IMPLEMENTACIÓN DE ALGORITMOS GENÉTICOS EN UN SISTEMA DISTRIBUIDO COMO HERRAMIENTA PARA LA GENERACIÓN AUTOMÁTICA DE REDES NEURONALES UTILIZANDO DATOS DE CRUDOS Y FRACCIONES.

ÁLVARO SÁNCHEZ VILLALBA

VANESSA REY GARCIA

INGENIERÍA DE SISTEMAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
UNIVERSIDAD INDUSTRIAL DE SANTANDER
BUCARAMANGA
2009

IMPLEMENTACIÓN DE ALGORITMOS GENÉTICOS EN UN SISTEMA DISTRIBUIDO COMO HERRAMIENTA PARA LA GENERACIÓN AUTOMÁTICA DE REDES NEURONALES UTILIZANDO DATOS DE CRUDOS Y FRACCIONES.

AUTORES
ÁLVARO SÁNCHEZ VILLALBA
VANESSA REY GARCÍA

Trabajo de grado para optar por el titulo de
Ingeniero de Sistemas

DIRECTOR
MPE. HENRY ARGUELLO FUENTES

CODIRECTOR
ING. ERICK RAMÓN MENESES CUADROS

INGENIERÍA DE SISTEMAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
UNIVERSIDAD INDUSTRIAL DE SANTANDER
BUCARAMANGA
2009

Este libro está dedicado a mis padres Sara y Álvaro que hicieron el mayor esfuerzo para que mi sueño UIS se realizara,

A mis hermanos Miguel Ángel y Martha Liliana, el sacrificio más grande que hice fue separarme de ellos,

A mis amigos y todo aquel que piensa diferente

Agradezco mi realización como profesional en la UIS:

A mis padres Sara y Álvaro por todo el esfuerzo y confianza depositada en mí,

A mi Tía Nubia por acogerme en su hogar y hacerme parte de éste,

A William García y Cristian Porras por su amistad y colaboración, sin ellos este libro no se pudiera haber escrito,

A CUSOL por ser un grupo cuna de grandes ideas y buenos amigos,

A Vanessa Rey García, gran amiga y compañera incondicional,

A Sergio Pino, gran amigo y compañero,

A "Los de Cuarto", por todos los momentos vividos,

Al Ingeniero Erick Meneses por creer en nosotros, por ser un amigo y brindarnos su colaboración de manera desinteresada en lo personal y académico,

A la familia Rey García por su amistad y colaboración desinteresada desde siempre,

A la familia Pedraza Pico por la confianza y colaboración brindada,

A Betty por los momentos vividos, por acompañarme en buenos y malos momentos,

A todas las personas que conocí en este largo camino y me han colaborado a recorrerlo.

Álvaro

Agradezco

A Dios por ser la base espiritual sobre la cual encamino mi vida.

A mis padres y hermanos por su apoyo y colaboración incondicional para la realización de mis sueños.

A la Universidad Industrial de Santander y a los profesores de la Escuela de Ingeniería de Sistemas por su apoyo a mi formación profesional.

A William y Cristian, quienes colaboraron con nuestra labor en este proyecto.

A Alvaro, buen amigo y compañero de trabajo.

A mis amigos y compañeros de universidad, grandes momentos vividos para nunca olvidar

A Erick Meneses, por su apoyo, colaboración e interés en nuestro trabajo, por creer en nosotros.

Dedico este libro

A mis padres Carmen y Víctor, son mi guía en todos los momentos de mi vida.

A mi novio Luis, por el cariño y felicidad que me brinda, por se mi soporte en todo momento.

Vanessa Rey

GLOSARIO

Back-End: El Back-end es el responsable de interactuar con el sistema a partir de los datos suministrados por el usuario.

Escalabilidad: Es la capacidad de un sistema informático de cambiar su tamaño o configuración para adaptarse a las circunstancias cambiantes.

Front-End: El Front-end es responsable de las entradas y salidas por parte del usuario y aportando estos datos a la especificación que el back-end puede utilizar. El front-end es una especie de interfaz entre el usuario y el back-end.

Granularidad: Granularidad consiste en la cantidad de cómputo con relación a la comunicación. En Granularidad Fina o Fine-grained, las tareas individuales son relativamente pequeñas en término de tiempo de ejecución. La comunicación entre los procesadores es frecuente. En cambio en Granularidad gruesa o Coarse-grained, la comunicación entre los procesadores es poco frecuente y se realiza después de largos periodos de ejecución.

Latencia: Es el tiempo o lapso necesario para que un paquete de información se transfiera de un lugar a otro. La latencia, junto con el ancho de banda, son determinantes para la velocidad de una red.

Mainframe (Computadora central): Es un computador grande, potente y costoso usado principalmente por una gran compañía para el procesamiento de una gran cantidad de datos; por ejemplo, para el procesamiento de transacciones bancarias.

Middleware: Es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre

plataformas heterogéneas, funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo, red). El Middleware abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas.

MPI (Message Passing Interface): Es un estándar que define la sintaxis y la semántica de las funciones contenidas en una librería de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores.

Rendimiento: Medida o cuantificación de la velocidad/resultado con que se realiza una tarea o proceso. En una computadora, su rendimiento no depende sólo del procesado como suele pensarse, sino de la suma de sus componentes como la memoria, el bus, los diversos dispositivos, etc. y sus software.

Índice de contenido

Índice de Ilustraciones

Índice de Tablas

Índice de Gráficas

RESUMEN

Titulo: IMPLEMENTACIÓN DE ALGORITMOS GENÉTICOS EN UN SISTEMA DISTRIBUIDO COMO HERRAMIENTA PARA LA GENERACIÓN AUTOMÁTICA DE REDES NEURONALES UTILIZANDO DATOS DE CRUDOS Y FRACCIONES. [*]

Autores: VANESSA REY GARCIA[**]
ÁLVARO SÁNCHEZ VILLALBA**

Palabras Claves: Redes Neuronales, Inteligencia Artificial, RNA, Algoritmos Genéticos Paralelos, AGP, Procesamiento Paralelo y/o Distribuido.

Los algoritmos de aprendizaje automático son un conjunto de técnicas que permiten a las computadoras aprender y ser capaces de generalizar comportamientos a partir de información suministrada en forma de ejemplos; una de las técnicas mas usadas son las redes neuronales artificiales que simula las propiedades observadas en los sistemas neuronales biológicos a través de modelos matemáticos recreados mediante mecanismos artificiales (computadores). Para lograr el correcto entrenamiento de una red Neuronal artificial, se debe elegir la configuración correcta de capas y neuronas, ya que una buena combinación de éstas permite obtener soluciones óptimas al problema. Por esta razón se considera tediosa la construcción de una de estas redes, debido a la infinidad de posibles configuraciones que podrían alcanzar bajos errores de entrenamiento, sin tener certeza de que la configuración utilizada sea la más óptima.

Se acude entonces a los algoritmos genéticos como herramienta generadora de configuraciones de redes neuronales artificiales, aprovechando su exitosa utilización en problemas de optimización y su vasta exploración del espacio de búsqueda, además se hace uso del procesamiento paralelo para realizar una distribución de trabajo y disminuir el tiempo empleado en este proceso. De esta forma se obtiene una solución mediante un sistema preciso, eficiente y escalable, que combina técnicas de aprendizaje automático y cálculo distribuido.

---

[*] Trabajo de Grado

[**] Facultad de Físico-Mecánicas, Ingeniería de Sistemas e Informática.

Director Henry Arguello Fuentes. Codirector Erick Meneses Cuadros

ABSTRACT

Title: IMPLEMENTATION OF GENETIC ALGORITHMS ON A DISTRIBUTED SYSTEM, AS A TOOL FOR THE AUTOMATIC GENERATION OF NEURONAL NETWORKS USING DATA ABOUT CRUDE OIL AND FRACTIONS. *

Authors: Álvaro Sanchez Villalba **
Vanessa Rey Garcia **

Key Words: Neural networks, artificial intelligence, RNA, Parallel Genetic Algorithms; AGP, Parallel and/or Distributed processing.

Description:
Automatic apprentice algorithms are a group of techniques which let the computers learn and be able to generalice behaviors beginning from supplied data as examples; the most utilized techniques are artificial neuronal networks that simulate observed properties on the biologic neuronal systems through mathematic models recreated by artificial mechanisms (computers). In order to achieve the right training of an Artificial Neuronal Network, it must be chose the right configuration of layers and neurons, because a good combination of those lets get optimum solutions to the problem. For this reason, the construction of one of those networks is considered difficult, due the infinite of possible configurations that might achieve low training mistakes, without having certainty about the used configuration being the optimum.

Then, the genetic algorithms are used as a generating tool of configurations of artificial neuronal networks, taking advantage of its successful utilization in optimization problems and its vast exploration of the searching space, although the parallel processing is used for realizing a work distribution and reduce the time employed in this process. This way the solution is gotten through an exact, efficient and scalable system, which combines automatic apprentice techniques and distributed calculus.

---

* Grade work

** Faculty of physics-mechanics engineering, engineering of systems and informatics

Director Henry Arguello Fuentes. Codirector Erick Meneses Cuadros

1. INTRODUCCIÓN

Durante muchos años el ser humano ha tratado de facilitar las tareas de cálculo matemático complejo o repetitivo. A lo largo de la historia se ha ingeniado mecanismos que han contribuido al desarrollo de instrumentos o máquinas que han impulsado esta labor.

Es así como después de algunos inventos como el ábaco y la máquina aritmética de Pascal, Jhon Von Newman introduce la idea de una computadora compuesta por una unidad central de procesamiento (CPU) y una memoria, dicha computadora fue el modelo estándar para la construcción de los hoy comúnmente conocidos computadores secuenciales, llamados de esta manera por que ejecutan una única secuencia de instrucciones, con una única secuencia de datos. Sin embargo, cuentan con limitaciones en la capacidad de procesamiento así como en la velocidad de comunicación entre la CPU y la memoria. Estas limitaciones representan un obstáculo para el tipo de procesamiento que hoy en día es necesario en aplicaciones científicas, o de ingeniería que requieren procesar gran cantidad de datos con un gran número de instrucciones en un corto periodo de tiempo.

Para solucionar estos inconvenientes se planteó la utilización de múltiples procesadores los cuales ejecutan varias subtareas simultáneamente e intercambian resultados, de esta manera se termina la ejecución de un problema mucho más rápido con la implementación conjunta de varios procesadores que con un único procesador.

De esta forma nace la Computación de Alto Rendimiento (HPC, en ingles High Perfomance Computing) que precisamente aprovecha la interconexión de procesadores para ganar capacidad de cómputo, cambiando así el paradigma del mainframe, donde al aumentar la capacidad de procesamiento de una

computadora central, se incurre en una gran cantidad de dinero; con el nacimiento de la supercomputación se ven reducidos dichos costos, gracias a la utilización de equipos de bajo precio fácilmente escalables permitiendo además tenerlos distribuidos geográficamente.

La computación de altas prestaciones va de la mano de aplicaciones que necesitan de estas arquitecturas para poder desarrollarse, y teniendo una idea de las mejoras de desempeño que representa el uso de*l HPC* cuando es comparado con un equipo de un solo procesador, es factible comenzar a utilizarlo para la resolución de una gran variedad de problemas reales; entre estas aplicaciones se incluyen problemas de cálculo como simulación con ecuaciones diferenciales y elementos finitos, que son aplicadas a distintas disciplinas, tales como ciencias nucleares, meteorología, astronomía, química, electrónica, diseño, y prácticamente cualquier disciplina que pueda requerir la observación del comportamiento de algún sistema físico, que sea factible de simularse y que se pueda beneficiar de la capacidad de un cluster para realizar grandes cantidades de cálculo.

Dentro del conjunto de aplicaciones escalables con altos requerimientos de procesamiento se encuentran los algoritmos genéticos ya que su ejecución demanda gran cantidad de tiempo para converger debido a su aleatoriedad y su extenso espacio de búsqueda para la solución del problema; estos algoritmos hoy en día son utilizados con éxito en campos como la medicina, geología, química y aeronáutica. Muchas de estas herramientas buscan imitar las actividades de un ser humano para resolver problemas de distinta índole que sirven como acompañamiento y apoyo a expertos en temas específicos automatizando procesos que realizados manualmente pueden tardar mucho tiempo y que la calidad de respuesta alcanzada no sea óptima. Cabe aclarar que con estos sistemas no se busca el reemplazo del experto, al contrario se le da un

acompañamiento al trabajo permitiendo que éste dedique su tiempo a actividades para las cuales puede ser más productivo.

El contenido de este trabajo se centra en la computación de alto rendimiento, siendo destinada a la ejecución de una aplicación implementada en un sistema distribuido, usando técnicas de inteligencia artificial donde por medio de un algoritmo genético se optimiza el proceso de consecución de una configuración de red neuronal artificial que prediga de forma confiable los resultados de los datos suministrados por Instituto Colombiano de Petróleos ICP.

2. DESCRIPCIÓN DEL PROYECTO

2.1. TITULO

IMPLEMENTACIÓN DE ALGORITMOS GENÉTICOS EN UN SISTEMA DISTRIBUIDO COMO HERRAMIENTA PARA LA GENERACIÓN AUTOMÁTICA DE REDES NEURONALES UTILIZANDO DATOS DE CRUDOS Y FRACCIONES.

2.2. DIRECTOR

MPE. Henry Arguello Fuentes
Universidad Industrial de Santander, Bucaramanga Colombia.
henarfu@uis.edu.co

2.3. CODIRECTOR

Ing. Erick Ramón Meneses Cuadros.
Universidad Industrial de Santander, Bucaramanga Colombia.
Erick.meneses@imag.fr

2.4. AUTORES

Vanessa Rey García
Estudiante de Ingeniería de Sistemas
Código: 2032376
vannerey@linux.com.co

Álvaro Sánchez Villalba
Estudiante de Ingeniería de Sistemas
Código: 2032370
asv@linux.com.co

3. ENTIDADES INTERESADAS EN EL PROYECTO

- **Universidad Industrial de Santander –** Escuela de Ingeniería de Sistemas e Informática, Bucaramanga – Colombia.

- **Instituto Colombiano del Petróleo** – ICP Convenio de Cooperación Tecnológica 003 de 2007.

- **Grupo de Investigación en Ingeniería Biomédica** – GIIB UIS, Bucaramanga – Colombia.

- **Centro de Tecnologías de la Información y la Comunicación de la UIS** – CENTIC, Bucaramanga -Colombia.

4. PLANTEAMIENTO DEL PROBLEMA

La ciencia y la tecnología han evolucionado basadas en el estudio del comportamiento de la naturaleza, formulando nuevos mecanismos, métodos, técnicas y teorías para solucionar, interpretar y mejorar nuestra relación con el mundo; hoy en día la informática en su afán de hacer los sistemas más eficientes y confiables, se basa en la observación de los fenómenos naturales para generar nuevas alternativas de solución a problemas relacionados con la computación y la matemática.

Debido a esto, los investigadores han centrado su atención en el desarrollo de nuevos sistemas de tratamiento de la información que permitan interpretar el mundo, tal como lo hace el cerebro humano; este órgano biológico cuenta con varias características deseables para cualquier sistema de procesamiento digital; éste recibe señales de entrada de muchas fuentes y las procesa con el fin de crear una apropiada respuesta de salida.

Una de las misiones de una Red Neuronal Artificial consiste en simular las propiedades observadas en los sistemas neuronales biológicos a través de modelos matemáticos recreados mediante mecanismos artificiales (como un circuito integrado, un ordenador o un conjunto de válvulas). El objetivo es conseguir que las máquinas den respuestas similares a las que es capaz de dar el cerebro que se caracterizan por su generalización y su robustez [1]. Dentro de las Redes Neuronales Artificiales, los elementos de procesamiento se encuentran agrupados por capas, una capa es una colección de neuronas, a su vez, estas neuronas tienen factores de peso para ajustar el valor de la fuerza de una interconexión con otras neuronas. Para lograr el correcto entrenamiento de una red se debe elegir la configuración de los elementos antes mencionados de la red neuronal, ya que la correcta combinación de estas nos puede o no dar solución a

nuestro problema. Por esta razón puede llegar a ser tediosa la construcción de una de estas redes, debido a la infinidad de configuraciones posibles en cuanto al número de capas y número de neuronas por capa, teniendo en cuenta que una red neuronal artificial para un problema específico podría tener varias configuraciones y resolver igualmente el problema.

Dado que no existe una metodología clara que indique el proceso para elegir correctamente la red, se debe recurrir a la técnica de prueba y error, la cual además de no garantizar una respuesta óptima, toma gran cantidad de tiempo. Como respuesta a esto aparecen los algoritmos genéticos que hacen parte de la computación evolutiva, están basados en la evolución natural biológica (Teoría de la evolución de Darwin), y pueden adaptarse a la resolución de múltiples problemas donde el dominio de la solución pueda resultar demasiado extenso mostrando ser una gran herramienta para la optimización de problemas, los AG son la opción acertada para la búsqueda de la configuración óptima de la red neuronal artificial del problema propuesto.

Ahora, las operaciones utilizadas por el Algoritmo Genético toman gran tiempo de procesamiento; algunas veces el tiempo que toma encontrar una configuración de red neuronal por medio de los AG's es el mismo o mayor al que le toma a una persona en el proceso de ensayo y error. En este sentido, este trabajo de investigación busca mejorar la capacidad de procesamiento de los algoritmos genéticos, esto traducido a menor cantidad de tiempo empleado.

Se plantea entonces como solución la paralelización de los Algoritmos Genéticos, dividiendo el problema global en subproblemas que son asignados a varios procesadores, los cuales disminuirán el tiempo y la labor que anteriormente realizaba una sola cpu; de esta forma al colaborar todos para llegar a la solución global del problema se puede ampliar el espacio de búsqueda y se encontrar mejores soluciones que las del algoritmo secuencial.

5. OBJETIVOS

5.1. OBJETIVO GENERAL

Desarrollar un sistema que determine la configuración óptima de una red neuronal artificial mediante algoritmos genéticos implementados en un ambiente distribuido para realizar el modelado de datos de crudos y fracciones del Instituto Colombiano de Petróleos (ICP).

5.2. OBJETIVOS ESPECÍFICOS

- Implementar una red neuronal artificial que usara para su entrenamiento datos proporcionados por el Instituto Colombiano de Petróleos (ICP), con el fin de analizar y evaluar los resultados en cuanto a tiempo de procesador y calidad de la respuesta.

- Diseñar e implementar el algoritmo genético paralelo que optimice la comunicación entre procesadores (disminuir frecuencia y volumen de información), basado en etapas de particionamiento, comunicación, agrupación y asignación; que permita obtener confiabilidad en los resultados y mayor velocidad en los tiempos de respuesta.

- Verificar el desempeño de las redes neuronales artificiales generadas automáticamente por el algoritmo genético paralelo sobre el ambiente distribuido, entrenadas con datos suministrados por el ICP, comparando las capacidades de generalización entre ellas y contrastándolas con las obtenidas por el algoritmo genético secuencial y con redes neuronales diseñadas por expertos u otros métodos.

6. JUSTIFICACIÓN

La tecnología avanza cada día a pasos agigantados, hoy en día la capacidad de procesamiento de un ordenador es asombrosa, sin embargo las demandas de procesamiento son cada día mayores, es decir, se tiene la necesidad de procesar enormes volúmenes de información científica y comercial y evaluarla con una velocidad de procesamiento que no limite los resultados esperados, por lo contrario debe proporcionarse los elementos necesarios que garanticen la rapidez y eficacia en el procesamiento de dicho volumen de información.

Debido al gran tiempo de procesamiento que emplean los algoritmos genéticos en un espacio de búsqueda amplio, se plantea la necesidad de hacer uso de múltiples procesadores en un ambiente distribuido, los cuales disminuirán el tiempo y la labor que anteriormente realizaba una sola cpu.

Afortunadamente el algoritmo es intrínsecamente paralelo, la evaluación de la función de desempeño se puede realizar de manera concurrente para diferentes cromosomas al disponerse de múltiples procesadores y de un canal de comunicación entre ellos; además muchos operadores de búsqueda pueden ser distribuidos fácilmente, y obtenerse así grandes incrementos de velocidad en problemas difíciles.

Las técnicas de procesamiento paralelo y distribuido se aplican al modelo clásico de AG con el objetivo de obtener mejoras desde el punto de vista de la eficiencia y para perfeccionar la calidad de la búsqueda genética. Desde la perspectiva de la eficiencia, paralelizar un AG permite afrontar la lentitud de convergencia para problemas cuya dimensión motiva el uso de poblaciones numerosas, o múltiples evaluaciones de funciones de optimización costosas. Desde el punto de vista algorítmico, los Algoritmos Genéticos Paralelos (AGP) pueden explotar el paralelismo intrínseco del mecanismo evolutivo, trabajando simultáneamente sobre varias poblaciones semi-independientes para resolver el mismo problema. Eventuales intercambios de soluciones (migraciones) introducen diversidad para evitar problemas de convergencia en óptimos locales. Así mismo, los AGP pueden

aprovechar características de paralelismo propias del problema, analizando concurrentemente diferentes secciones del espacio de búsqueda.

La implementación en este trabajo de investigación de algoritmos genéticos en un entorno distribuido, permitirá evaluar y comparar las verdaderas mejoras que pueden ser aportadas por las tecnologías paralelas en comparación con resultados en los ambientes secuenciales, además al encontrar una estrategia de paralelización adecuada se logrará encontrar un balance entre la frecuencia y volumen de información comunicada para garantizar el buen funcionamiento del algoritmo genético, sin limitar la capacidad de cómputo de alto rendimiento ofrecida por dicho entorno.

CAPITULO 7:

MARCO CONCEPTUAL

7. MARCO CONCEPTUAL

7.1. REDES NEURONALES

Las redes neuronales artificiales (RNA) son modelos que intentan reproducir el comportamiento del cerebro simulando las propiedades observadas en los sistemas neuronales biológicos a través de modelos matemáticos recreados mediante mecanismos artificiales.

Las neuronas se encuentran agrupadas formando capas, y existen tres tipos de ellas: capa de entrada, de salida y capas ocultas. Las unidades de entrada reciben señales desde el entorno; las de salida envían la señal fuera de la red, y las unidades ocultas son aquellas cuyas entradas y salidas se encuentran dentro del sistema.

Biológicamente, un cerebro aprende mediante la reorganización de las conexiones sinápticas entre las neuronas que lo componen. De la misma manera, las RNA tienen un gran número de “procesadores virtuales” interconectados que de forma simplificada simulan la funcionalidad de las neuronas biológicas. En esta simulación, la reorganización de las conexiones sinápticas biológicas se modela mediante un mecanismo de pesos, que son ajustados durante la fase de aprendizaje. En una RNA entrenada, el conjunto de los pesos determina el conocimiento de esa RNA y tiene la propiedad de resolver el problema para el que la RNA ha sido entrenada.

7.1.1. CONFIGURACIÓN DE RED NEURONAL

La configuración de las capas y las neuronas en cada capa es de vital importancia para conseguir un buen entrenamiento de la red, ya que son las neuronas, su ubicación en la red y los pesos de las conexiones entre ellas las que permiten que sea construido un buen modelo matemático durante el entrenamiento que simule con gran precisión entradas que se den a la red.

El proceso de consecución de una configuración de red neuronal generalmente se realiza a prueba y error, entrenando y comparando errores entre las diferentes redes; como es de esperarse este proceso es tedioso para el científico, además no garantiza la exploración de un gran número de configuraciones y no se tiene certeza que la configuración escogida sea le de menor error entre un gran número de configuraciones probadas.

7.1.2. ENTRENAMIENTO DE UNA RED NEURONAL

El aprendizaje o entrenamiento es el proceso por el cual una red neuronal modifica sus pesos en respuesta a una información de entrada. Los cambios que se producen durante el proceso de aprendizaje se reducen a la destrucción, modificación y creación de conexiones entre las neuronas,

El entrenamiento se realiza por medio de un algoritmo de aprendizaje denominado “Backpropagation” que es un método descendente de gradiente, con el cual podemos minimizar el error cuadrático total de la salida calculada por la red. En la fase de entrenamiento, se usa un conjunto de datos o patrones de entrenamiento (entradas y salidas) para determinar los pesos que definen el modelo neuronal. Una vez entrenado este modelo, se usará en la llamada fase de prueba, en la que se verificará el correcto funcionamiento de la red neuronal con un conjunto de datos que no se ha presentado previamente a la red. De esta manera se tiene un modelo que podrá ser utilizado para predecir las salidas de nuevos modelos.

7.1.2.1. VALIDACIÓN CRUZADA

El objetivo del aprendizaje es que la red se entrene bien de tal forma que aprenda lo suficiente acerca con los datos de entrenamiento para generalizar en el futuro.

En este contexto, una herramienta estándar en estadística conocida como validación cruzada provee una guía interesante [4]. El conjunto de entrenamiento

se particiona aleatoriamente en dos subconjuntos:

- El subconjunto de entrenamiento, usado para entrenar el modelo.
- El subconjunto de validación, usado para evaluar o validar el modelo.

El modelo se valida sobre un conjunto de datos diferente de aquel utilizado para el entrenamiento de la red neuronal. De este modo se puede usar el conjunto de entrenamiento para evaluar el desempeño de varios modelos candidatos, y así elegir el mejor.
El uso de la validación cruzada resulta particularmente interesante cuando se debe diseñar una red neuronal grande que tenga como objetivo la generalización.

7.1.2.2. DETENCIÓN TEMPRANA

Una red perceptrón multicapa entrenado con el algoritmo de back-propagation aprende en etapas, moviéndose desde la realización de funciones de mapeo bastante simples a más complejas a medida que progresa la sesión de entrenamiento. Esto se ejemplifica por el hecho de que en una situación típica el error cuadrático medio disminuye con el incremento del número de repeticiones durante el entrenamiento: comienza con un valor grande, decrece rápidamente, y luego continúa disminuyendo lentamente mientras la red hace su camino hacia un mínimo local sobre la superficie de error. Teniendo como meta una buena generalización, es muy difícil darse cuenta cuándo es el mejor momento de detener el entrenamiento si solamente se está observando a la curva de aprendizaje para el entrenamiento.
El subconjunto de entrenamiento se utiliza para entrenar a la red en el modo usual, deteniéndose periódicamente, y se evalúa la red con el conjunto de validación después de cada período de entrenamiento.
Cuando la fase de validación se completa, el entrenamiento se reanuda para otro período y el proceso se repite. Este procedimiento se denomina método de entrenamiento con detención temprana.

Normalmente, cuando la red comienza a sobre-entrenarse, el valor del error de entrenamiento sigue disminuyendo, pero el error de validación, que es monitoreado en todo momento, comienza a aumentar. Cuando el error de validación aumenta en varias iteraciones consecutivas, es el momento de detener el entrenamiento y tomar los parámetros de la red correspondientes al mínimo error de validación.

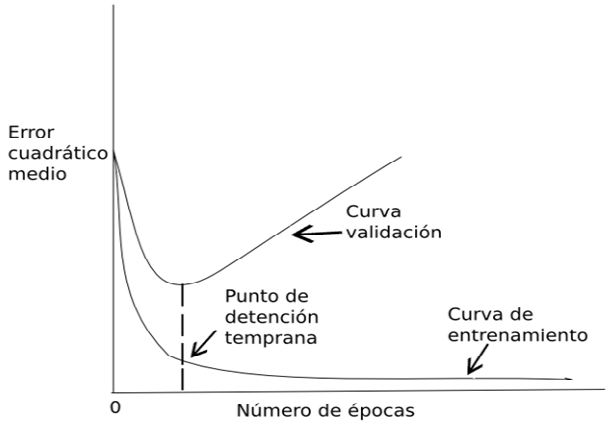


Ilustración 1: Regla de detención temprana basada en validación cruzada

Fuente: Autores proyecto

7.1.3. GENERALIZACIÓN DE UNA RED NEURONAL

El entrenamiento de una red, tiene un impacto significativo en el desempeño de la red y su habilidad para generalizar. Si una red no tiene suficientes conexiones entre neuronas, el algoritmo de entrenamiento puede no converger nunca y la red neuronal no es capaz de aproximar la función. Por otro lado, en una red densamente conectada, puede ocurrir un sobreajuste.

El sobreajuste es un problema de los modelos estadísticos donde se presentan demasiados parámetros. Esto es una mala situación porque en lugar de aprender

a aproximar la función presente en los datos, la red simplemente puede memorizar cada ejemplo de entrenamiento. El ruido en los datos de entrenamiento se aprende como parte de la función, afectando la habilidad de la red para generalizar.

7.2. ALGORITMOS GENÉTICOS

Los Algoritmos Genéticos son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización, están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin.

Los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real, la evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas; estas posibles soluciones son representadas en el algoritmo mediante los individuos que pertenecen a la población, éstos son seleccionados, cruzados y mutados permitiendo la evolución de la población hacia mejores individuos mas fuertes y mejores adaptados.

Los algoritmos genéticos establecen una analogía entre el conjunto de soluciones de un problema, llamado fenotipo[1], y el conjunto de individuos de una población natural, codificando la información de cada solución en una cadena, generalmente binaria, llamada cromosoma. Los símbolos que forman la cadena son llamados los genes. Cuando la representación de los cromosomas se hace con cadenas de dígitos binarios se le conoce como genotipo[2]. Los cromosomas evolucionan a través de iteraciones, llamadas generaciones. En cada generación, los

---

1 Fenotipo: Características externas observables de un organismo

2 Genotipo: Constitución genética de un organismo

cromosomas son evaluados usando alguna medida de aptitud. Las siguientes generaciones (nuevos cromosomas), llamada descendencia, se forman utilizando los operadores, de cruzamiento y de mutación.

7.2.1. TOPOLOGÍAS DE ALGORITMOS GENÉTICOS PARALELOS

Los algoritmos genéticos paralelos se clasifican según la forma en que se distribuye la carga de cómputo en los múltiples procesadores así como la topología y el diseño propio del algoritmo genético.

7.2.1.1. VARIOS ALGORITMOS GENÉTICOS SECUENCIALES EN PARALELO

Como su nombre lo indica varios algoritmos genéticos secuenciales se ejecutan en paralelo con el fin de obtener de cada uno respuestas óptimas, no existe interacción en el procesamiento de cada algoritmo ni en los resultados obtenidos de cada uno de ellos.

7.2.1.2. ALGORITMOS GENÉTICOS MAESTRO-ESCLAVO O COMUNICACIÓN ESTRELLA

Se distribuyen las funciones de aptitud entre procesadores esclavos, los resultados se reportan al procesador maestro el cual procesa la población de posibles soluciones aplicando los operadores restantes de selección, cruce y mutación de manera centralizada.

Durante la evaluación de una población, el programa maestro reparte los individuos entre los nodos esclavos disponibles y envía a cada uno de éstos el cromosoma correspondiente. Cuando un esclavo finaliza el cálculo de aptitud de un individuo, envía este valor al programa maestro y luego recibe de éste un

nuevo cromosoma. El proceso se repite hasta completar la evaluación de todos lo individuos de la población.

Una ventaja de esta estrategia es su escalabilidad, esto significa que se pueden utilizar tantos procesadores como se desee acelerando el proceso global manteniendo la eficiencia.

Esta implementación tiene la desventaja de hacer uso intensivo del canal de comunicación. De esta forma el tiempo que toman los cromosomas en el paso de mensajes podría ser comparable al tiempo de procesador del algoritmo secuencial. De hecho se ha demostrado que en su punto de operación óptimo, un algoritmo genético maestro-esclavo sólo utiliza los procesadores el 50% del tiempo.

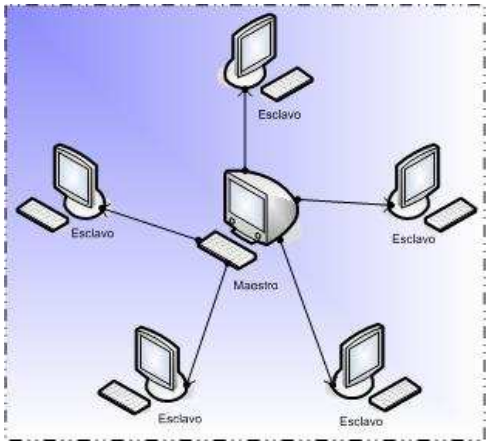


Ilustración 2: Implementación maestro esclavo

Fuente: Autores proyecto

7.2.1.3. ALGORITMOS GENÉTICOS DE GRANO GRUESO

En este tipo de algoritmos genéticos existen múltiples poblaciones, tantas como procesadores se emplean. Cada procesador implementa el algoritmo completo sobre una población ubicada en su espacio de memoria, cada una de las poblaciones evolucionan de manera independiente, hasta que un evento

preestablecido acontece, y los mejores cromosomas de cada población emigran hacia poblaciones vecinas. Los cromosomas que emigran substituyen algunos de los cromosomas de las poblaciones receptoras. La finalidad de la emigración es introducir nueva información que evite que la población converja a una solución óptima local (máximo o mínimo). El evento que produce la migración puede ser: la convergencia de todas las poblaciones, un tiempo preestablecido, un número definido de generaciones transcurridas.

La vecindad entre poblaciones no necesariamente corresponde a la vecindad física de los procesadores establecida por el tipo de canal de comunicación, sino a una vecindad lógica. La definición de dichas vecindades establece la topología del algoritmo genético paralelo.

El tiempo de convergencia de una población depende, entre otros factores, del número de cromosomas que la conforman. A mayor número de cromosomas mayor tiempo de convergencia. Esto se debe a dos factores: el primero es que entre más numerosa sea la población más tiempo toma por generación el evaluar los cromosomas y el segundo es que número de generaciones requeridas para el algoritmo converja es mayor entre mayor es el número de cromosomas de la población. Este tipo de algoritmos genéticos reducen el tiempo de convergencia al reducir el tamaño de las poblaciones. Una población pequeña tiene el problema de que puede converger hacia un óptimo local. La migración de individuos de una población a otra elimina este problema.

Existen varias topologías comunes para la comunicación entre subpoblaciones:

TOPOLOGÍAS:

- Modelo de las Islas:

Se permite enviar los individuos a cualquier subpoblación, se emplea el modelo de las islas para mostrar como los algoritmos genéticos paralelos de grano grueso se comportan como si el mundo fuera constituido por islas que se desarrollan en forma independiente, una de las otras. En cada una de las islas, la población es

libre de converger hacia un óptimo diferente. El operador de migración permite extraer de las diferentes subpoblaciones las buenas características para luego mezclarlas.

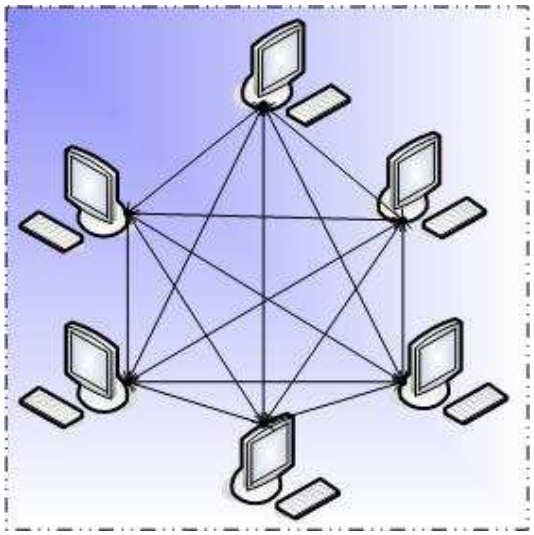


Ilustración 3: Modelo de Islas

Fuente: Autores proyecto

- Modelo de la Pasarela (Stepping Stone):

La migración esta limitada, ya que sólo se permite que los emigrantes se desplacen a las subpoblaciones vecinas.

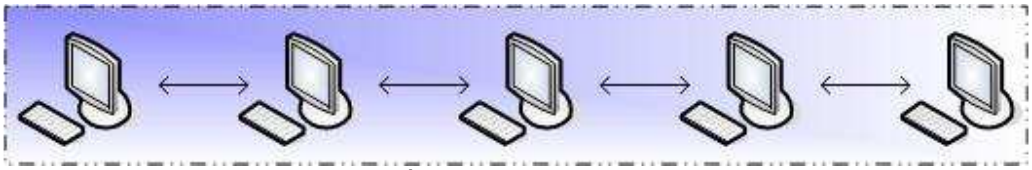


Ilustración 4: Modelo de la Pasarela

Fuente: Autores proyecto

7.2.1.4. ALGORITMOS GENÉTICOS DE GRANO FINO

La paralelización del programa se realiza a nivel de instrucción. Cada procesador

hace una parte de cada paso del algoritmo (selección, cruce y mutación) sobre la población común.

Este modelo asigna un individuo a cada elemento de procesamiento. Cada individuo forma parte de varias subpoblaciones, solapadas entre sí, que determinan la adyacencia entre individuos de acuerdo a una topología de conexión. La alta conectividad entre vecinos incrementa la difusión de individuos aptos, pero hace a la población susceptible al dominio de algún esquema de codificación que podría llevar a una convergencia prematura. Como al limitar las interacciones se reduce el desempeño, se plantea que el modelo debe lograr un compromiso entre estos factores para obtener soluciones de calidad, con buen desempeño.

7.2.1.5. ALGORITMOS GENÉTICOS EN IMPLEMENTACIÓN HÍBRIDA

Combinación de la implementación de grano fino y de grano grueso, existe una jerarquía de dos niveles el cual en el nivel superior suele haber siempre un algoritmo de grano grueso y en el nivel inferior puede haber una implementación maestro-esclavo, grano fino o grano grueso.

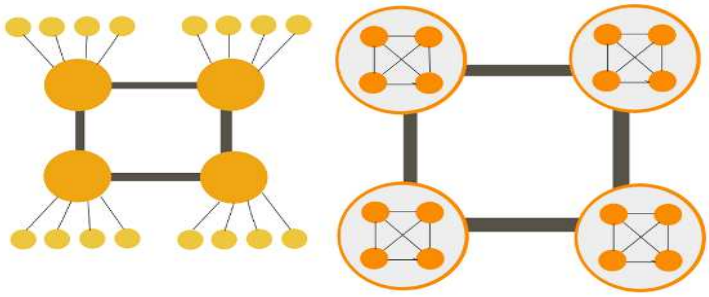


Ilustración 5: Implementación híbrida

a) Maestro-Esclavo en el nivel inferior b) Grano fino en el nivel inferior

Fuente: Autores proyecto

Un enfoque semi-distribuido propone resolver el equilibrio entre calidad de la búsqueda y desempeño del algoritmo, organizando los procesos en clusters sin generar un aumento excesivo en las comunicaciones.
De este modo, la propuesta comparte algunas de las ventajas de los modelos distribuidos y otras de los centralizados.

Bianchini y Brown [10] proponen una taxonomía de AGP, que toma en cuenta el modo de distribución de la población:

- Implementación centralizada: modelo maestro-esclavo de población única. Se introducen variantes para evitar los cuellos de botella en las comunicaciones, asignando varios individuos a los procesos esclavos y proporcionándoles la lógica para evolucionar aisladamente por un cierto número de generaciones.

- Implementación semi-distribuida: modelo que organiza clusters de procesadores, que trabajan al estilo del modelo centralizado, con esclavos dotados de lógica extendida y maestros de los clusters que intercambian sus mejores individuos. Esta organización reduce el problema de acceso a un único maestro, aunque incluye una sobrecarga para la replicación semi-distribuida y las nuevas comunicaciones entre procesos maestros.

- Implementación distribuida: modelo que elimina las poblaciones compartidas. Cada procesador mantiene una porción de la población y aplica un AG secuencial sobre ella, con migración de los mejores individuos.

- Implementación totalmente distribuida: corresponde al modelo anterior, sin migración entre subpoblaciones. La paralelización de grano

grueso tiene como atractivo la portabilidad, ya que se adapta perfectamente tanto a multiprocesadores de memoria distribuida como de memoria compartida. Este tipo de paralelización se puede a su vez realizar siguiendo tres estilos distintos de programación:

a) Paralelización en datos: El compilador se encarga de la distribución de los datos guiado por un conjunto de directivas que introduce el programador. Estas directivas hacen que cuando se compila el programa las funciones se distribuyan entre los procesadores disponibles. Como principal ventaja presenta su facilidad de programación. Los lenguajes de paralelismo de datos mas utilizados son el estándar HPF (High Performance Fortran) y el OpenMP.

b) Programación por paso de mensajes: El método más utilizado para programar sistemas de memoria distribuida es el paso de mensajes o alguna variante del mismo. La forma mas básica consiste en que los procesos coordinan sus actividades mediante el envío y la recepción de mensajes. Las librerías mas utilizadas son por este orden la estándar MPI (Message Passing Interface) y PVM (Parallel Virtual Machine).

c) Programación por paso de datos: A diferencia del modelo de paso de mensajes, la transferencia de datos entre los procesadores se realiza con primitivas unilaterales tipo put-get, lo que evita la necesidad de sincronización entre los procesadores emisor y receptor. Es un modelo de programación de bajo nivel pero muy eficiente, aunque en la actualidad son muy pocos los fabricantes que los soportan.

7.3. ALGORITMOS GENÉTICOS PARA LA GENERACIÓN AUTOMÁTICA DE CONFIGURACIONES DE RED NEURONAL

El diseño de una configuración óptima para una red neuronal [2] puede verse

como un problema de búsqueda en el espacio de configuraciones donde cada punto representa una configuración distinta. Dado un criterio de desempeño acerca de las configuraciones, como por ejemplo el error de entrenamiento más bajo, la complejidad de red más baja, entre otras; el nivel de desempeño de todas las configuraciones formará una superficie discreta en el espacio. Obtener el diseño de configuración óptima es equivalente a encontrar el punto más alto sobre esta superficie.

Debido a que esta superficie es infinitamente grande, en este proceso de búsqueda los algoritmos genéticos son una buena opción para el problema de optimización, ya que se basan en la teoría de la evolución genética y en el concepto de la supervivencia del más apto [3], son de naturaleza estocástica, tienen capacidad de considerar simultáneamente una población de soluciones, además de ser adaptables ante un rango amplio de problemas.

Se debe tener en cuenta que en el espacio de búsqueda, se encuentran configuraciones similares que pueden tener un desempeño bastante diferente, o diferentes configuraciones pueden tener un desempeño bastante similar, además el mapeo desde una configuración hacia su desempeño depende del método de evaluación utilizado.

Para implementar el algoritmo genético es necesaria la definición de la población de individuos así como su genotipo y fenotipo. Un genotipo es el contenido genético de un individuo, en forma de ADN, puede pensarse como un arreglo (una cadena) de genes, donde cada gen toma valores de un dominio definido adecuadamente. Cada genotipo codifica un fenotipo o solución candidata en el dominio de interés, en este caso, una clase de configuración neuronal. Tales codificaciones podrían emplear genes que toman valores numéricos para representar unos pocos parámetros o estructuras complejas de símbolos que se transforman en fenotipos (en este caso redes neuronales) por medio del proceso de decodificación apropiado. Se calcula a las redes neuronales resultantes (los fenotipos) la función aptitud, es decir, son entrenadas y evaluadas con datos; esta evaluación de un fenotipo, determina que tan bueno es el genotipo

correspondiente.

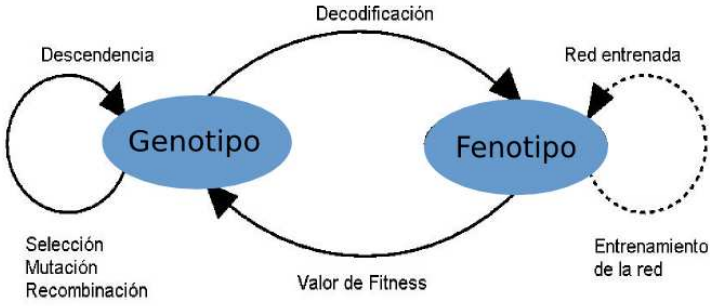


Ilustración 6: Proceso de diseño evolutivo de configuraciones de neuronales

Fuente: Autores proyecto

Como indica la figura anterior, el procedimiento evolutivo trabaja en una población de individuos, preferentemente se seleccionan los genotipos que después de la decodificación producen fenotipos con valores de aptitud altos, estos valores son generados a partir de los errores obtenidos al simular las redes que representan. Posteriormente se aplican los operadores genéticos tales como la mutación y cruzamiento, para introducir variedad dentro de la población y probar variantes de soluciones candidatas representadas en la población actual. Así, sobre varias generaciones, la población gradualmente evolucionará hacia genotipos que corresponden a fenotipos de aptitud alta.

La evaluación de la función aptitud es ruidosa, es decir puede introducir cierto error, ya que al evaluar la aptitud del genotipo se realiza con la configuración real (es decir, el fenotipo creado a partir del genotipo), y el mapeo entre fenotipo y genotipo no es uno-a-uno. En la práctica, un modo de evaluar genotipos es transformando un genotipo (una configuración codificada) en un fenotipo (una configuración completamente especificada) y posteriormente entrenar al fenotipo a partir de distintos pesos iniciales de conexión generados al azar. El resultado del entrenamiento se utiliza como función de aptitud para evaluar el genotipo. En otras palabras, se utiliza la aptitud del fenotipo para estimar la aptitud del genotipo.

Existen dos fuentes principales de ruido:

1. La primera fuente es la inicialización de los pesos al azar. Diferentes pesos iniciales aleatorios pueden producir diferentes resultados de entrenamiento. Así, el mismo genotipo puede tener una aptitud bastante distinta debido a los distintos pesos iniciales aleatorios utilizados en el entrenamiento.
2. La segunda fuente es el algoritmo de entrenamiento. Diferentes algoritmos de entrenamiento pueden producir diferentes resultados de entrenamiento aún partiendo del mismo conjunto de pesos iniciales.

Tal ruido puede engañar al algoritmo genético evolutivo, por el hecho de que la aptitud de un fenotipo generado a partir del genotipo G1 sea más alto que aquel generado a partir del genotipo G2 no implica que G1 tenga verdaderamente mejor calidad que G2.

Un método para reducir el ruido de la función aptitud es entrenar a cada configuración muchas veces partiendo de diferentes pesos iniciales aleatorios. Se toma entonces el mejor resultado para estimar la aptitud del genotipo.

7.4. COMPUTACIÓN PARALELA

En la actualidad el procesamiento paralelo es una de las herramientas más importantes que existen para resolver problemas computacionalmente intensivos. Algunos problemas que al ejecutarse en un solo procesador requieren días o meses de procesamiento, en una computadora puede realizarse en fracciones de horas.

Una máquina paralela es básicamente aquella que cuenta con dos o más procesadores y por lo tanto puede ejecutar varias instrucciones de manera simultánea, el poder de computación de estas máquinas puede ser incrementado haciendo crecer el número de procesadores con los que cuentan, en vez de reemplazar un sólo procesador con otros cada vez más complejos como ocurre en las computadoras tradicionales.

Con la computación en paralelo se resuelven problemas demandantes en recursos como procesamiento, almacenamiento y comunicación, estos problemas pueden ser implementados sobre arquitecturas distribuidas que serán descritas en la siguiente sección

7.4.1. SISTEMAS DISTRIBUIDOS

La definición mas concreta acerca de los sistemas distribuidos es tal ves la que proporciona Tanenbaum[5] en su libro:

"*Un sistema distribuido es una colección de computadoras independientes que aparecen ante los usuarios como una única unidad de procesamiento".*

Esta definición se remite a dos aspectos diferentes pero a la vez mutuamente dependientes, uno es el hardware cuando se menciona a las maquinas autónomas y software cuando menciona que para los usuarios es trasparente el sistema distribuido.

Hacia los años 80 el experto en computadoras Herb Grosch enunció lo que después se conocería como la ley de Grosch "El poder de cómputo es proporcional al cuadrado de su precio.", es decir, si se paga el doble se obtiene cuatro veces el desempeño. Esta declaración encajó bien en la tecnología de mainframe de su tiempo y provocó que muchas empresas compraran una sola máquina muy costosa. Con el nacimiento de el microprocesador la ley de Grosch ya no es válida, por unos pocos miles de pesos es posible obtener un microcircuito de CPU que puede ejecutar más instrucciones por segundo de las que realizaba una de las mas grandes de estas mainframe de la década de los 80.

De esta manera, la utilización de varios equipos de bajo costo unidos en un mismo sistema es la solución más eficiente, ya que se obtiene el mismo poder de procesamiento con un costo muy reducido.

Aunque los sistemas distribuidos constan de varias CPU existen diversas formas de organizar el hardware en la forma de interconectarlo y comunicarlo entre sí.

Flynn[6] propone una clasificación para estos sistemas entre los cuales se encuentra MIMD (múltiple instrucción, múltiples datos); diferenciando dos clases de sistemas:

- Memoria Compartida, que por lo general se conocen como multiprocesadores, aunque los tiempos de retardo son cortos, la escalabilidad es limitada y presentan cuellos de botella en el acceso a memoria, además son demasiado costosos.
- Memoria Distribuida, también conocidos como multicomputadores, son altamente escalables, fáciles de construir, pueden hacer uso de recursos no utilizados u ociosos o de bajo de costo, la implementación más conocida son los clusters computacionales.

7.4.1.1. CLUSTER COMPUTACIONAL

El término de cluster es asignado a los conglomerados de computadores construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen un único computador [7].

La computación sobre Clusters surge como resultado de la convergencia de varias tendencias que incluyen, la disponibilidad de microprocesadores de alto rendimiento más económicos y redes de alta velocidad, el desarrollo de aplicaciones de software para cómputo distribuido de alto rendimiento, y la creciente necesidad de potencia computacional para aplicaciones en las ciencias computacionales y comerciales. Los Clusters son usualmente empleados para mejorar el rendimiento y/o la disponibilidad por encima de la que es provista por un solo computador típicamente siendo más económico que computadores individuales de rapidez y disponibilidad comparables.

Las aplicaciones paralelas escalables requieren: buen rendimiento, baja latencia, comunicaciones que dispongan de gran ancho de banda, y acceso rápido a archivos. Un cluster puede satisfacer estos requerimientos usando los recursos que tiene asociados a él.

La tecnología cluster permite a las organizaciones incrementar su capacidad de procesamiento usando tecnología estándar, tanto en componentes de hardware como de software que pueden adquirirse a un costo relativamente bajo.

Los clusters pueden clasificarse con base en sus características:

- **CLUSTER DE ALTA DISPONIBILIDAD:** Un cluster de alta disponibilidades un conjunto de dos o más máquinas que se caracterizan por mantener una serie de servicios compartidos y por estar constantemente monitorizándose entre sí, cuando uno falla el otro los sustituye.
- **CLUSTER DE BALANCEO DE CARGA:** Está compuesto por uno o más ordenadores (*nodos*) que actúan como front-end del cluster, y que se ocupan de repartir las peticiones de servicio que reciba el cluster, a otros ordenadores del cluster que forman el back-end de éste
- **CLUSTER DE ALTO RENDIMIENTO:** Un cluster de alto rendimiento es un conjunto de ordenadores que está diseñado para dar altas prestaciones en cuanto a capacidad de cálculo. Los motivos para utilizar un cluster de alto rendimiento son el tamaño del problema por resolver y el precio de la máquina necesaria para resolverlo. Por medio de un cluster se pueden conseguir capacidades de cálculo superiores a las de un computador más caro que el costo conjunto de los ordenadores del cluster.

Para garantizar esta capacidad de cálculo, los problemas deben ser paralelizables, ya que el método con el que los clusters agilizan el procesamiento es dividir el problema en más pequeños y realizar las operaciones de cálculo en los nodos, por lo tanto, si el problema no cumple con esta característica, no puede utilizarse el cluster para su cálculo.

7.4.2. ARQUITECTURA DEL CLUSTER

En esta sección se expone las diferentes capas en la arquitectura de un Cluster computacional recorriendo la siguiente imagen de manera ascendente se

explicarán cada una de estas.

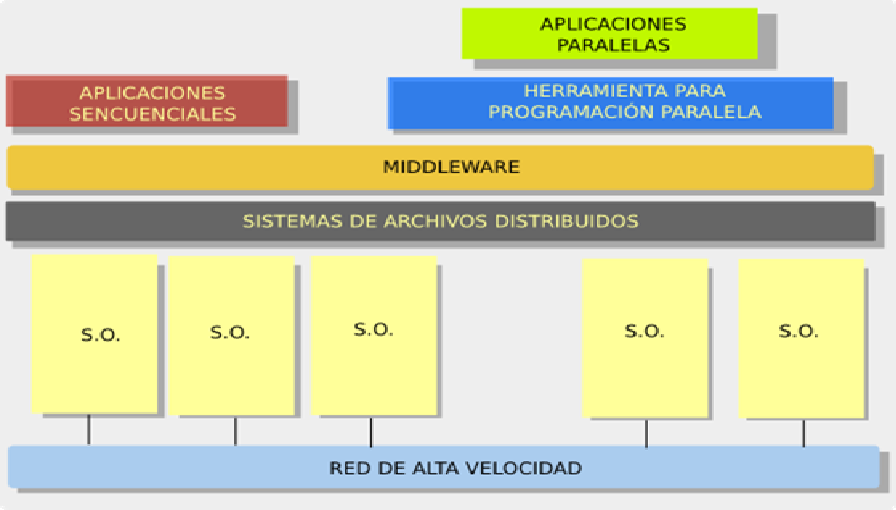


Ilustración 7: Capas de la arquitectura cluster

Fuente: Autores proyecto

7.4.2.1. CAPA DE RED

Las redes son una pieza fundamental de la capa subyacente al hardware, estas se caracterizan según su cobertura (local, nacional, internacional) y su desempeño, entendido como la cantidad de datos que puede transmitir en determinado intervalo de tiempo. El desempeño generalmente se mide en Kilo, Mega o Giga Bits por segundo, en un cluster es necesario contar con un red de área local para lograr la comunicación entre los nodos. En esta capa se encuentran protocolos estándar de seguridad y comunicación para transacciones de red. Los protocolos de comunicación permiten el intercambio de datos entre la capa más inferior y la de recursos mientras que los protocolos de seguridad brindan mecanismos de criptografía para identificar usuarios y recursos.

7.4.2.2. SISTEMA OPERATIVO

El sistema operativo es el software más importante de un ordenador. Para que

funcionen los otros programas, cada ordenador de uso general debe tener un sistema operativo. Los sistemas operativos realizan tareas básicas, tales como reconocimiento de la conexión del teclado, enviar la información a la pantalla, no perder de vista archivos y directorios en el disco, y controlar los dispositivos periféricos tales como impresoras, escáner, etc.

En sistemas grandes, el sistema operativo tiene incluso mayor responsabilidad y poder, es como un policía de tráfico, se asegura de que los programas y usuarios que están funcionando al mismo tiempo no interfieran entre ellos. El sistema operativo también es responsable de la seguridad, asegurándose de que los usuarios no autorizados no tengan acceso al sistema.

Para el montaje del cluster es necesario tener un sistema base donde se aloja todo los elementos software necesarios para la implementación, existen varios sistemas operativos sobre los cuales se puede hacer la implementación del cluster, como es el caso de GNU/Linux, Solaris, MacOSX, etc.

7.4.2.3. SISTEMAS DE ARCHIVOS DISTRIBUIDOS

Con el sistema de archivos distribuido (DFS, Distributed File System), los administradores de sistemas pueden facilitar a los usuarios el acceso y la administración de archivos que están físicamente distribuidos en una red. DFS le permite presentar a los usuarios archivos distribuidos por múltiples servidores de modo que parezca que residen en un solo sitio de la red. Los usuarios ya no tendrán que saber y especificar la ubicación física real de los archivos para tener acceso a éstos.

Un sistema de archivos distribuidos, es una implementación distribuida del clásico modelo de tiempo compartido de un sistema de archivos, donde varios usuarios comparten archivos y almacenan recursos. El propósito de los sistemas de archivos distribuidos es soportar la misma clase de comportamiento cuando los archivos están dispersos físicamente entre los diversos sitios de un sistema

distribuido.

7.4.2.4. CAPA MIDDLEWARE

El Middleware es el software que organiza e integra los servicios en un cluster, consiste en un conjunto de servicios software que implementan el acceso uniforme a los datos y recursos, la autenticación y autorización, el manejo de la información de los recursos y la planificación de la asignación de los recursos.

El objetivo del Middleware es presentar una completa transparencia al usuario de forma que éste no tenga que preocuparse por detalles de bajo nivel siendo una interfaz entre las tareas/procesos y el hardware o sistema operativo.

El middleware utilizado por las aplicaciones distribuidas es MPI ("Message Passing Interface", Interfaz de Paso de Mensajes), es un estándar que define la sintaxis y la semántica de las funciones contenidas en una librería de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores. Su principal característica es que no precisa de memoria compartida, por lo que es muy importante en la programación para sistemas distribuidos.

7.4.2.5. APLICACIONES SECUENCIALES

Sobre un cluster computacional además de ejecutarse aplicaciones paralelas y/o distribuidas existe la posibilidad de ejecución de herramientas secuenciales que con la ayuda de las herramientas de programación paralela logra realizar software de aplicación distribuida.

7.4.2.6. HERRAMIENTAS DE PROGRAMACIÓN PARALELA

Para la realización de aplicaciones paralelas existen diversas herramientas de programación, que permiten interacturar con las rutinas de las librerías MPI.

7.4.2.7. APLICACIONES PARALELAS:

Teniendo la unión de las aplicaciones secuenciales y de las herramientas de programación paralela nacen las aplicaciones que aprovecha los recursos paralelos; existen herramientas comerciales para varias aplicaciones y se desarrollan en empresas y universidades en el mundo.

7.4.3. DISEÑO DE APLICACIONES PARALELAS

Esta metodología estructura el proceso de diseño como cuatro etapas distintas: el particionamiento, la comunicación, aglomeración, y mapeo, propuestas por Ian Foster.

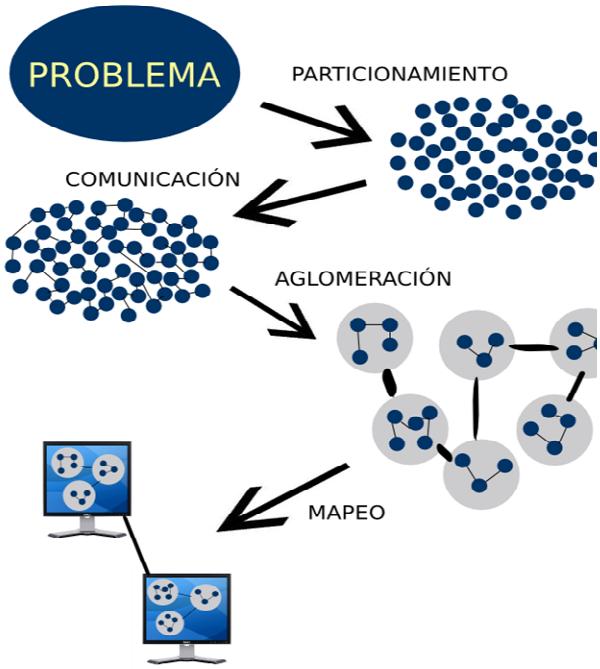


Ilustración 8: Fases de diseño de una aplicación paralela

Fuente: Autores proyecto

7.4.3.1. PARTICIONAMIENTO

En esta primera fase de diseño se define un gran número de pequeñas tareas a fin de producir lo que se denomina una descomposición de grano fino del problema, esta descomposición proporciona mayor flexibilidad en términos de potencial del algoritmo paralelo. Una buena partición divide en trozos pequeños, tanto las operaciones de cómputo asociadas al problema como los datos con los que se opera.

De este particionamiento se obtiene un número de tareas, cada una compuesta por algunos de los datos y un conjunto de operaciones sobre esos datos.

Existen dos técnicas diferentes para realizar el particionado de los datos, estas son: Descomposición de dominio la cual particiona primero los datos y posteriormente estudia la forma de asociarlos a las operaciones de cálculo, y Descomposición funcional primero realiza un división del cálculo a realizar y, a continuación, se refiere a los datos.

Una de las reglas básicas del particionado se centra en primer lugar en la estructura de datos más grande o en la que se accede con mayor frecuencia.

7.4.3.2. COMUNICACIÓN

Las tareas que se generan del particionamiento se realizan simultáneamente pero no independientemente ya que muchas de ellas requieren datos de otras tareas, por esta razón se deben crear canales de vinculación que permitan a las tareas enviar y recibir mensajes.

Esta fase de diseño se realiza con el fin de optimizar el rendimiento mediante la distribución de acciones de comunicación sobre muchas tareas y la organización de las actividades de comunicación de manera que permita la ejecución concurrente.

7.4.3.3. AGLOMERACIÓN

Después de definir la comunicación de las tareas se continua con la siguiente fase de diseño: La Aglomeración, en la cual las tareas y estructuras de comunicación definidas en las dos primeras etapas son evaluadas en lo que concierne a exigencias de funcionamiento y gastos de puesta en marcha. Si es necesario, se combinan en grandes tareas para mejorar el rendimiento o para reducir los costes de desarrollo.

Durante la fase de aglomeración se busca reducir el número de tareas aumentando la granularidad del algoritmo, de igual manera se busca reducir los costos de comunicación enviando menos volumen de datos o menos número de mensajes con igual volumen datos. Sin embargo, hay que tenerse en cuenta que la cantidad de comunicación realizada para una unidad de cómputo disminuye a medida que aumenta el tamaño de tarea.

En la aglomeración muchas veces se opta por replicar el cómputo con el fin de reducir la comunicación y el tiempo de ejecución, todo esto después de realizar un análisis al algoritmo y decidir la mejor opción de aglomeración para el problema particular, buscando mantener la flexibilidad con respecto a la granularidad y la escalabilidad del algoritmo, es decir su resistencia a los cambios de numero de procesadores.

7.4.3.4. MAPEO

El mapeo es la cuarta y última etapa del proceso de diseño de algoritmos paralelos, en el se especifica donde se ejecuta cada tarea, esta fase tiene como objetivo minimizar el timpo de ejecución, y para lograr este objetivo se utilizan dos estrategias principalmente: colocar las tareas que se pueden ejecutar simultáneamente en diferentes procesadores, a fin de aumentar la concurrencia y colocar las tareas que se comunican con frecuencia en el mismo procesador, a fin de aumentar la localidad.

Se han definido distintas técnicas de mapeo principalmente a aquellos problemas

con tareas que cambian dinámicamente, este es el caso de problemas desarrollados utilizando descomposición de dominio, para ellos se utilizan estrategias dinámicas de balanceo de carga donde un algoritmo se ejecuta periódicamente para determinar una nueva aglomeración y mapeo. Aquellos algoritmos basados en la descomposición funcional tienen tareas de cómputo que consisten en muchas tareas cortas que coordinan con otras tareas sólo al principio y al final de la ejecución. En este caso, podemos usar algoritmos que programan tareas (task-scheduling), que asignan tareas a procesadores que en determinado momento estén ociosos o se van a volver ociosos.

CAPITULO 8:

IMPLEMENTACIÓN

8. NIMPLEMENTACIÓN

8.1. DISEÑO DE LA APLICACIÓN PARALELA

Como primera medida se desarrolla el algoritmo genético secuencial, dejando a un lado los problemas de tiempo y carga de procesador, centrándonos simplemente en la funcionalidad de éste. Posteriormente, se encuentra necesario el uso de la computación de alto rendimiento para solucionar los problemas de procesamiento, recordando que para nuestro problema los datos son configuraciones de redes neuronales artificiales y el procesamiento sobre estos datos es el entrenamiento de éstas; por esta razón se decide implementar un cluster de alto rendimiento.

Se comienza realizando el diseño de la aplicación paralela definiendo el problema a paralelizar, y siguiendo las cuatro etapas de diseño de Ian Foster; posteriomente se realiza la implementación del cluster de acuerdo a las especificaciones de la aplicación diseñada.

8.1.1. DEFINICIÓN DEL PROBLEMA A PARALELIZAR

El problema que se aborda es el hallazgo de configuraciones de redes neuronales artificiales, donde se define el número de capas y las neuronas por cada capa además de las funciones de activación.

Para realizar la consecución de la configuración óptima de red se ha recurrido a los algoritmos genéticos para conseguir la mejor solución, pero con el inconveniente que los algoritmos genéticos toman demasiado tiempo y recursos de cómputo para su ejecución. Por esta razón se acude a los métodos de la computación de alto rendimiento con el fin de reducir los costos que se incurren al correr el algoritmo genético.

8.1.2. DEFINICIÓN DE LA POBLACIÓN EN EL ALGORITMO GENÉTICO

Los individuos definidos en el algoritmo genético están formados por el número de neuronas por cada capa, seguido de las funciones de activación; posteriormente se genera el genotipo, codificándo los individuos mediante una cadena de dígitos binarios.

Los individuos, al ser configuraciones de redes neuronales artificiales (RNA) son evaluados por medio del entrenamiento de dicha red, la aptitud de los individuos es medida según el error arrojado al simular la red que los individuos representan después de ser entrenada; por lo tanto se tiene una relación inversamente proporcional entre el error obtenido y su valor de aptitud o fitness3. Los individuos con mayor fitness tienen mayor probabilidad de reproducirse, sin descartar la posibilidad de que un individuo con bajo fitness lo pueda hacer, esto para agregar aleatoriedad al algoritmo y darle al nuevo individuo genes que podrían hacerlo aproximarse a la mejor RNA 4.

Es precisamente en el momento de la evaluación donde se genera un cuello de botella ya que en este momento se requiere mayor tiempo de procesamiento mientras se realiza el entrenamiento de la RNA, factor por el cual se decide distribuir el algoritmo genético.

En busca de que las redes neuronales artificiales con diferentes números de capas tengan la misma posibilidad de ser solución, se subdividió la población según su número de capas ocultas, encontrando entonces subpoblaciones de individuos en cada procesador del sistema distribuido; de esta forma al aplicar los operadores genéticos a individuos con diferente número capas no predomina ningún tipo de individuos.

Por esta razón en el algoritmo genético se han definido “caminos”, es decir, corrientes diferentes por donde el algoritmo se ejecuta independientemente,

---

[3] Fitness – Valor ponderado que se asigna a los individuos según su proximidad al valor óptimo.

[4] RNA- Red Neuronal Artificial

además como se mencionó anteriormente la población de cada camino esta dividida por número de capas.

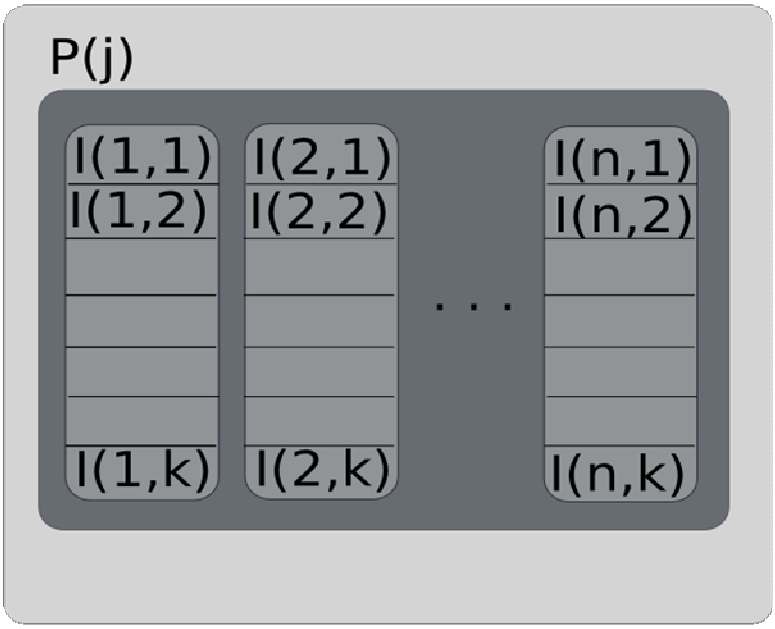


Ilustración 9: Estructura de la población de cada nodo del sistema dividida en subpoblaciones según número de capas ocultas

Fuente: Autores proyecto

Se define **P** el camino que sobre el sistema distribuido se traduce en un nodo del cluster, el número de caminos (**j**) depende de los parámetros iniciales suministrados por el usuario. Dentro de los caminos se ha subdivido la población definiendo en cada uno de éstos subpoblaciones de individuos con diferente número de capas. Se define un individuo como **I(a,b)**, donde **a** corresponde al número de capas ocultas hasta un tamaño máximo **n** ,y **b** el número del individuo en el subconjunto siendo **k** el último individuo. **n** y **k** son asignados previamente por el usuario.

Sobre el sistema distribuido un camino no es más que un nodo donde corre el algoritmo con parámetros y poblaciones diferentes que evolucionan independientemente aplicando cada una por separado sus operadores genéticos.

Cabe destacar que se resolvió estructurar de esta forma la población de individuos después de aplicar una a una las distintas fases de diseño de un algoritmo paralelo propuestas por Ian Foster[11]. En la siguiente sección se explica cada fase y su respectiva implementación en el algoritmo genético.

8.1.3. FASES DE DISEÑO

Ian Foster en su libro “Designing and Building Parallel Programs” define las etapas de particionamiento, comunicación, aglomeración y mapeo para realizar el diseño de una aplicación paralela, escalable, que optimice la frecuencia y el volumen de información comunicada.

8.1.3.1. PARTICIONAMIENTO

Una de las reglas básicas del particionado se centra en primer lugar en la estructura de datos más grande o en la que se accede con mayor frecuencia, sin lugar a dudas nos estamos refiriendo a la población total del algoritmo genético. Basándose en la técnica de descomposición de domino, se realiza la descomposición de los datos al problema analizado los tipos de datos existentes, éstos son los parámetros y la población de individuos.

Se resolvió particionar la población ya que como se dijo anteriormente, estos datos son los mas accedidos en la ejecución del algoritmo y no los parámetros que son usados solo al iniciar la ejecución.

Las particiones resultantes son los individuos diferenciados según su número de capas ocultas, de esta forma se tienen subconjuntos que tienen individuos con características similares a las cuales se pueden aplicar cada una de las operaciones de cómputo.

Al particionar las operaciones de cómputo del algoritmo se llega a descomponer los operadores genéticos y diferenciarlos en 4 grupos: selección, cruce, mutación y migración.

De ahí se obtienen las tareas que son resultado de la asociación de las particiones de los datos con las operaciones también particionadas. Las tareas corresponden a la ejecución de los operadores genéticos en cada uno de los subconjuntos de datos.

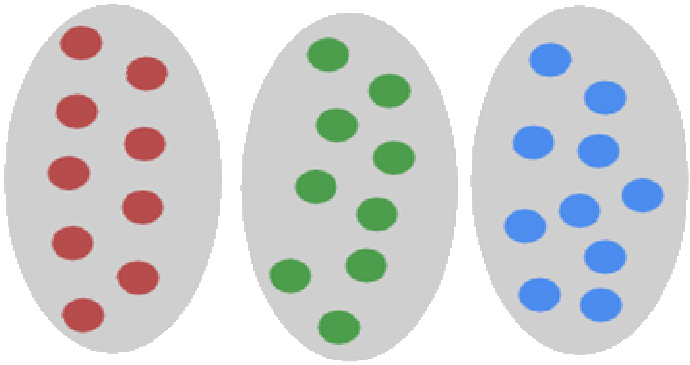


Ilustración 10: Particionado de datos diferenciados por número de capas ocultas

Fuente: Autores proyecto

En la ilustración 10 se observan los subconjuntos de datos particionados, el primer subconjunto contiene individuos de una capa oculta, el siguiente subconjunto tiene los individuos de dos capas ocultas, así, hasta el máximo de capas ocultas definido.

8.1.3.2. COMUNICACIÓN

Debido a la decisión de implementar un algoritmo genético paralelo de grano grueso con topología de islas, se incluye el operador migración del cual surge la necesidad de comunicar las diferentes particiones de datos para realizar el intercambio de individuos de las subpoblaciones. Este requerimiento de comunicación entre las particiones de datos, se efectúa después de ejecutar el algoritmo un número definido de generaciones, por esta razón se decide optar por

la comunicación global en la cual participan muchas tareas y se define una estructura de comunicación en la que cada tarea comunica su individuo emigrante a un administrador de tareas quien es el encargado de ejecutar el operador "migración" en el algoritmo.

Este tipo de comunicación centralizada no impide la computación concurrente ya que las tareas que ejecutan los demás operadores genéticos pueden realizarse de forma simultanea sin depender del administrador de tareas quien solo interviene para ejecutar el operador migración, de esta manera se obtiene una estructura de comunicación con el menor número de canales de comunicación y con tareas que se ejecutan independientemente sin afectar su desempeño.

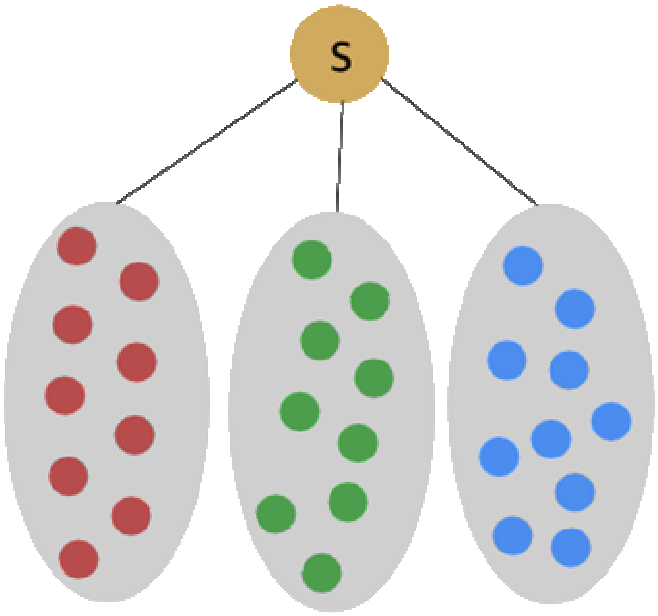


Ilustración 11: Comunicación Centralizada

Fuente: Autores proyecto

En la ilustración 11 se identifica un administrador de tareas *S,* al cual se envían los individuos emigrantes desde cada partición de datos, mediante una comunicación centralizada.

8.1.3.3. AGLOMERACIÓN

Después de estudiar la estructura del algoritmo genético y analizar sus requerimientos de comunicación y cómputo, se encuentra que según el particionado de los datos, cada subdivisión tiene datos definidos de diferente forma, y al aplicar las operaciones de cómputo asociadas a los operadores genéticos, debido a la distinta estructura de los datos particionados, requieren mayor tiempo de ejecución ciertas subdivisiones que otras.
En otras palabras, tardaría menos tiempo de ejecución una partición de individuos con una sola capa oculta que otra partición con individuos de tres capas ocultas, ya que el entrenamiento de las redes que estos individuos representan tarda mas tiempo que los individuos mencionados anteriormente.
Por otro lado, con este particionamiento no se podría realizar la migración entre particiones ya que el individuo inmigrante tendría diferente estructura.

Por esta razón, se decide aglomerar las particiones anteriormente realizadas, agrupando una sección de cada subdivisión y formando una división nueva conformada por todos los tipos de particiones; se tiene entonces varios conjuntos de datos subdivididos cada uno por subconjuntos de individuos según su numero de capas ocultas.

Realizando de esta forma la aglomeración se evita el tiempo ocioso que tendrían las particiones de datos que terminan su ejecución anticipadamente al esperar que las demás finalicen, ya que a cada conjunto de datos pertenecen individuos con diferente número de capas ocultas, consiguiendo de esta forma un tiempo de procesamiento regular entre conjuntos. Además se permite la migración de individuos de igual estructura a otras subdivisiones de los diferentes conjuntos de datos.

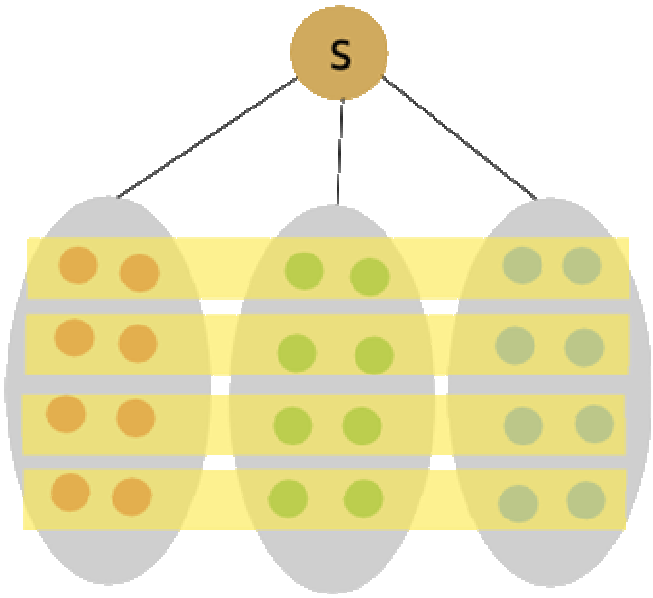


Ilustración 12: Aglomeración de individuos

Fuente: Autores proyecto

La ilustración 12 muestra la agrupación de dos individuos del primer conjunto de datos, de otros dos individuos del segundo conjunto de datos, y otros dos individuos del tercer conjunto de datos, formando una población con individuos con diferente número de capas ocultas.

8.1.3.4. MAPEO

El mapeo en el algoritmo genético paralelo se implementa de forma sencilla al encontrase ya con un diseño estructurado formado a partir de las fases de diseño. Teniendo los conjuntos de datos definidos con subpoblaciones pertenecientes a ellos, se asigna cada conjunto de datos a un procesador o nodo del sistema distribuido. De esta manera a cada nodo se asigna una población conformada por subpoblaciones de individuos con diferente numero de capas ocultas, a cada una de estas subpoblaciones se aplican todos los operadores genéticos incluyendo la migración la cual se realiza entre subpoblaciones de igual tipo de individuos

emigrando de un nodo a otro; la ejecución de los operadores genéticos en cada nodo implica una replicación de cálculo aumentando la localidad en cada nodo y a su vez se aumenta la concurrencia ya que en cada nodo evoluciona su población independientemente ratificando también la comunicación global entre procesadores.

Por otro lado, el diseño del algoritmo permite su escalabilidad ya que al incluir nuevos procesadores al sistema, simplemente se agregaría una nueva población que aumentaría su aleatoriedad.

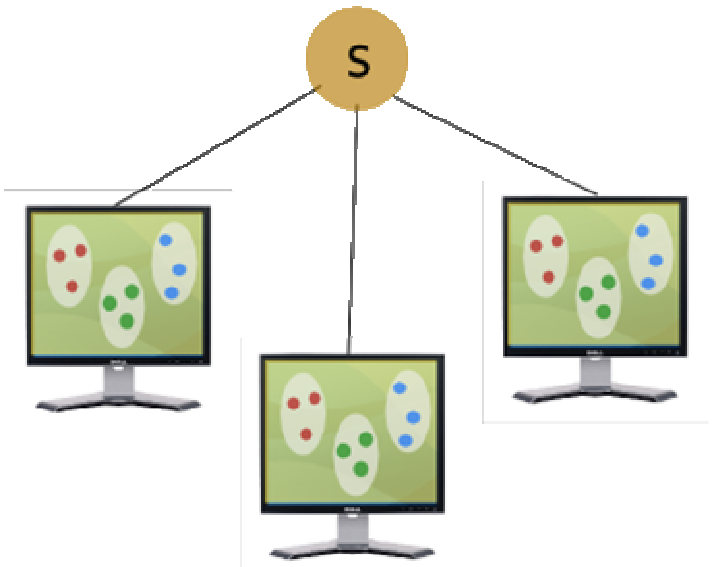


Ilustración 13: Mapeo: Asignación de tareas a los procesadores

Fuente: Autores proyecto

La ilustración 13 muestra la asignación en cada procesador de una subpoblación que contiene parte parte de individuos del primer subconjunto de datos (color rojo), del segundo subconjunto (color verde) y del tercer subconjunto de datos (color azul).

8.2. ARQUITECTURA DEL SISTEMA DISTRIBUIDO

Se dispuso de 18 computadores en las instalaciones del CENTIC[5] en la sala 1-6, donde se ha montado el cluster de alto rendimiento. En adelante se expone la implementación del cluster recorriendo de forma ascendente la ilustración 14.

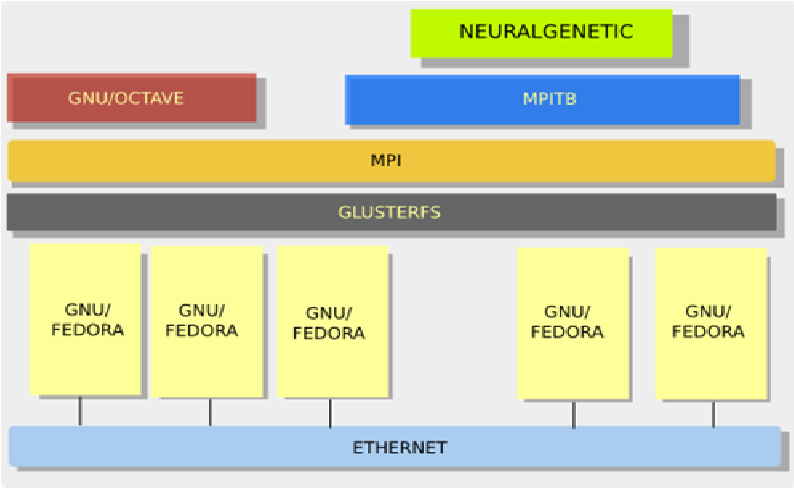


Ilustración 14: Arquitectura del cluster implementado

Fuente: Autores proyecto

8.2.1. CAPA DE RED

Se cuenta con una subred Ethernet ya configurada en la sala, con IP estáticas asignadas previamente.

8.2.2. SISTEMA OPERATIVO

Después de un periodo de vigilancia tecnológica hacia las posibles herramientas y sistemas operativos con los que podríamos contar para la realización del proyecto, se determinó como mejor solución la utilización de Software Libre[12], ya que si

5 Centro de Tecnologías de Información y Comunicación de la Universidad Industrial de Santander

se trabajara con herramientas de software propietario conllevaría a un gasto de dinero debido a la compra de licencias para cada nodo del cluster, restringiendo su escalabilidad, debido a las limitadas licencias de los nodos.

Se decidió entonces utilizar el sistema operativo GNU/Linux por la facilidad que le ofrece al usuario al tener acceso a todos los archivos de configuración. Luego de hacer pruebas con otras distribuciones de GNU/Linux como es el caso de GNU/Debian, se optó trabajar sobre la distribución GNU/Fedora 8, ya que las herramientas de programación a utilizar, son nativas de esta distribución. La instalación del sistema operativo es básica, solo se instalan los paquetes del sistema base, desprendiendo la interfaz gráfica del sistema operativo, ya que estos equipos están totalmente disponibles para ejecución de los programas paralelos y/o distribuidos, asegurando así que no hayan procesos que puedan estropear el funcionamiento del cluster o la ejecución del algoritmo.

8.2.3. SISTEMA DE ARCHIVOS DISTRIBUIDOS

Debido a la necesidad de paso de datos entre los nodos para realizar el operador migración, específicamente el paso de estructuras de datos como lo son las redes neuronales, se hace necesario el uso de un sistema de archivos ya que el middleware MPI[6] solo permite el paso de variables, vectores y matrices, excluyendo algunos tipos de datos como las estructuras.

Para la implementación en el cluster se decidió usar GlusterFS que es un sistema de archivos distribuido a prueba de fallas con menor latencia que otros sistemas de archivos como NFS[7], es escalable a varios petabytes, e implementa el modelo cliente-servidor.

GlusterFS es uno de los pocos proyectos con soporte para conexiones TCP/IP e Infiniband. Los nodos de almacenamiento pueden ser de cualquier hardware tal

---

[6] MPI: Interfaz de paso de mensajes

[7] NFS: Network file system

como servidores x86-64 con SATA II RAID e Infiniband.

Para montar el sistema de archivos GlusterFS, los nodos cliente necesitan el soporte de FUSE[8] en el kernel. Hasta la fecha los servidores GlusterFS han sido probados en sistemas operativos como GNU/Linux, FreeBSD y OpenSolaris y los clientes solo corren en maquinas GNU/Linux.

8.2.4. MIDDLEWARE MPI

Se decidió implementar OpenMPI[9], la cual es una implementación de código abierto de los estándares MPI-1 y MPI-2. OpenMPI es un proyecto que combina las tecnologías y recursos de otros proyectos (FT-MPI, LA-MPI, LAM/MPI, y PACX-MPI) para construir la mejor librería MPI.

Se decidió usar OpenMPI después de analizar los beneficios de esta implementación tales como: distribución de procesos de forma dinámica, alto rendimiento en todas las plataformas, administración de trabajos rápida y fiable, tolerancia a fallos de red y procesos, además es la única librería soporta todas las redes.

8.2.5. APLICACIONES SECUENCIALES: OCTAVE

Para la implementación de las rutinas propias del algoritmo genético así como la definición, entrenamiento y simulación de las redes neuronales, se usó el software libre GNU/OCTAVE, el cual maneja un lenguaje de alto nivel inspirado en el software comercial MATLAB (Laboratorio de Matrices, realizado inicialmente para cálculos numéricos de vectores y matrices) producto propietario de The Mathworks.

De igual manera Octave fue creado en 1988 para dar soporte a estudiantes de ingeniería química de las Universidades de Wisconsin-Madison y Texas en un curso de diseño de reactores químicos.

---

[8] FUSE: Sistema de archivos en espacio de usuario. http://fuse.sourceforge.net/

[9] OpenMPI: http://www.open-mpi.org/

Posteriormente se decide extender este proyecto y continuar con su desarrollo bajo la dirección de John W. Eaton[10], además de otras contribuciones realizadas por usuarios que han contribuido con la evolución de este software añadiendo librerías y otras funcionalidades.

Hoy en día octave no se limita a un simple trabajo con vectores y matrices, ahora resuelve aplicaciones puramente matemáticas o numéricas para diferentes campos de ciencias y de ingeniería, entre ellos problemas de álgebra numérica, encuentra soluciones de ecuaciones no lineales, realiza integrales de funciones ordinarias, manipula polinomios, e integra ecuaciones diferenciales ordinarias y ecuaciones diferenciales algebraicas; además del procesamiento de señales e imágenes, y otras áreas como la inteligencia artificial al poseer librerías para redes neuronales, sistemas de control y dibujo vectorial.

Las librerías son programadas bajo el lenguaje interpretado propio de octave, o usando cualquier lenguaje que sea soportado por el compilador gcc como C, C++, pascal y fortran. De esta manera Octave puede ser extendido con funciones y procedimientos por medio de módulos dinámicos.

GNU/Octave posee una interfaz en forma de shell, con una línea de comandos similar a la de Matlab, para su edición se usa la librería GNU[11] Readline, así como también el Bash[12] entre otros programas GNU.

Cabe aclarar que aunque GNU/Octave corre en plataformas Unix[13] es posible su ejecución en Windows. Así mismo, se facilita la transición del software Matlab a GNU/Octave ya que éste crea funciones con extensión *.m,* que son las utilizadas por Matlab.

---

10 John W. Eaton: http://jbrwww.che.wisc.edu/people/eaton.html

11 GNU: Proyecto con el objetivo de crear un sistema operativo completamente libre, compatible con Unix.

12 Bash:Es un shell de Unix (intérprete de órdenes de Unix) escrito para el proyecto GNU

13 Unix: Es un sistema operativo portable, multitarea y multiusuario;

8.2.5.1. TOOLBOX DE GNU/OCTAVE UTILIZADOS EN LA IMPLEMENTACIÓN

Como se mencionó anteriormente, las capacidades de GNU/Octave pueden ser extendidas mediante librerías de funciones; un toolbox de GNU/Octave es una agrupación de dichas librerías desarrolladas para diferentes aplicaciones específicas.

- **PAQUETE DE REDES NEURONALES NNET**

Contiene funciones que definen, entrenan y simulan las redes neuronales, funciona con versiones de octave mayores a 2.9.9. Este paquete debe ser instalado independiente de GNU/Octave y puede ser descargado de Octave Forge[14]. La versión utilizada es la 0.1.8.

8.2.6. HERRAMIENTAS PARA PROGRAMACION PARALELA

Para ejecutar las funciones de Octave en ambientes paralelos es necesario utilizar el toolbox MPITB (MPI toolbox) de Octave, mediante el cual se llama las rutinas de las librerías MPI desde el entorno de GNU/Octave que se ejecuta sobre el cluster.

8.2.7. APLICACIONES PARALELAS

Después de implementar en el cluster, OpenMPI, Fuse y GlusterFS, podemos decir que se tiene la base para desarrollar aplicaciones distribuidas; sobre el cluster se trabaja con GNU/Octave para la construcción del algoritmo, y en el último nivel se encuentra la aplicación resultante NeuralGenetic.
En el siguiente capitulo se expone a fondo la estructura y el funcionamiento la aplicación desarrollada.

---

[14] Octave Forge: http://octave.sourceforge.net/nnet/index.html

CAPITULO 9:

NEURALGENETIC

9. NEURALGENETIC

El algoritmo genético paralelo está regido por un nodo maestro que ejecuta el programa principal, y ***n*** nodos quienes son los encargados de realizar las generaciones definidas por el usuario en cada uno de ellos (ilustración 15); de esta manera en cada nodo evolucionará una población diferente y se aplicarán los operadores genéticos de selección, cruce y mutación independientemente de los demás.

Ilustración 15: Estructura del cluster

Fuente: Autores proyecto

9.1. CODIFICACIÓN DEL INDIVIDUO (FENOTIPO GENOTIPO)

Partiendo de un modelo biológico, los seres vivos en el núcleo de las células albergamos material genético compuesto por cromosomas con los cuales conformamos nuestro ADN, se define entonces el genotipo de un individuo como la constitución genética de sus cromosomas, y el fenotipo son las características físicas que representan su genotipo.

En una población natural el fenotipo es el conjunto de individuos que pertenecen a ella con las características físicas de dicha población; haciendo una analogía con los algoritmos genéticos el fenotipo de una población es el conjunto de soluciones de un problema y su genotipo es la representación de estas soluciones codificando su información en cadenas de dígitos binarios

En nuestro problema particular del entrenamiento de redes neuronales el fenotipo de los individuos representa los parámetros necesarios para crear una red neuronal.

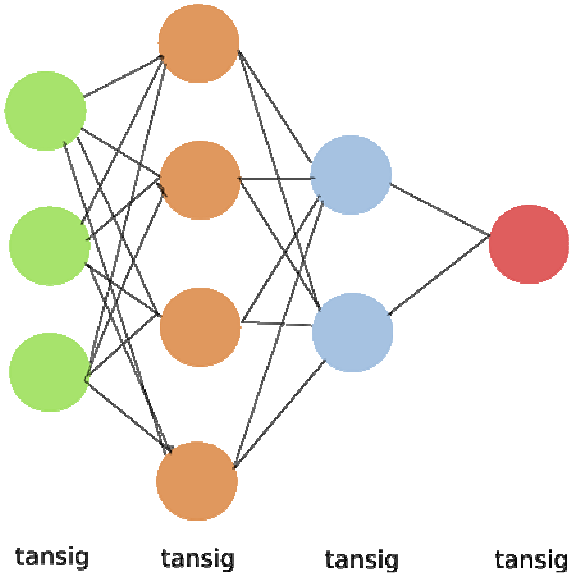


Ilustración 16: Fenotipo de la red neuronal

Fuente: Autores proyecto

La ilustración 16 muestra la red neuronal (fenotipo) que representa el individuo, se diferencia cada capa de la red mediante distintos colores.

Ilustración 17: Representación del fenotipo

Fuente: Autores proyecto

El fenotipo es representado como muestra la ilustración 17; a partir de estar representación se genera el genotipo (ilustración 18) codificando los individuos mediante una cadena de dígitos binarios que representan el número de capas ocultas, número de neuronas en cada capa y cada una de sus funciones de activación; se llamó al conjunto de estos parámetros “configuración de la red”.

0011|0100|0010|0001|tansig|tansig|tansig|tansig

Ilustración 18: Genotipo de la red neuronal

Fuente: Autores proyecto

Independientemente del número de capas que tengan los individuos, estos tendrán una longitud estándar de bits, de esta manera se completa con ceros las capas no existentes en cada caso.

1100|1000|0110|tansig|tansig|tansig
1100|1000|0000|tansig|tansig|0000
1100|0000|0000|tansig|0000|0000

Ilustración 19: Longitud de los individuos

Fuente: Autores proyecto

9.2. ESTRUCTURA DE LA POBLACIÓN

Las poblaciones se desarrollan independientemente en cada nodo, su estructura ha sido descrita en la sección 8.1.2.

9.3. ESTRUCTURA DEL ALGORITMO GENÉTICO

Al iniciar la ejecución del algoritmo genético, se inicia un ciclo iterativo de

generaciones, en cada una de ellas se ejecutan los operadores de selección, cruce y mutación.

```
INICIO %Algoritmo genetico
   Generar una poblacion inicial
WHILE(numero de migraciones no cumplido)
       PARA 1:Numero de generaciones por nodo
         %inicia ciclo reproductivo
           Seleccionar dos individuos de la anterior generación, para el cruce.
           Cruzar con cierta probabilidad los individuos obteniendo dos descendientes.
           Mutar los dos descendientes con cierta probabilidad.
           Compular la función de evaluación de los descendientes mutados.
           Insertar los dos descendientes mutados en la nueva generación.
        FIN
      HACER MIGRACIÓN

END
```

Ilustración 20: Pseudocódigo del algoritmo genético

Fuente: Autores proyecto

El criterio de parada del algoritmo genético es el número de generaciones definidas por el usuario antes de comenzar su ejecución.

9.4. INICIALIZACIÓN DE LOS NODOS

El nodo maestro ejecuta la función “inicializar_nodo” en la que se da la orden a los nodos esclavos de crear sus poblaciones iniciales, así como el porcentaje de cruce, porcentaje de mutación para cada uno, así mismo, los individuos son evaluados por primera vez.

El porcentaje de mutación fue definido de tal manera que aumenta en cada nodo, así, en el último nodo se espera que ocurran mas mutaciones que el primer nodo esclavo, esto para introducir mayor aleatoriedad al algoritmo.

La población creada en cada nodo diferencia una sección del individuo que corresponde a las neuronas y otra que corresponde a las funciones de activación

de cada capa, estas dos secciones son creadas separadamente y son producto de la aleatoriedad de los dígitos binarios.

9.5. FUNCIÓN DE EVALUACIÓN

El objetivo de la función de evaluación es asociar al individuo un valor de calidad el cual representa la capacidad de éste para adaptarse al medio donde vive, y es calculado por la función objetivo.

La función objetivo varia según la naturaleza del problema, para nuestro caso en particular la aptitud de los individuos es medida según el error arrojado al simular la red que estos representan después de ser entrenada; por lo tanto se tiene una relación inversamente proporcional entre el error obtenido y su valor de aptitud o fitness.

Para calcular la función de evaluación se procede a definir la red, entrenarla y posteriormente probarla con datos diferentes a los utilizados durante su entrenamiento, de esta manera se obtiene un error calculado teniendo en cuenta la tolerancia de error que es un parámetro definido por el usuario; las redes que presenten errores menores a la tolerancia tendrán mayor valor fitness, además se tiene en cuenta el error cuadrático medio, el cual es inversamente proporcional al valor fitness de la red.

El cálculo del valor fitness se realiza de acuerdo a la formula (1):

$$f = \frac{X \square c_1}{v \square c_2} \quad (1)$$

donde c_1 y c_2 son constantes diferentes, cercanas a cero.

X se calcula mediante la fórmula (1.1)

v se calcula mediante la fórmula (1.4)

$$X = \sum_{i=1}^{n} a_i \quad (1.1)$$

$$a_i = 1 \; si \; e_i \leq c$$
$$0 \; si \; e_i \leq c \quad (1.2)$$

Donde *c* es la constante que representa la tolerancia de error definido por el usuario y e_i se define en la fórmula (1.3)

$$e_i = |y_i - x_i| \quad (1.3)$$

Donde y_i es el valor experimental de cada una de las *n* muestras con que se prueba la red neuronal y x_i es el valor obtenido por la red neuronal para las mismas muestras.

$$v = \sum_{i=1}^{n} |y_i - x_i|^2 \quad (1.4)$$

Para efectuar la evaluación de los individuos en el algoritmo, se ejecuta la función “EvaluaIndividuos” en la cual se convierte la cadena de bits del individuo a su fenotipo, es decir se obtiene la configuración de red que representa dicho individuo.

Teniendo ya el fenotipo del individuo se procede a definir la red que se entrenará con dicha configuración; para esto se utiliza la función “newff” implícita en el paquete nnet de octave.

Se procede entonces a realizar el entrenamiento de la red neuronal definida anteriormente, este entrenamiento se realiza tantas veces como haya definido el usuario, ya que los resultados arrojados por las redes neuronales dependen en gran proporción por la asignación aleatoria de los pesos y estos son los que determinan la intensidad de las conexiones entre las neuronas y por lo tanto la calidad del entrenamiento de la red.

Cabe aclarar que cada vez que es entrenada la red con los mismos parámetros,

se obtiene diferentes pesos, por esta razón, después de realizar el número de entrenamientos especificado se escoge la red que gracias a sus conexiones sinápticas obtuvo menor error.

9.6. DATOS DE ENTRENAMIENTO DE LA RED

Para realizar las predicciones de los datos se utilizó la técnica de validación cruzada expuesta anteriormente, la cual define subconjuntos de datos para realizar el entrenamiento y validación de la red.

El entrenamiento de las redes es realizado con datos llamados “Datos de entrenamiento”, con los cuales la red define los pesos, y comienza su entrenamiento usando el algoritmo BackPropagation, el cual minimiza el error cuadrático medio de la red por medio de gradiente descendiente. Por ser éste un algoritmo de aprendizaje supervisado, es necesario conocer los valores de salida que se asocian a los valores de entrada, para garantizar el aprendizaje de estos valores y una posterior predicción de ciertos valores deseados.

Si se decide hacer uso de detención temprana es necesario hacer uso de otro subgrupo de datos o “Datos de verificación”.

Finalmente, se tiene otro subgrupo de datos llamado “Datos de pruebas”, el cual usaremos para hacer el cálculo del error, de la red; es necesario entonces proporcionar las entradas con sus respectivas salidas.

9.7. SELECCIÓN

El operador de selección juega un papel muy importante dentro del algoritmo genético ya que éste guía la búsqueda y provoca la convergencia de la población. Por lo tanto, en él se decide los individuos que pasaran a la siguiente generación, siendo seleccionados para pasar automáticamente, o siendo producto de la reproducción o mutación de unos padres seleccionados.

La selección se realiza teniendo en cuenta que igual que en la naturaleza, los

individuos que mejor se adaptan a su entorno, son los que mas sobreviven y los que mas se reproducen, por esta razón el objetivo de este operador es que aquellos individuos con un mejor fitness tienen una mayor probabilidad de ser seleccionados.

Existen varios mecanismos de selección interesantes como el elitismo donde normalmente se escogen a los mejores individuos de la población para pasar a la siguiente generación garantizando de esta forma la perpetuación de los mejores individuos, otra de ellas es la ruleta donde se hace la elección de los individuos de manera aleatoria asignando una probabilidad de selección proporcional a su valor de aptitud de esta forma se garantiza que los mejores individuos tienen mayor probabilidad de ser seleccionados; además de otras variantes existentes de estos modelos que modifican de alguna forma la probabilidad de selección.

Conociendo entonces, la aptitud o el fitness de cada individuo, se procede a realizar la selección de los individuos que se cruzarán utilizando el método de ruleta, asignando como se dijo anteriormente mayor rango en la ruleta a el individuo con mejor fitness. Los individuos seleccionados son utilizados para escoger los progenitores en el cruce, así como para escoger el individuo que será mutado, esto para las dos secciones que se diferenciaron del individuo (neuronas y funciones de activación)

Existen distintos enfoques sobre la política de reemplazo que se utiliza después de realizar la selección de los individuos:
Reemplazo Generacional o Plan Reproductivo en el que los hijos reemplazan a los padres en al nacer, también existe otra versión de este tipo de reemplazo en la que se sustituyen solo los peores individuos. En el caso del algoritmo genético se implementó el enfoque de Plan Reproductivo reemplazando los padres por los nuevos individuos.

9.8. CRUCE

En este operador se realiza el proceso de herencia de genes de dos progenitores a su hijo, es decir se simula la reproducción de dos individuos de la población, obteniendo como resultado un nuevo individuo intermedio entre sus dos padres.

Al ejecutarse este operador realiza una búsqueda de nuevos puntos en el espacio de soluciones ya que al combinar el material genético de los dos padres se producen nuevos individuos que contribuyen en la búsqueda de una solución óptima del problema.

Para escoger los individuos a cruzar entre los seleccionados anteriormente mediante el método de ruleta, se elige una pareja de individuos (Ilustración 21).

El cruce es realizado basándose en dos puntos obtenidos aleatoriamente, donde los dos progenitores son recombinados por medio de estos dos puntos; se escogió el cruce en dos puntos para mayor facilidad en la exploración del espacio de búsqueda (Ilustración 22 y 23).

Al aplicar este operador a los individuos seleccionados, estos son reemplazados por los nuevos hijos, manteniendo de esta forma una población de igual tamaño a la inicial (ilustración 24). Posteriormente, se tiene en cuenta los nuevos individuos ingresados a la población para su posterior evaluación en la siguiente generación.

1010|0111|1010|0010|0101|0111
1011|1001|0001|0110|0111|0001

Ilustración 21: Individuos progenitores

Fuente: Autores proyecto

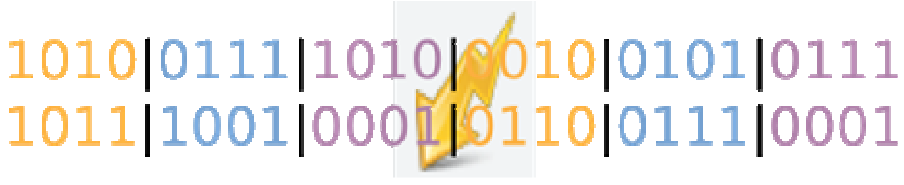


Ilustración 22: Punto de cruce aleatorio

Fuente: Autores proyecto

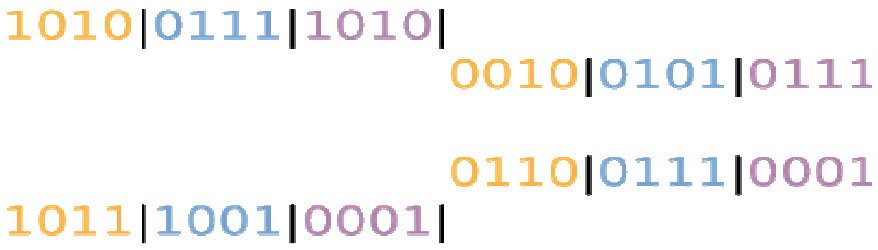


Ilustración 23: Fragmento a intercambiar

Fuente: Autores proyecto

Ilustración 24: Nuevos individuos

Fuente: Autores proyecto

9.9. MUTACIÓN

Este operador consiste en generar una alteración de un gen de ADN del individuo para producir diversidad en la población, de esta forma se introduce nueva información permitiendo así escapar de óptimos locales.

El operador mutación fue implementado escogiendo aleatoriamente el número de bits a mutar en los individuos seleccionados anteriormente; igualmente se hallan las posiciones de cada uno de los bits a mutar y finalmente se cambia el bit si es 0 lo cambia por 1, si es 1 es cambiado por un 0.

Al igual que en el cruce, se guardan las posiciones de los nuevos individuos para ser evaluados posteriormente.

9.10. ENVÍO DE INFORMACIÓN AL MAESTRO

Después de realizar todos estos operadores en cada una de las subpoblaciones

de los nodos esclavos a lo largo de las generaciones definidas inicialmente, se debe realizar el operador migración. Para esto cada subpoblación debe enviar la información al maestro haciendo uso del sistema de archivos GlusterFS en el cual una serie de equipos (clientes) haciendo uso del servidor glusterfs, acceden y envian datos a este sistema de archivos. Cuando se habla de la información enviada, se hace referencia a los individuos (genotipo) de cada población, la red neuronal entrenada correspondiente a cada individuo y su valor fitness.

Ilustración 25: Envío de datos al maestro, los datos son almacenados en el sistema de archivos distribuido GlusterFS del cual se puede leer y escribir desde cualquier nodo.

Fuente: Autores proyecto

9.11. MIGRACIÓN

Como se expuso anteriormente, cada procesador trabaja independientemente en una población aislada de individuos, el operador migración se incluye para intercambiar material genético entre poblaciones compartiendo los mejores individuos con las otras poblaciones desarrolladas en los demás procesadores, cada cierto número de generaciones se realiza el intercambio de información entre poblaciones.

Teniendo presente el tipo de individuos de las subpoblaciones existentes en cada procesador se realiza la migración de individuos con una (1) capa oculta a las demás subpoblaciones de los demás procesadores que sean de una (1) capa oculta, posteriormente se realiza el intercambio de individuos de dos (2) capas ocultas, así sucesivamente hasta terminar la migración de todas las subpoblaciones en todos los procesadores.

La elección del emigrante se realiza hallando de cada subpoblación el mejor individuo según su fitness, éste entra a todas las subpoblaciones de su tipo, es decir de su mismo número de capas ocultas reemplanzando al peor individuo de cada una.

El encargado de realizar este operador es el nodo maestro, el cual toma del sistema de archivos los datos de cada procesador y los almacena en una gran estructura donde contiene la información de las poblaciones, después de redefinir las poblaciones con los individuos emigrantes, envia nuevamente los datos al sistema de archivos, donde cada nodo esclavo lee nuevamente sus datos.

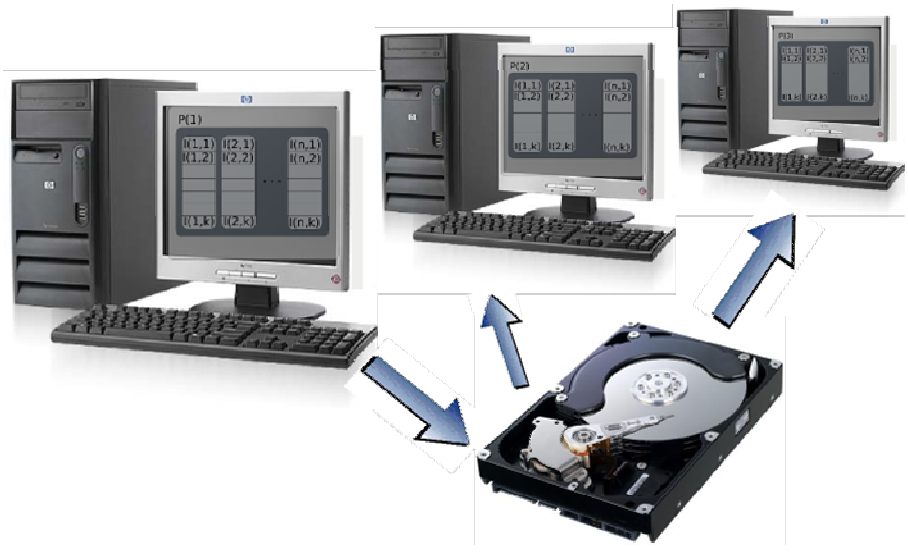


Ilustración 26: Migración de individuos

Fuente: Autores proyecto

En la ilustración 26 se presenta la redefinición de las poblaciones que realiza el maestro con los individuos emigrantes y el posterior envío al sistema de archivos, desde el cual leen los nodos esclavos sus nuevas poblaciones.

CAPITULO 10:

INTERFAZ GRÁFICA

10. INTERFAZ GRÁFICA DE LA APLICACIÓN

Para la ejecución del algoritmo genético el usuario final debe enfrentarse a la edición de comandos desde la consola del sistema operativo GNU/Linux distribución Fedora core 8. Para facilitar la interacción del usuario con la máquina se ha desarrollado una interfaz gráfica, la cual permite acceder al cluster computacional de forma remota; así un usuario que no se encuentre en el espacio físico del cluster puede iniciar la ejecución del algoritmo y colectar los resultados.

10.1. INTERFAZ WEB

Con la creación de la Interfaz Web para el sistema se logra separar al usuario del cluster donde se encuentra implementado el algoritmo genético, facilitando a su vez el manejo del mismo, siendo que este sistema puede ser utilizado por personas de diferentes disciplinas ajenas a la computación.

Con esta interfaz el usuario logra el envío de trabajo, recuperación resultados y monitoreo del cluster sin importar su ubicación geográfica, solo contando con conexión a internet, gracias a que ésta se encuentra disponible en un dominio público (http://200.21.228.175/neuralweb/).

Gracias a esta interfaz se hace transparente para el usuario el sistema de alto rendimiento al cual está accediendo, ya que él solo ingresa a un entorno web, desconociendo que el servidor donde se encuentra alojado dicho entorno se encarga de la comunicación y envío de los parámetros al nodo maestro, manejador del cluster sobre donde se ejecuta el algoritmo genético distribuido.

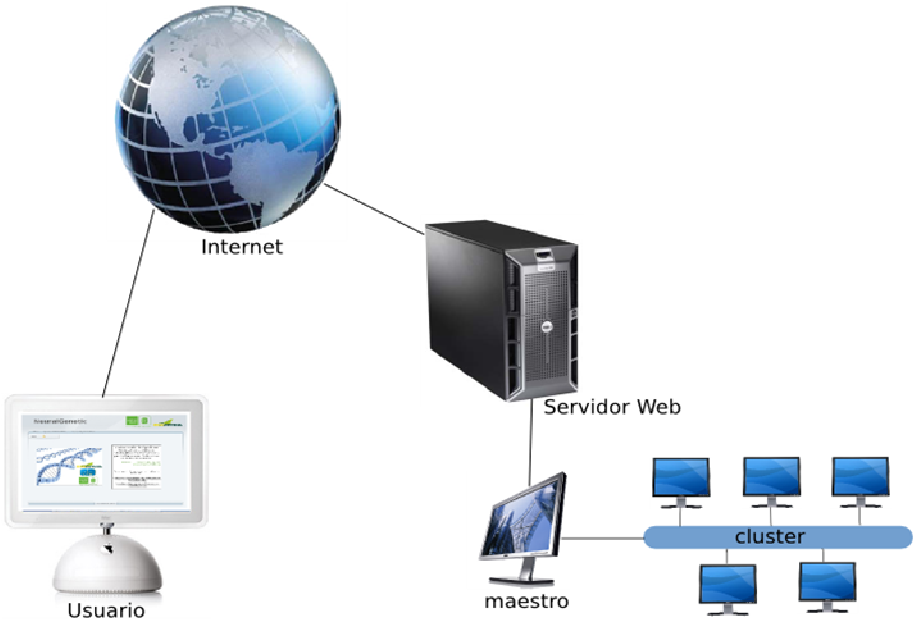


Ilustración 27: Interacción del usuario con el sistema.

Fuente: Autores proyecto

La interfaz web ha sido desarrollada utilizando el CMS[15] Joomla[16], el cual permite crear un framework[17] para la edición y administración de contenidos; en él se presenta información descriptiva de NeuralGenetic, así como una explicación de como ingresar los parámetros del algoritmo.

La interfaz es vista por el usuario como se muestra en la imagen anterior, en ella es posible la exploración de la información general del algoritmo y la información necesaria para el envío de trabajos al sistema distribuido sin necesidad de autenticarse en el sistema.

---

15 CMS: Sistema manejador de contenidos

16 JOOMLA: http://www.joomla.org

17 Framework: estructura de soporte que usa diseño reutilizable para un sistema o software.

A continuación se profundiza sobre cada sección de la interfaz.

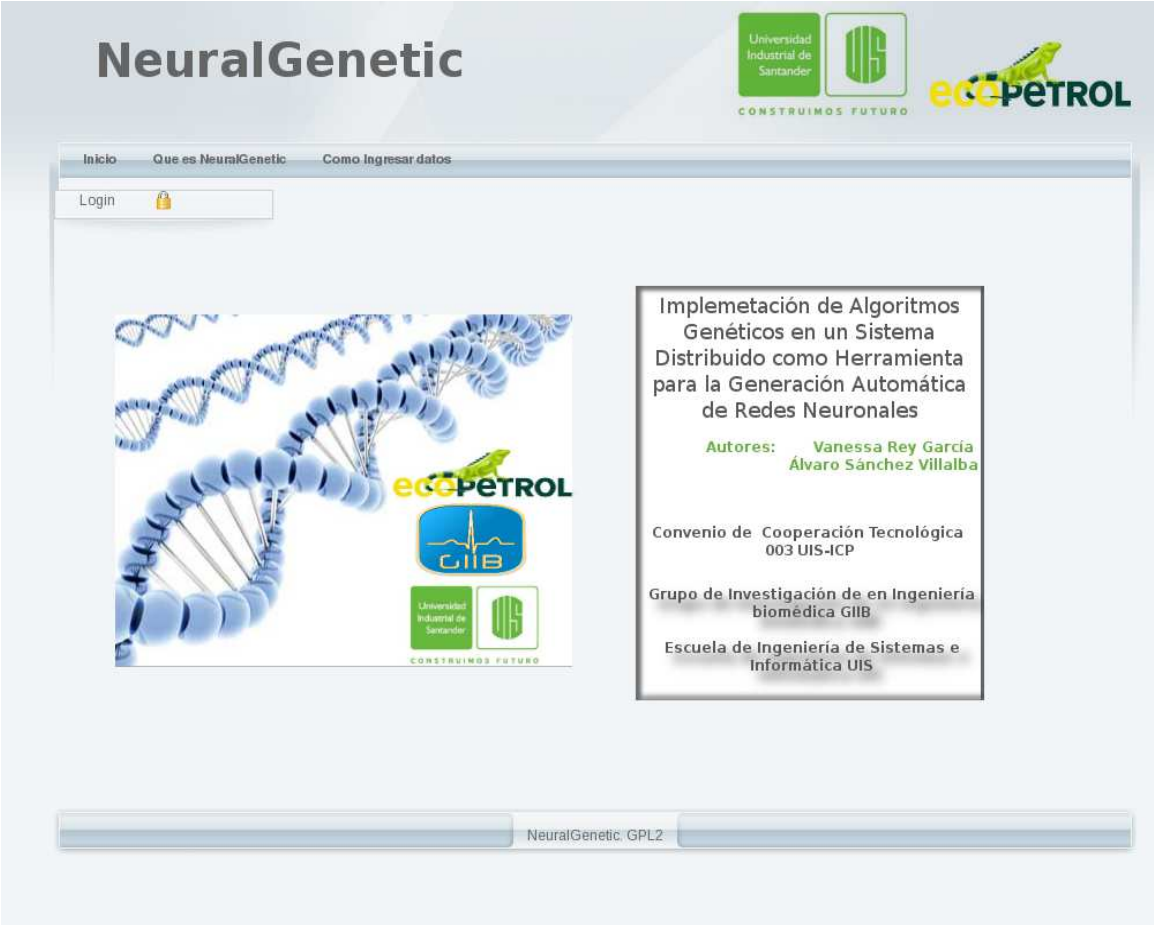


Ilustración 28: Inicio Interfaz web

Fuente: Autores proyecto

Para la interacción con el cluster es necesario ser un usuario del sistema, este usuario es creado por los administradores de éste; una vez autenticado el usuario podrá enviar trabajos desde el formulario de ingreso de parámetros del algoritmo, recibir los resultados arrojados por el sistema, simular las redes neuronales artificiales conseguidas como resultado del algoritmo genético y monitorear el

cluster

10.2. FORMULARIO INGRESO DE DATOS

Fue desarrollado un formulario basado en el lenguaje de programación web PHP[18], el cual agrupa los parámetros necesarios para la ejecución del algoritmo y los envía mediante el protocolo ssh[19] al nodo maestro del cluster.

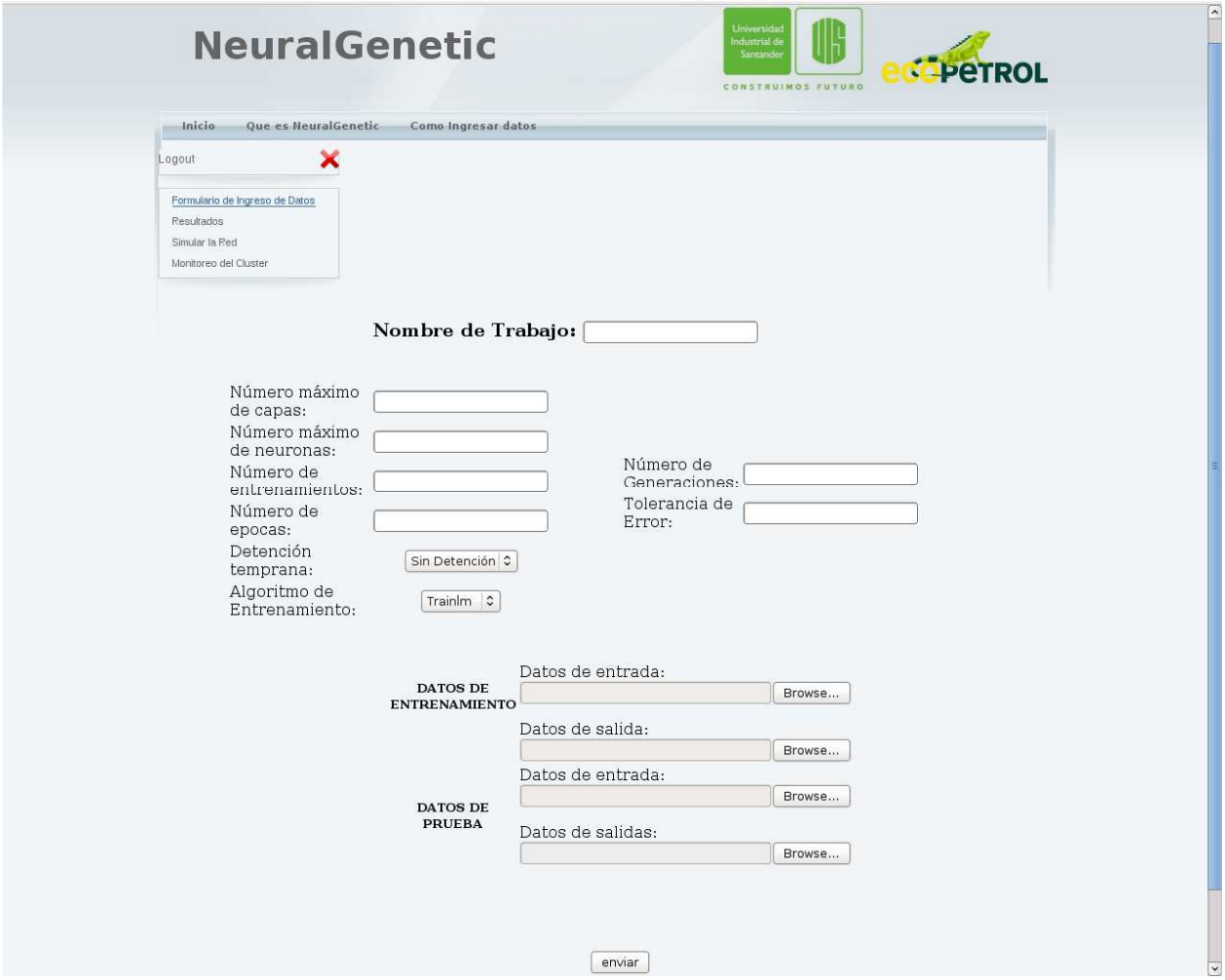


Ilustración 29: Formulario de ingreso de datos sin detención temprana

[18] PHP es un lenguaje script diseñado para la producción de páginas web dinámicas.

[19] SSH: Secure Shell: intérprete de órdenes seguro

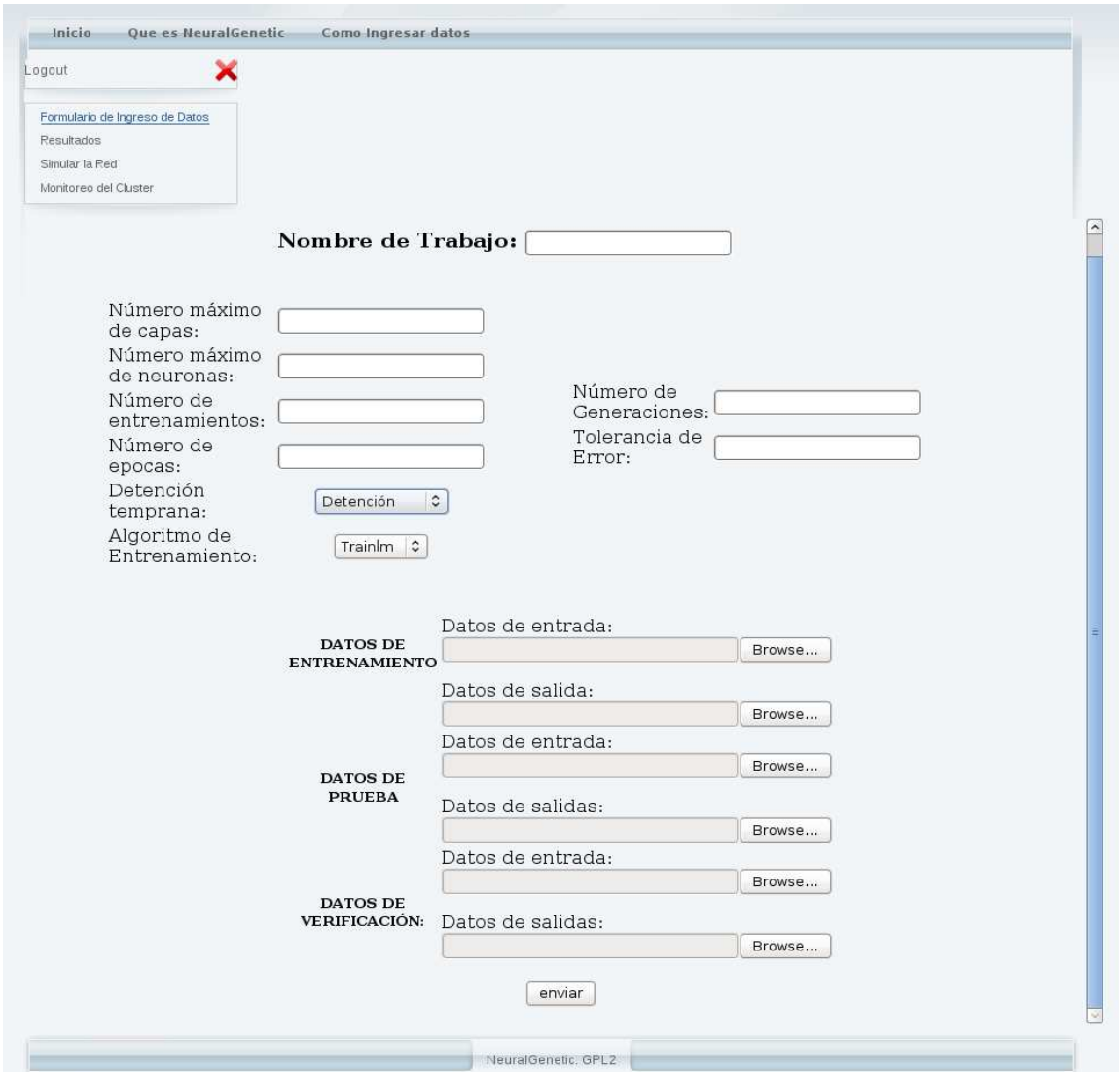
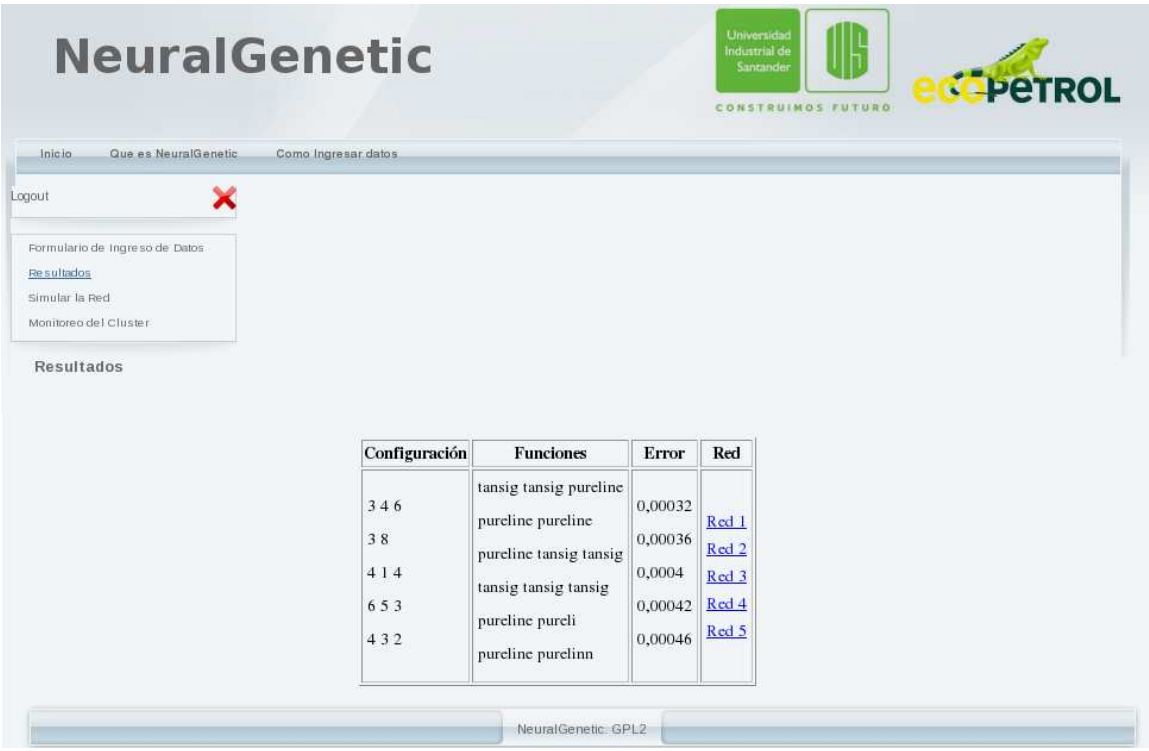


Ilustración 30: Formulario de ingreso de datos con detención temprana

Fuente: Autores proyecto

Ilustración 31: Mensaje de error, no ha terminado de ejecutarse el algoritmo

Fuente: Autores proyecto



Configuración	Funciones	Error	Red
3 4 6	tansig tansig pureline	0,00032	Red 1
3 8	pureline pureline	0,00036	Red 2
4 1 4	pureline tansig tansig	0,0004	Red 3
6 5 3	tansig tansig tansig	0,00042	Red 4
4 3 2	pureline pureli	0,00046	Red 5
	pureline purelinn		

Ilustración 32: Presentación de resultados

Fuente: Autores proyecto

resultado. El objetivo de la simulación es presentar a la red datos nuevos que no han sido utilizados en el entrenamiento ni en las pruebas realizadas a ésta, con el fin de que la red prediga la salida.

Para realizar la simulación de una red, es necesario elegir que red se desea simular e insertar las entradas correspondientes y el sistema retornará la salida que arrojará la red simulada.

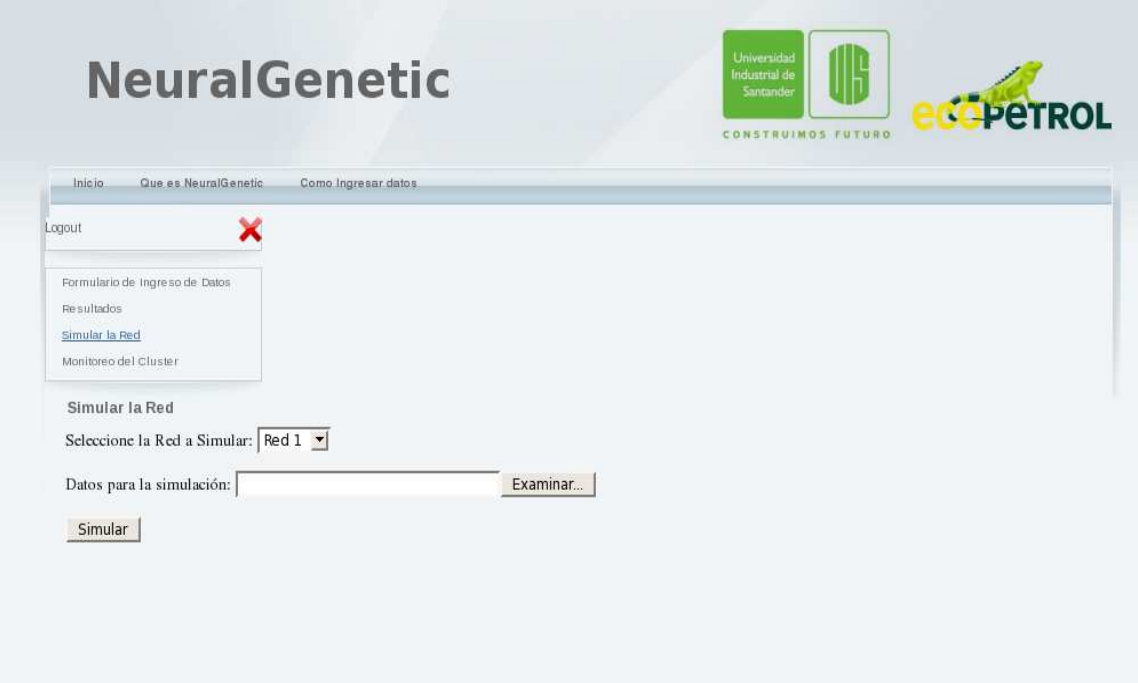


Ilustración 33: Simulación de la red neuronal

Fuente: Autores proyecto

10.5. SISTEMA DE MONITOREO GANGLIA

Con el fin de controlar remotamente la evolución y el estado de algunos recursos del cluster como lo son la memoria RAM, el procesador y la red, se instaló el sistema de monitoreo Ganglia[20], el cual permite supervisar en tiempo real, de acuerdo a tiempos y métricas variables definidas por el usuario como porcentajes de carga del procesador, bytes transmitidos por unidad de tiempo, entre otros. Estas estadísticas se presentan para cada uno de los nodos pertenecientes al cluster.

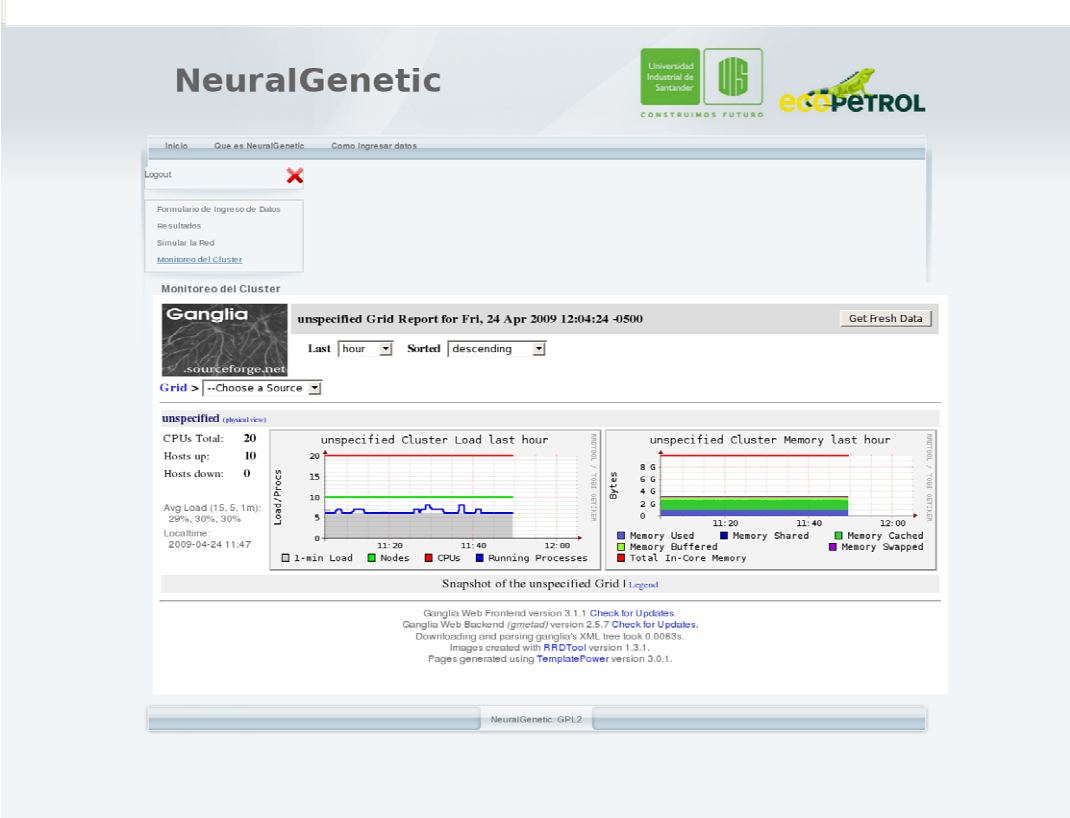


Ilustración 34: Sistema de monitoreo Ganglia

Fuente: Autores proyecto

[20] Ganglia: Es un sistema de control distribuido escalable de alto rendimiento para sistemas de computación, tales como Cluster's y Grids.

CAPITULO 11:

PRUEBAS

11. PRUEBAS

En esta etapa de la implementación se verifica el funcionamiento, la escalabilidad y el cumplimiento en la disminución de tiempos y tareas de procesamiento.

11.1. PRUEBAS DE PREDICCIÓN DE PROPIEDADES QUÍMICAS DE CRUDOS

Estas pruebas fueron realizadas con el fin de verificar la precisión y confiabilidad en los resultados arrojados por las redes neuronales artificiales producto de la ejecución del algoritmo genético; fueron desarrollados apartir de datos proporcionados por el ICP y corresponden a 12 espectros de resonancias magnéticas nucleares (RMN)[21] realizados a 30 crudos Colombianos; además se cuenta con datos experimentales de algunas propiedades físico-químicas de dichos crudos, como la densidad, gravedad API[22], gravedad específica, entre otras.

La tabla 1 muestra las 12 regiones espectrales, por razones de confidencialidad en los datos no se especifican los nombres de los crudos que son propiedad de Ecopetrol S.A[23], en su lugar, son enumerados de 1 a 30.

---

[21] RMN: Permiten estudiar la información estructural o química de una muestra.

[22] Gravedad API: Medida de densidad que describe que tan pesado o liviano es el petróleo comparándolo con el agua.

[23] Principal compañía petrolera en Colombia, pertenece al grupo de las 39 petroleras más grandes del mundo.

CRUDO	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12
Crudo1	0,1759	0,6154	3,4417	3,2756	0,0892	0,3682	4,0312	3,9484	3,9567	14,4607	39,6891	25,9476
Crudo2	0,1359	0,4933	3,0593	3,2626	0,1222	0,2265	3,2856	3,6957	4,0823	13,7599	39,3277	28,5489
Crudo3	0,1191	0,4363	2,9873	3,1565	0,0831	0,2411	3,4925	3,7230	3,6722	12,7316	43,2267	26,1306
Crudo4	0,1414	0,4828	3,2477	3,1797	0,1279	0,2173	2,9207	3,3638	3,5230	12,8640	42,8274	27,1042
Crudo5	0,1581	0,5674	3,4810	3,1760	0,1111	0,2975	3,2340	3,5705	3,6351	13,3480	41,9822	26,4392
Crudo6	0,1434	0,4995	2,7970	2,5512	0,1109	0,3643	2,6868	3,0224	3,1263	12,4421	45,6034	26,6527
Crudo7	0,1031	0,3574	2,2531	2,4336	0,1166	0,1516	2,0674	2,6700	2,8229	11,6692	46,0557	29,2994
Crudo8	0,1215	0,4480	2,7837	2,6738	0,1197	0,2603	2,2970	2,8166	3,0085	12,5727	44,4998	28,3985
Crudo9	0,1296	0,4335	2,4544	2,5825	0,1287	0,1168	1,8589	2,3002	2,5754	11,7253	48,1174	27,5773
Crudo10	0,1271	0,4217	2,6687	2,6856	0,1176	0,2257	2,9013	3,0369	3,0813	11,9104	47,9017	24,9220
Crudo11	0,1472	0,4890	3,0550	2,5972	0,0976	0,2259	2,6962	3,1827	3,0138	11,1678	48,5548	24,7727
Crudo12	0,1117	0,4025	2,6787	2,5967	0,1009	0,1124	2,0239	2,7699	2,7658	10,4178	49,4345	26,5852
Crudo13	0,0983	0,3449	2,2711	2,5171	0,1184	0,2057	2,1336	2,7332	2,9631	11,9581	45,9158	28,7407
Crudo14	0,1027	0,3508	2,2810	2,5447	0,1413	0,1978	2,0731	2,7102	2,9179	11,5315	46,2977	28,8513
Crudo15	0,1168	0,4192	2,7673	2,4862	0,1112	0,2259	2,4475	2,9786	2,8961	11,0932	49,1761	25,2817
Crudo16	0,0941	0,3101	2,0250	2,1571	0,1038	0,1553	1,7548	2,2021	2,3159	9,5537	52,5917	26,7363
Crudo17	0,1002	0,3867	2,8977	2,5784	0,1115	0,0796	1,8009	2,8314	2,7182	9,6699	50,8746	25,9509
Crudo18	0,1038	0,4005	2,9855	4,1761	0,2815	0,3314	1,9554	2,3671	3,2095	8,9825	48,4097	26,7968
Crudo19	0,1095	0,3927	2,9380	2,8498	0,1845	0,0936	1,8324	2,7947	2,7968	10,1830	50,4420	25,3830
Crudo20	0,0774	0,2391	1,7023	1,9724	0,1037	0,0644	1,0117	1,7625	2,1562	9,6240	51,9872	29,2991
Crudo21	0,0669	0,2561	1,1112	2,3309	0,2916	0,2345	0,6127	1,3923	1,7877	5,5442	62,4780	23,8939
Crudo22	0,0887	0,2880	2,2325	2,2260	0,1205	0,0782	1,1072	1,9624	2,3445	9,3845	52,7951	27,3725
Crudo23	0,0769	0,2810	2,4531	2,1230	0,1199	0,1151	1,1098	2,1119	2,4012	7,7296	55,3147	26,1639
Crudo24	0,1081	0,3720	2,8305	2,7798	0,2198	0,0577	1,2167	2,2822	2,6285	11,3248	48,9168	27,2633
Crudo 25	0,0687	0,2191	2,0161	2,2507	0,0000	0,0000	0,2373	1,5247	4,0982	6,2880	51,8585	31,4387
Crudo 26	0,1145	0,3475	2,6588	2,0135	0,0000	0,0242	0,5358	1,7997	3,1628	5,8457	50,9075	32,5900
Crudo 27	0,1465	0,4509	2,1179	1,2211	0,0000	0,0301	1,5054	2,0978	1,5305	11,1217	50,0964	29,6816
Crudo 28	0,2109	0,5560	2,4567	2,0478	0,0032	0,0087	1,8357	2,8245	2,6005	13,2104	43,2574	30,9881
Crudo 29	0,1622	0,5117	2,6089	1,8419	0,0055	0,1096	2,1706	2,9420	2,5202	13,5300	43,8289	29,7686
Crudo 30	0,3202	0,8910	3,9256	2,7144	0,0174	0,1085	3,1673	4,1748	3,0762	13,5521	40,1994	27,8531

Tabla 1: 12 regiones espectrales de resonancias magnéticas realizadas a 30 crudos

Densidad, g/mL	Gravedad específica, dcrudo15°C/dagua15°C	Gravedad API, °API	IC	Azufre, %m	Punto de Fluidez, °C	Punto de Chispa, °C	Número de ácido, mg KOH/g	Nitrogeno Total, ppm	%RCM, %m	Niquel, ppm
1,0186	1,0195	7,2915	204,3127	3,2470	36,0000	74,0000	0,2230	6610,0000	18,4700	132,5400
0,9928	0,9937	10,8983	77,2699	1,5970	15,0000	138,0000	6,0270	5212,0000	9,2500	66,0800
0,9765	0,9774	13,2752	67,1741	2,6450	-3,0000	61,0000	4,0470	5109,0000	11,2000	66,7600
0,9558	0,9567	16,4107	62,6813	1,4710	-24,0000	-3,0000	1,8570	4454,0000	10,5500	85,3000
0,9501	0,9510	17,2980	62,2800	1,9960	-27,0000	5,0000	0,0620	4066,0000	13,2600	77,7100
0,9388	0,9396	19,0891	61,1016	1,3710	0,0000	45,0000	<0.1	2902,0000	9,7900	51,5900
0,9277	0,9285	20,8909	53,1529	0,9490	<-33.0	47,0000	2,0660	2709,0000	5,4500	21,9200
0,9244	0,9252	21,4349	55,9800	0,9410	-30,1000	15,5000	<0.1	2136,0000	7,7900	21,6200
0,9201	0,9209	22,1496	53,2965	0,8650	21,0000	-2,0000	0,2510	1717,0000	6,8600	29,2300
0,9201	0,9209	22,1496	48,9882	1,5070	-21,0000	-5,0000	0,5560	4059,0000	8,3500	72,2600
0,9047	0,9055	24,7651	46,1974	0,8690	-24,0000	-38,0000	<0.1	2623,0000	8,2300	46,6000
0,8965	0,8973	26,1944	46,4528	0,3870	9,2000		0,1500			
0,8942	0,8950	26,6000	43,3654	0,7910	<-33.0	-44,0000	1,7710	2479,0000	4,5600	21,0950
0,8891	0,8899	27,5069	41,5975	0,7610	<-33.0	-30,0000	1,3010	2614,0000	5,2000	30,9400
0,8831	0,8839	28,5872	41,3903	0,7230	3,0000	<-56.0	<0.1	2100,0000	6,0100	28,3600
0,8687	0,8695	31,2409	34,6019	0,9320	-12,0000	-34,0000	0,1820	1439,0000	4,4400	9,7500
0,8640	0,8648	32,1262	37,0549	0,4330	3,0000	<-50.0	0,0550	1539,0000	3,9900	18,5000
0,8603	0,8611	32,8299	30,2826	0,1280	18,0000	<-40.0	<0.1	448,0000	1,6900	1,1800
0,8602	0,8610	32,8490	36,7671	0,2780	6,0000		0,0650			
0,8580	0,8588	33,2704	34,7974	0,2290	-2,3000		0,4960			
0,8433	0,8441	36,1426	26,4176	0,3080	27,0000	-9,0000	<0.1	376,0000	1,0700	0,4573
0,8143	0,8150	42,1129	29,8178	0,0520	-6,0000	-29,0000	<0.1	482,1300	0,7900	1,2700
0,8119	0,8126	42,6261	24,8038	0,0760	15,0000	<-50.0	<0.1	261,2000	0,5700	0,6105
0,8079	0,8086	43,4883	36,6158	0,0810	<-33.0	<-45.0	<0.1	193,9000	0,9800	0,4913
0,8010		45,1000	17,9315	0,0450	9,0000		<0.05	179,0000	0,2300	0,0871
0,8103		43,0000	19,5986	0,1290	6,0000		<0.05	264,0000	0,7673	1,0500
0,8809		29,0000	36,2574	0,5340	-9,0000		0.154	2477,0000	5,6900	40,4100
0,9106		23,8000	44,4000	0,8080	<-33.0		2,0340	2520,0000	5,1800	20,7100
0,9169		22,7000	48,1306	1,2760	-18,0000		1,3430	3228,0000	6,6900	47,7700
0,9836		12,3000	72,9801	2,1850	3,0000		0,1000	4856,0000	15,8500	82,6000

Tabla 2: Propiedades físico-químicas de los 30 crudos

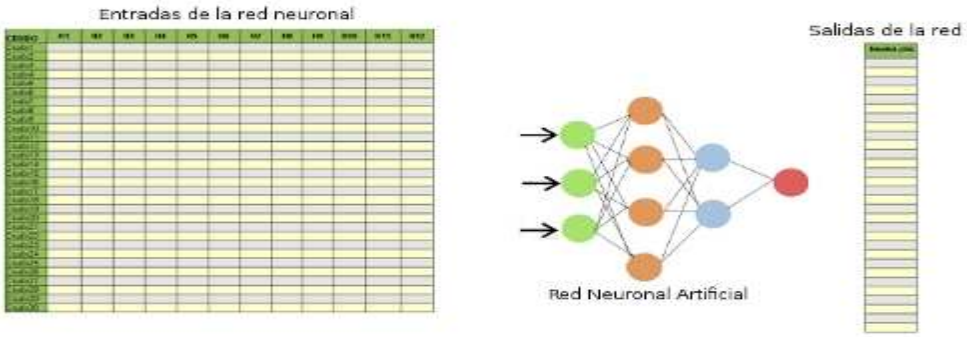


Ilustración 35: Entradas y Salidas de la red neuronal artificial

Fuente: Autores proyecto

11.1.1. PRUEBA: PROPIEDAD DENSIDAD

Para realizar la prueba de la propiedad “Densidad” se ejecutó el algoritmo genético con los siguientes parámetros:

Max de Capas: 4
Max de Neuronas: 30
Número de Entrenamientos: 15
Detención Temprana: No
Algoritmo de Entrenamiento: Trainlm[24]
Número de Generaciones: 9
Tolerancia de error: 0,01
Épocas en la RNA: 500

Se escogió como muestra de entrenamiento los 28 primeros crudos, de los cuales 24 son usados como datos de entrenamiento (crudo 1 a crudo 24), y 4 son usados como datos de pruebas (crudo 25 a crudo 28), y el crudo 29 y crudo 30 son usados como muestra de validación con los cuales se realiza la predicción de la propiedad “Densidad"

La tabla 3 muestra los datos arrojados por la mejor red hallada por el algoritmo genético.

[24] Algoritmo de Entrenamiento de la red neuronal Levenberg-Marquardt

CRUDO	EXPERIMENTAL	RNA	ERROR RNA
1	1,0186	1,0081	0,0105
2	0,9928	0,9848	0,0080
3	0,9765	0,9702	0,0063
4	0,9558	0,9515	0,0043
5	0,9501	0,9464	0,0037
6	0,9388	0,9362	0,0026
7	0,9277	0,9263	0,0014
8	0,9244	0,9233	0,0011
9	0,9201	0,9194	0,0007
10	0,9201	0,9194	0,0007
11	0,9047	0,9056	0,0009
12	0,8965	0,8982	0,0017
13	0,8942	0,8961	0,0019
14	0,8891	0,8915	0,0024
15	0,8831	0,8861	0,0030
16	0,8687	0,8732	0,0045
17	0,8640	0,8689	0,0049
18	0,8603	0,8656	0,0053
19	0,8602	0,8655	0,0053
20	0,8580	0,8635	0,0055
21	0,8433	0,8503	0,0070
22	0,8143	0,8242	0,0099
23	0,8119	0,8220	0,0101
24	0,8079	0,8184	0,0105
25	0,8010	0,8096	0,0086
26	0,8103	0,8239	0,0136
27	0,8809	0,9034	0,0225
28	0,9106	0,9990	0,0884
29	0,9169	0,9781	0,0612
30	0,9836	1,0039	0,0203

Tabla 3: Valores arrojados por la red neuronal con respecto a valores experimentales para la prueba de densidad

Fuente: Autores proyecto

La columna “Error RNA” es calculada mediante la fórmula 2:

$$e = |y_i - \hat{y}_i| \quad (2)$$

donde $\hat{y}_i$ es el valor predicho.

Los respectivos errores de entrenamiento, prueba, y simulación fueron hallados calculando el valor promedio de la columna “Error RNA”

Error simulación= 0,040747

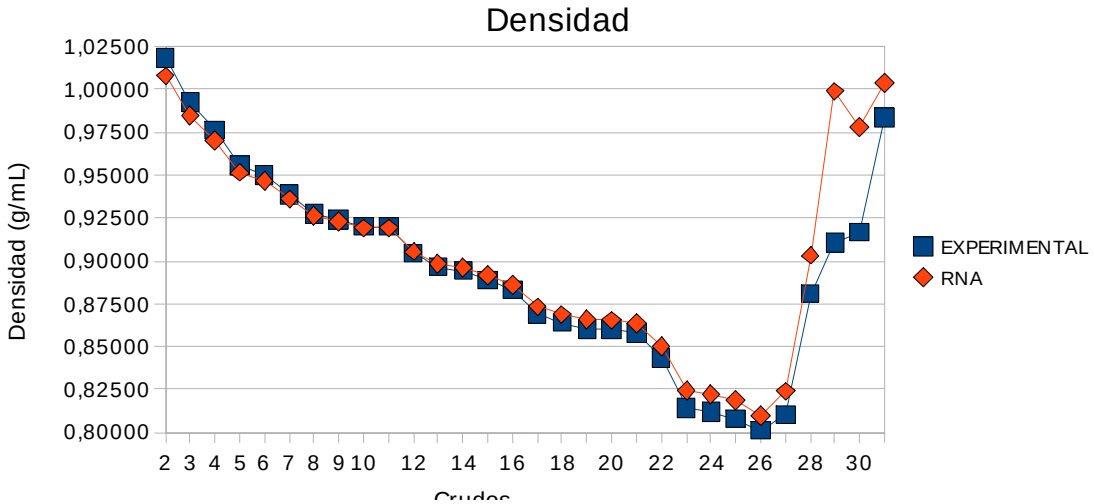
Se calcula posteriormente el error cuadrático medio mediante la fórmula 3:

$$EMC = \sqrt{\frac{\sum_{i=1}^{n} (y_i - y_i)^2}{n}} \quad (3)$$

2	0,0000633
3	0,0000400
4	0,0000181
5	0,0000136
6	0,0000065
7	0,0000021
8	0,0000012
9	0,0000005
10	0,0000005
11	0,0000007
12	0,0000028
13	0,0000036
14	0,0000058
15	0,0000091
16	0,0000198
17	0,0000243
18	0,0000280
19	0,0000281
20	0,0000305
21	0,0000489
22	0,0000979
23	0,0001027
24	0,0001110
25	0,0000734
26	0,0001851
27	0,0005047
28	0,0078190
29	0,0037425
30	0,0004128
	0,0135076

$$EMC = \sqrt{\frac{\sum_{i=1}^{n} (y_i - y_i)^2}{n}} = \sqrt{\frac{0,0135076}{30}} = 0.021219 = 2.1219$$

Gráfica 1: Valores arrojados por la red neuronal para la propiedad Densidad



Detención Temprana: No
Algoritmo de Entrenamiento: TrainIm[25]
Número de Generaciones: 9
Tolerancia de error: 0,01
Épocas en la RNA: 500

Se escogió de igual manera que la prueba anterior las muestras de entrenamiento y de validación, 28 primeros crudos, de los cuales 24 son usados como datos de entrenamiento (crudo 1 a crudo 24), y 4 son usados como datos de pruebas (crudo 25 a crudo 28), y el crudo 29 y crudo 30 son usados como muestra de validación con los cuales se realiza la predicción de la propiedad “Gravedad Específica" .

La tabla 4 muestra los datos arrojados por la mejor red hallada por el algoritmo genético.

[25] Algoritmo de Entrenamiento de la red neuronal Levenberg-Marquardt

CRUDO	EXPERIMENTAL	RNA	ERROR RNA
1	1,0195	1,0086	0,0109
2	0,9937	0,9859	0,0078
3	0,9774	0,9710	0,0064
4	0,9567	0,9525	0,0041
5	0,9510	0,9473	0,0036
6	0,9396	0,9371	0,0025
7	0,9285	0,9273	0,0012
8	0,9252	0,9242	0,0010
9	0,9209	0,9203	0,0006
10	0,9209	0,9199	0,0010
11	0,9055	0,9067	0,0012
12	0,8973	0,8982	0,0009
13	0,8950	0,8982	0,0032
14	0,8899	0,8924	0,0025
15	0,8839	0,8823	0,0016
16	0,8695	0,8794	0,0099
17	0,8648	0,8704	0,0056
18	0,8611	0,8663	0,0052
19	0,8610	0,8664	0,0054
20	0,8588	0,8612	0,0025
21	0,8441	0,8511	0,0070
22	0,8150	0,8289	0,0138
23	0,8126	0,8190	0,0063
24	0,8086	0,8195	0,0109
25	0,8017	0,8184	0,0166
26	0,8110	0,8248	0,0138
27	0,8817	0,9661	0,0844
28	0,9114	0,9816	0,0702
29	0,9177	0,9766	0,0589
30	0,9845	0,9819	0,0025

Tabla 4: Valores arrojados por la red neuronal con error respecto a valores experimentales para la prueba de Gravedad Específica

La columna “Error RNA” es calculada mediante la fórmula 2.

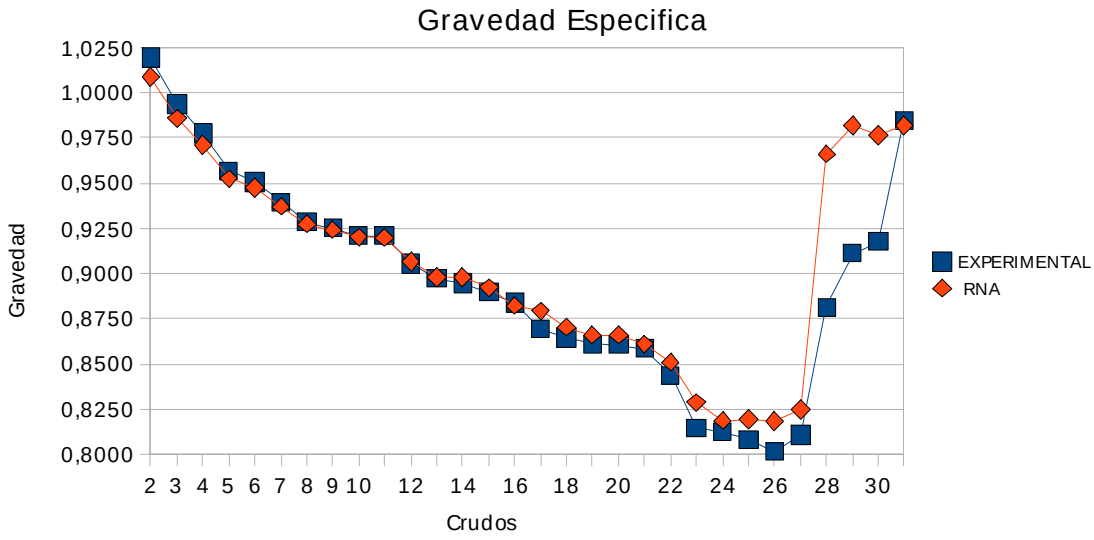
Los respectivos errores de entrenamiento, prueba, y simulación fueron hallados calculando el valor promedio de la columna “Error RNA”.

Error simulación= 0,030704

Se calcula posteriormente el error cuadrático medio mediante la fórmula 3:

CRUDO	$(y_i - \hat{y}_i)^2$
1	0,000119
2	0,000061
3	0,000041
4	0,000017
5	0,000013
6	0,000006
7	0,000002
8	0,000001
9	0,000000
10	0,000001
11	0,000001
12	0,000001
13	0,000010
14	0,000006
15	0,000002
16	0,000098
17	0,000032
18	0,000027
19	0,000029
20	0,000006
21	0,000049
22	0,000191
23	0,000040
24	0,000119
25	0,000277
26	0,000190
27	0,007117
28	0,004930
29	0,003466
30	0,000006

$$EMC = \sqrt{\frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}} = \sqrt{\frac{0.016860}{30}} = 0.023706 = 2.3706\,\%$$



Gráfica 2: Valores arrojados por la red neuronal para la propiedad Gravedad Específica

11.1.3. PRUEBA: PROPIEDAD %CERAS

Para realizar la prueba de la propiedad "Propiedad Específica" se ejecutó el algoritmo genético con los siguientes parámetros:

Max de Capas: 4
Max de Neuronas: 30
Número de Entrenamientos: 15
Detención Temprana: No
Algoritmo de Entrenamiento: TrainIm
Número de Generaciones: 9
Tolerancia de error: 0,01
Épocas en la RNA: 500

Se contaba con datos experimentales de 27 crudos, por lo tanto se tomaron 22 crudos para entrenamiento, 4 para pruebas (crudo7, crudo15, crudo21 crudo3) y 1 para predicción (crudo 27).

La tabla 5 muestra los datos arrojados por la mejor red hallada por el algoritmo

genético.

Tabla 5: Valores arrojados por la red neuronal con error respecto a valores experimentales para la prueba de %Ceras

CRUDO	EXPERIMENTAL	RNA	ERROR RNA
1	0,27	0,27	0,00
2	0,65	0,65	0,00
3	0,33	0,41	0,08
4	2,85	2,85	0,00
5	3,81	3,81	0,00
6	7,34	7,34	0,00
7	8,55	8,66	0,11
8	4,14	4,14	0,00
9	7,80	7,8	0,00
10	1,40	1,4	0,00
11	11,33	11,33	0,00
13	1,13	1,13	0,00
14	2,64	2,64	0,00
15	8,51	8,83	0,32
16	10,97	10,97	0,00
17	8,26	8,26	0,00
18	9,28	9,28	0,00
21	19,28	18,89	0,39
22	7,62	7,62	0,00
23	9,34	9,34	0,00
24	4,61	4,61	0,00
25	15,84	15,84	0,00
26	7,78	7,78	0,00
27	6,57	6,29	0,28
28	0,18	0,18	0,00
29	4,44	4,44	0,00
30	1,58	1,58	0,00

La columna “Error RNA” es calculada mediante la fórmula 2.

Los respectivos errores de entrenamiento, prueba, y simulación fueron hallados calculando el valor promedio de la columna “Error RNA”.

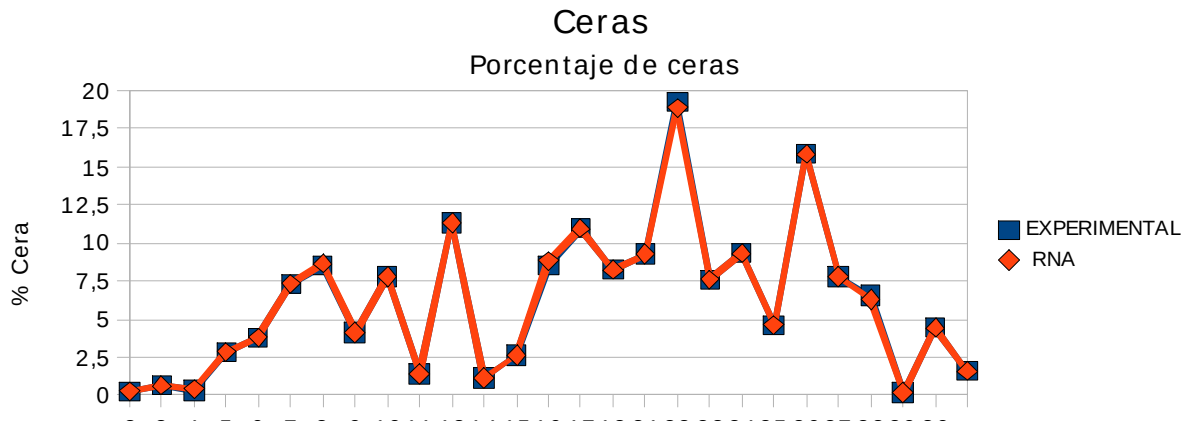
Se calcula posteriormente el error cuadrático medio mediante la fórmula 3:

error entrenamiento: 0,00
error prueba: 0,225
error simulación: 0,28

1	0,000
2	0,000
3	0,007
4	0,000
5	0,000
6	0,000
7	0,012
8	0,000
9	0,000
10	0,000
11	0,000
13	0,000
14	0,000
15	0,101
16	0,000
17	0,000
18	0,000
21	0,153
22	0,000
23	0,000
24	0,000
25	0,000
26	0,000
27	0,078
28	0,000
29	0,000
30	0,000
	0,351

$$EMC = \sqrt{\frac{\sum_{i=1}^{n} (y_i - y_i)^2}{n}} = \sqrt{\frac{0,351}{27}} = 0.11401 = 11,4\%$$

Gráfica 3: Valores arrojados por la red neuronal para la propiedad de %Ceras



11.1.4. PRUEBA: PROPIEDAD NC7

Para realizar la prueba de la propiedad “nc7” se ejecutó el algoritmo genético con los siguientes parámetros:

Max de Capas: 4
Max de Neuronas: 30
Número de Entrenamientos: 15
Detención Temprana: No
Algoritmo de Entrenamiento: TrainIm
Número de Generaciones: 9
Tolerancia de error: 0,01
Épocas en la RNA: 500

Se escogió como muestra de entrenamiento los 28 primeros crudos, de los cuales 20 son usados como datos de entrenamiento (crudo 1 a crudo 20), y 8 son usados como datos de pruebas (crudo 21 a crudo 28), y el crudo 29 y crudo 30 son usados como muestra de validación con los cuales se realiza la predicción de la propiedad “nc7".

La tabla 6 muestra los datos arrojados por la mejor red hallada por el algoritmo

genético.

Ilustración 36: Valores arrojados por la red neuronal con error respecto a valores experimentales para la prueba de propiedad
nc7.

CRUDO	EXPERIMENTAL	RNA	ERROR RNA
1	14,39	14,39	0,0004
2	2,07	2,07	0,0000
3	6,93	6,93	0,0000
4	8,75	8,75	0,0000
5	11,39	11,39	0,0000
6	10,4	10,4	0,0000
7	1,65	1,65	0,0000
8	7,52	7,52	0,0000
9	6,00	6	0,0000
10	5,52	5,52	0,0000
11	6,22	6,22	0,0000
12	2,23	2,23	0,0000
13	1,92	1,92	0,0000
14	3,2	3,2	0,0000
15	4,5	4,5	0,0000
16	2,00	2	0,0000
17	1,72	1,72	0,0000
18	0,86	0,86	0,0000
19	1,18	1,18	0,0000
20	0,83	0,83	0,0002
21	0,07	0,03	0,0395
22	0,17	0,15	0,0179
23	0,11	0,13	0,0215
24	0,67	0,63	0,0392
25	0	0,05	0,0500
26	0,14	0,16	0,0250
27	2,16	2,01	0,1518
28	1,29	1,29	0,0044
29	4,27	4,3067	0,0367
30	14,58	14,49	0,0912

La columna “Error RNA” es calculada mediante la fórmula 2.

Los respectivos errores de entrenamiento, prueba, y simulación fueron hallados calculando el valor promedio de la columna “Error RNA”.

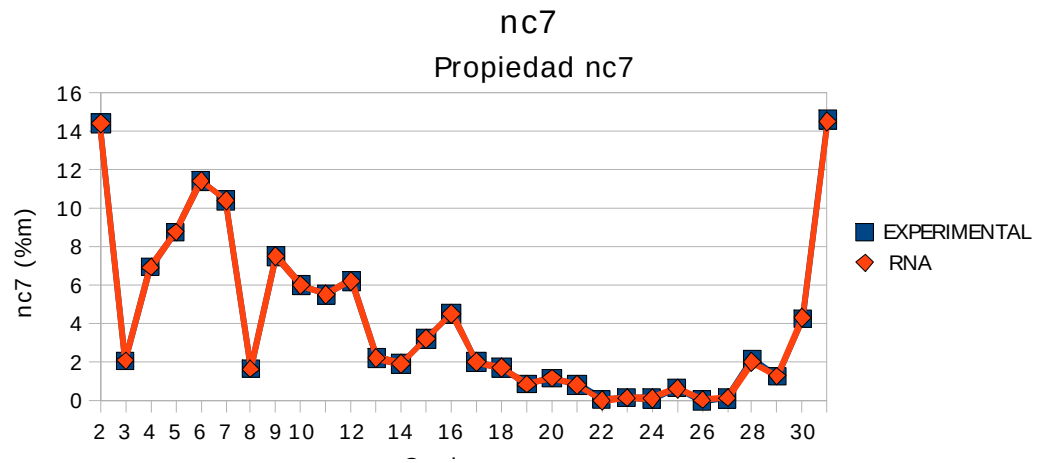
Error prueba: 0,044
Error simulación: 0,064

Se calcula posteriormente el error cuadrático medio mediante la fórmula 3:

1	0,000
2	0,000
3	0,000
4	0,000
5	0,000
6	0,000
7	0,000
8	0,000
9	0,000
10	0,000
11	0,000
12	0,000
13	0,000
14	0,000
15	0,000
16	0,000
17	0,000
18	0,000
19	0,000
20	0,000
21	0,002
22	0,000
23	0,000
24	0,002
25	0,003
26	0,001
27	0,023
28	0,000
29	0,001
30	0,008
	0,0397

$$EMC = \sqrt{\frac{\sum_{i=1}^{n} (y_i - y_i)^2}{n}} = \sqrt{\frac{0,0397}{30}} = 0.03605 = 3.6$$

Gráfica 4: Valores arrojados por la red neuronal para propiedad nc7



11.2. PRUEBAS AL CLUSTER

Con el fin de evaluar la escalabilidad y rendimiento del sistema distribuido se realizaron una serie de pruebas en las que se midió tiempo de procesamiento.

11.2.1. MEDICIÓN DEL TIEMPO DE PROCESAMIENTO

En esta prueba se verifica el rendimiento del cluster computacional variando el número de nodos esclavos, el algoritmo fue ejecutado con los siguientes parámetros:

Población total: 32 individuos (distribuidos en cada nodo)
max Neuronas: 10
max Capas: 3
Num Entrenamientos: 5
Tolerancia: 0.01
Migraciones: 3
Generaciones por nodo: 2
Épocas: 500

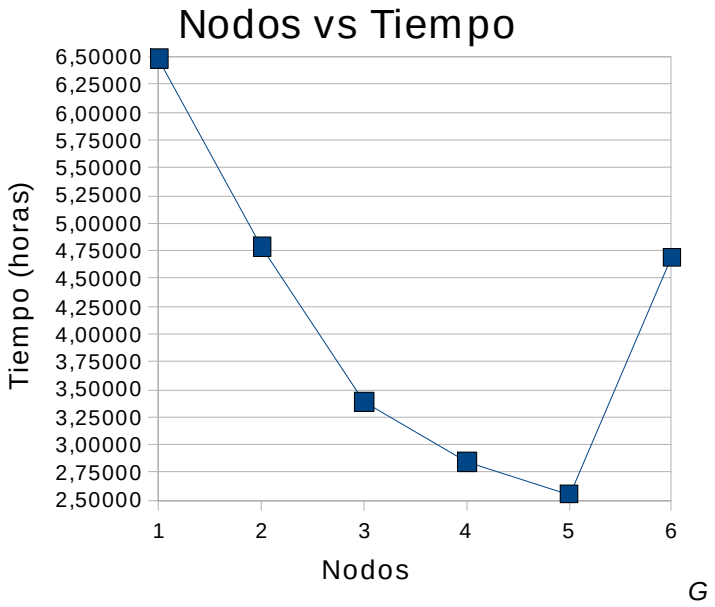
Mediante esta prueba se obtiene el número óptimo de nodos del cluster computacional, teniendo en cuenta que al trabajar con sistemas de archivos

distribuidos se tiene una latencia en la lectura y escritura sobre el disco, llegando a disminuir el rendimiento a partir de un determinado número de nodos.
La prueba consiste en el incremento de los nodos esclavos en el cluster ejecutando el algoritmo con los parámetros anteriormente mencionados, cabe destacar que la población es repartida equitativamente en los nodos esclavos.

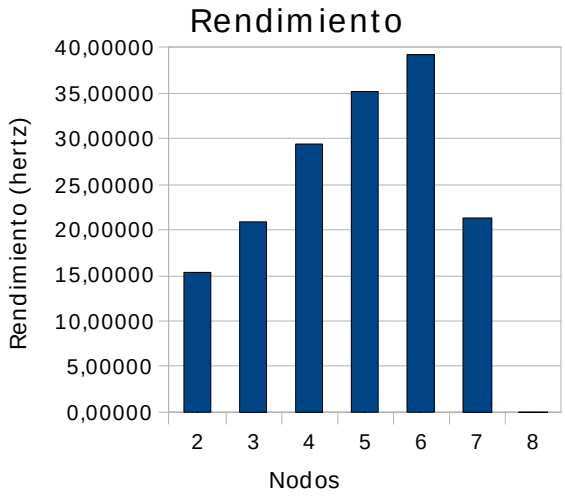
En la tabla 5 se muestra los tiempos y valores de rendimiento obtenidos luego de promediar los resultados arrojados al ejecutar la prueba cuatro (4) veces con la misma cantidad de nodos.

1	6,48875	15,41100
2	4,79167	20,86956
3	3,38667	29,52750
4	2,84361	35,16650
5	2,55486	39,14107
6	4,69167	21,31434

Tabla 6: Tiempo de procesamiento y rendimiento variando número de nodos.



Gráfica 5: Tiempo de Procesamiento vs Nodos



Gráfica 6: Rendimiento vs número de nodos

Se observa de la gráfica 3 que a medida que se incrementa el número de nodos el tiempo de procesamiento se reduce; al realizar la prueba con 6 nodos esclavos se nota que el tiempo de procesamiento incrementa considerablemente determinando de esta manera que la cantidad óptima y recomendable de nodos para trabajar en el cluster es 5 nodos, ya que con este número de nodos el sistema presenta el mejor rendimiento (Gráfica 4), al conseguir un equilibrio entre la frecuencia y volumen de información comunicada y la carga computacional de cada nodo.

El valor de rendimiento está dado por:

$$R(t) = 1/t \quad (4)$$

donde R representa el rendimiento en función del tiempo representado por t como el tiempo en ejecución.

Esta prueba es muy importante debido a que determinando la cantidad de nodos óptimos se elimina el tiempo ocioso de los nodos del cluster.

11.3. COMPARACIÓN CON OTRAS TÉCNICAS

Con el fin contrarrestar las redes neuronales obtenidas mediante el algoritmo genético con otros métodos de predicción, se realiza la siguiente prueba utilizando el método de regresión de mínimos cuadrados parciales (PLS), en el cual se encuentra un modelo lineal que describe algunas variables predichas en términos de otras variables experimentales.

La regresión mediante PLS fue realizada por el profesor de la Universidad Industrial de Santander Dr. Daniel Molina [13], para 6 de los 30 crudos que se trabajaron (crudo 25 al crudo 30), la prueba fue realizada para la propiedad "porcentaje de azufre" y se hallaron los datos expuestos en la tabla 6.

Crudo	Valor Experimental	Ycal	error total	(Ycal-Yexp)^2
Crudo 25	0,0450	0,0747	0,0297	0,0009
Crudo 26	0,1290	0,3319	0,2029	0,0412
Crudo 27	0,5340	0,1775	0,3565	0,1271
Crudo 28	0,8080	1,3405	0,5325	0,2836
Crudo 29	1,2760	1,0298	0,2462	0,0606
Crudo 30	2,1850	1,9451	0,2399	0,0576
Σ			1,6077	0,5709
			PRESS =	0,5709

Tabla 7: Valores hallados mediante la técnica PLS para la propiedad %Azufre

Donde la columna "Ycal" representa los valores hallados mediante el método PLS , la columna "error total" se calcula mediante la fórmula 5.

$$e_i = ycal_i - yexp_i \quad (5)$$

El valor PRESS es calculado cuando es utilizado el método de estimación del error

de generalización “Leave one out” (LOO)[26] y es calculado según la formula 6:

$$PRESS=\sum_{i=1}^{n} e_{[i]}^2 \quad (6)$$ donde n es el número de datos predichos

Los datos predichos de los 6 crudos correspondientes arrojados por la red neuronal para la propiedad de %Azufre se observan en la tabla 7.

Cabe aclarar que para la construcción el modelo de la red neuronal se usaron los 30 crudos; se realizaron 6 ejecuciones del algoritmo dejando por fuera del conjunto de entrenamiento a cada uno de los 6 crudos anteriormente descritos.

25	0,0450	0,2117	0,1667	0,0278
26	0,1290	0,2271	0,0981	0,0096
27	0,5340	0,7770	0,2430	0,0591
28	0,8080	0,5063	0,3017	0,0910
29	1,2760	1,2060	0,0700	0,0049
30	2,1850	2,5228	0,3378	0,1141
Σ			1,2174	0,3066
			PRESS =	0,3066

Tabla 8: Valores hallados mediante Redes Neuronales para la propiedad %Azufre

Mediante esta prueba se verifica el desempeño de las redes neuronales artificiales generadas automáticamente por el algoritmo genético paralelo sobre una arquitectura distribuida, al ser contrastada con valores obtenidos mediante otras técnicas distintas a la inteligencia artificial como lo es en este caso el método de regresión de mínimos cuadrados parciales (PLS). Se observan mejores resultados en el valor PRESS de la estimación mediante redes neuronales obteniéndose 0.3066, contra el valor PRESS 0.5709 obtenido mediante PLS para la propiedad

[26] LOO : Dejar una observación afuera y predecir el valor de ésta, así con todas las observaciaones.

química “Porcentaje de Azufre”.

12. CONCLUSIONES

- Se ha desarrollado un sistema que determina una configuración de red neuronal con excelentes capacidades de generalización mediante el uso de algoritmos genéticos sobre un sistema distribuido.

- Mediante redes neuronales artificiales generadas por el algoritmo genético fue posible modelar datos obtenidos de la resonancia magnética nuclear de crudos y sus propiedades físico-químicas y a partir de estos realizar el análisis y evaluación de los mismos, obteniendo resultados altamente confiables.

- El algoritmo genético distribuido fue desarrollado siguiendo lineamientos de diseño de algoritmos paralelos, implementando un algoritmo que optimiza la comunicación entre los procesadores, disminuyendo la frecuencia y el volumen de la información, al dividir la población entre el número de nodos permitiendo de esta manera la evaluación de gran cantidad de individuos en un menor tiempo y cantidad de procesamiento.

  - Por medio del algoritmo genético se ha logrado obtener redes neuronales artificiales con gran desempeño, además de la generalización de los datos analizados para modelar las propiedades de los crudos y sus fracciones, teniendo la capacidad de inferir o analizar con acierto datos no presentados en la fase de aprendizaje.

  - Aunque se obtiene confiabilidad en los resultados consiguiendo redes neuronales artificiales con el algoritmo genético capaces de predecir propiedades fisicoquímicas de los crudos, no se puede suprimir la necesidad de un experto, por el contrario, se ofrece una herramienta de soporte bastante confiable que sustituye otras técnicas químicas y matemáticas utilizadas para la consecución de éstas propiedades.

  - Se verificó el desempeño de las redes neuronales generadas por el algoritmo genético al contrastar los resultados con la técnica de regresión de mínimos cuadrados PLS; se consiguieron errores más bajos realizando pruebas con las propiedades físico-químicas de los crudos. Así mismo al comparar con el algoritmo secuencial se consiguieron mejores tiempos de procesamiento al encontrar un equilibrio entre la frecuencia de información comunicada y la carga computacional en cada nodo al disminuir la cantidad de individuos a evaluar.

13. RECOMENDACIONES

- Para tener un algoritmo genético que se ejecute sobre un cluster de producción estable que soporte una gran cantidad de procesos se recomienda invertir en hardware de mejores caracteristicas que el actual, de igual manera tener una infraestructura de redes de alta velocidad con el objetivo de que estos aspectos no influyan negativamente en el rendimiento de la infraestructura cluster**.**

- El sistema de archivos GlusterFS utilizado en la implementación del algoritmo genético genera buenos resultados frente a la usabilidad, no obstante se recomienda la implementación de un sistema de archivos más eficiente con menos latencia en lectura y escritura, permitiendo disminuir el tiempo de procesamiento.

- Se propone a partir de este proyecto explorar otras técnicas de Inteligencia Artificial, tales como las colonias de hormigas o abejas, para medir la calidad de respuesta y el rendimiento adquirido con estas técnicas frente al algoritmo genético.

- Es recomendable continuar con el crecimiento de estas herramientas de alto procesamiento mediante convenios inter-institucionales que permitan a la Universidad Industrial de Santander compartir recursos, experiencias, y trabajos colaborativos con otras entidades similares con el fin de que se aproveche el resultado de este trabajo ya que posibilita grandes oportunidades para la labor de investigación.

14. REFERENCIAS

[1] Elaine Rich , Kevin Knight. Artificial Intelligence: International Edition, McGraw-Hill, 1991.

[2] Rios J., Pazos A., NR Brisaboa, S Caridad, Estructura, Dinámica y Aplicaciones de las Redes Neuronales Artificiales, Madrid: Editorial Centro de Estudios Ramón Areces, 1991

[3] Holland J. H., Adaptation in natural and artificial systems. Ann Arbor: The University of Michigan Press, 1975. Holland, 1975; 1980; 1986;

[4] Stone M., Cross Validatory Choice and Assessment of Statistical Predictions En Journal of the Royal Statistical Society *B*, Vol. 36, No. 1. (1974), pp. 111-147.

[5] Tanenbaum, Andrew S, Sistemas Operativos Distribuidos, México: Prentice Hall Hispanoamericana. 1996.

[6] Flynn Michael J, Computer Architecture: Pipelined and Parallel Processor Design. Jones and Bartlett, Boston: 1995

[7] Bernal C. Iván, Mejía N. David y Fernández A. Diego, Computación de Alto Rendimiento con Cluster's de PC's, Escuela Politécnica Nacional, Quito, Ecuador, 2005 [en línea]
<http://clusterfie.epn.edu.ec/clusters/Publicaciones/HTML/articulo1.htm>

[8] MPI: A message passing interface standard [en línea] <http://www.mpi-forum.org>

[9] PVM: Parallel Virtual Machine [en línea] < http://www.csm.ornl.gov/pvm/>

[10] R. Bianchini, C. M. Brown, Parallel Genetic Algorithms on Distributed-Memory Architectures. Technical Report 436, University of Rochester. Computer Science Department, 1993

[11] Foster, Ian, Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering, Addison-Wesley Longman Publishing Co. Boston:MA, USA ,1995

[12] Comunidad Universitaria de Software Libre (CUSOL): Software Libre [en

línea<http://www.linux.com.co/cusol/index.php?title%C2%BFQue_es_el_Software_Libre%3F>

[13] Molina, Daniel Partial Least Squares (PLS) Correlation between refined product yields and physicochemical properties with the 1H nuclear magnetic resonance (NMR) spectra of Colombian crude oils, Colombia, Santander: Universidad Industrial de Santander, 2007

ANEXOS

A. MONTAJE CLUSTER

Para el montaje del Cluster se instalan los siguientes paquetes, ya que la distribución GNU/Fedora los trae por defecto por tanto utilizamos los siguientes comandos

```
apt-get install build-essential
apt-get install ssh
```

La red debe estar configurada en todos los nodos con IP's estáticas y se añade al archivo /etc/hosts las ip's de los nodos con el alias con el que queremos identificar estos nodos.

```
192.168.109.10 nodo01          %nombre del servidor
192.168.109.11 nodo02          %nombre de los nodos
192.168.109.12 nodo03
192.168.109.13 nodo04
192.168.109.14 nodo05
192.168.109.15 nodo06
192.168.109.16 nodo07
```

NOTA: Todos los nodos que se van a realizar el cluster deben tener la misma información anterior para evitar errores. Luego creamos las llaves publicas como usuario del sistema.

Este comando crea las llaves públicas

```
ssh-keygen -t rsa
```

Se pasan las llaves al maestro con el siguiente comando

```
scp /home/cluster/.ssh/id_rsa.pub cluster@nodo01:/home/cluster/$HOSTNAME
```

Después el servidor concatena los nodos de la siguiente manera

```
cat nombre_del_nodo >> /home/cluster/.ssh/authorized_keys
```

Así con todos los nodos que existen y al final concatena el servidor

```
cat .ssh/id_rsa.pub >> /home/cluster/.ssh/authorized_keys
```

Por último el servidor procede a pasar la authorized_keys a cada nodo así:

```
scp /home/cluster/.ssh/authorized_keys cluster@nodo1:/home/cluster/.ssh/
```

Una vez realizada la creación y traspaso de llaves públicas, se puede decir que nos podemos comunicar entre todos los nodos sin ningún problema de acceso por clave, o contraseñas de permiso.

Ya teniendo el cluster montado se procede a la instalación de OpenMPI.

B. INSTALACIÓN DE OPENMPI Y MONTAJE DE GLUSTERFS

Teniendo el Sistema Operativo instalado en los nodos que están dispuestos para el montaje del Cluster, nos dirijimos a la carpeta personal.

```
cd /home/cluster
```

Se instala los paquetes necesarios para iniciar la instalación (octave, gcc, gcc-gfortran. Octave-devel)

```
yum install octave gcc gcc-gfortran octave-devel
```

Se instala el toolbox nnet de Octave desde el prompt de Octave

```
pkg install package-filename.tar.gz
```

package-filename es el nombre del archivo descargado de la página de Octave-Forge (http://octave.sourceforge.net/)
Se descarga y se instala los paquetes de openmpi como interfaz de paso de mensajes.

```
wget -t 0 http://www.open-mpi.org/software/ompi/v1.3/downloads/openmpi-1.3.tar.gz
```

```
tar -xvzf openmpi-1.3.tar.gz
```

```
mkdir /home/cluster/openmpi
```

```
cd openmpi-1.3
```

Se instala openmpi con librerías compartidas para el correcto funcionamiento de mpitb.

```
./configure --prefix=/home/cluster/openmpi/ --enable-shared --with-modules --with-trillium --with-rsh=ssh --with-fc=gfortran –disable-static
```

```
make
```

```
make install
```

Se crea un link virtual de las librerías de openmpi hacia las librerías del sistema.

```
ln -s /home/cluster/openmpi/lib/* /lib/
```

Se agrega la PATH del sistema las direcciones necesarias para manejar las

librerías de OpenMPI.

```
set variable0="PATH=/home/cluster/openmpi/bin:$PATH"
set variable1="export PATH"
set variable2="PATH=/home/cluster/openmpi/bin:$PATH"
set variable3="export PATH"
echo $variable0 >> /home/cluster/.bashrc
echo $variable1 >> /home/cluster/.bashrc
echo $variable2 >> /home/cluster/.bashrc
echo $variable3 >> /home/cluster/.bashrc
```

Se descarga los paquetes, se instala y se crean los links virtuales

```
wget -t 0
http://atc.ugr.es/%7Ejavier/investigacion/mpitb/mpitb-beta-
FC6-OCT2912-LAM713-OMPI123.tar.bz2
tar -xvf mpitb-beta-FC6-OCT2912-LAM713-OMPI123.tar.bz2
cd /home/cluster/octave/mpitb/src
make
ln -s /home/cluster/mpitb/startups-2.9.12/startup_MPITB.m
/home/cluster/.octaverc
```

Como super-usuario se realiza la instalación del sistema de archivos compartidos. Se instala las librerías.

```
yum install libibverbs
```

En el nodo que se utilizará como el servidor del sistema de archivo, se descarga el paquete de glusterfs para el servidor.

```
wget -t 0
ftp://ftp.univie.ac.at/systems/linux/fedora/updates/8/i386.ne
wkey/glusterfs-server-1.3.8-0.7.fc8.i386.rpm
```

Se instalan estos paquetes descargados en el nodo maestro.

```
rpm -ivh glusterfs-server*
```

Para los nodos esclavos se descarga el paquete de glusterfs y respectivas librerías para clientes.

```
wget -t 0
```

ftp://ftp.univie.ac.at/systems/linux/fedora/updates/8/i386.newkey/glusterfs-client-1.3.8-0.7.fc8.i386.rpm

```
wget -t 0
ftp://ftp.univie.ac.at/systems/linux/fedora/updates/8/i386.newkey/glusterfs-libs-1.3.8-0.7.fc8.i386.rpm
```

Se instalan estos paquetes descargados en los nodos clientes.

```
rpm -ivh glusterfs-libs*
rpm -ivh glusterfs-client*
```

Se crea el archivo en el directorio /et c/glusterfs/ en el nodo servidor llamado glusterfs-server.vol

```
toutch /etc/glusterfs/glusterfs-server.vol
```

Se agrega al archivo la siguiente configuración.

```
echo "type storage/posix" >> /etc/glusterfs/glusterfs-
server.vol
echo "option directory /tmp/export" >>
/etc/glusterfs/glusterfs-server.vol
echo "end-volume" >> /etc/glusterfs/glusterfs-server.vol
```

```
echo "volume server" >> /etc/glusterfs/glusterfs-server.vol
echo "type protocol/server" >> /etc/glusterfs/glusterfs-
server.vol
echo "subvolumes brick" >> /etc/glusterfs/glusterfs-
server.vol
echo "option transport-type tcp/server # For TCP/IP
transport" >> /etc/glusterfs/glusterfs-server.vol
echo "option auth.ip.brick.allow *" >>
/etc/glusterfs/glusterfs-server.vol
echo "end-volume" >> /etc/glusterfs/glusterfs-server.vol
```

Se crea en los nodos clientes el archivo glusterfs-server.vol con la siguiente configuración.

```
toutch /etc/glusterfs/glusterfs-server.vol

echo "volume client" >> /etc/glusterfs/glusterfs-server.vol
echo "type protocol/client" >> /etc/glusterfs/glusterfs-
server.vol
echo "option transport-type tcp/client" >>
/etc/glusterfs/glusterfs-server.vol
echo "option remote-host 192.168.109.PONGA_LA_IP_MANO" >>
/etc/glusterfs/glusterfs-server.vol
echo "option remote-subvolume brick" >>
/etc/glusterfs/glusterfs-server.vol
echo "end-volume" >> /etc/glusterfs/glusterfs-server.vol
```

En el nodo maestro se añade los siguientes parámetros a los archivos de configuración.

```
echo "/usr/sbin/glusterfsd -f /etc/glusterfs/glusterfs-
server.vol" >> /etc/rc.local
```

```
echo "/sbin/iptables -F" >> /etc/rc.local
echo "/sbin/iptables -X" >> /etc/rc.local
echo "/sbin/modprobe fuse" >> /etc/rc.local
echo "/usr/sbin/glusterfs -f /etc/glusterfs/glusterfs-
client.vol /mnt/gluster" >> /etc/rc.local
```

Para los demás nodos se le agrega la siguiente configuración.

```
echo "/sbin/iptables -F" >> /etc/rc.local
echo "/sbin/iptables -X" >> /etc/rc.local
echo "/sbin/modprobe fuse" >> /etc/rc.local
echo   "/usr/sbin/glusterfs   -f   /etc/glusterfs/glusterfs-
client.vol /mnt/gluster" >> /etc/rc.local
```

y por último se reinicia el sistema.

```
Reboot
```

Al abrir octave ya se cuenta con las herramienta de mpi

C. INSTALACIÓN DE SISTEMA DE MONITOREO GANGLIA

Ganglia usa gmond en cada computador monitoreado. Cada proceso gmond debe informarle a otros procesos similares las estadísticas mediante una de dos formas:

- Enviando mensajes UDP a un computador central (o a un grupo de computadores) que ejecutan gmond
- Enviando mensajes multicast UDP

Un proceso gmond puede escuchar mensajes UDP multicast en la red. También puede escuchar mensajes unicast que le sean enviados directamente.

La instalación de Gmond, luego de haberlo descargado de la pagina principal del proyecto en cada nodo se realiza de la siguiente manera:

```
tar xzvf ../ganglia-3.0.X.tar.gz && cd ganglia-3.0.X
```

```
./configure
make
make install
```

La configuración generada automáticamente para gmond usa multicast.
Gmetad agrupa la información de todos los nodos. Antes de instalar este programa, se instalaron los programas y librerías que vienen con rrdtool. En Debian:

```
aptitude install librrd2-dev rrdtool
```

Ahora la compilación y configuración de gmetad.

```
tar xzvf ../ganglia-3.0.X.tar.gz && cd ganglia-3.0.X
./configure --with-gmetad
make
make install
```

Se necesita un archivo de configuración /etc/gmetad.conf. Este lo usamos en la prueba:

```
data_source "NeuralCluster" nodo01 nodo02 nodo03 nodo04 nodo05
setuid_username "ganglia"
```

nodo01 - nodo05 significa que si nodo01 no puede ser alcanzado por red, gmetad usará nodo05 para obtener información. Este es un **or** lógico y no un **and**.
En la instalación fue creado un usuario del sistema ganglia, así que no ejecutaremos el servicio como nobody. El servicio no arrancará a menos que exista un directorio /var/lib/ganglia/rrds/ propiedad del uid que corresponde al usuario ganglia.
Se inicializa el servicio de ganglia.

```
/etc/init.d/gmetad start
```

Y de esta manera en nuestro servidor LAMP ya tenemos el sistema de monitoreo Ganglia.