de problemas, puede darse una solución híbrida que mejores las obtenidas con anterioridad.

4.3. Conceptos básicos de computación evolutiva

4.3.1. Población inicial

Se genera de manera aleatoria teniendo en cuenta que pueden utilizarse poblaciones que han sido sometidas a algún proceso que asegure cierto grado de optimización; esto causará algunos problemas en determinados casos, por ejemplo que converja la solución demasiado rápido o nos arroje una solución de mínimo local.

Cada elemento de población está constituido por “cromosomas” los cuales integran el conjunto de posibles soluciones, los “cromosomas” están integrados por genes o elementos más simples, estos genes son los que se modifican de tal forma que el cromosoma pueda ser distinto y obtener un nuevo elemento de población.

4.3.2. Función aptitud

También es llamada función *fitness* o función objetivo, es una de las partes más complicadas de diseñar un algoritmo genético, debe evaluar qué tan óptimo es el elemento poblacional que se ha producido y arrojar un resultado estable de modo que, para elementos parecidos devuelva un valor numérico similar. Existen distintas recomendaciones al momento de crear una función objetivo, entre ellas que deben dar valores numéricos reales, también se puede hacer devaluando algún valor de manera que si no cumple, disminuya su capacidad de ser un elemento aceptable.

Algunos investigadores han propuesto que las funciones de evaluación cambien a lo largo de las generaciones, de forma que los individuos cercanos entre sí, devalúen su función objetivo y de esta manera ganen diversidad.

4.3.3. Selección

Existen diferentes criterios para seleccionar los elementos de población, el más utilizado es por función objetivo, se elige el elemento que haya salido mejor calificado, este método también es conocido como elitista y el elemento es elevado a padre.

Está también la selección por torneo en la cual se escoge un número de elementos al azar y de este grupo se selecciona el mejor, además hay métodos dinámicos en los cuales los criterios de selección varían con cada generación.

Si todos los elementos de la población tienen garantizada una probabilidad de selección distinta a cero, el método de selección se denomina preservativo; si alguna probabilidad de selección es cero, se denomina extintivo.

4.3.4. Cruce

Es el mecanismo principal de reproducción de los algoritmos genéticos, en ellos se toman elementos “cromosomas” de los padres y se mezclan obteniendo un elemento poblacional “hijo” que conserve ciertas características de los padres.

4.3.5 Reemplazo

El reemplazo o mutación es el mecanismo por medio del cual se explora el universo de soluciones posibles al igual que en la naturaleza las mutaciones se producen de manera totalmente aleatoria de modo que un gen dando lugar a una nueva combinación que puede ser una posible solución al problema lo que resultaría ser un nuevo cromosoma que pasaría a ser parte integral del elemento de población hijo, el cual sería analizado por la función aptitud la cual determinaría el grado de aptitud de este nuevo elemento y por ende del cromosoma que ha tenido la mutación.

4.4 Ejemplo básico de un algoritmo genético

A continuación se incluye un algoritmo en pseudocódigo tratando de representar la forma de operación de un algoritmo genético.

```
INICIO
    Generar población inicial
        MIENTRAS  NO terminado hacer
        INICIO
            Seleccionar individuos de generación anterior
            para cruce
            Cruzar elementos para obtener descendientes
            Mutar alguno de los elementos
            Reemplazar los elementos mutantes por alguno
            proveniente de los padres
        TERMINAR
        SI población ha convergido
                Terminado = verdadero
        TERMINAR
TERMINAR
```

4.5. Aplicaciones de los algoritmos genéticos

"Algunas aplicaciones importantes que en los últimos años ha tenido la computación evolutiva:

- Diseño automatizado, incluyendo investigación en diseño de materiales y multiobjetivo de componentes automovilísticos: mejor comportamiento ante choques, ahorro de peso, mejora de aerodinámica, etc.
- Diseño automatizado de equipamiento industrial
- Diseño automatizado de sistemas de comercio en el sector financiero
- Construcción de árboles filogenéticos

- Optimización de carga de contenedores
- Diseño de sistemas de distribución de aguas
- Diseño de topologías de circuitos impresos
- Diseño de topologías de redes computacionales
- En teoría de juegos, resolución de equilibrios
- Análisis de expresión de genes
- Aprendizaje de comportamiento de robots
- Aprendizaje de reglas de lógica difusa
- Análisis lingüístico, incluyendo inducción gramática, y otros aspectos de procesamiento de lenguajes naturales, tales como eliminación de ambigüedad de sentido
- Infraestructura de redes de comunicaciones móviles
- Optimización de estructuras moleculares
- Planificación de producción *multicriteria*
- Aplicación de Algoritmos Genéticos al Dilema del Prisionero Iterado;
- Optimización de sistemas de compresión de datos, por ejemplo, usando *wavelets*
- Predicción de plegamiento de proteínas
- Optimización de *layout*
- Predicción de estructura de RNA
- En bioinformática, Alineamiento múltiple de secuencias
- Aplicaciones en planificación de procesos industriales, incluyendo planificación *Job-shop"*[5].

[5] http://es.wikipedia.org/wiki/Algoritmo_gen%C3%A9tico

5. SISTEMAS HÍBRIDOS INTELIGENTES

A lo largo del tiempo la naturaleza nos sigue dando lecciones importantes, una de ellas es que para resolver problemas no utiliza mecanismos individuales aislados sino más bien combina diferentes medios que ayudan a encontrar mejores soluciones de forma más rápida. Investigadores como Minsky, Sloman o Deb Roy creen que estamos en el momento de integrar sistemas complejos para generar grandes sistemas de inteligencia.

Un sistema híbrido inteligente está formado por diferentes subsistemas, cada uno mantiene independencia al momento de inducir soluciones a problemas dados. El objetivo principal de este tipo de sistema es mejorar la eficiencia y potencia de razonamiento que puede dar un sistema inteligente aislado.

5.1. Clasificación de sistemas híbridos inteligentes

En el año 1994 se publicó una clasificación que engloba los sistemas híbridos inteligentes, considerando el grado de complejidad con que interactúan los subsistemas que lo componen, a esta clasificación se le llama ABC, identifica tres niveles de integración:

Artificial
Biológico
Computacional

Los niveles aumentan al pasar del nivel de inteligencia computacional al de inteligencia artificial y de éste al de inteligencia biológica; en cada uno se puede mostrar cómo las unidades básicas de conocimientos se utilizan para reconocer patrones y de este modo crear sistemas inteligentes.

Los sistemas de inteligencia artificial pueden construirse de manera que se combinen unidades básicas de conocimiento integrándose para crear procesos computacionales y estructuras de datos. La inteligencia biológica es objeto de imitación y utiliza sensores, memoria y una serie de técnica propias de redes neuronales biológicas para procesar grandes cantidades de información aumentando la complejidad de sus operaciones. El nivel computacional únicamente utiliza datos numéricos, puede en algún momento reconocer ciertos patrones, pero no utiliza el conocimiento en el sentido al que se refiere la inteligencia artificial.

Para el modelo ABC un sistema híbrido inteligente no es más que tomar inteligencia computacional extendida hacia la artificial de modo que se pueda obtener inteligencia biológica.

Otro sistema de clasificación es el IRIS (Integración de Razonamiento, Información y Servicio) el cual nos facilita un diseño eficiente de sistemas inteligentes, en esta clasificación la creación de un sistema híbrido requiere de la integración de distintas disciplinas científicas como biología, psicología, lingüística e informática.

Medsker y Bailey definieron cinco modos de integración los cuales se describen a continuación.

5.1.1. Modelos independientes

Esta es la primera forma real de hibridar un sistema inteligente ya que el anterior únicamente nos permite trabajar con distintos sistemas que no llegan a comunicarse o interactuar entre sí, por medio de una forma sencilla de comunicación como lo es por ejemplo un archivo, estos sistemas pueden presentar distintas características como:

Preprocesador: utiliza un sistema que se encargue de transformar distintos datos antes de enviarlos a otro que se encargará de un proceso más complicado. Por ejemplo, se puede entrenar una red neuronal que identifique ciertos datos y los transforme a determinada codificación, antes de enviarlos a un sistema experto que los utilice en bases de datos u otro proceso.

Pos-procesador: uso de un sistema inteligente para procesar datos de salida y clasificarlos o seleccionar de otro sistema inteligente, por ejemplo una red neuronal que obtiene datos de un sistema experto y selecciona la información que se guardará.

Coprocesador: requiere de la cooperación entre dos sistemas inteligentes, aunque no es una técnica muy utilizada tiene potencial para resolver problemas complicados que requieren dualidad para obtener mejores soluciones.

Interfaz de usuario: un sistema inteligente que puede facilitar la interacción con el usuario final, a través de una red neuronal que procese los datos obtenidos de un sistema basado en reglas, para presentarlos ordenados al usuario.

Entre de las ventajas más importantes se encuentra que nos provee una implementación más sencilla que la de sistemas híbridos con una copulación más fuerte. Como limitaciones encontramos que consumen más tiempo y recursos, debido a su tipo de comunicación, pueden generar duplicación de esfuerzo por el hecho de desarrollar sistemas separados, con comunicación entre ellos.

5.1.2. Modelos fuertemente acoplados

Estos modelos son similares al anterior, la diferencia radica en que no utilizan métodos como ficheros para la comunicación de los sistemas, sino ambos se comunican directamente de memoria logrando comunicación más rápida y una colaboración que puede llevar menor coste. También en éste aplican los submodelos presentados en el apartado anterior, teniendo en cuenta que supera algunas desventajas presentadas por los sistemas ligeramente acoplados.

5.1.3. Modelos totalmente integrados

Dentro de las características más importantes de estos modelos encontramos que los sistemas comparten estructuras de datos y representación, el razonamiento se hace de modo cooperativo y utilizando algún componente controlador que optimiza el híbrido.

Como ventajas de este tipo de modelos esta la obtención de uno final robusto y sólido que mejora las capacidades de resolución de problemas, durante el desarrollo del sistema no existe redundancia y supera distintas capacidades con respecto a sistemas no integrados.

Entre las desventajas que presenta está que aumenta la complejidad, dificulta la construcción de herramientas capaces de generar híbridos, la validación y verificación requiere mayor esfuerzo.

5.2. Integración de algoritmos genéticos y redes neuronales

El principal objetivo de este trabajo se basa en crear un sistema integrado por una red neuronal tipo ART y un algoritmo genético para integrar un sistema híbrido inteligente, a continuación veremos cómo se han utilizado ambos conceptos en diferentes ambientes.

Una de las aplicaciones que se puede dar al combinar ambos elementos es entrenar las redes neuronales, haciendo evolucionar los pesos de las redes neuronales con la utilización de algoritmos genéticos, de forma que se tengan poblaciones iníciales de los distintos pesos y se usen las salidas de la red neuronal como función aptitud para converger de manera más rápida hacia soluciones óptimas de los pesos. También se puede hacer evolucionar la arquitectura como analogía al cerebro humano creando conexiones o neuronas en tiempo de ejecución, para obtener mejores resultados.

Un sistema ART+AG puede utilizarse en el entrenamiento para eliminar los máximos y mínimos locales que nos traerían problemas de soluciones falsas y en general, para hacer evolucionar cualquier parámetro dentro de la red neuronal.

Pueden utilizarse redes neuronales como función aptitud de los algoritmos genéticos, de modo que la función aptitud pueda aprender de manera similar y variar los resultados en el tiempo.

5.3. Aplicaciones de sistemas híbridos inteligentes

Estos sistemas tienen diferentes aplicaciones, puede afirmarse que se logran aplicar en cualquier campo que requiera un grado de inteligencia siguiendo una serie de pasos para obtener mejores soluciones al momento de presentarse algún problema. Se puede aplicar, entre otros a:

a. Minería de datos

b. Robótica

c. Economía

d. Modelación de sistemas

e. Física

6. IMPLEMENTACIÓN DE SISTEMA HÍBRIDO INTELIGENTE RED ART+ALGORITMO GENÉTICO

En esta sección se detalla la implementación de un sistema híbrido inteligente formado por una red neuronal tipo ART1 y un algoritmo genético, describiendo la arquitectura de la aplicación así como la interacción de ambos.

6.1. Definición del problema

El problema consistió en combinar dos sistemas, el primero simuló el funcionamiento de una red neuronal tipo ART la cual se ha estudiado en capítulos anteriores, éste se integró totalmente a un simulador algoritmo genético tratando de autogenerar estados para el sistema de red neuronal con lo cual se buscó obtener mejores resultados, en cuanto a tiempo y eficiencia, en comparación con el trabajo de los sistemas por separado.

El sistema híbrido se implementó de manera general para lograr ser alimentado por distintos orígenes de datos, sin necesidad de demasiada programación para su adaptación y poder ser utilizado en distintos campos; se desarrolló con un enfoque orientado a objetos y aunque el análisis y diseño de la solución puede adaptarse a cualquier lenguaje que maneje el concepto de objetos, este sistema específico se construyó en lenguaje java bajo las especificaciones J2EE.

6.2. Análisis de la solución

6.2.1. Análisis de riesgos

Riesgos de requerimientos: se deben estudiar y analizar exhaustivamente los algoritmos que conforman la red neuronal ART y el funcionamiento de los algoritmos genéticos utilizados, ya que únicamente la implementación correcta de éstos puede asegurar el éxito del proyecto; también analizar y escoger la mejor forma de acoplar los sistemas, sin afectar las funciones básicas de ambos conceptos, de modo que produzca salidas acorde a las esperadas. Sabemos que tanto las redes ART como los algoritmos genéticos han sido implementados exitosamente por lo que la posibilidad del fracaso del proyecto es muy baja y se basa en el éxito de encontrar el mejor lugar para copular ambos sistemas.

Riesgos tecnológicos: el sistema será desarrollado bajo conceptos orientados a objetos para facilitar la abstracción de los datos utilizados, en la codificación se utilizara el lenguaje java bajo los estándares J2EE y el *hardware* asociado que soporte dichas tecnologías; hoy en día el riesgo de no tener acceso a un *hardware* con estas características es muy bajo y aprovechando que la tecnología J2EE es de uso libre, se espera finalizar con éxito la aplicación.

Se analizarán todos los componentes para minimizar el riesgo si en determinado momento éstos no pudieran acoplarse, por medio de diagramas y con análisis de sus respectivas clases. Una de las ventajas que se tiene para finalizar con éxito el proyecto se sustenta en que no necesita conectarse a ninguna base de datos externa o sistema que no se integre a los estándares mencionados, todas las herramientas requeridas se encuentran bajo dicha especificación.

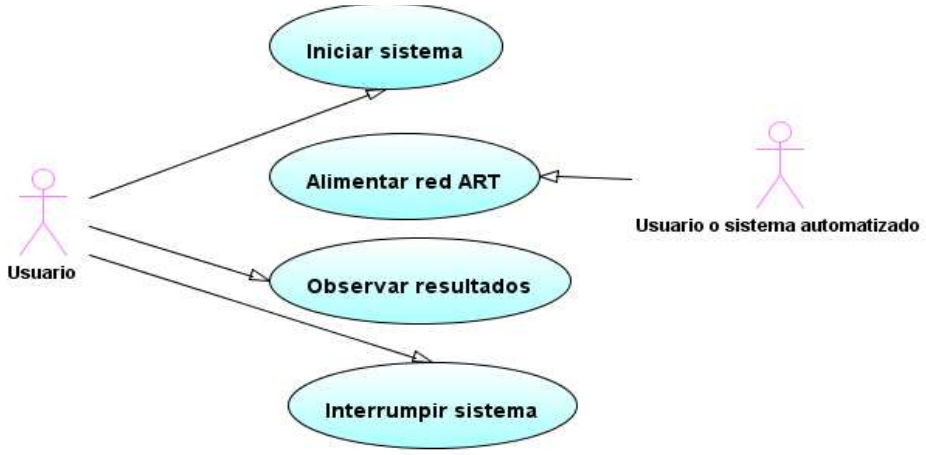
Riesgos de habilidad: se cuenta con la ayuda de un asesor experto en el tema con experiencia en manejo de aplicaciones orientadas a objetos así como en el uso del lenguaje java y tecnologías J2EE por lo que no debería existir ningún problema en codificar la aplicación, ni en desarrollar la lógica necesaria.

Riesgos políticos: el sistema se desarrollará como trabajo de graduación para lo cual se requiere la autorización respectiva y la finalización en un tiempo determinado, además de cumplir con las normas y especificaciones de la Facultad de Ingeniería.

6.2.2. Análisis de casos de uso del sistema

La aplicación que se desarrollará, como todo sistema enfocado en inteligencia artificial, tiene un alto grado de autopoiesis y por lo tanto se genera a sí misma en determinados momentos, por eso su interacción especifica con agentes externos, está reducida a los siguientes casos de uso.

Figura 11. **Diagrama de casos de uso sistema híbrido red ART+AG**



Fuente: elaboración propia.

Actor usuario: encargado de supervisar el correcto funcionamento del sistema como ente humano inteligente.

Actor sistema automatizado: robot o una estructura de datos que ingrese elementos a la red para que ésta produzca los respectivos resultados.

Caso iniciar sistema: Usuario será encargado de poner en marcha el sistema para que el mismo se entrene generando los resultados esperados, contará con una función especial para hacerlo.

Caso alimentar red ART: la red ART necesita ingresar matrices de datos en este caso binarios, para poder entrenar la red y funcionar e manera correcta; esta entrada se hará manualmente por medio de un usuario que ingrese los datos o de manera automatizada en lotes.

Caso observar resultados: el sistema debe ser aplicable a algún área específica, como robótica por ejemplo y el usuario debe ser encargado de observar los datos que se producen para interperetarlos y decidir si son los que se esperan.

Caso interrumpir sistema: si el sistema se sale de sus limites o pierde el control, la aplicación debe tener un mecanismo especial para interrumpir el funcionamiento del mismo.

6.2.3. Análisis de algoritmos de red ART1

Primero se analizarán las estructuras y algoritmos que conforman la red neuronal tipo ART1, de manera que se pueda comprender lo esperado que la red genere como resultados, ademas de su funcionamiento interno.

6.2.3.1. Algoritmo ART1

Es preciso seleccionar el tamaño de las distintas capas que conformarán la red ART1 y los valores de los distintos parámetros del sistema, pueden surgir las limitaciones siguientes, teniendo en cuenta que se utilizarán dos capas F1 y F2 en la cual M denota el número de unidades en F1 y N el número de unidades en F2.

$$A1 \geq 0$$

$$C1 \geq 0$$

$$D1 \geq 0$$

$$\max\{D1,1\} < B1 < D1 + 1$$

$$L > 1$$

$$0 < \rho \leq 1$$

Los valores de los pesos sinápticos reciben valores iniciales descendentes, de acuerdo con la siguiente fórmula.

$$z_{ij}(0) > \frac{B1 - 1}{D1}$$

Los pesos sinápticos ascendentes iniciales se basan en:

$$0 < z_{ij}(0) < \frac{L}{L - 1 + M}$$

Las actividades de F2 reciben valores iníciales de cero y, de acuerdo con el modelo, las actividades de F1 reciben valores iguales a

$$X_{1i}(0) = \frac{-B1}{1 + C1}$$

Debe recordarse que la red ART1 procesa valores binarios de la forma

$$I_i \in \{0,1\}$$

A continuación se presenta el algoritmo básico que debe seguir la red neuronal.

1. Se aplica un vector de entradas a F1. Las actividades de F1 se calculan con la siguiente fórmula:

$$x_{1i} = \frac{I_i}{1 + A_1(I_i + B_i) + C_1}$$

2. El vector de salida correspondiente a F1, se calcula de la siguiente manera:

$$S_i = h(x_{1i}) = \begin{cases} 1, & x_{1i} > 0 \\ 0, & x_{1i} \leq 0 \end{cases}$$

3. Se propaga S hacia adelante, hasta F2 y se estiman las actividades de acuerdo a:

$$T_j = \sum_{i=1}^{M} S_i z_{ji}$$

4. Únicamente el ganador de F2 posee una salida no nula:

$$u_j = \begin{cases} 1, & T_j = \max\{T_k\} \forall k \\ 0, & en\ cualquier\ otro\ caso \end{cases}$$

5. La salida de F2 se propaga retrocediendo hasta llegar a F1. Se calculan las entradas netas procedentes de F2 que llegan a las unidades de F1.

$$V_i = \sum_{j=1}^{N} u_j z_{ij}$$

6. Luego se calculan las actividades de acuerdo con:

$$X_{1i} = \frac{I_i + D_1 V_1 - B_1}{1 + A_1(I_i + D_1 V_i) + C_1}$$

7. Se determinan los nuevos valores de salida de la misma manera que en el paso 2.

8. Determinar el grado de coincidencia entre la trama de entrada y la plantilla descendente.

$$\frac{|S|}{|S|} = \frac{\sum_{i=1}^{M} S_i}{\sum_{i=1}^{M} I_i}$$

9. Si |S|/|I| < ρ , se marca V_j desactivada, se ponene a cero las salidas de F2 y se vuelve al paso 1 empleando la trama de salida original. Si |S|/|I| ≥ ρ continuamos.

10. Se actualizan únicamente los pesos ascendentes de V_j

$$z_{Ji} = \begin{cases} \dfrac{L}{L - 1 + |S|}, & si\ V_i\ esta\ activa \\ 0, & si\ V_i\ \text{no esta activa} \end{cases}$$

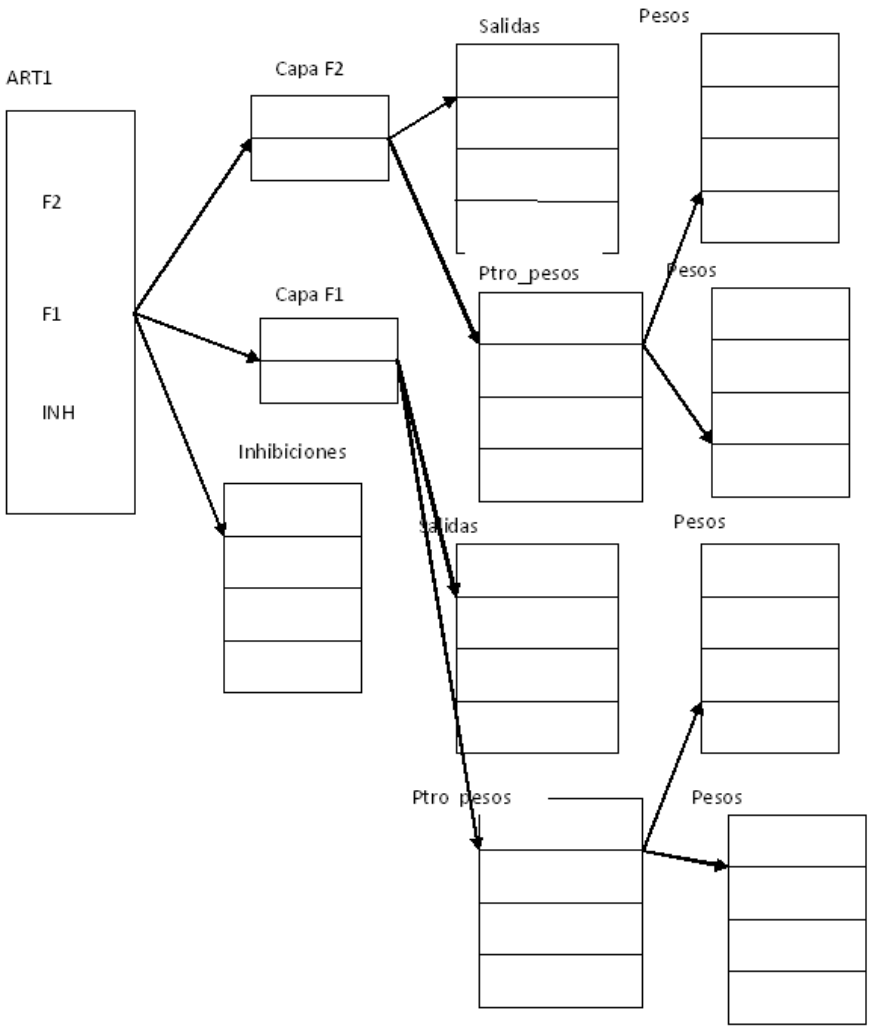
11. Se actualizan solo los pesos descendentes que provienen de V_j y llegan a todas las unidades de F1.

$$z_{ij} = \begin{cases} 1, & si\ V_i\ \text{esta activa} \\ 0, & si\ V_i\ \text{no esta activa} \end{cases}$$

"Se elimina la trama de entrada y restauran todas las unidades inactivas de F2. Se vuelve al paso 1 con una nueva trama de entrada.

En la figura 12 puede observarse una representación gráfica de la red neuronal con dos capas, cada capa apunta a sus respectivas salidas y pesos, cabe mencionar que las salidas son vectores planos mientras que los pesos son matrices de datos.

Figura 12. **Representación de red ART en memoria**



Fuente: elaboración propia.

A continuación se muestra una iteración de cómo deberia comportarse la red neuronal. En la tabla VI pueden observarse los vectores de entrada que se darán a la red y las salidas que produce; la tabla VII incluye los pesos modificados que produce el vector de entrada en la capa F1. En la tabla VIII se observa cómo se modifican los pesos en F2, el resultado de esta entrada es que reconocerá el primer patrón y lo identificará como patrón 0 siempre que se introduzca en la red.

Tabla VI. **Entradas y salidas esperadas de la red ART**

Entradas	Salidas
0	0
0	0
0	0
0	0
1	1

Fuente: elaboración propia.

Tabla VII. **Matriz de pesos en F1**

Matriz de Pesos F1				
0	0,756	0,756	0,756	0,756
0	0,756	0,756	0,756	0,756
0	0,756	0,756	0,756	0,756
0	0,756	0,756	0,756	0,756
1	0,756	0,756	0,756	0,756

Fuente: elaboración propia.

Tabla VIII. **Matriz de pesos en F2**

Matriz de Pesos de F2				
0	0	0	0	1
0,329	0,329	0,329	0,329	0,329
0,329	0,329	0,329	0,329	0,329
0,329	0,329	0,329	0,329	0,329
0,329	0,329	0,329	0,329	0,329

Fuente: elaboración propia.

6.2.3.2. Análisis de sistema híbrido con algoritmo genético

Se analizó el comportamiento de un simulador de red neuronal tipo ART1 paso a paso observando como los valores de entrada se van trasnformando hasta dar una salida. En este punto se buscará el mejor lugar para desarrollar un algoritmo genético que ayudará al desempeño del sistema.

Se necesita que el algorimo genético no altere ninguna de las ecuaciones fundamentales con las cuales funciona el algoritmo ART1 y que mejore el desempeño de éste, durante la recepción del vector de entrada y durante su respectivo proceso, para reconocerlo podría afectar el comportamiento general del algoritmo ART, se tratará de ubicar el algoritmo en la matriz de pesos sinápticos, mientras no se esté procesando alguna entrada.

El algoritmo genético tomará pesos anteriores y generarndo un nuevo peso sináptico, con esto se espera ahorrar tiempo ya que cuando una nueva entrada que no ha sido procesada por la red es tomada, se necesita cierto tiempo para procesarla, reconocerla y guardarla de modo que en la siguiente ocasión que dicho patrón se obtiene ya se ha identificado en alguna de las columnas de la matriz de pesos sinápticos.

Al utilizar el algoritmo genético éste se auto-genera de modo que ahorra tiempo integrando el nuevo patrón a las matrices de pesos.

6.2.3.3. Algoritmo genético

El algoritmo inicia en un momento en el que deben existir dos o más patrones observados en la matriz de pesos en F1, estos son los padres y a partir de ellos pueden generarse nuevos valores, este estado inicial puede observarse en la tabla IX.

Tabla IX. **Matriz de pesos en F1 al momento de generar un nuevo patrón para matriz de pesos sinápticos**

Matriz de Pesos F1				
0	0	0,756	0,756	0,756
0	0	0,756	0,756	0,756
0	1	0,756	0,756	0,756
0	0	0,756	0,756	0,756
1	0	0,756	0,756	0,756

Fuente: elaboración propia.

A partir de dichas entradas o padres éstas se cruzan produciendo un nuevo valor llamado hijo el cual contiene características de ambos padres, puede observarse en la tabla X.

Tabla X. **Nuevo patrón generado a partir de padres**

Padre 1	Padre 2	Hijo
0	0	0
0	0	0
0	1	1
0	0	0
1	0	1

Fuente: elaboración propia.

Este nuevo patrón puede sufrir alguna variación llamada mutación, en referencia al hijo generado en la tabla X, puede observarse en la tabla XI.

Tabla XI. **Mutación de un patrón a partir de un hijo existente**

Hijo original	Mutación
0	1
0	0
1	0
0	0
1	1

Fuente: elaboración propia.

La mutación debe pasar un proceso de selección de modo que se optimice el sistema y no existan nuevos valores que retrasen el algoritmo ART1, en lugar de darle mayor velocidad.

Para evaluar los vectores hijos existen varios criterios, por ejemplo que no todos los valores del patrón pueden ser uno o no todos pueden ser cero ni idéntico a uno de los padres.

Para cada uno de los criterios se asignará un valor númerico buscando crear una fórmula matemática que indique un porcentaje con que el patrón cumple el crieterio; por ejemplo, se contará cada una de las posiciones, si el valor es igual el criterio se cumple en un 0%, si el valor es diferente se cumple en 1/x% donde x es el total de elementos del patrón. La sumatoria de todos los niveles indica qué tan eficiente es dicho patrón, según este criterio si es mayor a 80% como en la tabla XII, se acepta, de lo contrario se desecha intentando con una nueva iteración.

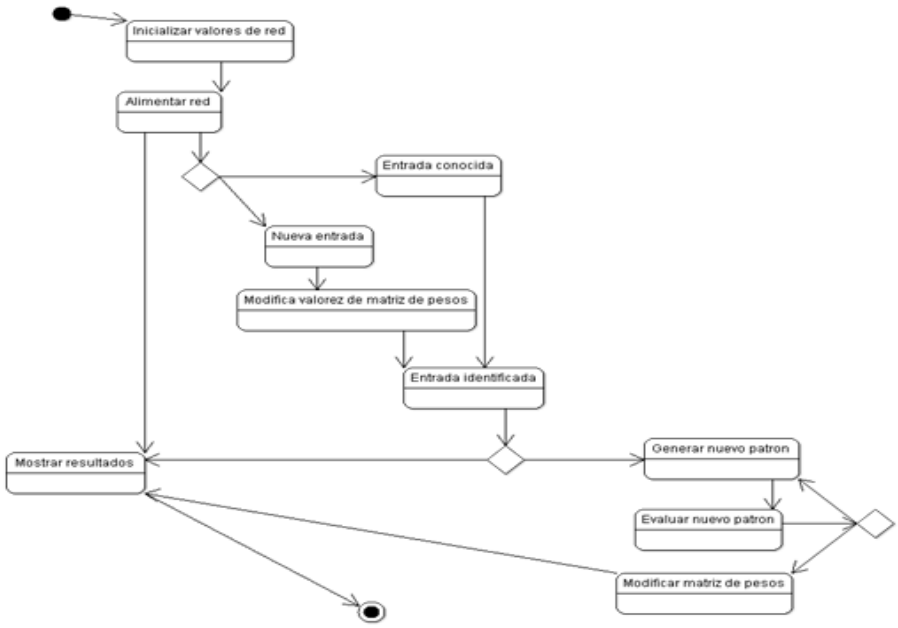
Tabla XII. **Total de eficiencia con que el nuevo valor cumple con el criterio de selección**

Padre 1	Mutación	% de eficiencia
0	1	1/5
0	0	1/5
0	0	1/5
0	0	1/5
1	1	0
Total		4/5%

Fuente: elaboración propia.

6.2.4. Diagrama de estados

Figura 13. **Diagrama de estados sistema híbrido red ART+AG**

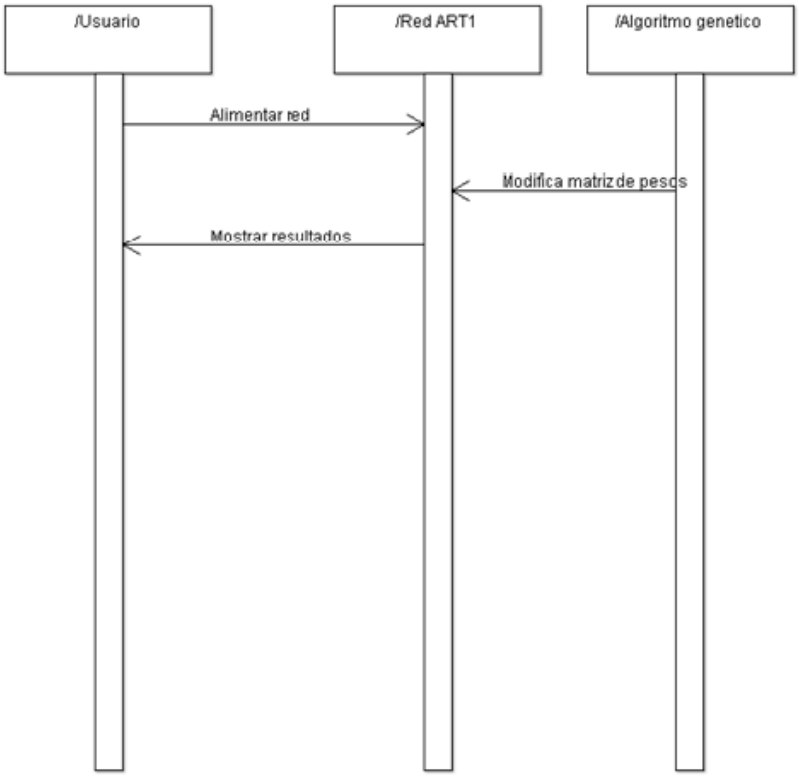


Fuente: elaboración propia.

El simulador híbrido inteligente puede tener distintos estados durante su ejecucion; al iniciar se deben establecer parámetros iníciales para los valores, luego el sistema espera se le ingresen vectores o patrones; se calculan, si son conocidos se ubican en la matriz de pesos sinápticos, de lo contrario se modifica para que el patrón quede registrado en ésta. Luego de identificar el patrón en algún momento aleatorio se genera uno nuevo, evaluando qué tan factible es usar éste, si es positivo modifica la matriz de pesos, si no se busca una nueva iteración del algoritmo genético hasta que se obtenga resultado satisfactorio y por último muestra los resultados obtenidos en cualquier momento, luego de esto se puede finalizar la aplicación. Figura 13.

6.2.5. Diagrama de secuencia

Figura 14. **Diagrama de secuencia aplicación red ART+AG**



Fuente: elaboración propia.

El sistema en general debe tener el comportamiento mostrado en la figura 14, se inicializa y espera recibir un vector de entrada por parte del usuario u algún otro sistema; el algoritmo de red ART1 se encarga de modificar todo lo necesario para recordar e identificar el patrón ingresado y espera que en algún tiempo aleatorio o al terminar, se modifiquen los pesos sinápticos de la matriz, utilizando el algoritmo genético.

Las clases potenciales pueden verse en . Tabla XIII.

Tabla XIII. **Clases potenciales de sistema red ART1+AG**

Clase potencial	**Descripción**	**Decisión**
Capa	Las redes ART1 están integradas por distinto número de capa, éstas contienen información necesaria para realizar el proceso así como algunos procesos intrínsecos	Aceptada
Simulador ART1	Debe contener los distintos mecanimos para que las entradas recorran las capas y estas se transformen en la respectivas salidas	Aceptada
Pesos	Debe contener los distintos pesos en una matriz específica	Rechazada, por el momento se utilzarán estructuras de datos que nos proporciona la herramienta
Simulador algoritmo genético	Se debe contener la información necesaria así como los procesos para ejecutarlo	Aceptada
Red ART1	Incluye distintos mecanismos que llevan el control de una lista con información de la red	Aceptada
Interfaz de usuario	Se debe contener una clase especial para mantener la comunicación con el usuario respectivo	Aceptada

Fuente: elaboración propia.

6.3. Diseño de la solución

A partir de la información con que se cuente pueden diseñarse los algoritmos que ayudarán a solucionar el problema así como a definir una serie de clases que se deben utilizar, al momento de desarrollar este sistema.

6.3.1. Algoritmos de Aplicación

Algoritmo de inicializar valores

1. Se ingresan valores iníciales
2. La matriz de pesos descendentes recibe valores iníciales
3. La matriz de pesos ascendentes recibe valores iníciales
4. F2 recibe valores iníciales iguales a cero
5. Las actividades de F1 reciben valores iníciales

Todas las fórmulas se especifican en el punto 6.3.2.1

Algoritmo de procesamiento de datos en la red

1. Se aplica un vector de entrada a las actividades en F1
2. Se calcula el vector de salida de F1
3. Se propaga S hacia F2 y se calculan las actividades
4. Si es un nodo ganador posee una salida no nula, de lo contrario tiene una salida nula
5. Se propaga la salida de F2 hacia F1
6. Se calculan las entradas netas procedentes de F2 que llegan a las unidades de F1
7. Se calculan las nuevas actividades para la red
8. Se calculan los nuevos valores de salida
9. Se calcula el grado de coincidencia entre la trama de entrada y la plantilla descendente.
10. Si $|S|/|I| < \rho$ se marca vj desactivada y se colocan a cero las salidas en F2 y se regresa al paso 1, de lo contrario se continúa al paso 11
11. Se actualizan los pesos descendentes
12. Se actualizan únicamente los pesos descendentes de vj y llegan a todas las unidades F1
13. Se elimina la trama de entrada y se vuelve al paso 1

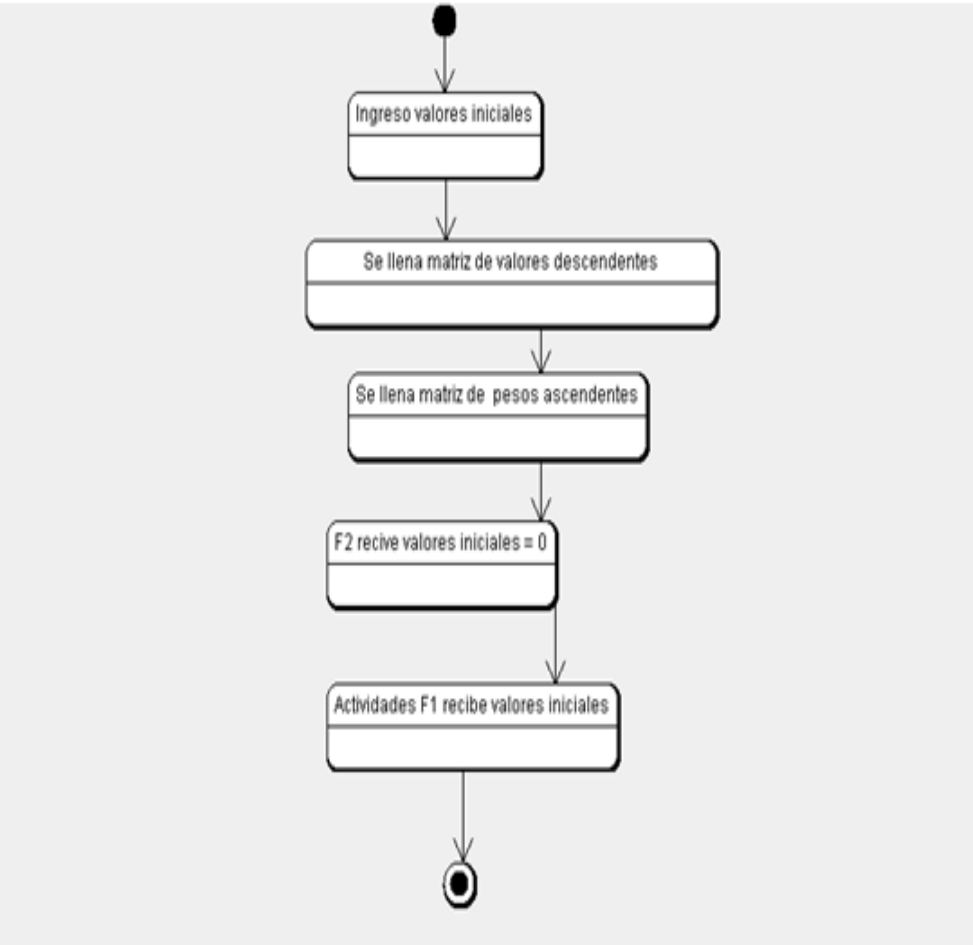
Todas las fórmulas se especifican en el punto 6.3.2.1

Algoritmo genético

1. Se toma la matriz F1 como F2
2. Si existen x>=2 columnas con pesos definidos se procede al paso 3, de lo contrario se retorna a paso 1
3. Se selecciona una columna como padre 1
4. Se selecciona una columna como padre 2
5. Se toma un elemento de cada padre de forma que se deja uno de por medio, por ejemplo 12121212 donde 1 son hijos del padre 1 y 2 hijos del padre 2
6. Se genera aleatoriamente una mutación que consistirá en cambiar uno o varios elementos de cero a uno, o viceversa
7. El vector generado se evalúa según los criterios establecidos
8. Si el elemento supera los criterios de selección, se copia en una nueva posición para las actividades F1 y F2, de lo contrario el elemento se rechaza y se vuelve al paso 3
9. Se intercambian las nuevas matrices en la red neuronal ART1

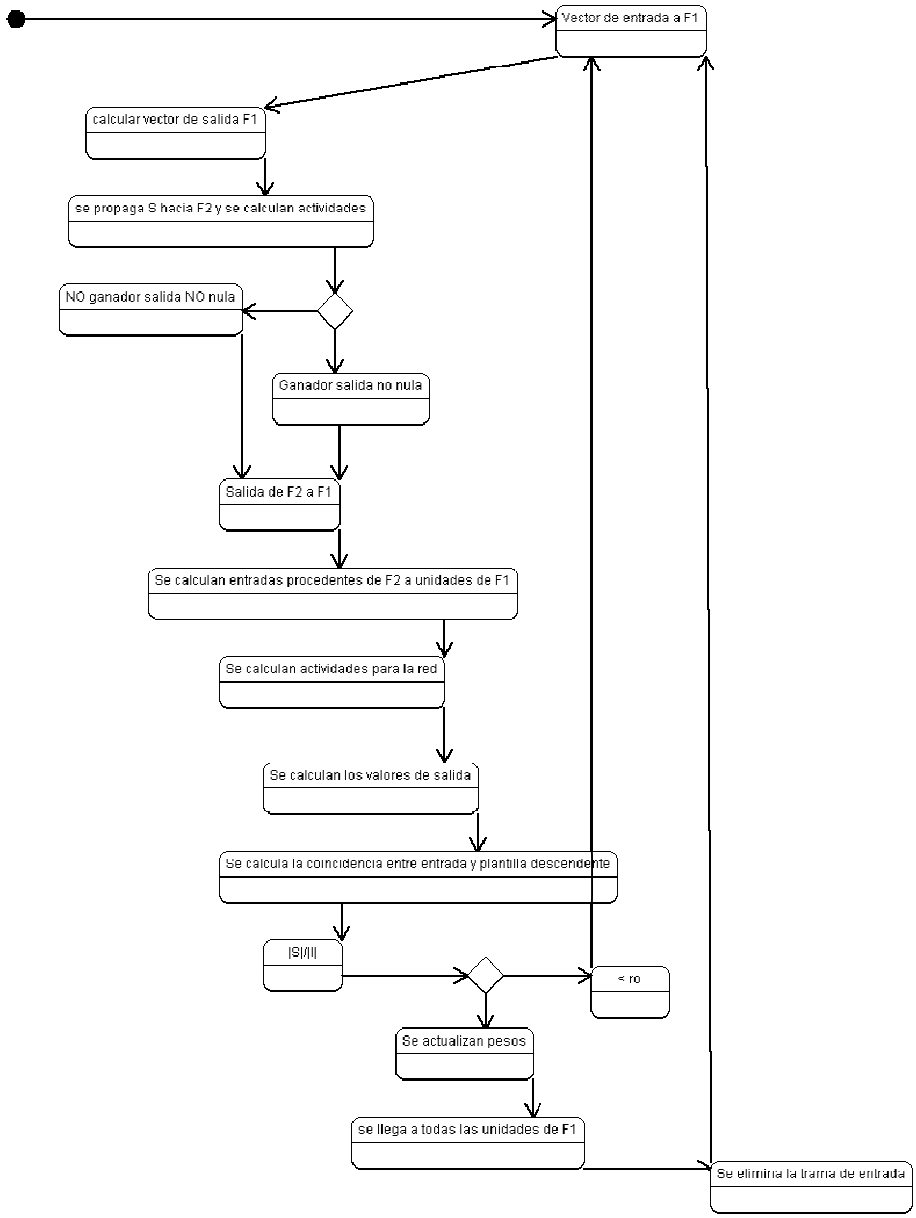
6.3.2. Diagramas de flujo

Figura 15. **Diagrama de flujo inicialización de datos**



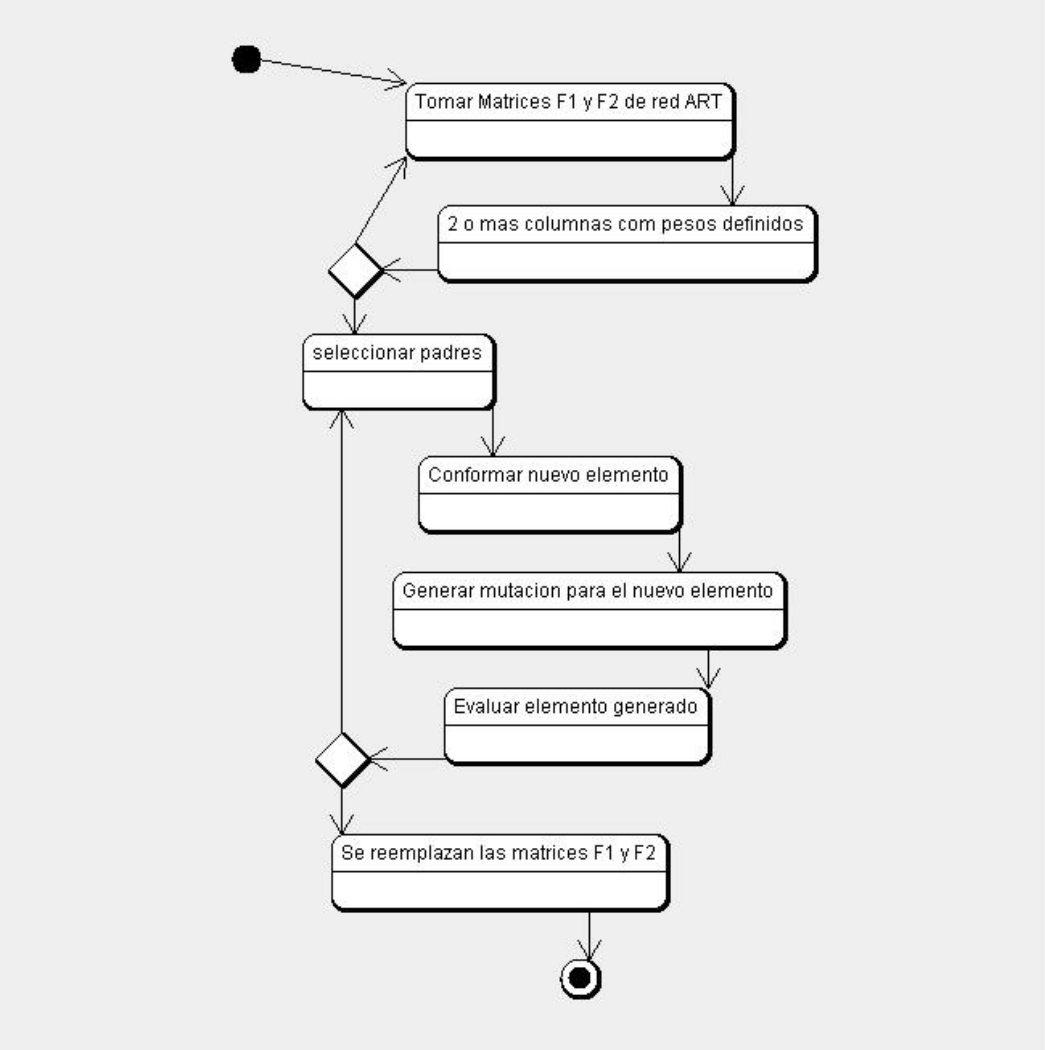
Fuente: elaboración propia.

Figura 16. **Diagrama de flujo para la propagación de los datos en la red**



Fuente: elaboración propia**.**

Figura 17. **Diagrama de flujo que representa el comportamiento del algoritmo genético.**

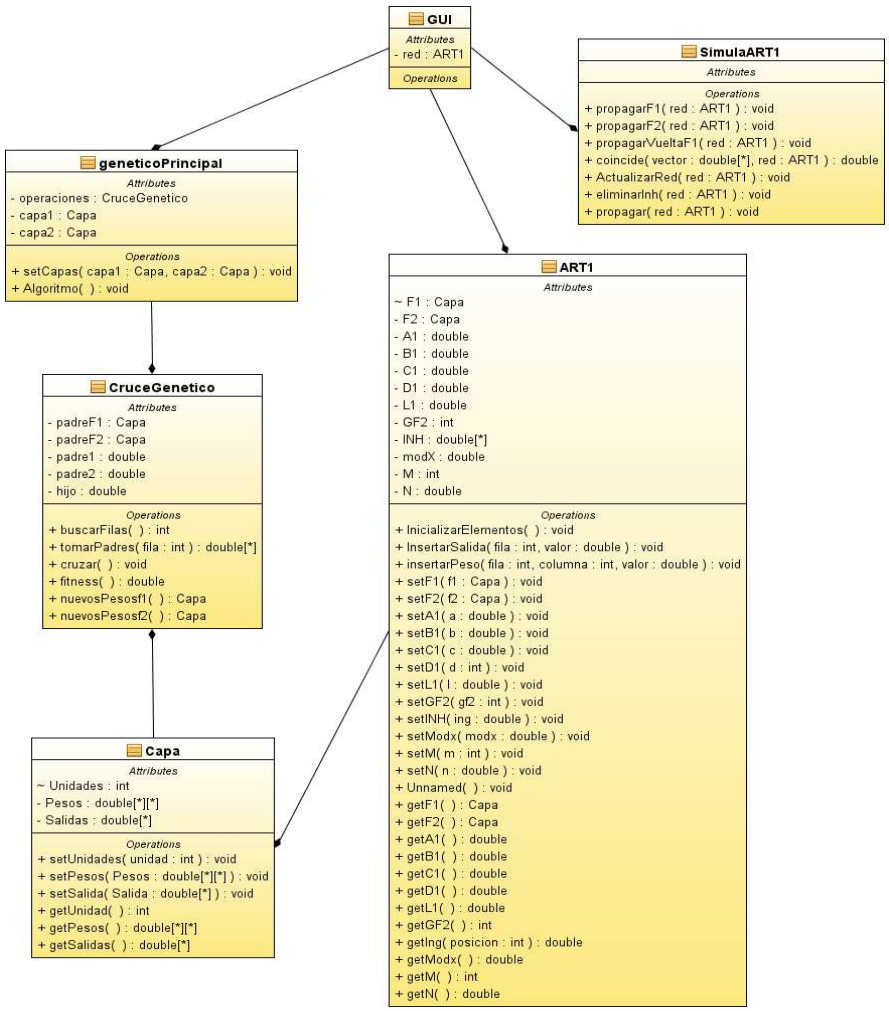


Fuente: elaboración propia.

6.3.3. Diagrama de clases

A continuación, a partir de las clases previstas en la fase de análisis de la solución se crea un diagrama especificando la interacción de cada una de estas clases así como se propondrá su estructura especifica en la cual se detallaran los posibles datos a contener y las acciones que tendrá destinadas.

Figura 18. **Diagrama de clases**



Fuente: elaboración propia.

6.3.4. Descripción de clases

GUI*:* debe integrar toda la funcionalidad necesaria para que el usuario pueda interactuar con el sistema híbrido inteligente, dentro de lo importante a resaltar se subraya que en esta clase encontramos instancias de las clases ART1, simulaArt y genético principal las cuales encapsulan el funcionamiento del sistema, no se especifica a detalle en el diagrama cada uno de los componentes ya que éstos pueden cambiar dependiendo del sistema al que se desee integrar.

GeneticoPrincipal*:* encapsula toda la funcionalidad necesaria que llevaran a cabo las operaciones para generar una iteración del algoritmo genético, también se encarga de tomar las capas almacenadas en la red neuronal ART1.

CruceGenetico: contiene todas las operaciones que debe llevar a cabo un algoritmo genético, entre ellas escoger a los padres, llevar a acabo el cruce, así como la mutación de genes y comprobar la eficiencia del nuevo elemento de población; por medio de la función *fitness* se debe también incluir una operación que sustituya las matrices de pesos originales por las generadas por en el algoritmo genético.

ART1: encontramos todos los elementos contenidos en la red neuronal que ayudará a encapsular dichos elementos y operaciones para interactuar con ellos, de tal forma que todos los datos por manipular en la red, se encuentren contenidos en una instancia que facilite la interacción con otros elementos y se reduzca la complejidad durante el acoplamiento con otro tipo de sistema.

SimulART1: incluye las operaciones específicas que lleva a cabo una red neuronal como propagarla, actualizar elementos o banderas; este diseño logra tener una mayor independencia de los datos, de tal forma que realizar las operaciones no dependa de ellos, se desea lograr con esto un número n de redes dentro del sistema necesitando una instancia de este elemento para manipular estas redes.

Capa: contiene las unidades de almacenamiento de datos dentro del sistema híbrido, ayuda a que los datos puedan ser manipulados por los demás elementos, contiene la matriz de pesos sinápticos, así como los distintos vectores de salida para que toda la infomacion se encuentre encapsulada por él.

6.4. Implementación del sistema

En esta sección se detallan los pormenores de la implementación del sistema como nombres de los archivos utilizados, nombres y descripciones de las variables así como de las funciones y operaciones utilzadas.

Para la implementación y funcionamiento del sistema se requieren cinco archivos que están agrupados en dos paquetes representados para su almacenaje en disco, como carpetas. Éstas son: paquete redneuralart que contiene:

A. Capa.java: este archivo incluye la clase capa, contiene la variable “pesos” que se representa como una matriz de tipo *double* y la variable “salidas” la cual es un vector también de tipo *double*. Ambas son de ese tipo con el objetivo de no perder precisión en cuanto a los decimales manejados.

También contiene la variable unidades de tipo int, representa el número de éstas que podrá manejar la red neuronal, dentro de los métodos se incluyen los que pueden establecer un valor a las variables y los que les devuelven un valor.

B. Art1.java: contiene la clase ART1 y la variable F1 de tipo capa, representa la capa de entrada de la red neuronal, la variable F2 que es de tipo Capa y representa la de salida de la red neuronal. las variables A1, B1, C1, D1, L1, ro, son de tipo *double* para mantener la precisión de los decimales y demuestran los parámetros iníciales de la red neuronal.

   La variable GF2 es de tipo entero, muestra el vector ganador de la matriz de pesos sinápticos, la variable INH es de tipo *double* y se representa como un vector en memoria que contiene al de inhibición de la red neuronal. Las variables M y N representan las unidades de F1 y F2, respectivamente.

   Entre las operaciones de mayor importancia en esta clase se encuentra el constructor de ART1 en el cual se deben establecer todos los parámetros inciales con los que se desea que la red funcione; luego de establecerlos se ejecuta la operación inicializarElementos() encarga de realizar a detalle cada una de las operaciones descritas en el funcionamiento de la red neuronal, también se encuentran todas las operaciones relacionadas con establecer y devolver el valor para cada una de las variables contenidas en esta clase.

C. SimulaArt1.java: incluye únicamente operaciones que se encargan de simular las actividades de una red neuronal de tipo ART1, éstas son prop_F1() que calcula salidas de la capa F1 para un vector de entrada,

esta función contiene la variable i de tipo entero y funciona como un contador de iteraciones; la variable unidad abarca las salidas de la capa F1 y es un *array* de tipo *double,* este método no devuelve ningún valor ya que es de tipo *void* y recibe como parámetros, red de tipo ART1 construida en memoria.

Vectent es un *array* de tipo *double,* vector de entrada dado para una iteración de la red. La función prop_F2() propaga las señales calculadas para F1 hacia F2, esta función contiene las variables i,j que son de tipo int y funcionan como contadores de iteración, la variable unidad de tipo double la cual almacena las unidades de salida de la capa F2, la variable entradas de tipo *double* la que almacena las unidades de salida de la capa F1, la variable mayor de tipo double se almacena la mayor activación de la red.

La variable ganadora de tipo int se inicializa en cero e indica que esta unidad en esa posición es la ganadora para luego transferirse a la variable global Gf2, misma que almacena a la ganadora actual de la red. La variable suma es de tipo *double*, acumula una relación entre el peso sináptico y el vector de entrada y un elemento de las salidas de F1; este método no devuelve ningún valor y es de tipo *void,* recibe como parámetro una red construida en memoria de tipo ART1 a la que modifica.

La función prop_vuelta_F1() al tener una ganadora en F2 propaga la señal de dicho ganador de vuelta a F1, contiene las variables ganadoras de tipo int las que toman de la variable global GF2 el índice de la ganadora en F2, la variable entera i funciona como un contador de iteraciones.

La variable unidad almacena las salidas de F2, la variable X de tipo *double* nos indica si su valor es mayor a cero que una salida de F2 debe ser activada = 1 o bien si se debe desactivar = 0 si su valor es igual a cero; esta función recibe una red de tipo ART1 la cual debe ser con la que se viene trabajando y un vector de tipo double el cual se propaga de la salida anterior.

La función coincide() compara el vector de entrada con los vectores de activación de modo que si éstos no coinciden se inhiban los ganadores de F2, es de tipo *double* ya que retorna una razón de coincidencia; las variables que contiene esta función son i la cual es de tipo int y es un contador de iteraciones, la variable unidad que es un *array* de tipo *double* la cual contiene las salidas de F1.

La variable modX almacena el módulo de la plantilla de salidas de F2 y modI de tipo *double* que contiene el módulo del vector de entrada; los parámetros de esta función son la red tipo ART1 y debe ser la misma con la que se ha trabajado u un vectent el cual es un vector de entrada al que se le sacará el módulo para observar el grado de coincidencia con el de los almacenados de modo que se actualice o no la red.

La función actualizar(), actualiza los pesos sinápticos de la red neuronal para que recuerde las tramas que entran; sus variables son i de tipo int y es un contado de iteraciones, ganadora de tipo int e indica el índice de la posición de peso ganador, la variable entradas es un *array* de tipo *double,* contiene las salidas de F1 y la variable conexiones la cual es una matriz del mismo tipo con los pesos sinápticos de la capa F2.

La función propagar() se encarga de tomar un vector y llevarlo por las anteriormente descritas de tal manera que se pueda simular de forma electrónica el comportamiento de una red neuronal matemáticamente ya descrita.

Otro paquete importante, abordado en la sección de diseño, es el genético, contiene las clases que toman los pesos sinápticos de la red neuronal y a partir de ellos crea nuevos pesos, utilizando un algoritmo genético para lograr el objetivo. Los archivos importantes son:

D. Cruce.java: contiene todas las funciones necesarias para tomar pesos sinápticos y realizar las operaciones de un algoritmo genético, sus principales variables son padreF1 de tipo capa que debe contener la capa F1 de la red y padreF2 de tipo capa que contiene la F2 de la red neuronal.

Las variables padre1 son un *array* de tipo *double* que contiene una fila de la matriz de pesos y sirve como padre uno, la variable padre2 es también un *array* de tipo *double* y contentiene otro padre del algoritmo genético; la variable hijo es un *array* de tipo *double*, contiene el resultado de generar un nuevo peso sináptico, a partir de los padres anteriores.

Los métodos importantes son: buscarFilas() que no tiene parámetros de ingreso y retorna un valor entero el cual indica el número de filas con pesos específicos recordados que posee la red; sus variables importantes son noFilas de tipo entero y es un contador que indica cuántas filas están almacenando actualmente un peso.

La función tomarPadres(int filas) devuelve un *array* de tipo *double* conteniendo una fila especificada de la matriz de pesos sinápticos. El método cruzar() no recibe ni devuelve ningún parámetro pero llena la variable hijo a partir de los padres, tomando elementos aleatorios de cada uno y creando una mutación que consiste en colocar un valor de los dos posibles, uno o cero, para un elemento del peso. Las variables importantes son probMutacion de tipo int que indica si el elemento debe mutar o no y probPadre que es de tipo int e indica que el hijo hereda un gen de un padre u otro.

La función *fitness()* evalúa qué tan apto es un elemento para ser hijo y pasar a formar parte de la matriz de pesos sinápticos, las variables importantes son fitIgual de tipo *double* que nos da una razón que indicando si el valor está más cerca de cero; el elemento hijo tiene un grado de coincidencia más elevado con alguno de los padres. La variable fitCeros es de tipo *double* e indica si todos los elementos del hijo son ceros, la variable fitUnos es de tipo *double*, e indica si el elemento hijo está compuesto únicamente por elementos de activación uno.

El método capaNuevosPesosF1(*double* i,int j) reemplaza los valores anteriores de la capa en la red neuronal, por el generado mediante el algoritmo genético para la capa F1, sus parámetros son un *array* de tipo *double* pesos que contiene los nuevos a utilizar y un int llamado posCol que indica la columna dentro de la matriz de pesos sinápticos que se sustituirán.

E. genPrincipal.java: servirá como interfaz entre el paquete de red ART y las operaciones de algoritmo genético, esta contiene la variable algoritmo que es de tipo cruce y realiza las acciones necesarias para ejecutar un algoritmo genético, retorno F1 y retorno F2 las cuales son de tipo capa y contendrán las capas F1 y F2.

La función algoritmoGen realiza las acciones necesarias llamando funciones del algoritmo de tipo cruce para llevar a cabo todas las acciones de un algoritmo genético, validando previamente que existan al menos dos o más padres para lograr un hijo que contenga características de ambos padres.

7. EXPERIMENTO SOBRE RECONOCIMIENTO DE PATRONES Y ROBÓTICA

En esta sección se expone un experimento sobre el reconocimiento de patrones como ejemplo de aplicación del sistema híbrido de inteligencia artificial, orientado al control de un robot pequeño en un ambiente controlado y simple que pretende simular a escala menor, una aplicación industrial, se tratará de exponer la finalidad del experimento así como la descripción de los elementos necesarios para llevarlo a cabo y los resultados obtenidos.

7.1. Definición del problema

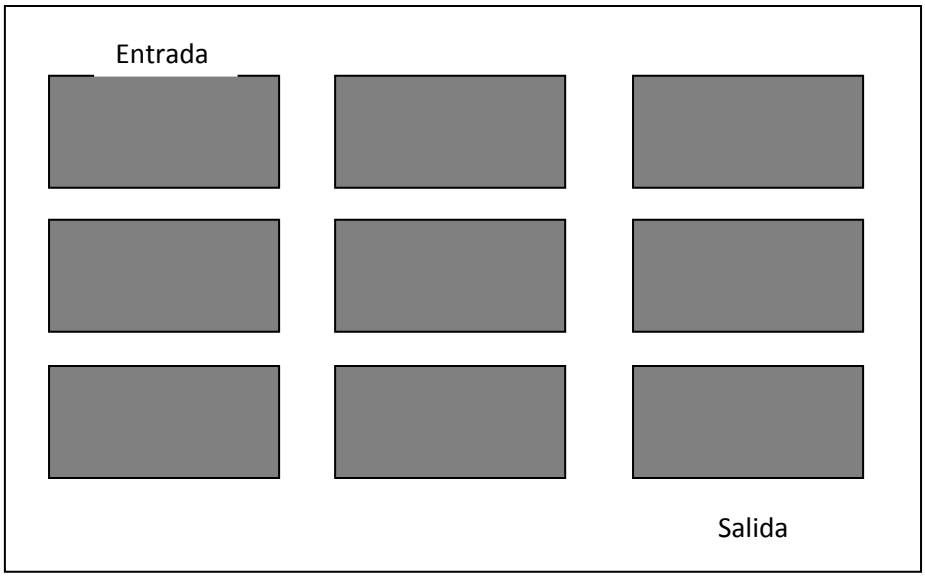
Las redes neuronales de tipo ART1 están enfocadas específicamente a resolver problemas que se relacionan con el reconocimiento de patrones. Puede decirse que las redes ART1 ayudan a reconocer modelos por medio de los cuales se logra resolver problemas abordados por dichos modelso. De manera que una vez detectado un problema puede resolverse siempre que siga el mismo patrón de comportamiento.

El problema seleccionado para el experimento es el siguiente. Se trata de una bodega en la que se almacenan aparatos eléctricos grandes, se necesita reemplazar la mano de obra humana por un robot que pueda mantenerla ordenada y con una salida.

La intervencion debe prever minimizar los accidentes disminuyendo las demandas y los gastos que este rubro puede ocasionar a la empresa, además de ahorrar en otros rubros y teniendo un empleado dispuesto a trabajar las veinticuatro horas, sin amenazas de huelga.

Lo anterior se puede lograr a nivel industrial a escala de este experimento se utilizó un robot Lego® NXT®, configurado para medir 45cm de largo y 20cm, de ancho se colocado sobre un tablero el cual representa la bodega.

Figura 19. **Tablero de simulación de bodega**



Fuente: elaboración propia.

Cada uno de los rectángulos de color gris indican posiciones que pueden ser ocupadas por determinado objeto. Al momento de ser ocupadas ni el robot ni persona alguna pueden ocupar o pasar por esa área siendo necesario que el robot limpie dicha posición si interfiere, en algún punto, entre la entrada y la salida de la bodega.

El robot no detecta automáticamente cuál posición está ocupada, necesitará de un operador encargado de indicarlo, por medio de la interfaz de usario.

El robot por medio de la red neuronal debe ser capaz de determinar si existe o no un camino libre desde la entrada hasta la salida del sistema si no existiera esta última debe liberar el camino moviendo las posiciones ocupadas; si la bodega está demasiado llena envia a la sala de ventas los excedentes, la entrada del sistema es en el área de descarga y la salida hacia sala de ventas.

Figura 20. **Tablero de mando en pantalla**

Fuente: elaboración propia.

7.2. Observación

Al observar el proceso se comprueba que existe una pequeña matriz de tres filas por tres columnas, el comportamiento de una red tipo ART1 de la manera en la cual se diseñó e implementó en este proyecto opera reconociendo patrones que se comportan como vectores lineales. La entrada es un vector lineal que se propaga por las diferentes capas que conforman la red neuronal tipo ART1, podemos asociar ambos escenarios tomando cada una de las columnas del tablero que se utiliza como simulador del espacio en bodega y convertirla en una entrada lineal vectorial para la red ART1.

El algortmo genético que se acopla para modificar los pesos de dicha red, influye directamente entrenándola para reconocer en qué momento se encuentra ocupada o libre alguna posición de la misma.

7.3. Inducción

Cada una de las filas en la matriz de posiciones dentro del panel puede manejarse por medio de una red neuronal independiente, de tal forma que para cada una de ellas hay distintos estados y por lo tanto distintas operaciones que el robot debe realizar. En la primera posición está ocupada la fila, el robot la limpia dejándola libre para poder transitar, en la segunda posición el robot puede ejecutar de igual manera distintas acciones, limpiando si es necesario o avanzar si está libre el paso hasta llegar a la salida.

El robot avanza a través de la matriz descubriendo fila a fila qué patrón de comportamiento es el que más se ajusta a sus operaciones, la red debe ser entrenada previamente para reconocer los moldes pre-existentes en los cuales se apoyará, para realizar las distintas acciones en ese momento los algoritmos genéticos pueden intervenir de manera que se reduzca el tiempo de respuesta.

7.4. Hipótesis

Con el experimento se pretendió comprobar lo siguiente:

A. Utilizando el concepto de redes neuronales puede mantenerse despejado, un camino en el tablero;

B. El uso del algoritmo genético, como medio de entrenamiento de la red neuronal, favorece mejorando la velocidad de respuesta de las redes neuronales.

7.5. Experimentación

7.5.1. Construcción del tablero

El tablero tiene dimensiones de 90cm de largo por 60cm de ancho, está estructurado utilizando cartón como base, forrado con papel construcción color negro para minimizar la respuesta de luz. Sobre él se colocó cada una de las posiciones hechas de papel esmalte dorado para maximizar el retorno de luz, con dimensiones de 15cm de largo por 8cm de ancho.

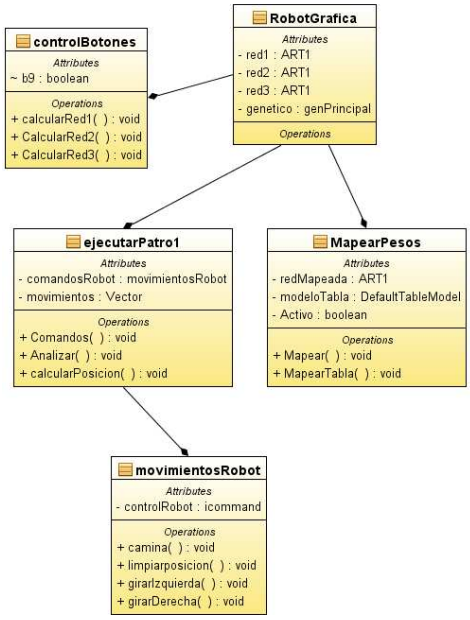
7.5.2. Construcción del robot

Para el robot se utilizaron piezas originales incluidas en el mismo, a través del modelo de construcción Tribot(vehículo), Apéndice 1. Los motores derecho e izquierdo están conectados a los puertos A y B respectivamente, el puerto de motor C no se utiliza y aunque el motor integra al robot, éste no tiene cable que lo una a su respectivo puerto.

7.5.3. Desarrollo de *software*

El *software* toma como base el desarrollo de la red neuronal tipo ART1 y el algoritmo genético expuesto en el capítulo anterior con el agregado en el paquete GUI o interfaz de usuario, el cual debe desarrollarse como se muestra en la figura 21.

Figura 21. **Diagrama de clases para la aplicación del robot**



Fuente: elaboración propia.

La clase robotGrafica: es la clase que contiene la interfaz entre el usuario, el sistema y el robot, une todos los elementos necesarios que integran tanto el sistema híbrido inteligente como los sistemas necesarios para obtener la instrucción del usuario, plasmarla en pantalla y en órdenes para el robot.

En la clase robotGrafica se observa que integra redes de tipo ART1 encargadas de reconocer y almacenar patrones vistos y algoritmos genéticos que realizarán las diferentes funciones de entrenamiento, contiene las distintas tablas botones y todo lo necesario para interactuar con el sistema.

La clase controlBotones: calcula el estado de cada uno de los botones de la interfaz gráfica, dependiendo cual está ocupado, convierte dicha información reconocible a usuario en una reconocible al sistema, de tal forma que represente vectores para introducirlos en las redes neuronales y así procesar la informacion.

La clase MapearPesos: permite tomar las redes que están armadas en memoria armadas y plasmarlas para que el usuario pueda observarlas y de esta manera comprenda mejor la información que contienen las redes y los modelos de tablas sobre los cuales se trabaja para conseguir resultados óptimos.

La clase ejecutar patrón: recibe los vectores de entrada para las distintas redes, por medio de sus métodos analiza y revisa si el patrón existe previamente, de ser así automáticamente realiza la acción con la cual se asoció.

Si el patrón es desconocido se revisa el vector de salida y a partir de éste se decide la acción. Si la red que se analiza es la red uno y el vector de salida indica que la primera posición está ocupada, se dispara la acción de limpiar la primera posición y se asocia con la salida de la red, de tal forma que la próxima vez que dicho patrón se reconozca se dispare una acción determinada; si la red analizada es la red dos y la siguiente posición a la derecha no se encuentra ocupada, simplemente el robot se posiciona en ese espacio y se asocia la salida con dicha acción.

Al analizar la red tres el robot busca siempre la salida, por lo tanto limpiará cualquier posición que se interponga entre el y la salida.

La clase movimientosRobot: Importa la librería *icommand* encargada de controlar el robot lego *mindstorms* por medio de java en forma remota, esta librería contiene los métodos para cada uno de los motores *forward()* que mueven los motores hacia adelante, *backward()* que los mueve hacia atrás, *stop()* que los detiene.

Contiene el método pausa(n) a cargo de detener la compilación durante n milisegundos para que el robot pueda moverse; para girar a la izquierda o derecha se detiene alguno de los motores, de modo que el robot pueda girar hacia el lado seleccionado.

7.5.4. Recolección de datos

A continuación se presentan los datos recolectados durante distintos experimentos, utilizando los sistemas descritos anteriormente.

Las redes neuronales incluidas en el experimento se inicializaron de la manera como se muestra en la tabla XIV, en la tabla XV se muestran los datos al momento de utilizar la red neuronal ART1 de forma individual, en la tabla XVI incluye datos de la utilizacion un algoritmo genético para entrenar la red ART1, dicho de otra manera utilizando el sistema híbrido ART1+AG.

Tabla XIV. **Valores iníciales de red**

Variable	**Valor inicia**
A1	1,0
B1	1,5
C1	5,0
D1	0,9
L1	3,0
Ro	0,9
M	3,0
N	5,0
GF0	0,0

Fuente: elaboración propia.

Tabla XV**.** **Entrenamiento de red**

Entrada	**Salida**	**Patrón**	**Pesos**
1-0-0	1-0-0	0	1-0-0 0,5-0,5-0,5 0,5-0,5-0,5
1-1-0	1-0-0	0	1-0-0 0,5-0,5-0.5 0,5-0,5-0,5
0-1-0	0-1-0	1	1-0-0 0-1,0-0 0,5-0,5-0,5
0-1-1	0-1-0	1	1-0-0 0-1,0-0 0,5-0,5-0,5
0-0-1	0-0-1	2	1-0-0 0-1.0-0 0-0-0,1

Fuente: elaboración propia.

Tabla XVI. **Entrenamiento red utilizando algoritmo genético**

Entrada	Salida	Patrón	Pesos
1-0-0	1-0-0	0	1-0-0 0,5-0,5-0,5 0,5-0,5-0,5
1-1-0	1-0	0	1-0-0 0,5-0,5-0,5 0,5-0,5-0,5
Algoritmo	0-1-1	1	1-0-0 0-1-1 0,5-0,5-0,5
0-0-1	0-0-1	2	1-0-0 0-1-0 0-0-1
0-1-0	0-1-0	1	1-0-0 0-1-0 0-0-1

Fuente: elaboración propia.

A continuación se muestra un segundo caso utilizando el entrenamiento por algoritmo genético, tabla XVII.

Tabla XVII. **Segundo caso entrenamiento con AG**

Entrada	**Salida**	**Patrón**	**Pesos**
1-0-0	1-0-0	0	1-0-0 0,5-0,5-0,5 0,5-0,5-0,5
1-1-0	1-0	0	1-0-0 0,5-0,5-0,5 0,5-0,5-0,5
(algoritmo)	0-0-1	1	1-0-0 0-0-1 0,5-0,5-0,5
0-0-1	0-0-1	1	1-0-0 0-0-1 0,5-0,5-0,5
0-1-0	0-1-0	2	1-0-0 0-0-1 0-1-0

Fuente: elaboración propia.

Se muestra tambien una serie de salidas tomadas del algoritmo genético al momento de entrenar la red, la asociamos con un número o patrón que aumenta cada vez que aparezca uno nuevo, como puede observarse en la tabla XVIII.

Tabla XVIII. **Salidas de algoritmo genético**

Prueba	**Salida**	**Patrón**
1	0-0-1	1
2	0-0-1	1
3	1-0-0	2
4	0-0-0	3
5	0-1-1	4
6	0-0-1	1
7	1-0-1	5
8	0-0-0	3
9	1-0-1	5
10	0-0-1	1

Fuente: elaboración propia.

A continuación se resalta la efectividad del robot para ejecutar las tareas propuestas con diferentes configuraciones. La columna prueba nos indica qué número de prueba se realizará, la columna patrón la configuración de la red que se desea probar, la columna red el número de red que se está probando y la columna efectivo 1 = si es efectivo y 0= no efectivo esto se muestra en la tabla XIX.

Tabla XIX. **Experimento sobre ejecución de acciones**

Prueba	**Patrón**	**Red**	**Efectivo**
1	1-0-0	1	1
2	0-1-0	1	1
3	0-1-0	2	1
4	0-1-0	3	1
5	0-0-1	1	1
6	0-0-1	2	1
7	0-0-1	3	1
8	0-1-1	1	1
9	0-0-1	2	1
10	0-0-1	3	1
11	0-0-1	1	1
12	1-1-0	2	1
13	0-0-1	3	1

Fuente: elaboración propia.

7.6. Análisis de resultados

En los resultados del primer experimento, entrenar la red sin utilzación de algoritmos genéticos, podemos ver que las redes se comportaron tal y como la teoría lo indica. Al ingresar el primer patrón 1,0,0 automáticamente fue reconocido por la red y puesto a disposición para su uso en el tiempo o bien recordado por la red, se le asignó un número de patrón igual a cero, de modo que cuando se ingresó el segundo patrón 1,1,0 la red reconocía que coincidía efectivamente con el patrón anterior, de forma que la salida para este patrón fue igual que el anterior 1,0,0, lo asoció con el número de patrón igual a cero.

Esto es conveniente ya que el patrón para la primera red se asocia con limpiar la posición, de forma que sin importar si la entrada es 1,0,0 o bien 1,1,0 el robot debe limpiar dicha posición.

En la siguiente entrada 0,1,0 vemos que la red lo reconoció como un patrón distinto ya que no inicia como el anterior y el robot debe hacer algo diferente, de este modo se sabe que la primera posición está limpia y con espacio libre, por lo tanto se debe dirigir a la siguiente posición.

La entrada 0,1,1 es un patrón asociado con el anterior así que el robot debe hacer lo mismo que antes, la red lo reconoce como salida 0,1,0 y como patrón 1 por lo tanto se ha logrado la meta de reconocer los distintos patrones. Para la última entrada 0,0,1 la salida es un nuevo patrón, la orden del robot será llegar hasta la segunda posición y de allí avanzar o limpiar esperando lo que viene en la siguiente red.

Puede decirse con certeza que las redes se han comportado como se esperaba hipotéticamente al comprobar la utilidad de las mismas, reconociendo patrones y adaptando entradas previamente establecidas.

Se generaron dos experimentos distintos para comprobar la utilidad del entrenamiento con algoritmos genéticos, en cada uno de ellos se pudo comprobar su utilidad. En el primero se generó un peso necesario para poder generar automáticamente los siguientes que serían autopoiéticos, sin la necesidad de una entrada exterior. Para funcionar de esta manera se lograría entrenar la red sin necesidad de nuevas entradas sino simplemente el robot se entrenaría a sí mismo al ejecutar el algoritmo, luego del primer ingreso los pesos se conformaron 0-1-1.

Lo anterior puede parecer que no se acopla a ninguno de los patrones observados al ejecutar la red sin la ayuda de algrotimos genéticos pero vemos que se genera el patrón 1 y cuando damos la nueva entrada 0-0-1 los pesos del patrón anterior son reducidos a 0-1-0 sin necesidad de intervención humana, debido a las propiedades de la red ART1. El experimento logra generar un peso sin intervención, de modo que para los distintos pesos que surgen la red se auto acomoda logrando un funcionamiento óptimo.

En el segundo experimiento se observa que el peso sináptico generado es el patrón 0-0-1 que satisface las condiciones necesarias de un peso, cuando aparece la siguiente entrada 0-0-1 es reconocida como patrón que previamente se ha encontrado. En general, podemos decir que utilizar algoritmos genéticos para el entrenamiento de la red puede ser exitoso, en algunas ocasiones.

Aunque en otras puede llevar a ampliación del tiempo que le conduzca a reconocer algún patrón ya que gasta tiempo en reacomodarse, en algunas ocasiones y considerando que el algoritmo genético contiene un alto grado de limitaciones para tratar de satisfacer las restricciones establecidas, puede llegar a generar hijos que no satisfagan por completo algún experimento.

Esto recaería directamente en un decremento importante del tiempo para entrenar la red, aunque la ventaja más importante se basa en poder autoentrenarse, de esta manera la misma red permanece en constante aprendizaje, inclusive en ambientes únicamente teóricos pero que lograrían disminuir la pérdida de tiempo y recursos del entrenamiento.

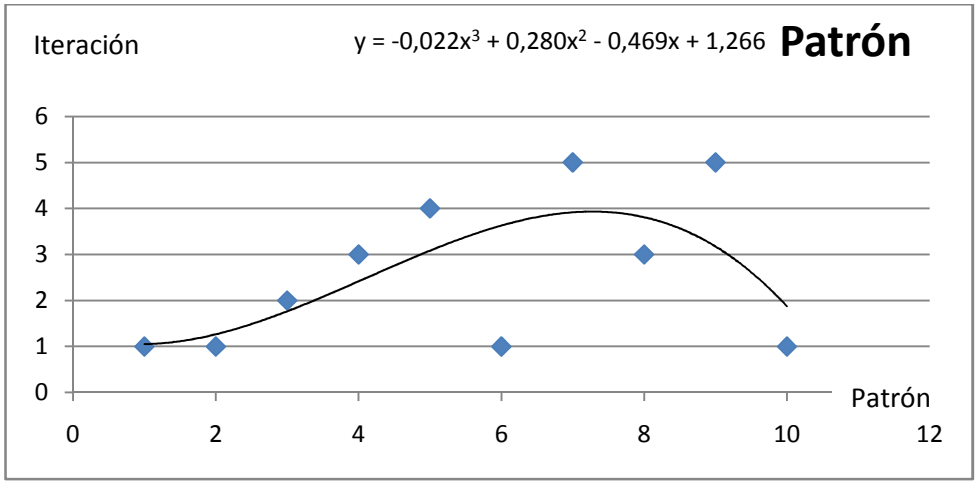
El funcionamiento del robot guiado por la red se obtuvo con éxito, la red fue capaz de reconocer cada uno de los patrones y asociarlos a determinada acción en el tiempo, por lo tanto podemos decir que con un buen entrenamiento llevado a cabo de manera correcta, se puede conseguir el objetivo especificado, mantener un camino transitable dentro de la bodega.

7.7. Propuesta de modelo

A partir de los distintos experimentos realizados podemos lograr un modelo que describa el comportamiento del algoritmo genético que controla la red y de tal forma, predecir el comportamiento de dicho sistema. Se puede inferir si es recomendable o no en determinada ocasión y en ciertas condiciones utilizarlo.

Con los datos mostrados en la tabla XVII, generados por el algoritmo genético, se estructuró una gráfica y el modelo respectivo.

Figura 22. **Gráfica y modelo de generación de datos por medio de algoritmo genético**



Fuente: elaboración propia.

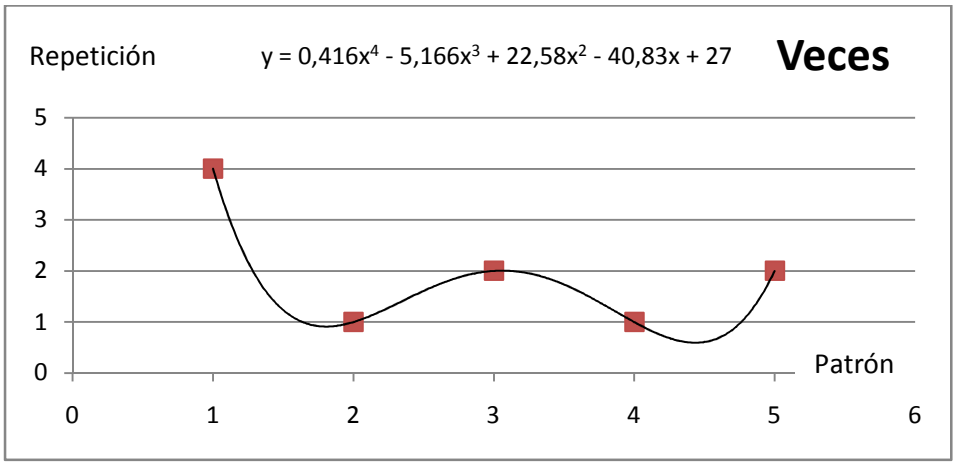
Los datos anteriores se agruparon por cada distinto patrón generado tal como se muestra en la tabla XX. La figura 23 contiene la gráfica perteneciente a dicha tabla, junto a la ecuación que modela el comportamiento de los datos.

Tabla XX. **Repetición en aparición de determinado patrón**

Patrón	Veces repetidas
1	4
2	1
3	2
4	1
5	2

Fuente: elaboración propia.

Figura 23. **Figura y modelo de repetición de patrones en experimento**



Fuente: elaboración propia.

7.8. Conclusión al experimento

En el experimento anterior se comprobó que por medio de la utilización de una red neuronal y algoritmos genéticos, es posible abordar la intervención de un robot organizando una bodega y que al emplear tiempos pequeños la velocidad entre utilizar la red ART1 por separado, o un sistema híbrido, es imperceptible.

8. PRUEBAS DE RENDIMIENTO Y ANÁLISIS DE RESULTADOS

8.1. Del por qué del capítulo

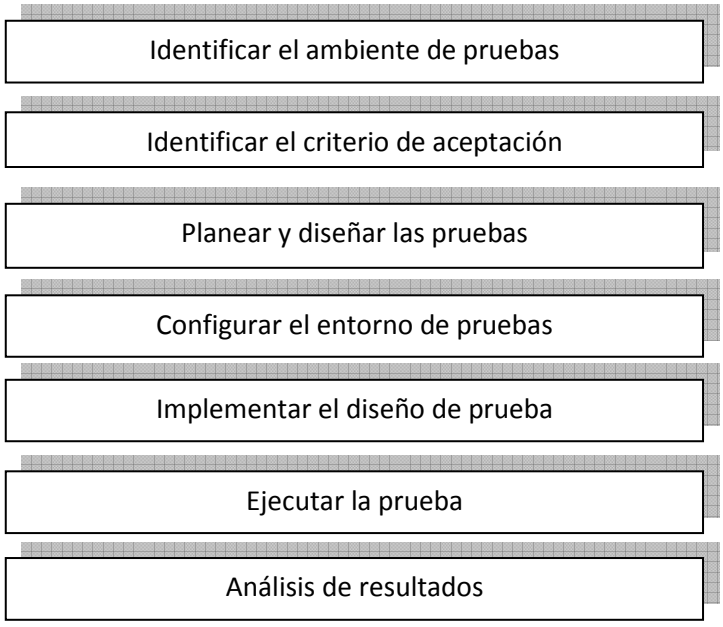
En este apartado se pretende comparar el rendimiento de un sistema de red neuronal tipo ART1 contra uno que además del anterior integra algoritmos genéticos como medio de entrenamiento automatizado; se describe la metodología utilizada para probar el rendimiento de ambas aplicaciones, además los métodos estadísticos del análisis comparativo sobre rendimiento entre ambos sistemas.

8.2. Metodología de pruebas de rendimiento en aplicaciones

En el análisis comparativo de rendimiento entre ambas aplicaciones se utilizó la metodología[6] de pruebas de rendimiento que incluye las actividades descritas en la figura 24.

[6] http://msdn.microsoft.com/en-us/library/bb924356.aspx

Figura 24. **Actividades de metodología de prueba**



Fuente: elaboración propia.

8.3. Ambiente de pruebas

Las pruebas de rendimiento se ejecutaron en un equipo con especificaciones de *hardware* descritas en la tabla XXI, la arquitectura física del sistema de pruebas es sencilla, toda la arquitectura lógica corrio sobre el equipo que se describe.

Tabla XXI. **Especificaciones equipo de pruebas**

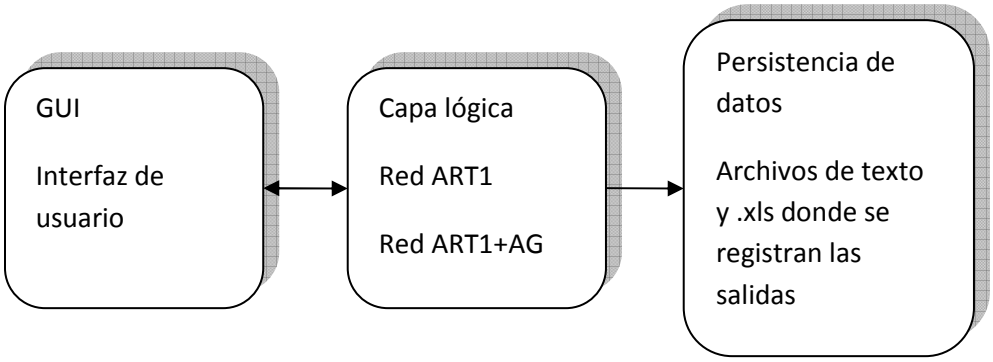
Componente	**Equipo**
Procesador	Intel Pentium core 2 Quad, Q6600
Frecuencia	2,4GHz
Bus	1066 Mts/s
Cache L1	4*32 Kb
Cache L2	8 MB
Memoria RAM	3 GB
Placa Base	Intel DG33BU

Fuente: elaboración propia.

Como se describió en capítulos anteriores el sistema se desarrolló utilizando java, el JDK de *Sun Microsystem*® en su versión 1,6 en cuya ejecución se incluyó el JRE utilizado por *Sun Microsystem*® en su versión 6. Estos sistemas corren bajo el ambiente *Microsoft Windows XP*®, el sistema a probar fue desarrollado específicamente para este trabajo y la arquitectura lógica de la aplicación se describe a continuación.

La aplicación se organiza en tres capas que son: capa de presentación, la interfaz de usuario GUI la cual se encarga de tener una interacción en la que se establecen los parámetros de la prueba que se realiza. La capa lógica que abarca la aplicación de red neuronal y la red neuronal entrenada por medio de un algoritmo genético, descrito en capítulos anteriores. La capa de persistencia de datos se utiliza para guardar los resultados de las pruebas a las cuales se sometieron los sistemas.

Figura 25. **Arquitectura lógica del sistema de pruebas**



Fuente: elaboración propia.

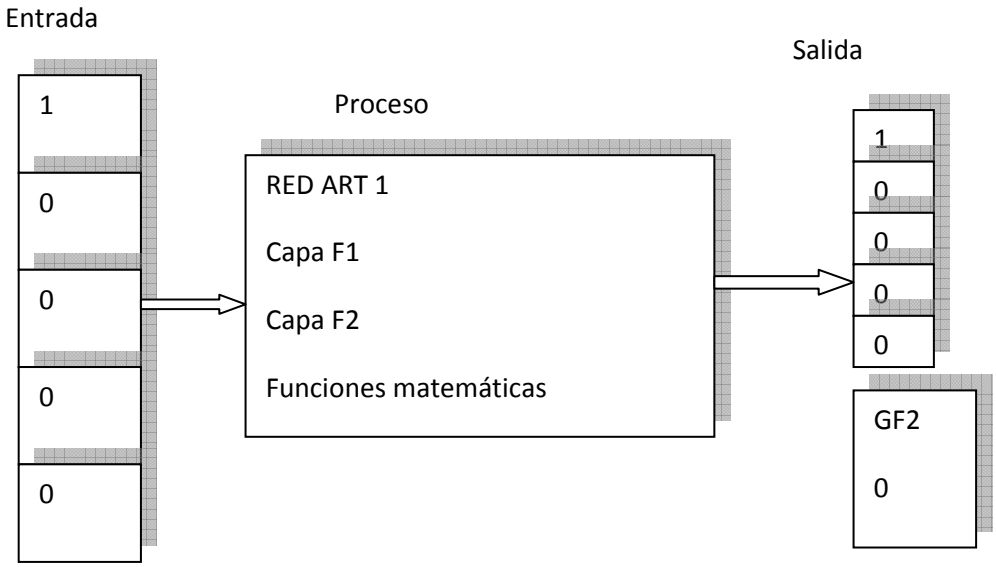
8.4. Criterio de aceptación

Los objetivos principales del desarrollo de las pruebas de rendimiento, tanto para la red neuronal tipo ART1 como para el sistema híbrido ART1+AG, son los siguientes, en cada uno de los sistemas:

1. Encontrar los puntos críticos para el número de entradas al sistema;
2. Encontrar los tiempos de respuesta para entrenamiento y sus estadísticas asociadas;
3. Encontrar los tiempos de respuesta para determinada entrada;
4. Encontrar la cantidad del recurso actividad de CPU utilizado por el sistema para analizar los datos;
5. Comparar y analizar, desde el punto de vista estadístico, los resultados para los datos de rendimiento tiempos y recursos.

La funcionalidad y arquitectura del sistema a probar se describe detalladamente en capítulos anteriores. En general a la aplicación de red ART1 debe ingresársele un vector de entrada binaria, dicho vector es procesado por la red en distintas capas de pesos sinápticos y distintas funciones matemáticas, de modo que el vector de entrada quede registrado en dichos pesos y la próxima vez que se ingrese un patrón parecido o el mismo, pueda reconocerse; como salida se obtiene el patrón reconocido y una etiqueta del mismo.

Figura 26. **Funcionalidad del sistema de red ART1**

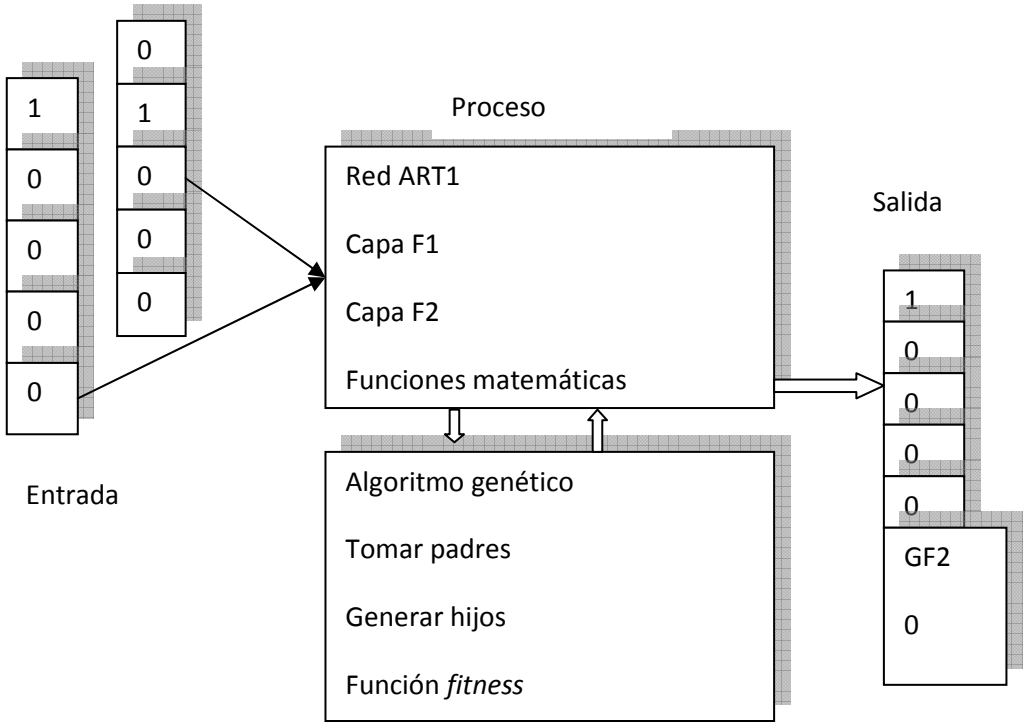


Fuente: elaboración propia.

La aplicación desarrollada, como tema central de este trabajo, combina la funcionalidad de la red ART1 con un sistema de algoritmo genético descrito a detalle anteriormente.

En esta se deben ingresar inicialmente dos o más vectores binarios y a partir de este punto el sistema puede recibir más vectores de entrenamiento externo y/o generar, de manera autopoiética, nuevos pesos de forma que si en el futuro se ingresa un vector de entrada que en determinado momento ha sido generado sistematizadamente por medio de un algoritmo genético, la entrada se salte ciertos pasos y en lugar de reconocer dicho patrón el algoritmo de red ART1 lo reconozca como un patrón que previamente se ha ingresado, ahorrando de este modo tiempo y recursos del equipo.

Figura 27. **Funcionalidad del sistema híbrido ART1+AG**



Fuente: elaboración propia.

Estas pruebas tuvieron como limitación que son específicamente desarrolladas para este trabajo, en una única red alimentada por distinta cantidad de datos en varias pruebas en las cuales se comprueba el número máximo de unidades que puede almacenar, los tiempos de respuesta y recursos que consume.

Se midieron los distintos tiempos en milisegundos por la rapidez en que se ejecutan los procesos, unidades de vectores ingresados al sistema, el porcentaje de uso de CPU en cada uno de los sistemas, la cantidad de *Bytes* y el porcentaje de uso del recolector de basura de java.

Tabla XXII. ***Codebook* métricas utilizadas para medir el rendimiento**

Variable	**Id**	**Descripción**
Unidades	U	Cantidad de entradas a la red
Milisegundos	TimeResp	Tiempos de respuesta de sistemas
UsoCPU	CPUU	Indica el porcentaje de uso de CPU
Bytes	B	Bytes usados por el sistema
patronReconocido	PR	Patrón reconocido con éxito 1= correcto 0= incorrecto
tipoRed	Tr	1= red ART 0= red ART+AG
Experimento	Nexp	1= Experimento 1 2= Experimento 2 3= Experimento 3 4= Experimento 4 5= Experimento 5

Fuente: elaboración propia.

8.5. Planeación y diseño de pruebas

Las pruebas se desarrollaron en un ambiente controlado. Para esto se estructuró una aplicación como interfaz de usuario y de persistencia de datos que es capaz de leerlos de manera fácil y rápida, desde un archivo de hoja de cálculo, y escribir la salida hacia otro a continuación se muestra la forma como se desarrollaron las pruebas.

- Prueba de stress*:* inicialmente se ejecutó la prueba de *stress* o punto de quiebre en la cual se introducen los datos a la red, iniciando con cinco y aumentando cinco unidades cada vez que se ejecuta una nueva iteración del sistema.

  Esta prueba termina al momento en que el sistema muestra algún error ejecutando la iteración, luego de tomar los datos necesarios como tiempo y recursos consumidos por cada una de las iteraciones. Después se calculan los parámetros estadísticos asociados como medias, mínimos, máximos, etc. También se muestran los gráficos de los distintos avances en las pruebas y las reacciones de la red ante los conjuntos de datos.

- Entrenamiento lineal: luego de comprobar el máximo de unidades que pueden ser ingresadas a la red, esta se entrena linealmente para observar el tiempo y recursos gastados al momento de ser entrenada, ingresando cada patrón en forma ordenada, observándose las estadísticas y graficas respectivas.

  Reconocimiento de patrones: después de ser entrenada de forma lineal, la red somete a una cantidad del doble de la máxima capacidad de entradas y se observándose los tiempos y recursos en cada una de las unidades ingresadas para encontrar las medias, máximas, mínimos y demás estadísticos pertinentes, así como gráficos.

- Entrenamiento con AG: la siguiente prueba consiste en entrenar linealmente intercalando cada cinco unidades una iteración de entrenamiento, con algoritmo genético, se encuentran los estadísticos pertinentes asociados a los recursos, y se mostraran las graficas conteniendo datos.

- Reconocimiento de patrones*:* se toman los mismos datos ingresados a la red neuronal entrenada linealmente integrándose a la red mixta entrada, se obtienen datos de tiempo y recursos a los que se le estiman los estadísticos pertinentes, junto a sus respectivas gráficas.

  En los experimentos que se consideran paralelos se observa si los datos obtenidos tienden a ser normales o no, en el primer caso se aplicó la prueba paramétrica de *T student* para determinar la diferencia entre los mismos, de lo contrario se aplica la prueba U de *Mann-whitney* la cual es no paramétrica.

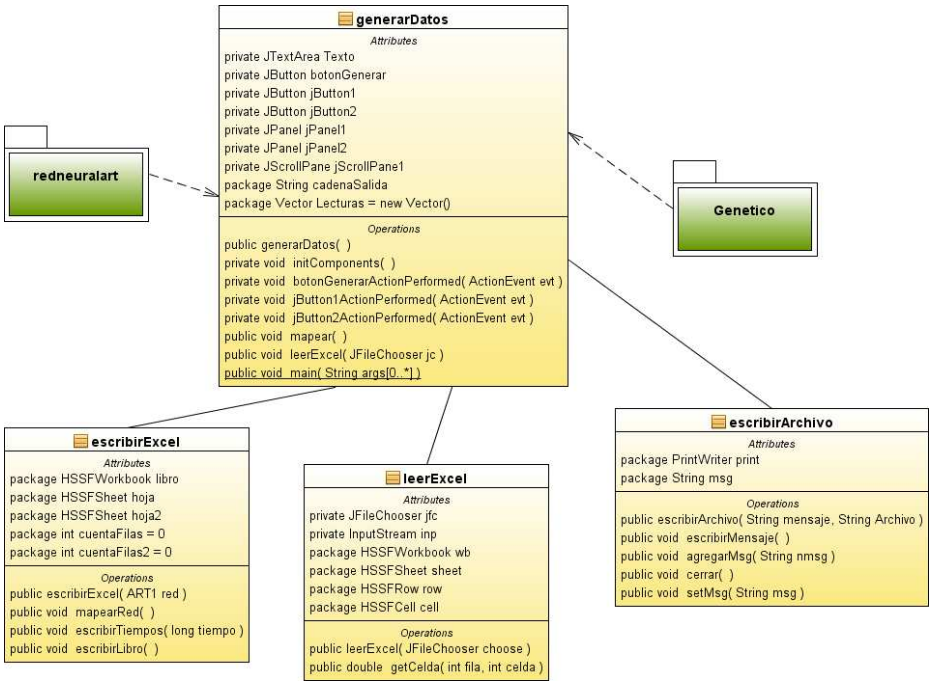
Los experimentos considerados paralelos son el entrenamiento lineal y el entrenamiento por red neuronal, la prueba es la de ingresar unidades aleatorias a la red entrenada linealmente y a la entrenada con algoritmos genéticos.

8.6. Configuración del entorno de prueba

8.6.1. Configuración de herramienta de generación de datos

En la prueba se utiliza un pequeño sistema diseñado específicamente para tomar elementos de la red neuronal tipo ART1 y la red ART1+AG, la arquitectura de dicho sistema se presenta en la figura 28.

Figura 28. **Arquitectura de aplicación para obtener datos**



Fuente: elaboración propia.

La aplicación tiene su interfaz gráfica llamada generaDatos en la cual se utilizan los elementos de la capa lógica compuesta por los paquetes redneuralart y Genético los cuales se describieron en capítulos anteriores.

La clase leerExcel se encarga de abrir un archivo de hoja de cálculo y tomar los elementos que aparecen en ellos utilizando la librería apache poi[7] que es un API de java utilizada para manipular documentos *Microsoft*®, de manera rápida. En ella se toma un libro de Excel en una dirección introducida por el usuario se direcciona a la página 1 e inicia la toma datos celda a celda.

La clase escribirExcel se encarga de anotar las salidas de los sistemas de red ART1 y red ART1+AG a archivos de hoja de cálculo, para ello se utiliza el API apache poi en el cual se escribe un libro y en distintas hojas del mismo, las salidas como capa F1 y F2, salidas generales del sistema.

La clase escribirArchivo genera un archivo de texto donde se muestra que se han inicializado las redes neuronales a entrenar, para cada uno de los experimentos y sus condiciones iníciales.

8.6.2. Configuración de herramientas de monitoreo

Para monitorear los recursos como actividad de procesador que toma el sistema en su funcionamiento, se utiliza el programa visual VM[8] en su versión 1,3,1 en este encontramos la opción CPU *usage* que indica el porcentaje de recurso procesador para determinada aplicación en este momento, *GC activity* indica un porcentaje de actividad del recolector de basura de java y la gráfica *Heap* muestra el tamaño en *Bytes* de la pila, a lo largo del tiempo.

En la pestaña *sampler* encontramos el muestreo del CPU que incluye el tiempo utilizado en milisegundos por el procesador, en cada uno de los hilos del sistema, el botón de memoria nos muestra cuantos *Bytes* están siendo utilizados para cada uno de los tipos de datos que contiene el sistema, a lo largo del tiempo.

[7] http://poi.apache.org/
[8] https://visualvm.dev.java.net/

8.6.3. Configuración de herramientas de análisis

Luego de obtenidos los datos es importante analizarlos para tener una base sobre la cual se sustentan las conclusiones, se utilizan dos programas *Microsoft Excel*® y el sistema estadístico SPSS buscando estadísticas asociadas a los datos. Se presentan gráficas que ayudarán a tener una mejor comprensión de lo que sucedido con cada uno de los sistemas, en cuanto a su funcionamiento y rendimiento.

8.7. Ejecutar prueba

A continuación se muestra el resumen de los resultados obtenidos en distintas pruebas. El detalle del conjunto de datos obtenidos se presentan en el Apéndice 2.

8.7.1. Experimentos

Prueba de stress o punto de quiebre: en esta prueba se trató de determinar la cantidad máxima de unidades que puede albergar y procesar la red ART1. En la primera prueba se tiene una red neuronal inicial de 10x5 la cual aumentó en cinco unidades cada dimensión de la red, se terminó en el momento que la red no respondió de la manera esperada también se registraron los tiempos de respuesta en cada una de las iteraciones para la red que tiene las mismas dimensiones.

Se modificaron los parámetros M y N del constructor de la clase ART1 que se encuentra en el paquete redneuralart con las dimensiones de la red de la manera descrita anteriormente.

En el proyecto datosRed se modificó el procedimiento leerExcel ya que para esta prueba se utilizó el archivo de entrenamiento Entrenamientolienal.xls, mismo que cuenta con datos que sobrepasan las capacidades de cada una de las pruebas, En el procedimiento leerExcel se modificaron los ciclos que se ven afectados directamente por las variables locales i, j las cuales indican cuántos datos se tomaron del archivo de entrada en cada ocasión.

En el primer caso se mantuvo la variación de M cinco unidades por encima de N, en el segundo caso se mantuvo la variación de N cinco unidades por encima de M y para finalizar se variaron ambas unidades al mismo ritmo.

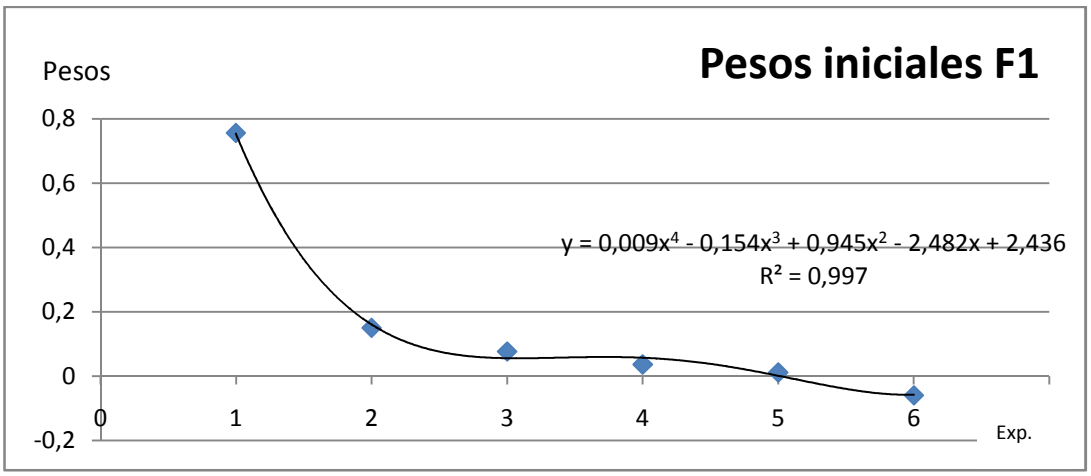
Tabla XXIII. **Resultados experimento punto de quiebre**

EXP1	**EXP2**	**EXP3**	**RESULTADO**
5x10	5x10	5X5	ÉXITO
10x15	15x10	10X10	ÉXITO
15x20	20X15	15X15	ÉXITO
20x25	25X20	20X20	ÉXITO
25x30	30X25	25X25	ÉXITO
30x35	30X35	30X30	FRACASO

Fuente: elaboración propia.

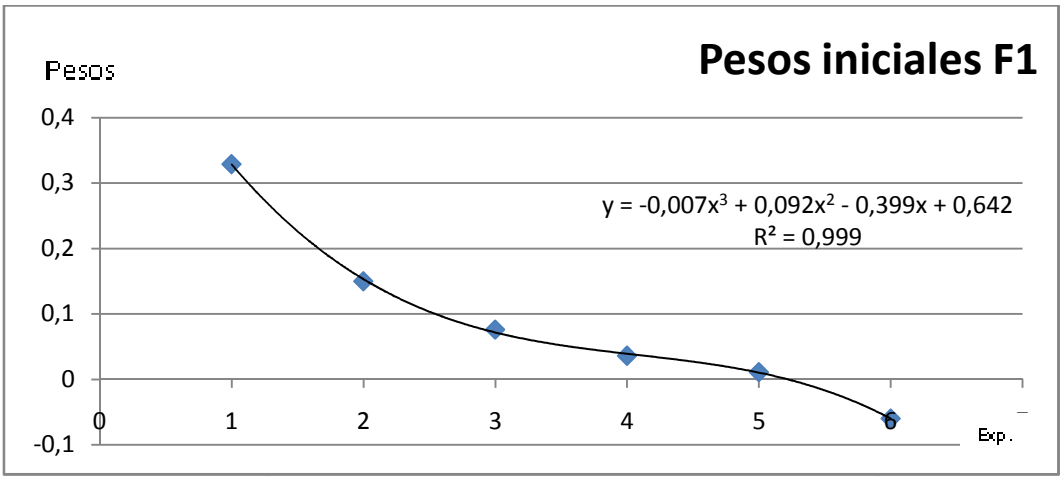
Como se aprecia en la tabla XXIII, para cada uno de los casos, se obtuvo un resultado similar, las redes neuronales se comportaban de manera normal produciendo resultados esperados cuando sus parámetros de entrada rondaban las 25 unidades, luego de este punto las redes producen un resultado errático para cualquier entrada que se desee ingresar. Para obtener más datos de por qué ocurre esto, se analizaron los pesos sinápticos iníciales, presentados para la capa F1 lo que arrojó resultados de los cuales presentamos las gráficas con los datos acompañados de la ecuación que describe su comportamiento. La totalidad de los datos puede verse en Apéndice 2.

Figura 29. **Gráfico comportamiento pesos sinápticos experimento 1**



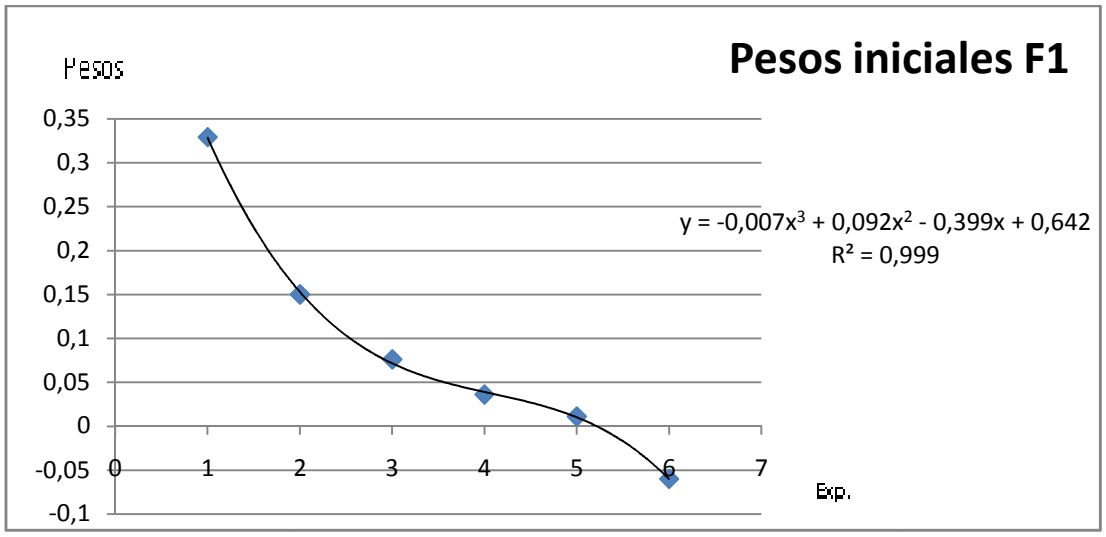
Fuente: elaboración propia.

Figura 30. **Gráfico comportamiento pesos sinápticos experimento 2**



Fuente: elaboración propia.

Figura 31. **Gráfico comportamiento pesos sinápticos experimento 3**



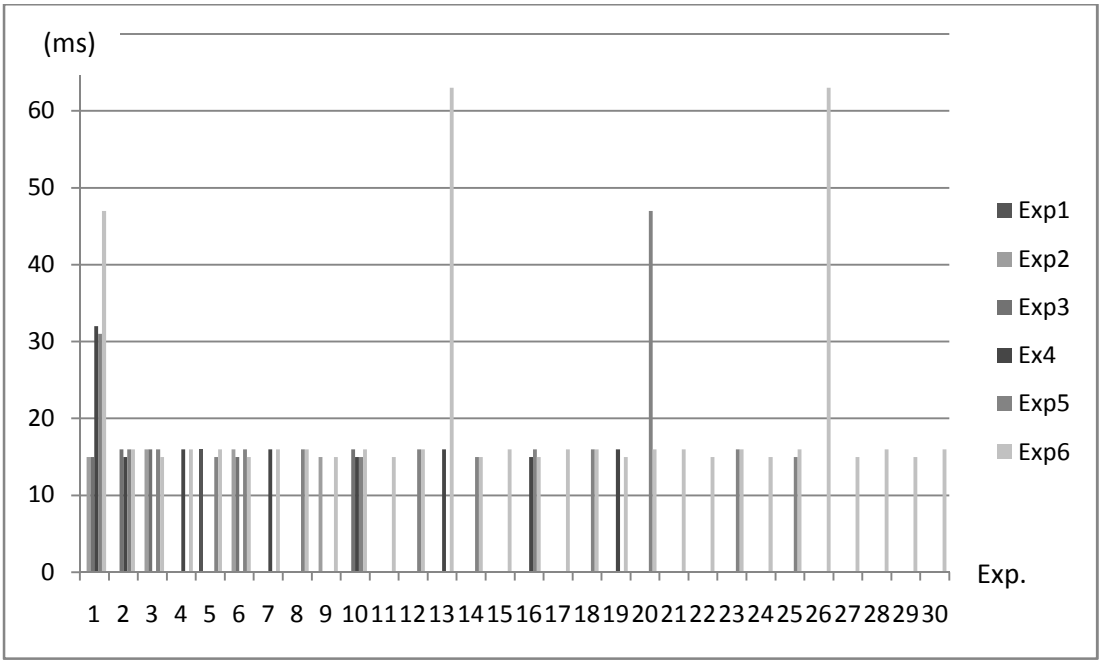
Fuente: elaboración propia.

En los gráficos se observa una tendencia clara; cuando las redes son más grandes y las cantidades de datos aumentan, los valores iníciales de los pesos sinápticos decrecen incluso a convertirse en números negativos. Cuando los valores iníciales son negativos, la red neuronal se comporta de manera errática.

El experimento presentado a continuación se enfoca en medir los tiempos de respuesta para cada iteración, se busca comparar si dichos tiempos tienden a un grado alto de variación según el tamaño de la red lo que se analiza en la figura 32, el tiempo es similar en los distintos experimentos.

Tienen mayor variación al inicio y luego tiende a estabilizarse, para la iteración Exp6 la cual excede el número máximo de iteraciones y nos presenta un comportamiento errático de respuesta, también se puede apreciar que mantiene tiempos altos para cada una de las iteraciones y cuando se le intenta ingresar algún vector de entrada. La serie de datos de tiempos obtenidos puede verse en Apéndice 2.

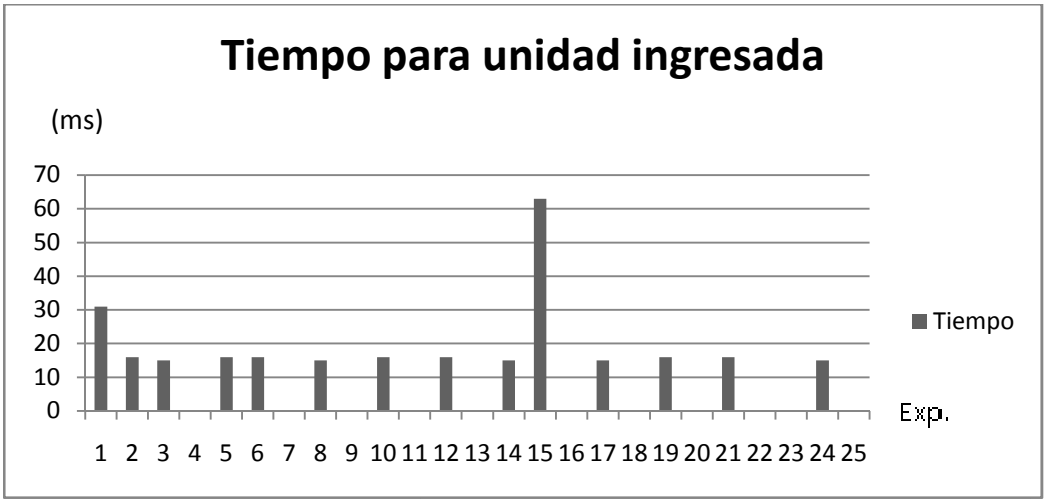
Figura 32. **Gráfico comparación de tiempos en milisegundos de experimentos**



Fuente: elaboración propia.

Prueba de entrenamiento lineal: en la siguiente prueba se preparó una red ART1 de 25x25, teniendo en cuenta, por pruebas anteriores que éste es el mayor número de unidades que se soporta. Se preparó un archivo de entrada de manera que las unidades tuvieran una relación de variar en una posición con el anterior, entrenamiento lineal, esto se puede observar en la figura 33 y tabla XXIV.

Figura 33. **Tiempos de procesamiento para cada unidad ingresada a red ART1**



Fuente: elaboración propia.

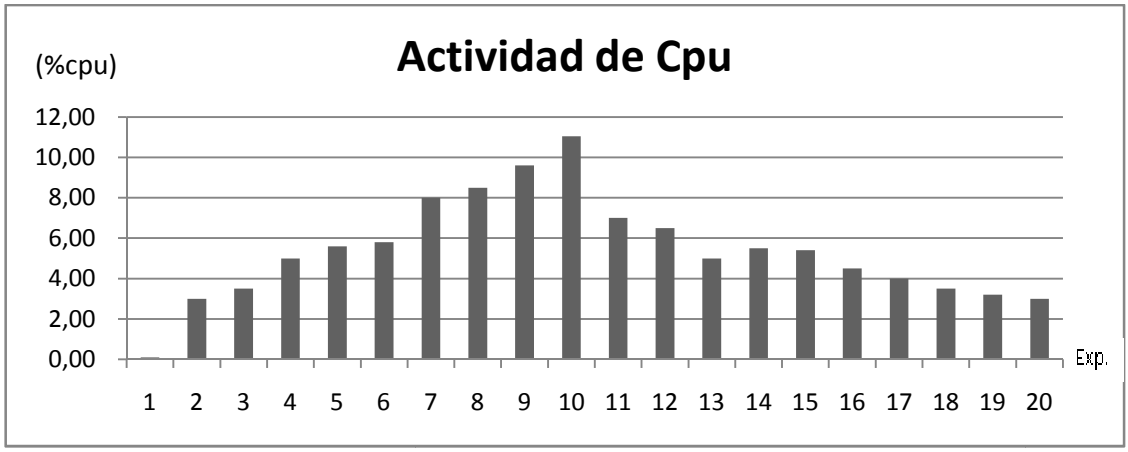
Tabla XXIV. **Resumen estadístico entrenamiento red ART1 lineal**

Resumen	
Media	11,24
Error típico	2,79540098
Mediana	15
Moda	0
Desviación estándar	13,9770049
Varianza de la muestra	195,356667
Curtosis	7,11782981
Coeficiente de asimetría	2,18588593
Rango	63
Mínimo	0
Máximo	63
Suma	281
Cuenta	25

Fuente: elaboración propia.

Con la herramienta visual VM se midió el porcentaje de CPU que utilizaba la aplicación en el momento de procesar las unidades, lo que se presenta en la figura 34.

Figura 34. **Gráfica de uso de CPU durante el entrenamiento lineal**



Fuente: elaboración propia.

Tabla XXV. **Resumen estadístico actividad CPU entrenamiento lineal red ART1**

Resumen Actividad CPU	
Media	5,3875
Error típico	0,57278033
Mediana	5,2
Moda	3
Desviación estándar	2,56155149
Varianza de la muestra	6,56154605
Curtosis	0,47262218
Coeficiente de asimetría	0,37402803
Rango	10,95
Mínimo	0,1
Máximo	11,05
Suma	107,75
Cuenta	20

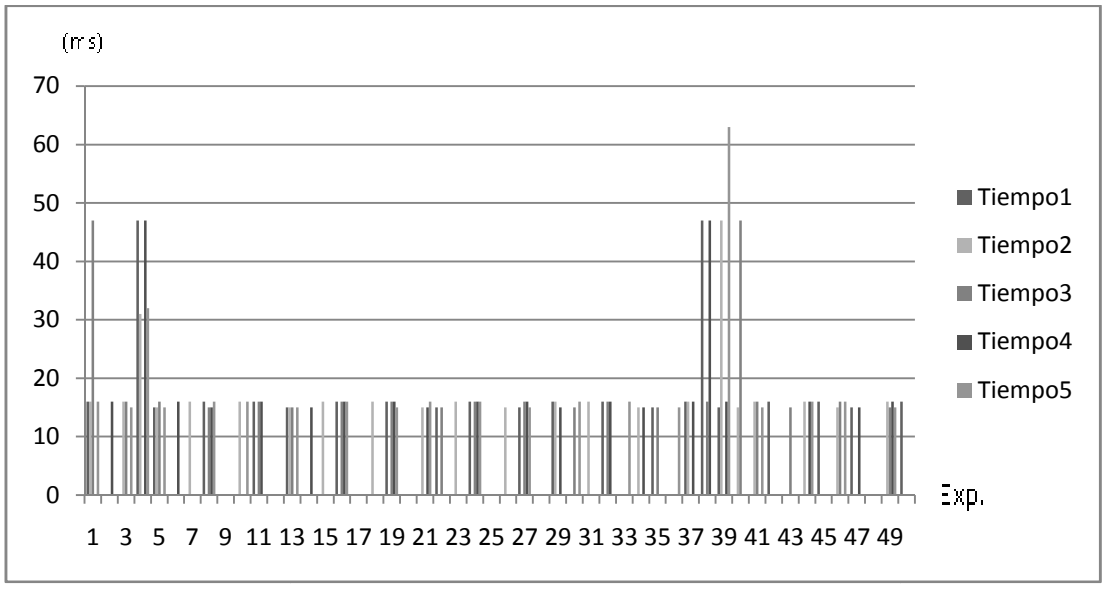
Fuente: elaboración propia.

Experimento reconocimiento de patrones: el objetivo de este experimento es analizar el rendimiento de la red neuronal, al momento de ingresarle una cantidad grande de datos, los cuales ya han sido aprendidos.

Primero se tomó la cantidad de milisegundos que ocupa la aplicación al momento de ingresar cada una de las unidades vectoriales de entrada para las redes neuronales, que ésta presente una salida.

En la gráfica 35 pueden apreciarse los resultados obtenidos en las cinco iteraciones del sistema sobre el conjunto de datos los mismos siguen un patrón similar, como se aprecia en la gráfica.

Figura 35**. Tiempos en ms de respuesta para 50 unidades de entrada a una red ART1**



Fuente: elaboración propia.

En la tabla XXIV se muestra el resumen estadístico de los datos generados, observando que existen medias parecidas para cada uno de los experimentos realizados. El error es relativamente bajo, la moda tiende a cero, puede decirse que el sistema tarda menos de un milisegundo en ejecutar la solución; el principal problema que se observa es que la varianza es alta debido a que existen muchos valores cercanos a cero y los extremos son altos, en la gráfica vemos que los datos afectados visualmente no tienen un comportamiento normal, lo cual se comprueba con los procedimientos respectivos.

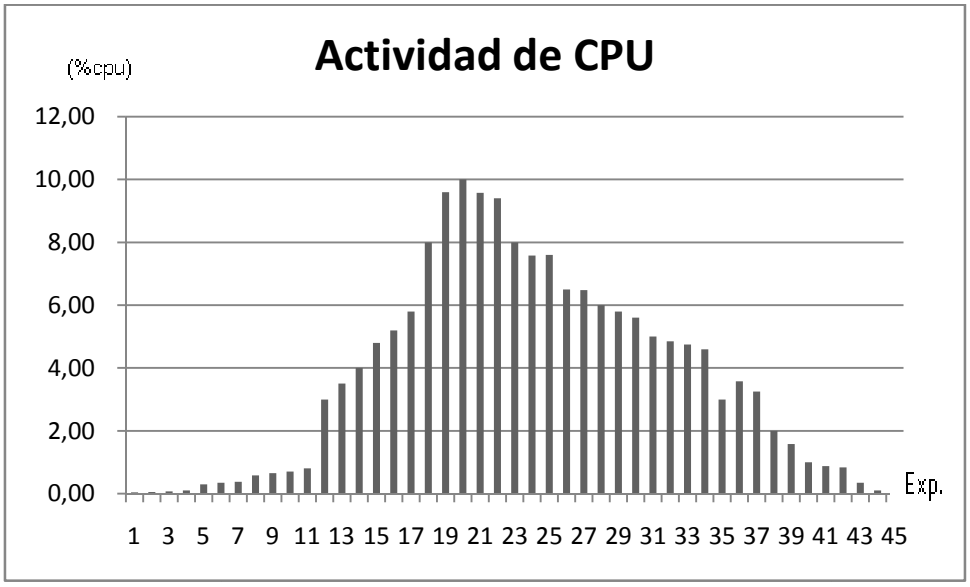
Tabla XXVI. **Resumen estadístico de tiempos en ms de procesamiento para unidad por la red ART1**

Estadísticos	***Prueba1***	***Prueba2***	***Prueba3***	***Prueba4***	***Prueba5***	***Promedio de datos***
Media	7,82	7,82	7,5	7,52	7,5	7,63
Error típico	1,56	1,43	1,56	1,56	1,63	1,55
Mediana	0	0	0	0	0	0
Moda	0	0	0	0	0	0
Desviación estándar	11,07	10,11	11,06	11,07	11,55	10,96
Varianza de la muestra	122,59	102,35	122,37	122,70	133,47	120,25
Curtosis	4,317	3,03	4,57	4,52	9,78	4,83
Coeficiente de asimetría	1,810	1,41	1,88	1,87	2,51	1,87
Rango	47	47	47	47	63	49,83
Mínimo	0	0	0	0	0	0
Máximo	47	47	47	47	63	49,83
Suma	391	391	375	376	375	381,52
Cuenta	50	50	50	50	50	50

Fuente: elaboración propia.

En las pruebas se observó también el comportamiento del uso en porcentaje del CPU que se describe en la figura 36 destacando una tendencia a la normalidad de los datos teniendo su pico en el uso del 10% del CPU.

Figura 36. **Actividad de CPU prueba ingreso de unidades aleatorias**



Fuente: elaboración propia.

Tabla XXVII. **Resumen estadístico Actividad CPU prueba red ART1**

Resumen pruebas red ART1	
Media	3,694
Error típico	0,47331588
Mediana	3,5
Moda	0,1
Desviación estándar	3,17509943
Varianza de la muestra	10,0812564
Curtosis	-1,03365088
Coeficiente de asimetría	0,43731503
Rango	10
Mínimo	0
Máximo	10
Suma	166,23
Cuenta	45

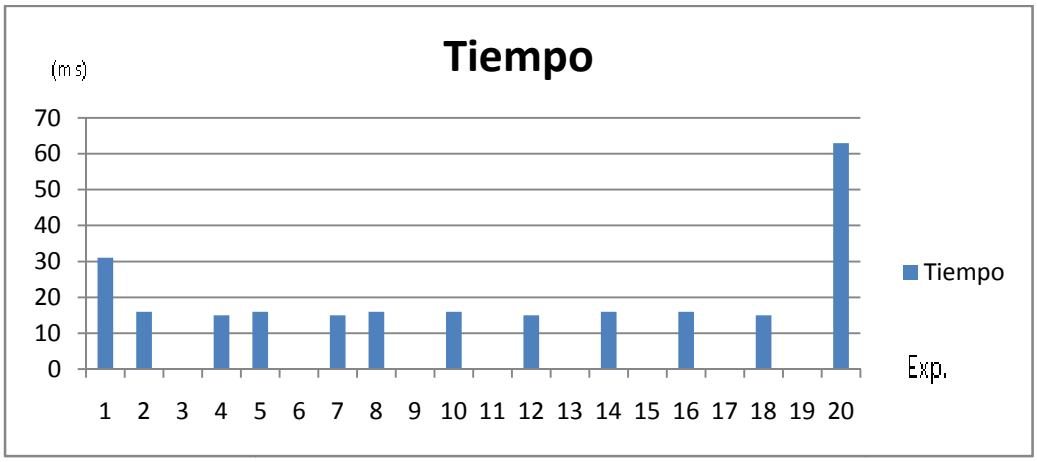
Fuente: elaboración propia**.**

Prueba de entrenamiento lineal ART1+AG: en esta prueba se busca tomar datos de la respuesta del sistema ART1, cuando se incluye un entrenamiento por medio de algoritmo genético.

Para ello se tomó la red neuronal de 25x25 en sus dimensiones y los datos que se utilizaron para entrenar la red neuronal linealmente, con la salvedad que cada cinco unidades se agregó una iteración de algoritmo genético.

En la figura 37 se observa una gráfica de comportamiento con los tiempos que el sistema ART1+AG utiliza para procesar una entrada.

Figura 37. **Tiempos de respuesta en ms para unidades ingresados al sistema ART1+AG**



Fuente: elaboración propia.

Tabla XXVIII. **Resumen estadístico de experimento tiempos de rendimiento sistema ART1+AG**

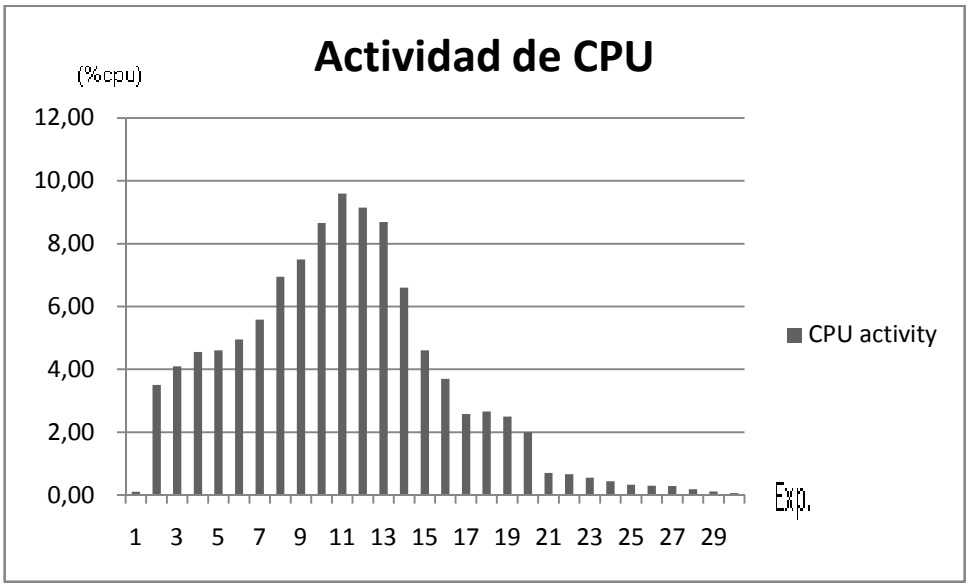
Experimento	
Media	12,5
Error típico	3,34073739
Mediana	15
Moda	0
Desviación estándar	14,9402318
Varianza de la muestra	223,210526
Curtosis	6,34531386
Coeficiente de asimetría	2,10106481
Rango	63
Mínimo	0
Máximo	63
Suma	250
Cuenta	20

Fuente: elaboración propia.

Al observar la figura 37 se aprecia que los tiempos se mantuvieron con un mayor grado de estabilidad, levemente tiende a la normalidad. La media de tiempo se colocó en cerca de 12 milisegundos y la varianza de la muestra indica que existe una distancia considerable entre cada uno de los tiempos y la media por lo que ésta puede considerarse que no es un referente con alto grado de confiabilidad.

Se calculó también el porcentaje de CPU que ocupa la aplicación observando un pico de 9,6% del total respecto al uso del CPU. Figura 38.

Figura 38. **Actividad de CPU red ART1+AG entrenamiento lineal**



Fuente: elaboración propia.

Tabla XXIX. **Resumen actividad CPU Entrenamiento lineal red ART1+AG**

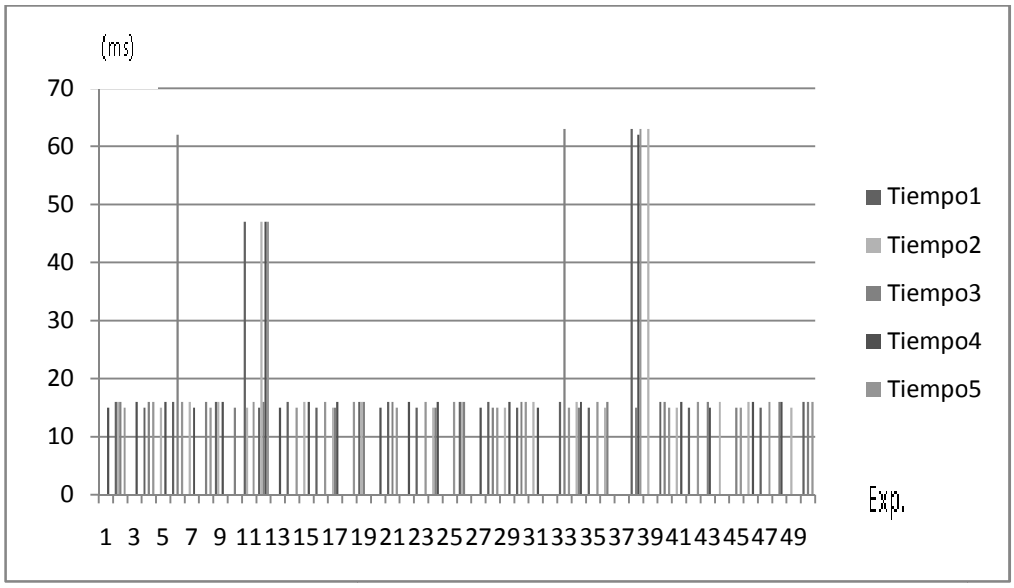
Resumen Actividad CPU	
Media	3,53966667
Error típico	0,57313987
Mediana	3,08
Moda	4,6
Desviación estándar	3,13921633
Varianza de la muestra	9,8546792
Curtosis	-0,97032461
Coeficiente de asimetría	0,52938515
Rango	9,54
Mínimo	0,06
Máximo	9,6
Suma	106,19
Cuenta	30

Fuente: elaboración propia.

Prueba de reconocimiento de patrones ART1+AG: esta prueba busca ingresar una cantidad grande de datos los cuales son similares a los ingresados en la prueba de red que se realizó utilizando únicamente el sistema ART1, también busca observar el rendimiento y la respuesta del sistema en cuanto a recursos como tiempo y procesador, utilizados para las entradas.

En la figura 39 se puede apreciar la gráfica que muestra el tiempo en milisegundos, ocupado por el sistema red ART1+AG, para procesar una entrada.

Figura 39. **Gráfico prueba de Ingreso aleatorio a sistema ART1+AG**



Fuente: elaboración propia.

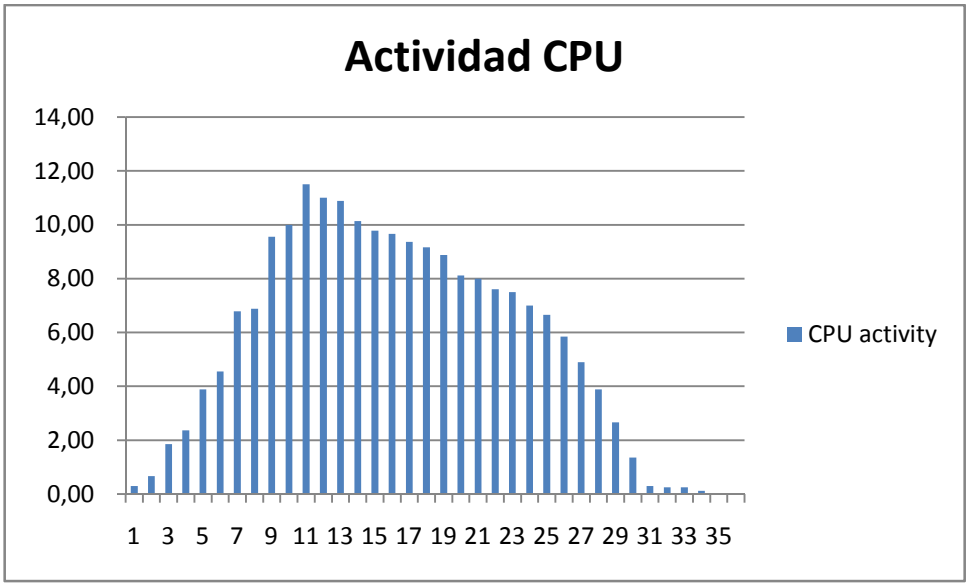
La tabla XXVI contiene el resumen estadístico de los distintos experimentos realizados con los tiempos de respuesta para la red ART1+ AG, resaltando que los tiempos medios se mantuvieron estables y la varianza de la muestra es muy pequeña por lo que puede decirse que hubo mayor estabilidad en el procesamiento de datos.

Tabla XXX. **Resumen estadístico pruebas de respuesta sistema ART1+AG**

Estadístico	*tiempo1*	*tiempo2*	*tiempo3*	*tiempo4*	*tiempo5*	*Media*
Media	8,12	7,80	8,76	8,12	8,44	8,24
Error típico	1,74	1,75	1,90	1,74	1,75	1,77
Mediana	0	0,00	0,00	0,00	0,00	0,00
Moda	0	0,00	0,00	0,00	0,00	0,00
Desviación estándar	12,37	12,37	13,46	12,29	12,36	12,56
Varianza de la muestra	153,01	152,98	181,21	151,09	152,82	157,46
Curtosis	8,18	8,46	8,68	7,82	7,94	8,20
Coeficiente de asimetría	2,43	2,50	2,60	2,38	2,37	2,45
Rango	63,00	63,00	63,00	62,00	63,00	62,80
Mínimo	0,00	0,00	0,00	0,00	0,00	0,00
Máximo	63,00	63,00	63,00	62,00	63,00	62,80
Suma	406,00	390,00	438,00	406,00	422,00	411,76
Cuenta	50,00	50,00	50,00	50,00	50,00	50,00

Fuente**: e**laboración propia.

Figura 40. **Actividad de CPU al utilizar sistema ART1+AG**



Fuente: elaboración propia.

Tabla XXXI. **Resumen estadístico actividad CPU pruebas red ART1+AG**

Resumen actividad CPU	
Media	5,60083333
Error típico	0,65107462
Mediana	6,725
Moda	0,3
Desviación estándar	3,90644769
Varianza de la muestra	15,2603336
Curtosis	-1,46191698
Coeficiente de asimetría	-0,18522302
Rango	11,5
Mínimo	0
Máximo	11,5
Suma	201,63
Cuenta	36

Fuente: elaboración propia.

En la figura 40 se observa el comportamiento de las actividades del CPU en porcentaje, el pico máximo es de cerca del 12% del CPU destinado para realizar el reconocimiento de patrones de manera aleatoria estando previamente la red entrenada, con ciertos patrones y un algoritmo genético.

8.7.2. Análisis estadísticos

Luego de haber obtenido los datos necesarios de los sistemas, tanto de la red neuronal ART1 como del sistema ART1+AG, se procedió a realizar distintas pruebas estadísticas para determinar la variación entre los experimentos, de tal manera que se puede asegurar que los llevados a cabo con parámetros idénticos en distintas iteraciones, arrojaron resultados coherentes y consistentes sin una variación indicativa de que el sistema es caótico y no puede establecerse un parámetro de referencia para observar el rendimiento.

Además que los experimentos que difieren a sistema que utilizan tengan una variante definida que indique que el uso o no de un algoritmo genético para entrenar la red puede tener inferencia en el desempeño del sistema.

8.7.2.1. Análisis para experimento de entrenamiento lineal

Como primera prueba se estimó la normalidad de cada uno de los experimentos, el realizado con la red ART1 y el correspondiente a la red AR1+AG, luego se compararon ambos para observar si existía variante pronunciada entre cada uno. Se realizó el test de normalidad con un nivel de significancia de 0,05, como puede observarse en la tabla XXXII.

Tabla XXXII. **Prueba de Normalidad para entrenamiento lineal**

Prueba de Normalidad

		Kolmogorov-Smirnov			Shapiro-Wilk		
	TipoRed	*Statistic*	df	Sig.	*Statistic*	df	Sig.
TimeResp	ART1ag	0,307	20	0,000	0,709	20	0,000
	Art	0,287	25	0,000	0,692	25	0,000

Fuente: elaboración propia.

Considerando que la muestra no supera los 50 datos, se usó la prueba de Shapiro-Wilk en la que se observa la columna Sig de la tabla XXII la cual indica que el nivel de significancia es igual a cero, menor al nivel de significancia 0,05 propuesto, lo que afirma que los datos no tienden a la normalidad.

Como los datos no tienden a la distribución normal se comparan utilizando la prueba U de Mann-Whitney, de esta prueba se obtiene la hipótesis

Lo anterior significa que no hay diferencia entre las medianas de cada grupo, y entre los tiempos de respuesta de ambos experimentos y $H_1: Me_M \neq Me_F$; hay diferencia en las medianas de cada grupo, por lo tanto los tiempos de respuesta son distintos. Los resultados de la prueba se muestran en la tabla XXXIII.

Tabla XXXIII. **Prueba Mann-Whitney para tiempos de entrenamiento lineal**

Prueba de Mann-Whitney

	TimeResp
Mann-Whitney U	237,000
Wilcoxon W	562,000
Z	-0,314
Asymp. Sig. (2-tailed)	0,753

a. Variable de Agrupacion: TipoRed

Fuente: elaboración propia.

En la tabla XXXIII se presenta los resultados de la prueba Mann-Whitney en la cual podemos observar que el valor de U es de 237 al cual se estandariza y se obtiene un valor z igual a -0,314 al cual corresponde una probabilidad, error tipo uno de 0,753 y por lo tanto se rechaza la hipótesis nula, de modo que podemos afirmar con certeza que hay una variación importante entre ambos grupos.

Luego se analizó el comportamiento de la actividad de CPU para el entrenamiento lineal de una red ART1 y un sistema ART1+AG, primero para observar si su comportamiento corresponde a la distribución normal y luego hacer una comparación entre las respuestas de ambos sistemas y observar si existe una diferencia importante entre ambos.

Primero se analizaron los datos para determinar si las distribuciones de ambos experimentos se ajustan a la distribución normal con un nivel de significancia igual a 0,05.

En la figura XXIV se puede apreciar los resultados de la prueba de normalidad para la cual se utiliza Shapiro-Wilks puesto que tenemos menos de 50 datos.

Al observar los niveles de significancia tenemos que para los tiempos de procesador de ART1 tiende a una distribución normal ya que su nivel de significancia es de 0,837 mayor al nivel de significancia propuesto pero ART+AG es de 0,10 y lo cual es significativamente menor al nivel de significancia de referencia y por lo tanto no tiende a la distribución normal, como uno de los experimentos no tiende a la distribución normal se debe utilizar la prueba de U Mann-Whitney para observar si hay diferencia significativas entre el porcentaje de uso de CPU durante la ejecución de cada uno de los experimentos.

Tabla XXXIV. **Prueba de normalidad para Uso de CPU en entrenamiento lineal**

Prueba de normalidad

		Kolmogorov-Smirnov			Shapiro-Wilk		
	TipoRed	*Statistic*	df	Sig.	*Statistic*	df	Sig.
UsoCpu	ARTag	0,173	29	0,026	0,900	29	0,010
	ART	0,140	19	0,200*	0,973	19	0,837

Fuente: elaboración propia.

Al tener en cuenta que los datos no tienden a la distribución normal se comparan utilizando la prueba U de autopoiético- Whitney, en esta prueba se tiene la hipótesis: $H_0 : Me_M = Me_F$, significa que no hay diferencia entre las medianas de cada grupo ni diferencia significativa entre el porcentaje de CPU utilizado por cada uno de los experimentos y $H_1 : Me_M \neq Me_F$. Hay diferencia en las medianas de cada grupo, por lo tanto el porcentaje de uso de CPU es significativamente distinto en ambos experimentos.

.

Tabla XXXV. **Prueba de Mann-Whitney para porcentaje de uso de CPU en entrenamiento lineal**

Prueba Mann-Whitney

	UsoCpu
Mann-Whitney U	179,000
Wilcoxon W	614,000
Z	-2,035
Asymp. Sig. (2-tailed)	0,042

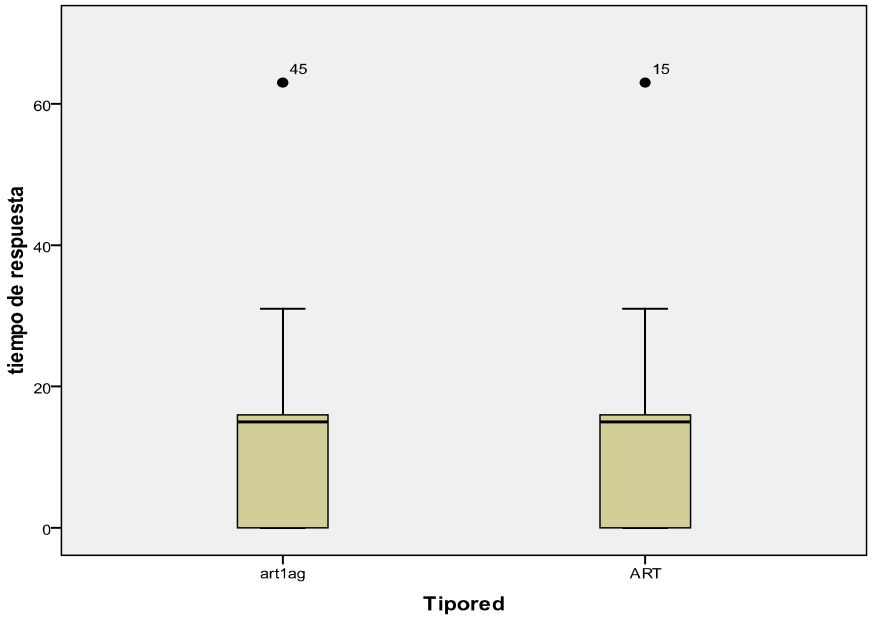
a. Variable de Agrupacion: TipoRed

Fuente: elaboración propia.

En la tabla XXXV puede observarse el parámetro U que es de 179, al estandarizarlo da un zeta de -2,035 a la que corresponde una probabilidad de, 0,42 de error tipo 1, este valor es menor al valor de significancia propuesto, por lo tanto no se rechaza la hipótesis nula pudiéndo afirmar que la actividad del CPU en ambos experimentos, no varía de manera significativa.

Como último material de análisis se presenta una gráfica de comparación de medias para el tiempo de respuesta de los sistemas, como se aprecia en la figura 41. En esta gráfica se observa que existe una leve variación cuya mediana general para el sistema ART1+AG, es superior en 30 ms al del sistema ART1.

Figura 41. **Representación gráfica de medianas de tiempos de Respuesta entre sistemas ART1 y ART+AG**



Fuente: elaboración propia.

8.7.2.2. Análisis estadístico de experimentos de reconocimiento de patrones

Se revisaron los datos de la pruebas de reconocimiento de patrones primero con una prueba de normalidad, de modo que se logra reconocer si los datos tienen una tendencia que se ajusta a la distribución normal para luego decidir qué tipo de pruebas realizar y observar las diferencias, entre los datos provistos por los experimentos.

Se realizó la prueba de normalidad con 5% o 0,05 de significancia, los resultados se muestran en la tabla 36, se aprecia que por ser conjuntos de datos iguales a 50 se utilizó la prueba de Kolmogorov-Smirnov para determinar el ajuste de los datos al comportamiento de la distribución normal y se observa en la columna Sig.

En ningún experimento se obtuvo un nivel de significancia importante, que se acerque o supere al nivel propuesto, por lo que concluimos que los datos proporcionados en los distintos experimentos no se ajustan a la distribución normal, para determinar si existen diferencias significativas entre experimentos se utilizo la prueba de Kruskal-Wallis.

Tabla XXXVI. **Prueba de normalidad Tiempos Experimento pruebas aleatorias red ART1**

Prueba de Normalidad

	NumExp	Kolmogorov-Smirnov			Shapiro-Wilk		
		Statistic	df	Sig.	*Statistic*	df	Sig.
TiempoRespuesta	Exp1	0,340	50	0,000	0,660	50	0,000
	Exp2	0,340	50	0,000	0,698	50	0,000
	Exp3	0,351	50	0,000	0,649	50	0,000
	Exp4	0,351	50	0,000	0,649	50	0,000
	Exp5	0,342	50	0,000	0,625	50	0,000

Fuente: elaboración propia.

Luego de analizar si los datos se ajustan a la distribución normal y se determina que esto no es así, se procede a definir si el tiempo entre los distintos experimentos es significativamente parecido o igual, esto nos ayudó a determinar si el sistema es consistente, tiene aplicaciones útiles y se puede estimar un punto de referencia. Al tener en cuenta que los datos no tienden a la distribución normal, se comparan utilizando la prueba Kruskal-Wallis, en esta prueba se tiene inicialmente las hipótesis: $H_0 : Me_M = Me_F$.

Lo anterior significa que no hay diferencia entre las medianas de cada grupo ni diferencia significativa entre los tiempos de respuesta del sistema para cada experimento y $H_1 : Me_M \neq Me_F$; hay diferencia en las medianas, por lo tanto el tiempo de respuesta por cada experimento es distinto. El nivel de significancia propuesto es del 5% o 0,05.

Tabla XXXVII. **Prueba Kruskal Wallis para experimentos en red ART1**

Prueba Kruskal wallis

	TimeResp
Chi-Square	0,308
df	4
Asymp. Sig.	0,989

Fuente: elaboración propia.

En la tabla XXVII se observan los resultados de la prueba de Kruskal-Wallis realizada a los datos obtenidos en cinco distintas iteraciones del sistema ART1, entrenada de manera lineal, en esta tabla el dato que interesa está en la columna *TimeResp* y en la fila *Asymp Sig*., nos indica el error estándar de tipo uno el cual es mayor al 5%, por lo tanto aceptamos la hipótesis nula respecto a que las medianas son iguales, de modo que entre los distintos experimentos no hay diferencias significativas en cuanto a los resultados, asegurando que el sistema es consistente.

Los datos del sistema red ART1+AG se obtuvieron de cinco experimentos que generaron distinta información acerca del tiempo de respuesta del sistema a unidades de entrada aleatoria, fue entrenado de forma lineal utilizando el mismo sistema.

Inicialmente se estimó la normalidad de los datos, para ello se realizó la prueba cuyos resultados pueden observarse en la tabla 38, ésta arroja un 5% o 0,05 de significancia inicial como puede verse en esta tabla en la columna Sig. de Kolmogorov-Smirnov ya que tenemos 50 datos.

Nos damos cuenta que es cero, por lo tanto se sitúa bajo del nivel de significancia pudiendo decir que los datos en ninguno de los casos se ajusta a la distribución normal, de ahí que no se puede realizar una prueba de ANOVA sino una prueba no paramétrica como la U de Kruskal-Wallis para determinar si existe diferencia significativa entre los tiempos de respuesta en los distintos experimentos realizados.

Tabla XXXVIII. **Prueba de Normalidad experimentos red ART1+AG**

Prueba de normalidad

	TimeResp	Kolmogorov-Smirnov			Shapiro-Wilk		
		Statistic	df	Sig.	*Statistic*	df	Sig.
TimeResp	Exp1	0,324	50	0,000	0,626	50	0,000
	Exp2	0,336	50	0,000	0,615	50	0,000
	Exp3	0,302	50	0,000	0,601	50	0,000
	Exp4	0,326	50	0,000	0,629	50	0,000
	Exp5	0,313	50	0,000	0,635	50	0,000

Fuente: elaboración propia.

Al comprobar que los datos no tienen una tendencia normal, se procede a realizar la prueba de Kruskal-Wallis la cual es extensión de la prueba U de Mann-Whitney con la hipótesis: $H_0 : Me_M = Me_F$, significando que no hay diferencia entre las medianas de cada grupo ni diferencia significativa entre los tiempos de respuesta del sistema para cada experimento. En $H_1 : Me_M \neq Me_F$ hay diferencia en las medianas de cada grupo, por lo tanto el tiempo de respuesta por cada experimento es distinto. El nivel de significancia propuesto es del 5% o 0,05.

Los resultados de la prueba de Kruskal-Wallis pueden observarse en la tabla XXXIX, el error estándar de tipo 1 es de 0,989, nos indica al compararlo con el nivel de significancia igual a 5% que es superior, por lo tanto aceptamos la hipótesis nula.

La anterior indica que no existe diferencia significativa entre los datos arrojados por los distintos experimentos, en condiciones similares, nos conduce a pensar que se puede definir una línea base en cuanto a la respuesta del sistema ya que éste es consistente en cuanto a las respuesta que se pueden observar, de modo que los resultados no tienen una variación importante bajo condiciones similares.

Tabla XXXIX. **Prueba de Kruskal-Walis experimentos sistema ART1+AG**

Prueba Krukal Wallis

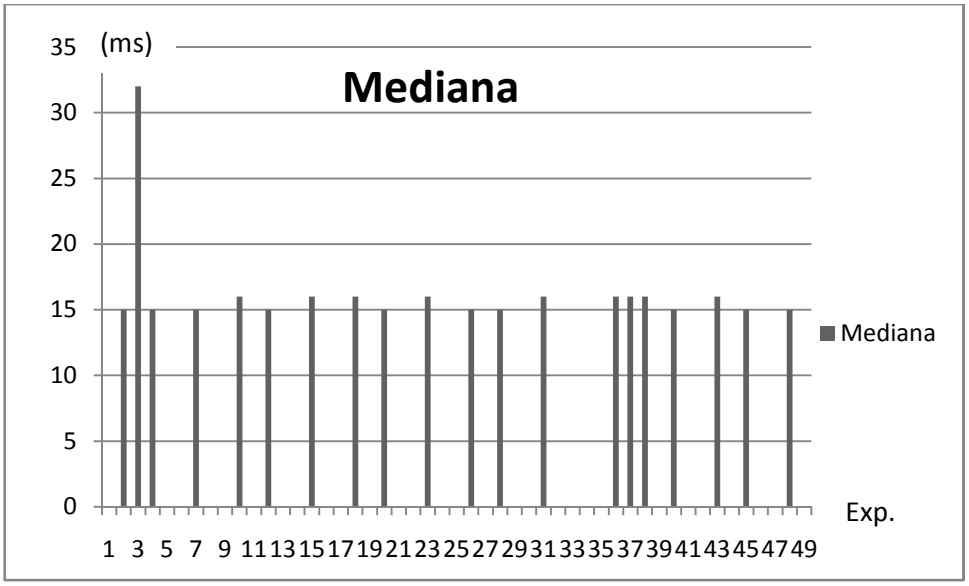
	TimeResp
Chi-Square	0,317
df	4
Asymp. Sig.	0,989

Fuente: elaboración propia.

En los análisis anteriores se concluyó que no existe diferencia significativa entre los experimentos realizados bajo las mismas condiciones y los mismos sistemas. Por lo tanto, podemos tomar cualquiera de estos conjuntos de datos para realizar la comparación de rendimiento de tiempos entre los sistemas ART1 y ART1+AG, para hacer dicha comparación se generó un conjunto de datos a partir de la mediana puesto que este es un valor original obtenido de los experimentos y no se ve afectada por valores extremos ocurridos durante el desarrollo del experimento como la media.

Los datos de la mediana, obtenidos en ambos experimentos, se presentan gráficamente en las figuras 42 y 43 en las cuales se puede apreciar que el rango de tiempos en milisegundos de respuesta del sistema ART1, es menor en máximos que el tiempo de respuesta en milisegundos del sistema ART1+AG. En contraste también se observa que existen más tiempos de respuesta que tienden a cero en el sistema ART1+AG que en el ART1.

Figura 42. **Tiempos de respuesta para sistema red ART1**



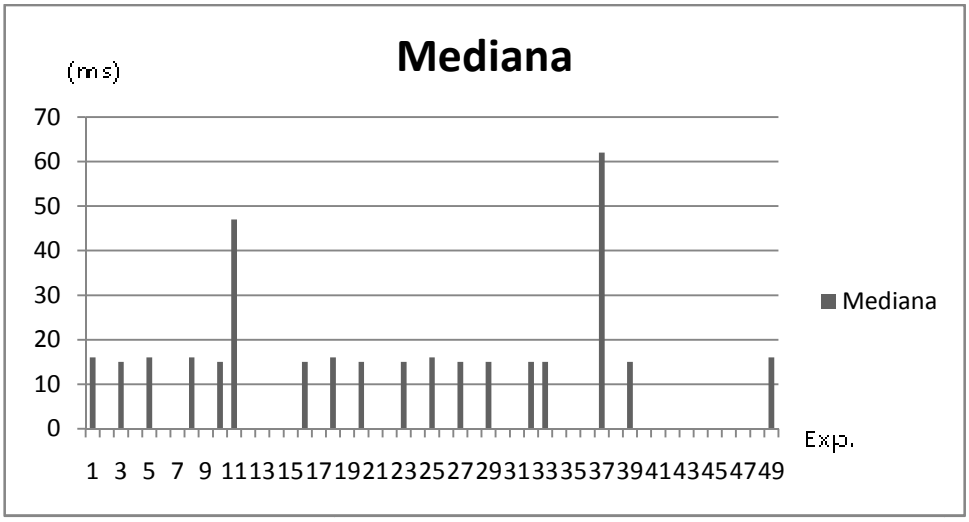
Fuente: elaboración propia.

Tabla XL. **Estadística descriptiva para medianas de tiempos de respuesta del sistema ART1**

Mediana tiempos de respuesta	
Media	6,84
Error típico	1,19432673
Mediana	0
Moda	0
Desviación estándar	8,44516526
Varianza de la muestra	71,3208163
Curtosis	-0,56736852
Coeficiente de asimetría	0,69762137
Rango	32
Mínimo	0
Máximo	32
Suma	342
Cuenta	50

Fuente: elaboración propia.

Figura 43. **Medianas de tiempos de respuesta sistema ART1+AG**



Fuente: elaboración propia.

Tabla XLI. **Estadística descriptiva Medianas tiempo de respuesta sistema ART1+AG**

Estadística descriptiva Medianas	
Media	7,1
Error típico	1,72833528
Mediana	0
Moda	0
Desviación estándar	12,221176
Varianza de la muestra	149,357143
Curtosis	8,98516289
Coeficiente de asimetría	2,64022069
Rango	62
Mínimo	0
Máximo	62
Suma	355
Cuenta	50

Fuente: elaboración propia.

Los análisis anteriores se inician con la prueba de normalidad para ambos conjuntos de datos determinando si se ajustan a la distribución normal para realizar la prueba ANOVA o de lo contrario, la prueba U de Mann-Whitney con un nivel de significancia del 5% o 0,05.

Tabla XLII. **Prueba de normalidad tiempos de respuesta de sistemas**

Prueba de normalidad

		Kolmogorov-Smirnov			Shapiro-Wilk		
	TipoRed	*Statistic*	df	Sig.	*Statistic*	df	Sig.
TimeResp	Artag	0,353	51	0,000	0,600	51	0,000
	ART	0,377	49	0,000	0,688	49	0,000

Fuente: elaboración propia.

En la tabla XL se aprecian los resultados para la prueba de normalidad en las medianas de tiempos de respuesta de los sistemas, para esta prueba nos interesa el análisis de Kolmogorov-Smirnorv ya que tenemos conjuntos de datos mayores o iguales a 50 unidades, en la columna Sig. Se observa que el error tiende a cero lo cual es menor que el nivel de significancia del 5%, por lo tanto concluimos que ambos conjuntos de datos no se ajustan a la distribución normal y para realizar un análisis de comparación, necesitaremos hacer la prueba U de Mann-Whitney.

Para saber si existe una variación importante entre el comportamiento de los tiempos de respuesta de ambos sistemas y determinar si el uso de un algoritmo genético puede afectar el rendimiento del mismo, se utilizó la prueba U de Mann-Whitney con la hipótesis: $H_0 : Me_M = Me_F$ la que significa que no hay diferencia entre las medianas de cada grupo ni diferencia significativa entre los tiempos de respuesta del sistema por experimento y $H_1 : Me_M \neq Me_F$, hay diferencia en las medianas de cada grupo, por lo tanto el tiempo de respuesta por experimento es distinto. El nivel de significancia propuesto es del 5% o 0,05.

La tabla XLI incluye los resultados para la prueba U de Mann-Whitney y las medianas de tiempo de respuesta de sistemas, se observa que el error es de 0,732 superior al nivel de significancia, por lo tanto no se acepta la hipótesis nula, concluyendo que existen diferencias significativas entre los dos grupos de datos.

Tabla XLIII. **Prueba U de Mann Whitney para medianas de tiempos de respuesta de sistemas**

Prueba U Mann-Whitney

	TimeResp
Mann-Whitney U	1206,500
Wilcoxon W	2532,500
Z	-0,340
Asymp. Sig. (2-tailed)	0,734

Fuente: elaboración propia.

Luego de realizar las distintas pruebas a los tiempos de respuesta, se procede a analizar las actividades de CPU para resultados provenientes de ambos sistemas y el análisis de comparación. Se inicia con el de normalidad para después determinar con qué pruebas se procederá a la comparación de datos, esta prueba tiene un nivel de significancia de 5% o 0,05.

Tabla XLIV. **Prueba de normalidad Actividad CPU pruebas aleatorias**

Prueba de Normalidad

		Kolmogorov-Smirnov			Shapiro-Wilk		
	tipoRed	*Statistic*	df	Sig.	*Statistic*	df	Sig.
TimeResp	ArtAg	0,135	36	0,097	0,903	36	0,004
	Art	0,180	45	0,001	0,902	45	0,001

Fuente: elaboración propia.

La tabla XLIV presenta los resultados para la prueba de normalidad en el porcentaje de actividad de CPU en cada sistema, para unidades menores a 50 se utilizó la prueba de Shapiro-wilk como referencia. En ésta observamos que el error en ambos casos es menor a 5% como nivel de significancia, se puede decir que los datos no se ajustan a la distribución normal.

Al tratarse de una distribución normal, se debe utilizar la prueba no paramétrica de u Mann-Whitney para determinar si existe o no, una diferencia significativa entre la actividad del procesador.

Se demostró que los datos para las actividades de CPU no se ajustan al comportamiento normal, por lo tanto se procede a realizar la prueba U de Mann-Whitney para la cual tenemos la hipótesis: $H_0 : Me_M = Me_F$, lo que significa que no hay diferencia entre las medianas de cada grupo ni diferencia significativa entre el porcentaje de CPU utilizado por cada uno de los experimentos y $H_1 : Me_M \neq Me_F$; hay diferencia en las medianas de cada grupo, por lo tanto el porcentaje de uso de CPU es significativamente distinto en ambos experimentos.

Tabla XLV. **Prueba U Mann-Whitney porcentaje uso de CPU pruebas aleatorias**

Prueba U de Mann-Whitney

	TimeResp
Mann-Whitney U	584,500
Wilcoxon W	1619,500
Z	-2,143
Asymp. Sig. (2-tailed)	0,032

Fuente: elaboración propia.

La tabla XLV presenta los resultados para la prueba U de Mann-Whitney, apreciamos en la fila Aymp. Sig el error estándar de tipo uno que está por debajo del nivel de significancia 5%, esto indica que no se debe rechazar la hipótesis nula, por lo tanto se puede afirmar que los porcentajes de uso de CPU no tienen ninguna variación importante siendo relativamente iguales.

8.8. Análisis de resultados

Al experimentar con la aplicación de red ART1 desarrollada específicamente para este trabajo, se observó que el tamaño de ésta no puede superar las dimensiones M y N mayor a 25, de modo que no se admite una matriz de pesos sinápticos mayor a esa cantidad, debido a que los pesos inician con valores altos pero convergen muy rápidamente a valores decimales pequeños e inclusive negativos, los que afectan directamente al momento de analizar las entradas. Por lo tanto, al tener pesos iníciales negativos se obtienen resultados erráticos en el momento de ingresar unidades vectoriales para su respectivo procesamiento a la red. Ésta no reconoce ni aprende ninguna entrada vectorial cuando la matriz de pesos sinápticos tiene alguna de sus dimensiones mayores a 25.

En la gráfica del entrenamiento progresivo de la red se observa claramente que para cada uno de los experimentos se conservó la misma tendencia, sin importar con qué dimensiones se inicializa la red, a excepción del momento cuando se efectúa con parámetros que no soporta; en este caso se obtienen datos aunque parejos siempre grandes, disminuye la cantidad de datos que tienden a cero y el sistema se esfuerza tratando de hacer su trabajo, sin lograrlo.

Para el experimento de entrenamiento lineal de red se comprobó que existen diferencias entre los tiempos de respuesta para los sistemas ART1 y ART+AG, por medio de pruebas estadísticas, al analizar la estadística descriptiva se observa que estas diferencias son leves de modo que la media para los tiempos de respuesta en el sistema ART1, difiere únicamente en 1,26 milisegundos del ART1+AG, la mediana para los resultados de dichos experimentos es similar.

La moda en ambos casos tiende a cero, de ahí que en el experimento el dato que más se repitió fue cero, lo que indica que el funcionamiento de estos sistemas en sus respuestas a las entradas, tiende a menos de unos milisegundos para analizar cierto vector de entrada.

La varianza y desviación estándar son levemente distintos indicando que los datos obtenidos en cada uno de los experimentos estuvo lejos de la media en casi la misma medida para ambos sistemas, eso confirma que los tiempos de respuesta obtenidos se mantienen constantes de la misma forma para ambos sistemas y los picos son similares.

Esto lleva a concluir que el agregar un sistema de algoritmo genético, como medio de entrenamiento de la red ART1, no influye significativamente en el desempeño de la misma.

En cuanto al uso del CPU se demostró estadísticamente que no existe ninguna diferencia significativa en el rendimiento del sistema de cómputo cuando se ingresan datos de manera lineal a ART1 que cuando se ingresan datos de manera lineal, ayudado por un sistema de AG en el entrenamiento del mismo. Se puede asegurar que el incluir un algoritmo genético, como medio de entrenamiento a un sistema simulador de red neuronal, no tiene incidencia en el porcentaje de uso del CPU.

En cuanto a las distintas iteraciones para probar el sistema de red ART1, con cincuenta datos, se comprobó estadísticamente que no existen diferencias significativas entre ellos y por lo tanto que la red se comporta de manera consistente.

En los experimentos realizados al probar un sistema ART1+AG con una serie de datos aleatorios se demostró estadísticamente que no existen diferencias significativas asegurando que al ingresar cualquier cantidad de datos, el tiempo de respuesta para procesar los mismos será consistente y no aleatorio, pudiendo establecerse un tiempo promedio o línea base de procesamiento de datos para este sistema.

En la comparación de medianas en tiempos de respuesta para cada uno de los sistemas estudiados, se demostró que existe una diferencia entre ambos conjuntos de datos, al analizar la estadística descriptiva de éstos se concluye que las medias difieren únicamente en 0,26 milisegundos y que la desviación estándar y varianza son similares.

La mayor diferencia que se puede observar es en relación al máximo en el cual hay una diferencia de 30 milisegundos, pero en general los otros datos siguen tiempos similares para las muestras; la moda y la mediana son las mismas para ambos grupos, por lo que se puede decir que aunque existen diferencias, éstas son muy pequeñas y por ello el rendimiento de ambos sistemas es similar.

Se comprobó estadísticamente que en la comparación de medianas de los tiempos de respuesta, para un ingreso aleatorio de vectores a los sistemas estudiados son iguales, por lo que se puede asegurar que usar un sistema de algoritmo genético para entrenar al simulador de ART1, no influye en cuanto al uso del porcentaje de CPU.

En general, al utilizar los sistemas desarrollados para este trabajo, no existe diferencia de rendimiento en los sistemas de cómputo al emplear un algoritmo genético para entrenar una red neuronal tipo ART1, de ahí que se pueden aprovechar las ventajas que provee el uso de AG sin preocuparse que éste aumentará tiempos de respuesta o consumirá más recursos que utilizando la red ART, de manera individual.

8.9. Aplicaciones

Las aplicaciones de un sistema híbrido red ART + AG son muchas, variadas y similares a las aplicaciones de redes neuronales como por ejemplo, el reconocimiento visual de objetos, usando como cerebro el de un robot lo cual se pudo comprobar en el capítulo 7. También el procesamiento de imágenes y textura, de tal forma que, en cierto momento, el entrenador de AG sea capaz de crear imágenes autogeneradas o predecir algunas como control borroso y predicción económica ya que la economía se comporta en ciclos y al poder almacenar sucesos anteriores en la red y tener la capacidad de autogenerar nuevos datos, se predicen ciertos comportamientos.

Entre las ventajas principales de un sistema como éste destaca que no sólo será capaz únicamente de reconocer un patrón aprendido, sino podrá generar patrones nuevos y con ello muchas posibilidades de acción.

CONCLUSIONES

1. Al desarrollar un sistema computacional que imite sistemas biológicos con comportamiento caótico, se deben buscar medios alternativos no deterministas de solución, puesto que los medios directos en cuanto a complejidad tienden a pertenecer a la clase de problemas NP-completos, para los cuales no está demostrado que pueda tener una solución determinista.

2. El término autopoiesis se acuñó en biología, pero es válido para describir sistemas que pueden crear o destruir elementos dentro del mismo sistema, por lo tanto es aplicable a la ciencia de la computación.

3. Las redes neuronales tipo ART presentan elasticidad y plasticidad, por eso reconocen nuevas operaciones y las aprenden o memorizan.

4. Los algoritmos genéticos tienen la capacidad de percibir perturbaciones del medio o entradas y autogenerarse, o bien destruir ciertos elementos que no cumplen con los requisitos, por lo tanto se puede afirmar que se comportan de manera autopoiética.

5. Los modelos híbridos de inteligencia artificial, buscan aprovechar lo mejor de cada modelo de simulación de inteligencia, para mejorar la potencia y eficiencia en el razonamiento de un sistema inteligente aislado.

6. El sistema híbrido inteligente de simulador ART1+AG, desarrollado específicamente para este trabajo, puede tener aplicaciones reales constituyéndose en cerebro de un robot para aplicaciones industriales.

7. El sistema híbrido inteligente de simulador ART1+AG, estructurado para fines de este trabajo, tiene un punto de quiebre en veinticinco unidades para cada dimensión de la matriz de pesos sinápticos, al exceder este parámetro el sistema generará resultados erráticos.

8. Para los sistemas desarrollados en este trabajo, es factible utilizar un sistema de algoritmo genético y entrenar un simulador de red ART no afecta el rendimiento en cuanto a tiempo y porcentaje de uso de CPU y de esta manera, aprovechar las ventajas de predecir los resultados esperados.

9. Las aplicaciones para sistemas híbridos inteligentes ART+AG son extensas, en los próximos años se generará una alta gama de posibilidades al respecto.

RECOMENDACIONES

1. Investigar los fractales, analizando cómo éstos se relacionan con los sistemas dinámicos y por consiguiente con la teoría del caos, de manera que se puedan generar nuevos puntos de vista para abordar temas que necesiten algún grado de inteligencia para resolver problemas.

2. Familiarizarse con los modelos biológicos de redes neuronales, adentrándose en el tema investigando en cuanto a los descubrimientos más recientes, de modo que se puedan generar sistemas de cómputo que imiten los sistemas biológicos.

3. Antes de intentar desarrollar un sistema de inteligencia artificial debe fundamentarse extensamente con la teoría que lo respalda.

4. Utilizar pruebas estadísticas al momento de analizar los datos, para garantizar la validez de los mismos.

5. No conformarse con leer sobre IA y discutir si es posible o no, sino ir más alla y crearla.

BIBLIOGRAFÍA

1 BAGNALL, Brian *LEGO NXT: Building Robots with java brain*. 1a ed. Winnipeg: Variant press, 2006. 524 p. ISBN: 09-738-6495-8

2 BALLESTEROS, Alfonso. *Neural Network Frameworks*. [en línea]. Texto ed. 1.0. [Estados Unidos de América]: 2009. [ref. 3 de diciembre de 2003]. Disponible en Web: http://www.redes-neuronales.netfirms.com/index.htm

3 FLORES, Raquel; FERNÁNDEZ, José Miguel. *Las Redes Neuronales Artificiales, fundamentos teóricos y aplicaciones prácticas*. 1a ed. Madrid: NETBIBLO, S. L., 2007. 352 p. ISBN: 978-84-9745-246-5

4 FREEMAN, James; SAKAPURA, David. *Redes Neuronales Algoritmos, aplicaciones y técnicas de* programación. García-Bermejo, Rafael (trad.); Aguilar, Joyanes(trad.). 1a ed. Madrid: Addison-Wesley, 1993. 431 p. ISBN: 84-797-8103-3

5 Ishtaki. *Tema 12. Kruskal-Wallis ANOVA*. [en línea]. Video ed. 1.0. [Lima, Perú]: Youtube, marzo de 2008 [ref. 3 diciembre de 2010]. Disponible en web: http://www.youtube.com/watch?v=4bWOn8HIpSE.

6 Ishtaki. *Tema 9c. Pruebas de Hipótesis (Parte 3)*. [en línea]. Video ed. 1.0. [Lima, Perú].Youtube, abril de 2008 [ref. 5 de diciembre de 2010]. Disponible en web: http://www.youtube.com/watch?v=bcOFrmCnJC8

7 LUHMANN, Niklas. *Organización y decisión. Autopoiesis, acción y entendimiento comunicativo*. Rodríguez, Darío (trad.). 1a ed. Santiago de Chile: Pontificia universidad católica de chile, 2005. 138 p. ISBN: 84-7658-772-4

8 PALLANT, Julie. SPSS *survival manual.* 3a ed. Sydney: Allen y Unwin, 2007. 352 p. ISBN: 17-4175-216-7

9 Universidad del país Vasco. *Algoritmos Genéticos*. [en línea] texto ed. 1.0. [País Vasco, España]. Página oficial, junio de 2008 [ref. 3 de diciembre de 2010]. Disponible en web: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/temageneticos.pdf

APÉNDICES

Apendice 1. Código fuente red ART1+AG

```
package redneuralart;
/**
 *
 * @author Antonio Prera
 * Guatemala 2010
 * Universidad de San carlos de Guatemala
 */
public class CAPA {
    public CAPA() {
        for(int i=0;i<15;i++){
           for(int j=0;j<15;j++){
             PESOS[i][j]=-5;
             SALIDAS[i]=-5;
           }
        }
    }
   private double[][] PESOS = new double[15][15];
   private double[] SALIDAS = new double[15];
   private int UNIDADES;
   public void insertElementoPeso(int fila,int col,double valor){
      PESOS[fila][col]=valor;
   }

    public double[][] getPESOS() {
        return PESOS;
    }

   public void insertElementoSalida(int fila,double valor){
      SALIDAS[fila]=valor;
   }
   public double[] getSalidas(){
     return this.SALIDAS;
   }
    public int getUNIDADES() {
        return UNIDADES;
    }
    public void setUNIDADES(int UNIDADES) {
        this.UNIDADES = UNIDADES;
    }
   public double getElementoSalida(int fila){
     return SALIDAS[fila];
   }
   public double getElementoPeso(int fila,int col){
     return PESOS[fila][col];
   }
}

CLASE RED ART1
package redneuralart;
/**
 *
 * @author Antonio
 * Guatemala 2010
 * Universidad de San carlos de Guatemala
 */
public class ART1 {
    private CAPA F1 = new CAPA(); //LOCALIZA  LA ESTRUCTURA DE LA CAPA F1
```

```
    private CAPA F2 = new CAPA(); //LOCALIZA LA ESTRUCUTURA DE LA CAPA F2
    private double A1;
    private double B1;
    private double C1;
    private double D1;
    private double L1;
    private double ro;
    private int GF2;
    private double[] INH=new double[15];
    private double modX;
    private int M; //unidades en f1
    private double N; //unidades en f2
    public ART1(double A1, double B1, double C1, double D1, double L1, double ro, int M, int N) {
        this.A1 = A1;
        this.B1 = B1;
        this.C1 = C1;
        this.D1 = D1;
        this.L1 = L1;
        this.ro = ro;
        this.M = M;
        this.N = N;
        F1.setUNIDADES(M);
        F2.setUNIDADES(N);
        GF2=0;
}
    public double getRo() {
        return ro;
    }
    public void setRo(double ro) {
        this.ro = ro;
    }
    public void setModX(double modX) {
        this.modX = modX;
    }
    public double getL1() {
        return L1;
    }
    public void setElementoInh(int indice,double valor){
       INH[indice]=valor;
    }
    public double getModX() {
        return modX;
    }
    public void setGF2(int GF2) {
        this.GF2 = GF2;
    }
    public double getINH(int pos) {
        return INH[pos];
    }
    public CAPA getF1() {
        return F1;
    }
    public CAPA getF2() {
        return F2;
    }
   public void insertarPeso(int fila,int col,double valor){
    this.F1.insertElementoPeso(fila,col,valor);
   }
    public void insertarSalida(int fila,double valor){
    this.F1.insertElementoSalida(fila,valor);
   }
    public int getGF2() {
        return GF2;
    }
    public double getA1() {
        return A1;
```

```
    }
    public double getB1() {
        return B1;
    }
    public double getC1() {
        return C1;
    }
    public double getD1() {
        return D1;
    }
    public void setF1(CAPA F1) {
        this.F1 = F1;
    }
    public void setF2(CAPA F2) {
        this.F2 = F2;
    }
 public void inicializarElementos(){
   for(int i=0;i<F1.getUNIDADES();i++){
     for(int j=0;j<F2.getUNIDADES();j++){
        F1.insertElementoPeso(i, j,((B1-1)/D1)+.2);
        F2.insertElementoPeso(j, i, (L1/(L1-1+M))-0.1);
       // F1.insertElementoSalida(i, j,-B1/(1+C1));
     }
   }
 }
}
```

Simulador ART1

```
package redneuralart;

/**

 *

 * @author Antonio

 * Guatemala, 2010

 * Universida de san carlos de guatemala

 */

public class SimulaArt1 {

    public SimulaArt1() {

    }

 public void prop_F1(ART1 red,double[] vectent){

    int i;// contador de iteraciones

    double[] unidad; //salidas de las unidades

    unidad = red.getF1().getSalidas(); //se localizan las salidas de las unidades

    //inicio operaciones

     for(i=0;i<red.getF1().getUNIDADES();i++){ //para todas las unidades de F1 se hace

        unidad[i]= vectent[i]/(1+red.getA1()*(vectent[i]+red.getB1())+red.getC1());
```

```
      if(unidad[i]>0){ // se transforma  la activacion en salida
        unidad[i]=1;
      }
      else{
        unidad[i]=0;
      }
    }
}// se calculan las salidas de la capa F1 para un vector de entrada dado
public void prop_F2(ART1 red){
 int i,j;  //contadores de iteraciones
 double[] unidad; //salidas de F2
 double[] entradas;//salidas de F1
 double mayor; //mayor activacion
 int ganadora=0;//indice de la ganadora
 double suma;//acumulador
  unidad = red.getF2().getSalidas();
  entradas=red.getF1().getSalidas();
  mayor = -100;

  for(i=0;i<red.getF2().getUNIDADES();i++){
    unidad[i]=0;
  }
  for(i=0;i<red.getF2().getUNIDADES();i++){
    suma =0;
    for(j=0;j<red.getF1().getUNIDADES();j++){
      suma += entradas[j]* red.getF2().getElementoPeso(i, j);
    }
    suma = suma * red.getINH(i);
    if(suma > mayor){
      ganadora = i;
      mayor = suma;
    }
```

```
   }
    unidad[ganadora]=1;
    red.setGF2(ganadora);
}
public void  prop_vuelta_F1(ART1 red, double[] vectent){
int ganadora,i;//ganadora es el indice de la ganadora , i contador de iteraciones
double[] unidad;//salidas de F2
double X;// activa una nueva entrada
double Vi;//peso de coneccion
 unidad = red.getF1().getSalidas();
 ganadora = red.getGF2();

  for(i =0; i<red.getF1().getUNIDADES();i++){
     Vi = red.getF1().getElementoPeso(i,ganadora);//obtiene el peso de la activacion
     X= (vectent[i]+red.getD1()*Vi-red.getB1())/(1+red.getA1()*(vectent[i]+red.getD1()*Vi)+red.getC1());

 if(X>0){
   unidad[i]=1;
 }
 else{
    unidad[i]=0;
 }
}
}//propagar de vuelta f1
public double coincide(ART1 red, double[] vectent){
   int i;
   double[] unidad;
   double modX;
   double modI;
   //inicio
   unidad = red.getF1().getSalidas();
   modX=0;
```

```
   modI=0;
   for(i=0;i<red.getF1().getUNIDADES();i++){
      modX=modX+unidad[i];
   }
 red.setModX(modX);
 return modX/modI;
}
 public void actualizar(ART1 red){
     int i ;
     int ganadora;
     double[] unidad;
     double[] entradas;
     double[][] conexiones;
     unidad = red.getF2().getSalidas();
     conexiones= red.getF2().getPESOS();
     ganadora = red.getGF2();
     entradas = red.getF1().getSalidas();
     for(i=0;i<red.getF2().getUNIDADES();i++){
       conexiones[ganadora][i]=(red.getL1()/(red.getL1()-1+red.getModX()))*entradas[i];
     }
     conexiones = red.getF1().getPESOS();
     for(i=0;i<red.getF1().getUNIDADES();i++){
       conexiones[i][ganadora]=(red.getL1()/(red.getL1()-1+red.getModX()))*entradas[i];
     }
 }//actualiza los pesos de una red para recordar una trama
 public void eliminar_inhibicion(ART1 red){
 int i;
   for(i=0;i<red.getF1().getUNIDADES();i++){
       red.setElementoInh(i, 1.0);
   }
}
public void propagar(ART1 red,double[] vectent){
```

```
  boolean hecho=false;
  this.eliminar_inhibicion(red);

  while(!hecho){
   this.prop_F1(red, vectent);
   this.prop_F2(red);
   this.prop_vuelta_F1(red, vectent);
     if(this.coincide(red, vectent)< red.getRo()){
        red.setElementoInh(red.getGF2(), 0);
     }
     else
     {
        hecho=true;
     }
  }
   this.actualizar(red);
}
}
```

CLASE CRUCE

```
public class cruce {
    CAPA padreF1;
    CAPA padreF2;
    double[] padre1= new double[15];
    double[] padre2= new double[15];
    double[] hijo= new double[15];
    public cruce() {
    }
    public CAPA getPadreF2() {
        return padreF2;
    }
    public CAPA getPadreF1() {
        return padreF1;
    }
    public void setPadreF1(CAPA padre) {
        this.padreF1 = padre;
    }
    public void setPadreF2(CAPA padreF2){
        this.padreF2=padreF2;
    }
    public double[] getPadre1() {
        return padre1;
    }
```

```
  public double[] getPadre2() {
     return padre2;
  }

  public void setPadre1(double[] padre1) {
     this.padre1 = padre1;
  }
  public void setPadre2(double[] padre2) {
     this.padre2 = padre2;
  }
  public int  buscarFilas(){
   int noFilas=0;
   boolean esPeso=false;
   double tomaNoTemp;
        for(int i=0;i<this.padreF2.getUNIDADES();i++){
          if(esPeso)
          { noFilas++;}
          esPeso=false;

           for(int j=0;j<this.padreF1.getUNIDADES();j++){
             tomaNoTemp= this.padreF1.getElementoPeso(j, i);
             if((tomaNoTemp==0)||(tomaNoTemp==1)){
               esPeso=true;
             }
          }
        }

    return noFilas;
  }//busca las filas que contienen patrones aprendidos
public double[] tomarPadres(int filas){
    double Nofit;
    int ran= (int)(Math.random()*filas);
    int ran2= (int)(Math.random()*filas);
      for(int i=0;i<this.padreF1.getUNIDADES();i++){
        padre1[i]=this.padreF1.getElementoPeso(i,ran);
      }
    for(int i=0;i<this.padreF1.getUNIDADES();i++){
        padre2[i]=this.padreF1.getElementoPeso(i,ran2);
        System.out.println((int)(Math.random()*2));
      }
     do{
      cruzar();
      Nofit= this.fitness();
       if(Nofit < 0.2){
         padre2=hijo; //sustituye el padre por el hijo para una nueva iteracion
       }
     }while(Nofit<0.2);
  return hijo;
}//toma dos de las columnas pesos y las convierte en los padres de una
//proxima columna lo cual hara por medio de un algoritmo genetico
public void cruzar(){
    int probMutacion=(int)(Math.random()*10);//indica si el elemento debe mutar o no
    int probPadre=(int)(Math.random()*1);//indica de que padre hereda el gen

    for(int ix=0;ix<this.padreF1.getUNIDADES();ix++){
        if(probMutacion!=5){
          if(probPadre >0){
```

```
            hijo[ix]=padre1[ix];
          }
          else{
            hijo[ix]=padre2[ix];;
          }
        }
        else
        {
          hijo[ix]=1.0;
        }
       probMutacion=(int)(Math.random()*10);
       probPadre=(int)(Math.random()*1);
     }
}//funicion que cruza elementos de dos poblaciones
public double fitness(){
   double fit =0;
   double contadorIguales=0;
   double fitIgual,fitIgualb;
   double fitCeros=0,fitUnos=0,fitInv=0;
   Double tempHijo;
   double temp2;
    for(int i=0;i<this.padreF1.getUNIDADES();i++)
      if(hijo[i]==0){
         contadorIguales=contadorIguales+1;
      }
    }
   fitCeros=contadorIguales/this.padreF1.getUNIDADES();
   contadorIguales=0;

    for(int i=0;i<this.padreF1.getUNIDADES();i++){
      if(hijo[i]==1){
         contadorIguales=contadorIguales+1;
      }
    }
   fitUnos= contadorIguales/this.padreF1.getUNIDADES();

   contadorIguales=0;
     for(int i=0;i<this.padreF1.getUNIDADES();i++){

        if(padre1[i]==hijo[i]){
          contadorIguales=contadorIguales+1;
        }
      }
      fitIgual=1-(contadorIguales/this.padreF1.getUNIDADES());
      contadorIguales=0;
      for(int i=0;i<this.padreF1.getUNIDADES();i++){
        if(padre2[i]==hijo[i]){
          contadorIguales=contadorIguales+1;
        }
      }
      fitIgualb=1-(contadorIguales/this.padreF1.getUNIDADES());
       if((fitUnos==1)||(fitCeros==1)){
          fit=0.5;
       }
       else{
          fit=.5*fitIgual+.5*fitIgualb;
       }
```

```
 return fit;
}//función aptitud que mide que tan bueno es el vector obtenido


 public CAPA capaNuevosPesosF1(double[] pesos,int posCol){

    for(int i=0;i<this.padreF1.getUNIDADES();i++){
       this.padreF1.insertElementoPeso(i,posCol, pesos[i]);

    }
    return this.padreF2;
 }

  public CAPA capaNuevosPesosF2(double[] pesos,int posCol){

    for(int i=0;i<this.padreF2.getUNIDADES();i++){
       this.padreF2.insertElementoPeso(posCol,i, pesos[i]);

    }
    return this.padreF1;
 }

}
```

APENDICE 2. Conjunto de datos obtenidos de los sistemas para su análisis.

Datos experimento prueba punto de quiebre.

Iteracion	Exp1	Exp2	Exp3	Ex4	Exp5	Exp6
1	0	15	15	32	31	47
2	0	0	16	15	16	16
3	0	16	16	0	16	15
4	0	0	0	16	0	16
5	16	0	0	0	15	16
6	0	16	15	0	16	15
7	0	0	0	16	0	16
8	0	0	0	0	16	16
9	0	15	0	0	0	15
10	0	0	16	15	15	16
11	0	0	0	0	0	15
12	0	0	0	0	16	16
13	0	0	0	16	0	63
14	0	0	0	0	15	15
15	0	0	0	0	0	16
16	0	0	0	15	16	15
17	0	0	0	0	0	16
18	0	0	0	0	16	16
19	0	0	0	16	0	15
20	0	0	0	0	47	16
21	0	0	0	0	0	16
22	0	0	0	0	0	15
23	0	0	0	0	16	16
24	0	0	0	0	0	15
25	0	0	0	0	15	16
26	0	0	0	0	0	63
27	0	0	0	0	0	15
28	0	0	0	0	0	16
29	0	0	0	0	0	15
30	0	0	0	0	0	16

Datos tendencias de los pesos prueba punto de quiebre

	Exp2
Iteracion	Pesos iníciales F1
1	0,329
2	0,15
3	0,076
4	0,036
5	0,011
6	-0,06
	Exp1
Iteracion	Pesos iníciales F1
1	0,756
2	0,15
3	0,076
4	0,036
5	0,011
6	-0,06
Exp3	
Iteracion	Pesos Iníciales F1
1	0,329
2	0,15
3	0,076
4	0,036
5	0,011
6	-0,06

Datos para entrenamiento lineal sistema ART1

Tiempo	Cpu Usage	Tiempo
21,10	0,10	31
21,20	3,00	16
21,30	3,50	15
21,40	5,00	0
21,50	5,60	16
21,60	5,80	16
21,70	8,00	0
21,80	8,50	15
21,90	9,60	0
22,00	11,05	16
22,10	7,00	0
22,20	6,50	16
22,30	5,00	0
22,40	5,50	15
22,50	5,40	63
22,60	4,50	0
22,70	4,00	15
22,80	3,50	0
22,90	3,20	16
23,00	3,00	0
		16
		0
		0
		15
		0

Datos entrenamiento lineal sistema ART1+AG

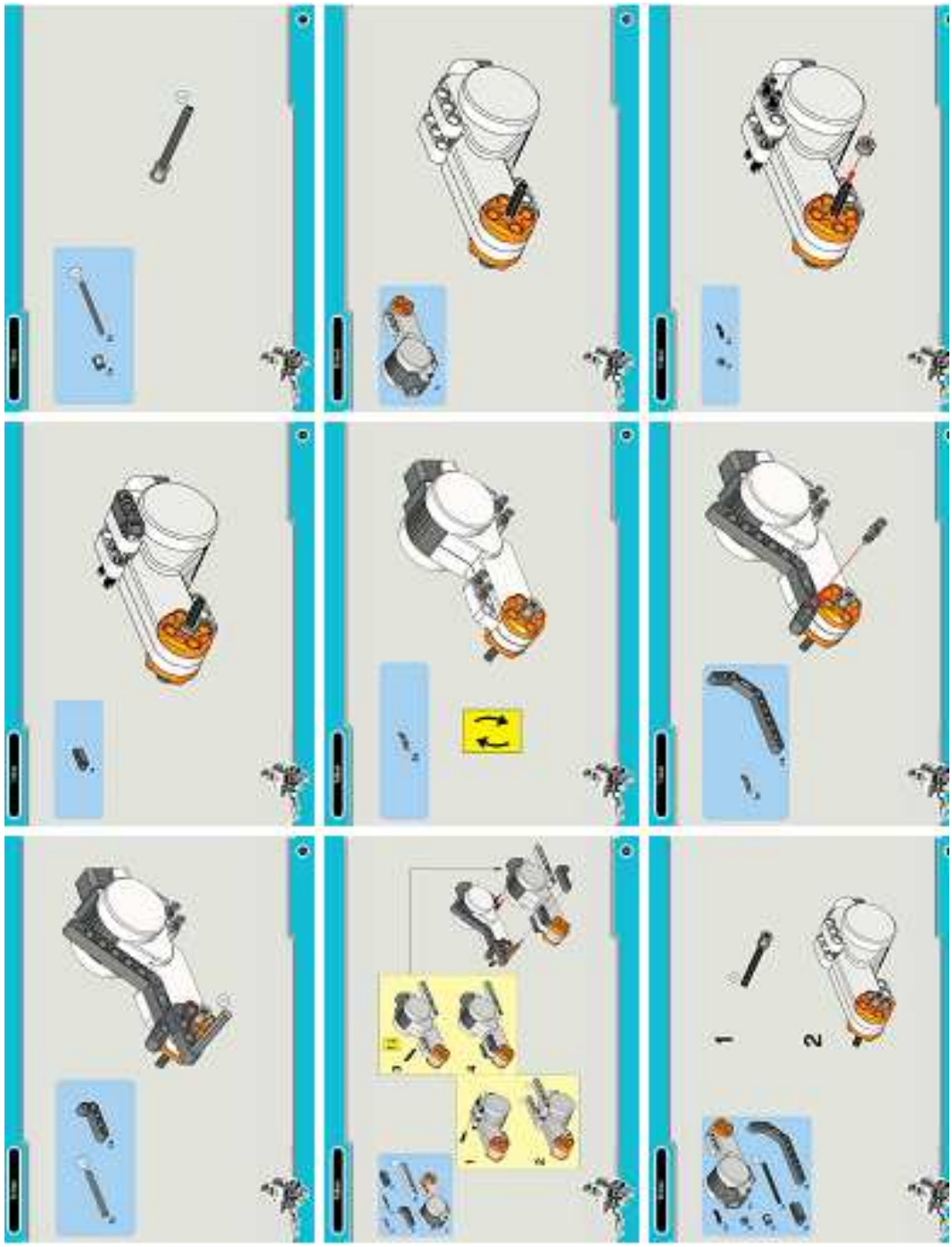
Tiempos respuesta		CPU activity
31	0,00	0,10
16	0,10	3,50
0	0,20	4,10
15	0,30	4,55
16	0,40	4,60
0	0,50	4,95
15	0,60	5,58
16	0,70	6,95
0	0,80	7,50
16	0,90	8,66
0	1,00	9,60
15	1,10	9,15
0	1,20	8,69
16	1,30	6,60
0	1,40	4,60
16	1,50	3,70
0	1,60	2,58
15	1,70	2,66
0	1,80	2,50
63	1,90	2,00
	2,00	0,70
	2,10	0,66
	2,20	0,55
	2,30	0,44
	2,40	0,33
	2,50	0,30
	2,60	0,29
	2,70	0,18
	2,80	0,11
	2,90	0,06

Datos pruebas aleatorias sistema ART1

Tiempo1	Tiempo2	Tiempo3	Tiempo4	Tiempo5	Mediana	Cpu Activity
16	16	47	0	16	16	0,04
0	0	0	16	0	0	0,05
0	16	16	0	15	15	0,07
47	31	0	47	32	32	0,10
15	15	16	0	15	15	0,30
0	0	0	16	0	0	0,35
0	16	0	0	0	0	0,38
16	0	15	15	16	15	0,58
0	0	0	0	0	0	0,65
0	16	0	0	16	0	0,70
16	0	16	16	0	16	0,80
0	0	0	0	0	0	3,00
15	15	15	0	15	15	3,50
0	0	0	15	0	0	4,00
0	16	0	0	0	0	4,80
16	0	16	16	16	16	5,20
0	0	0	0	0	0	5,80
0	16	0	0	0	0	8,00
16	0	16	16	15	16	9,60
0	0	0	0	0	0	10,00
0	15	0	15	16	15	9,58
15	0	15	0	0	0	9,40
0	16	0	0	0	0	8,00
16	0	16	16	16	16	7,58
0	0	0	0	0	0	7,60
0	15	0	0	0	0	6,50
15	0	16	16	15	15	6,48
0	0	0	0	0	0	6,00
16	16	0	15	0	15	5,80
0	16	0	0	0	0	5,00
16	0	16	16	0	16	4,85
0	0	0	0	16	0	4,75
47	0	16	47	0	16	2,00
15	47	0	16	63	16	1,58
0	15	47	0	0	0	1,00
0	16	16	0	15	15	0,88
16	0	0	0	0	0	0,84
0	0	15	0	0	0	0,35
0	16	0	16	16	16	0,10
16	0	0	0	0	0	0,00

ANEXOS

Anexo 1. Método utilizado para armar el robot



3
4
5
6

3
4
1
2
5
6