UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD No. 6

"Algoritmo para la distribución de Perceptrones Multicapa."

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autores: Isbel Ochoa Izquierdo.
José Carlos Pardo Cruz.

Tutores: Dr. Ramón Carrasco Velar.
Ing. Yuleidys Mejías Cesar.

Junio, 2009

Si nuestro cerebro fuera tan simple que lo pudiéramos entender,
nosotros seríamos tan simples que no lo podríamos entender.
Poppe

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter no exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Isbel Ochoa Izquierdo

_____

Firma del Autor

José Carlos Pardo Cruz

_____

Firma del Autor

Dr. Ramón Carrasco Velar

_____

Firma del Tutor

Ing. Yuleidys Mejías Cesar

_____

Firma del Tutor

DATOS DE CONTACTO

Tutores:

Dr. Ramón Carrasco Velar.

Universidad de las Ciencias Informáticas, Habana, Cuba.

rcarrasco@uci.cu

Ing. Yuleidys Mejías Cesar.

Universidad de las Ciencias Informáticas, Habana, Cuba.

ymejias@uci.cu

AGRADECIMIENTOS

Queremos agradecer primero que todo a la UCI por darnos la oportunidad única de desarrollarnos como profesionales y seres humanos. A nuestros padres por el esfuerzo realizado y por la confianza depositada en nosotros. A todos nuestros profesores por nutrirnos de los conocimientos que nos han permitido llegar a ser lo hoy somos. A Yendri, Julio, Carrasco, Yuli, Andy, Valenzuela, Tortosa, Cesar, Daniel, Téllez, Maikel por tendernos sus manos en los momentos más oportunos. A Isidro por brindarnos el soporte tecnológico que fue decisivo en la investigación. A todos nuestros amigos y en general a todas esas personas que de alguna forma u otra influyeron en el éxito alcanzado con este trabajo.

DEDICATORIA

De José Carlos:

Dedico este trabajo a mi mamá por sobre todas las cosas por ser mi motor impulsor, mi amiga, mi todo, ejemplo vivo de sacrificio y perseverancia y a mi papá porque aunque se haya ido, siempre ha estado junto a mí, diera todo porque me pudiera ver hoy, a ustedes les regalo este triunfo. A mis hermanos de sangre por siempre poder contar con ellos, especialmente Juli por ser mi punto de apoyo en los momentos importantes. A Yilian por ser mi compañera, mi amante, mi amiga y darme la posibilidad de contar con otra maravillosa familia, los quiero mucho a todos. A mis hermanos de la UCI por estar siempre, ellos saben quiénes son. A mis compañeros de apartamento por todo y principalmente por los juegos de dominó. A todos muchas gracias por ayudarme a cumplir mis sueños y por ser, de una forma u otra, parte de mi vida durante este maravilloso y corto tiempo.

De Isbel:

Dedico este trabajo a mi familia. A mis padres, gracias a su exigencia y enseñanzas he podido convertirme en un hombre de bien, los quiero y los admiro, desde pequeño han sido mi paradigma, mi guía, mi luz. A mi hermano Abel, a pesar de las veces que peleamos cuando niños, me alegra mucho que hoy tengamos tanto en común, confío en ti. A mis tías y tíos, sé que siempre podré contar con ustedes. A mis primos, los gordos, los flacos, los deportistas, los artistas, ustedes están presentes en cada momento. A Alicia, por ser una fuente inagotable de optimismo, te mereces todo la felicidad del mundo. A Iliana y a Idania, por confiar en mí y levantarme cuando ya bajaba la guardia frente a los golpes de la vida. A todas las buenas personas que he tenido el gusto conocer en estos cinco años de carrera y que hoy, indiscutiblemente, forman parte de mí, nunca los olvidaré. A los que me demostraron que la vida no es siempre color de rosa y que hasta el sol, fuente de infinita luz, tiene manchas oscuras, tardé mucho en entenderlo. A todos y a cada uno de ustedes mis más sinceros agradecimientos, por transformar la arcilla de mi personalidad en esta escultura única e indestructible.

RESUMEN

Dentro de la gama de redes de neuronas artificiales (RNA) que existen, el perceptrón multicapa (PMC) es uno de los que mejores resultados ha aportado en los estudios de relación estructura-actividad. Se conoce que los volúmenes de datos previstos para procesar, son definitivamente grandes por lo que se propuso evaluar algoritmos y aplicaciones para acortar el tiempo del entrenamiento de la red ya que este proceso de aprendizaje de la red puede ser computacionalmente poco rentable. Se evaluaron diferentes herramientas que trabajan con las RNA de las cuales se seleccionó el Weka para extraer el algoritmo de la red y la Plataforma de Tareas Distribuidas Tarenal para distribuir el entrenamiento del perceptrón multicapa. Se demostró que para una muestra de 60 000 compuestos con 576 descriptores, el tiempo que demora realizar un entrenamiento local dura aproximadamente 39 horas con 24 minutos, el cual se reduce cuando se trabaja con el algoritmo distribuido propuesto a un límite de 2 horas con 25 minutos, manteniendo aceptable la calidad del perceptrón obtenido.

PALABRAS CLAVES

Algoritmo distribuido, perceptrón multicapa, Plataforma de Tareas Distribuidas, redes neuronales artificiales.

TABLA DE CONTENIDO

INTRODUCCIÓN

La predicción de la actividad biológica de compuestos orgánicos es hoy día un objetivo fundamental dentro de la Industria Médico Farmacéutica Mundial. El alto costo del proceso de investigación - desarrollo de nuevos fármacos, ha obligado a este sector económico a adoptar la estrategia del uso de técnicas de la computación y la informática para acelerar el proceso y disminuir los costos. Esta necesidad, junto con los avances en el sector de la bio-orgánica, los extraordinarios progresos de la fisiología, la bioquímica, la medicina y las técnicas de computación han promovido una revolución en el ámbito del diseño y producción de fármacos. En los últimos años, la industria farmacéutica ha reorientado sus investigaciones y prestado más atención a aquellos métodos que permitan una selección racional o el diseño de nuevos compuestos con propiedades deseadas. [3]

El proceso de predicción de actividad biológica de compuestos orgánicos utilizando técnicas de programación como las Máquinas de Soporte Vectorial, Redes Neuronales Artificiales, requiere una capacidad de cómputo muy elevada por lo que se hace necesario utilizar una supercomputadora o varios computadores interconectados. Económicamente es mucho más factible usar las computadoras personales con las que se cuente, pues se aprovecha mucho más las potencialidades de una red de computadoras incluso no necesariamente tienen que estar físicamente en el mismo sitio, pero para esto se hace necesario el desarrollo e implementación de algoritmos y técnicas para distribución de estos cálculos entre los ordenadores.

El presente trabajo es una investigación que surge en el proyecto de la Facultad 6 de la Universidad de las Ciencias Informáticas denominado “Plataforma Inteligente para la Predicción de Actividad Biológica de Compuestos Orgánicos”, donde se cuenta con varias técnicas para predecir la actividad biológica, tales como: Máquinas de Soporte Vectorial, Lógica Difusa y Algoritmos Genéticos. Por la necesidad de enriquecer las potencialidades de la plataforma se realiza un estudio sobre las redes neuronales artificiales y específicamente el perceptrón multicapa.

Las redes neuronales artificiales (RNA) por el alto grado de paralelismo que ofrece es una gran candidata, para ser usada con estos fines, y específicamente dentro de las RNA los perceptrones multicapa (PMC). En el mundo se han desarrollado varias herramientas relacionadas con la presente investigación en cuanto a las redes neuronales artificiales, ejemplo de esto es la aplicación Joone (Java Object Oriented Neural Engine por sus siglas en inglés), una plataforma de software libre capaz de crear, entrenar y probar redes neuronales artificiales. El Javanns (Java Neuronal Network Simulator) es un eficiente simulador universal de redes neuronales artificiales. Debido a las

limitaciones del perceptrón multicapa antes expuestas el **problema científico** de la presente investigación es:

¿Cómo reducir el tiempo de entrenamiento del perceptrón multicapa con grandes volúmenes de datos?

Se plantea como **Objetivo de la investigación:** Proponer un algoritmo para reducir el tiempo de entrenamiento del perceptrón multicapa, útil para el diseño racional de fármacos, teniendo como **Objetivos Específicos** para dar cumplimiento al objetivo de la investigación**:**

- Evaluar diferentes arquitecturas de perceptrón multicapa.
- Diseñar algoritmo para la distribución de un perceptrón multicapa (MLP) para la predicción de actividad biológica.
- Validar el algoritmo diseñado.

Para lograr cumplir satisfactoriamente todos los objetivos se trazaron varias **Tareas:**

- El estudio el funcionamiento de las Redes Neuronales Artificiales y específicamente el Perceptrón Multicapa.
- El estudio las posibles funciones de transferencias.
- Selección de la función de transferencia que permita obtener los resultados de manera óptima.
- Realización pruebas con diferentes topologías para escoger la más adecuada para la predicción de la actividad biológica.
- El entrenamiento de la red neuronal con una colección de datos amplia.
- Pruebas de la red neuronal con otra colección de datos y evaluar los resultados obtenidos.
- El estudio del funcionamiento de los Sistemas Distribuidos.
- Implementación de un algoritmo para la distribución de manera eficiente del perceptrón multicapa.
- Realización de pruebas de dicho algoritmo para validarlo.

Después de concluido este trabajo se podrá contar con los siguientes **resultados**:

- Modelo de red neuronal que permita predecir de manera eficiente la actividad biológica de compuestos orgánicos.
- Algoritmo para la distribución de perceptrones multicapa basado en la plataforma de tareas distribuidas Tarenal.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

La bioinformática es una ciencia que surge a partir de la necesidad en la biología de agilizar los procesos de investigación y análisis, además de procesar grandes volúmenes de datos aprovechando la capacidad de cómputo que brinda la informática.

Un área sumamente interesante dentro del modelado molecular es el diseño de nuevos compuestos, para lo cual es de vital importancia conocer su actividad biológica. Se ha demostrado que las relaciones entre la estructura molecular y las propiedades físico químicas o la actividad biológica de los compuestos se pueden cuantificar matemáticamente a partir de parámetros estructurales simples (descriptores). Para estos fines se usan comúnmente diferentes técnicas de inteligencia artificial.

Las redes neuronales artificiales junto a los Sistemas Distribuidos constituyen una potente alternativa para el desarrollo de algoritmos predictivos.

Este capítulo es el resultado de un estudio realizado sobre las Redes Neuronales Artificiales y los Sistemas Distribuidos, que ayuda a comprender los posteriores capítulos, pues el modelo que se propone para predecir actividad biológica es un modelo neuronal que usa técnicas distributivas para su ejecución.

1.1 Redes Neuronales Artificiales.

Las Redes Neuronales Artificiales son métodos de inteligencia artificial que se basan en la analogía que existe en el comportamiento y función del cerebro humano, el cual está compuesto por redes de neuronas biológicas que poseen bajas capacidades de procesamiento, sin embargo toda su capacidad cognitiva se sustenta en la conectividad de éstas.

La primera red neuronal conocida, fue desarrollada en 1943 por Warren McCoulloch y Walter Pitts;

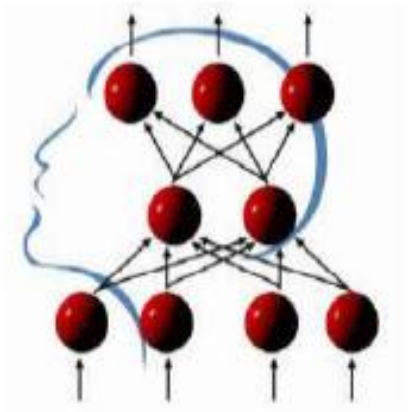


Figura 1. Las redes neuronales artificiales simulan el funcionamiento del cerebro humano.

esta consistía en una suma de las señales de entrada, multiplicadas por unos valores de pesos escogidos aleatoriamente. La entrada es comparada con un patrón preestablecido para determinar la salida de la red. Si en la comparación, la suma de las entradas multiplicadas por los pesos es mayor o igual que el patrón preestablecido la salida de la red es uno (1), en caso contrario la salida es cero (0). Al inicio del desarrollo de los sistemas de inteligencia artificial, se creyó que este modelo podía computar cualquier función aritmética o lógica. [11]

Elementos de la neurona U_j:

Valor o estado de activación a_j (t): Valor numérico que caracteriza a la neurona.
Función de transferencia f_j que transforma el estado actual de activación en una **señal de salida y_j**.
Estas señales se modifican de acuerdo a los **pesos sinápticos w** (Son las conexiones que unen a las neuronas, son los que hacen que la RNA adquiera conocimiento) asociados a cada canal.
Las entradas modulares que entran a la **neurona j** se combinan entre ellas generando la entrada total.

$$Xj = \sum_{i} yi\ wij$$

Una **regla de activación** determina el nuevo estado de activación $\mathbf{a_j}$ (t+1) de la neurona, teniendo en cuenta la entrada total calculada y el anterior estado de activación $\mathbf{a_j}$. [12]

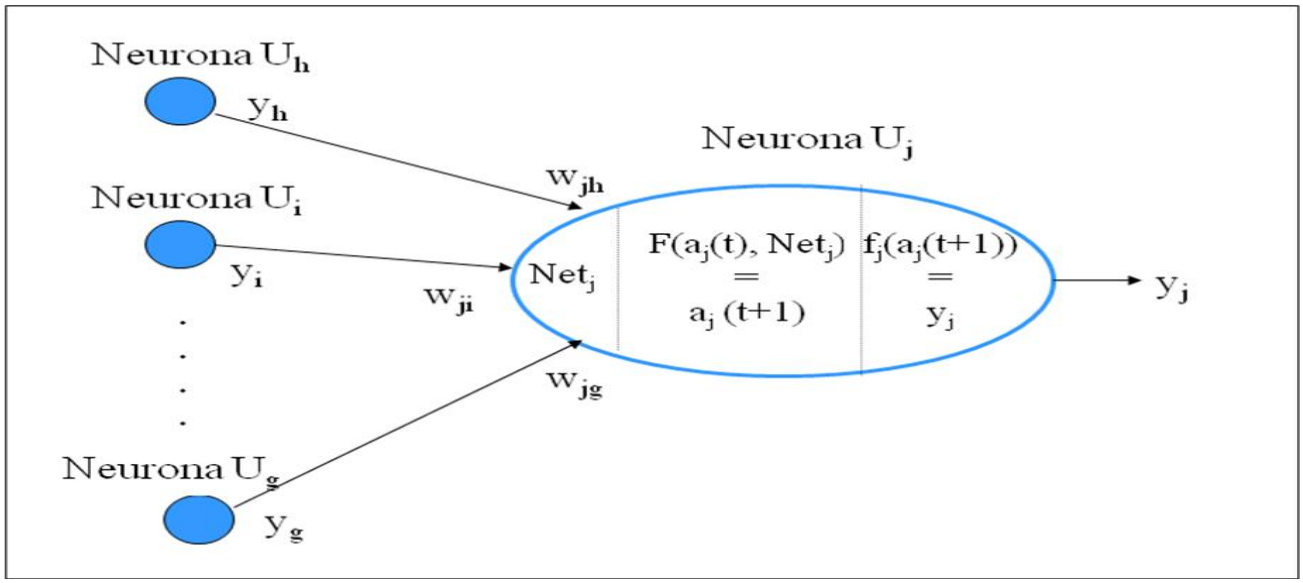


Figura 2. Estructura de una neurona artificial.

1.1.1 Características de las RNA.

Por lo tanto se puede señalar que una RNA es un sistema de computación distribuida caracterizada por:

- Un conjunto de unidades elementales, cada una de las cuales posee bajas capacidades de procesamiento.
- Una densa estructura interconectada usando enlaces ponderados.
- Pesos que deben ser ajustados para satisfacer los requerimientos de desempeño.
- Un alto grado de paralelismo.

Estas características hacen que ofrezcan numerosas ventajas y que este tipo de tecnología se esté aplicando a múltiples áreas. [12]

1.1.2 Ventajas de las RNA.

Aprendizaje adaptativo: Capacidad de aprender a realizar tareas basadas en un entrenamiento o en una experiencia inicial.
Auto-organización: Una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje.
Tolerancia a fallos: La destrucción parcial de una red conduce a una degradación de su estructura; sin embargo, algunas capacidades de la red se pueden retener, incluso sufriendo un gran daño.
Paralelismo: Los computadores neuronales pueden ser procesados en paralelo. [11]

1.1.3 Desventajas de las RNA.

Existen también algunos aspectos desventajosos que hay que considerar al hacer uso de las RNA. Entre sus desventajas están:
Elevada duración de los tiempos de entrenamiento.
El algoritmo de gradiente descendente puede alcanzar mínimos locales (es decir un mínimo en la función que relaciona los pesos con el error) más que un mínimo global.
El algoritmo de gradiente descendente también puede oscilar y nunca alcanzar.
Los valores de pesos que se aplican inicialmente y que suelen ser valores pequeños y aleatorios influyen en los resultados. [6]

1.1.4 Tipos de Aprendizajes.

Es importante señalar que la propiedad más importante de las redes neuronales artificiales es su capacidad de aprender a partir de un conjunto de patrones de entrenamientos, es decir, es capaz de encontrar un modelo que ajuste los datos. En el proceso de aprendizaje también conocido como entrenamiento de la red se dice que la red ha "aprendido" cuando los valores de los pesos se mantienen estables ***dwij/dt = 0***, el entrenamiento puede ser supervisado o no supervisado.

Aprendizaje supervisado.

Consiste en entrenar la red a partir de un conjunto de datos o patrones de entrenamiento compuesto por patrones de entrada y salida. El objetivo del algoritmo de aprendizaje es ajustar los pesos de la red (w) de manera tal que la salida generada por la RNA sea lo más cercanamente posible a la verdadera salida dada una cierta entrada. Es decir, la red neuronal trata de encontrar un modelo al proceso desconocido que generó la salida. Este aprendizaje se llama supervisado pues se conoce el patrón de salida el cual hace el papel de supervisor de la red.

Aprendizaje no supervisado.

En cambio en el aprendizaje no supervisado se presenta sólo un conjunto de patrones a la RNA, y el objetivo del algoritmo de aprendizaje es ajustar los pesos de la red de manera tal que la red encuentre alguna estructura o configuración presente en los datos.

1.1.5 Función de activación.

Esta función representa simultáneamente el estado de activación de la neurona en función de la entrada total y la salida de la neurona.

Tabla 1. Diferentes funciones de activación de las neuronas artificiales.

Nombre	**Relación Entrada/Salida**	**Icono**	**Función**
Lineal	**a = *n***		purelin
Lineal positiva	**a = 0, *n* < 0** **a = *n*, *n* ≤ 0**		poslin
Lineal Saturado	**a = 0, *n* < 0** **a = 0, 0 ≤ *n* ≤ 1** **a = 1, *n* > 1**		satlin
Lineal Saturado Simétrico	**a = -1, *n* < -1** **a = *n*, -1 ≤ *n* ≤ 1** **a = 1, *n* > 1**		satlins
Limitador Fuerte	**a = 0, *n* < 0** **a = 1, *n* ≥ 0**		hardlim
Limitador Fuerte Simétrico	**a = -1, *n* < 0** **a = 1, *n* ≥ 0**		hardlims
Sigmoidal Logarítmico	$a = \frac{1}{1 + e^{-n}}$		logsig
Tangente Sigmoidal Hiperbólica	$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$		tansig
Competitiva	**a = 1**, neurona con *n* Máx. **a = 0**, El resto de las neuronas		competa

La función de salida convierte el estado de la neurona en la salida hacia la siguiente neurona o a las siguientes neuronas de la próxima capa mediante los pesos sinápticos. Usualmente no se considera y se toma como salida el propio estado de activación de la neurona.

1.2 Clasificación de las Redes Neuronales Artificiales.

Se pueden clasificar las RNA teniendo en cuenta aspectos tales como: la transmisión de datos, número de conexiones, la topología, entre otros. [11]

1.2.1 Transmisión de datos.

Redes *feedforward*, o con conexiones hacia adelante: Este tipo de redes contienen solo conexiones entre capas hacia delante. Esto implica que una capa no puede tener conexiones a una que reciba la señal antes que ella.

Redes *feedback*, o con conexiones hacia atrás: Aparte del orden normal algunas capas están también unidas desde la salida hasta la entrada en el orden inverso en que viajan las señales de información. Este tipo de redes se diferencia de las anteriores en que si pueden existir conexiones de capas hacia atrás y por tanto la información puede regresar a capas anteriores en la dinámica de la red.

Redes *feedlateral* o con conexiones laterales: Son conexiones entre neuronas de la misma capa, este tipo de conexión son muy comunes en las redes mono capa*.*

1.2.2 Número de conexiones.

Redes neuronales totalmente conectadas: En este caso todas las neuronas de una capa se encuentran conectadas con las de la capa siguiente (redes no recurrentes) o con las de la anterior (redes recurrentes).

Redes neuronales parcialmente conectadas: En este caso no se da la conexión total entre neuronas de diferentes capas.

1.2.3 Topología.

Si se clasifica a las redes desde un punto de vista **topológico**, se puede hablar de redes monocapa y las redes multicapas.

Redes monocapa: Una red formada por sucesivas capas tendrá como forma más simple una capa de nodos de entrada y otra de nodos de salida, donde x_i representa el vector de entrada k, U_j a las unidades intermedias, W_{jk} las conexiones o pesos que relacionan a la neurona i de la capa de entrada con la neurona j de la capa intermedia.

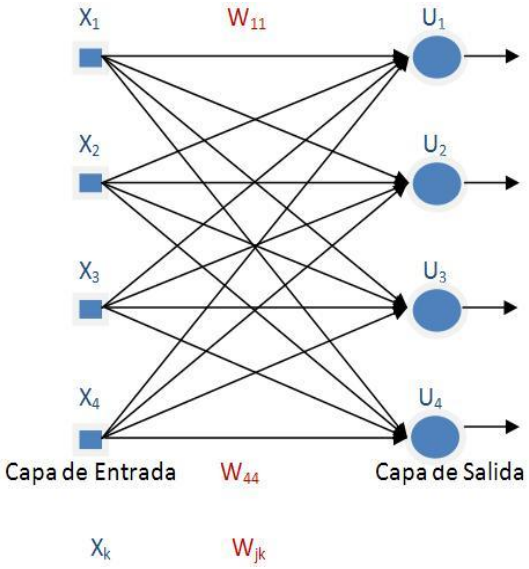


Figura 3. Red neuronal artificial mono capa.

Redes Multicapa: Una red multicapa consiste en una extensión de una red monocapa, con tantas capas ocultas como se desee. Existen dos variantes, totalmente conectadas, y las parcialmente conectadas.

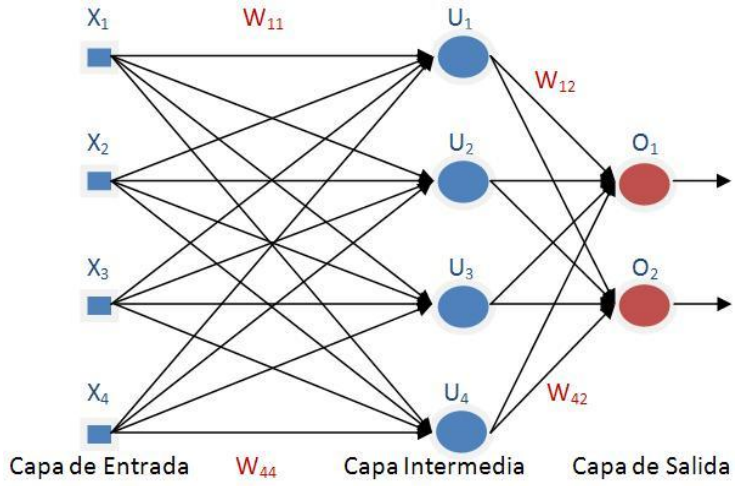


Figura 4. Red neuronal artificial multicapa.

Donde $\mathbf{X_i}$ representa el vector de entrada del patrón **K**, $\mathbf{U_j}$ a las unidades intermedias, $\mathbf{W_{jk}}$ las conexiones o pesos que relacionan a la neurona i de la capa de entrada con la neurona j de la capa intermedia y $\mathbf{W_{ij}}$ de la capa intermedia a las unidades de salida. El índice **i** siempre se referirá a la unidad de salida, **j** al de las unidades o neuronas ocultas, y **k** al nodo o neurona de entrada.

1.2.4 Tipo de Entrada.

Redes analógicas: procesan datos de entrada con valores continuos y, habitualmente, acotados. Ejemplos de este tipo de redes son: Hopfield, Kohonen y las redes de aprendizaje competitivo.

Redes discretas: procesan datos de entrada de naturaleza discreta; habitualmente valores lógicos booleanos. Ejemplos de este segundo tipo de redes son: las Máquinas de Bolzman y Cauchy, y la red discreta de Hopfield.

1.3 Perceptrón.

1.3.1 Perceptrón Simple.

La red tipo perceptrón fue inventada por el sicólogo Frank Rosenblatt en el año 1957. El primer modelo de perceptrón fue desarrollado en un ambiente biológico imitando el funcionamiento del ojo humano. El fotoperceptrón era un dispositivo que respondía a señales ópticas. El perceptrón es una red neuronal no lineal que consta de una combinación lineal entre las entradas y los pesos sinápticos, un valor de

ajuste externo (umbral o tendencia) y la función **signo** $sgn(x) = \begin{cases} -1; x < 0 \\ 1 \quad ; x \geq 0 \end{cases}$ como función de transferencia o activación. Inicialmente es un dispositivo de aprendizaje, en su configuración inicial no tiene la capacidad de distinguir patrones de entrada muy complejos, sin embargo mediante un proceso de aprendizaje puede adquirir esta capacidad.

El perceptrón está constituido por un conjunto de neuronas de entrada que reciben los patrones de entrada a reconocer o clasificar y una neurona de salida que se ocupa de clasificar a los patrones de entrada en dos clases, según que la salida de la misma sea 1 (activada) o 0 (desactivada). El perceptrón simple tiene una serie de limitaciones muy importantes. La más importante es su incapacidad para clasificar conjuntos que no son linealmente independientes. [10]

1.3.2 Perceptrón Multicapa.

Este modelo es una ampliación del perceptrón a la cual añade una serie de capas que, básicamente, hacen una transformación sobre las variables de entrada, que permiten eludir el problema anterior. Las capas presentan interconexión total.

Esto acaba con el problema del perceptrón, convirtiendo las funciones linealmente no independientes en linealmente independientes gracias a la transformación de la capa oculta. Su modo de propagación de los datos es hacia adelante (*feedforward*), es decir, que los patrones de entrada se presentan en la capa de entrada y se propagan hasta generar la salida. Las entradas y las salidas son continuas [0,1].

La función de activación de sus neuronas es sigmoidal. Los estados de las unidades de cada capa son determinados aplicando las ecuaciones **(1)** y **(2)** a las conexiones provenientes de las capas anteriores. La entrada total $\mathbf{X_j}$ a la unidad $\mathbf{U_j}$ es una función lineal de las salidas $\mathbf{Y_i}$ de las unidades que están conectadas a $\mathbf{U_j}$ y de los pesos $\mathbf{W_{ij}}$ sobre estas conexiones. Para cada unidad $\mathbf{U_j}$ existe una entrada extra con valor 1 y peso **W0j** que se trata como el resto.

$$X_j = \sum_i y_i * w_{ij} + b \qquad (1)$$

La salida de cada unidad de procesamiento, Y_j, se calcula usando una función no lineal de su entrada total (sigmoidal logarítmica).

$$Y_j = \frac{1}{1 + e^{-X_j}} \qquad (2)$$

Los perceptrones multicapa con aprendizaje *backpropagation* utilizan la regla Delta como forma de aprendizaje (Esta regla de aprendizaje, se fundamenta en la utilización del error entre la salida real y esperada de la red para modificar los pesos). Estas redes adaptan la regla Delta de tal forma, que se facilite el entrenamiento de todas las conexiones entre los distintos niveles de la red.
El perceptrón multicapa admite valores reales. Se puede decir que el perceptrón multicapa es un modelador de funciones universal. [5]

1.4 Método de aprendizaje *Backpropagation*.

Tiene como objetivo ajustar los pesos para reducir el error cuadrático medio de las salidas. Genera una señal de error a partir de la diferencia entre la salida actual de la red y la salida real. Los pesos (fuerzas sinápticas) se actualizan de forma proporcional al producto de la señal de error y la señal de entrada a la red, con lo cual se disminuye el error en dirección del gradiente (la dirección de cambio más rápido).
De forma simplificada, este algoritmo es utilizado para el entrenamiento de una red multicapa. La red se inicializa con una serie de pesos aleatorios y tras hacer pasar los patrones de entrada a través de la estructura de la red se compara la salida con el patrón que se desea que aprenda.
La diferencia entre entradas-salidas, estimación de error, se utiliza para derivar los errores de los pesos de las sucesivas capas de la red haciendo pasar estos valores retrógradamente a la dirección de activación de la red.
La técnica de entrenamiento *backpropagation* requiere el uso de neuronas cuya función de activación se continua, y por lo tanto, derivable.
Encontrar el error $\mathbf{E_j}$ para todas las unidades. Si $\mathbf{D_j}$ es el valor de la salida deseada y $\mathbf{Y_j}$ es el valor de salida real para la j-ésima unidad, los errores se calculan por:
Para las unidades de salida:

$$E_j = (Y_j - D_j) * Y_j * (1 - Y_j) \qquad (3)$$

Para las unidades en las capas ocultas:

$$E_j = Y_j * (1 - Y_j) * \sum_k z_k * w_{kj} \qquad (4)$$

Donde **k** recorre todas las unidades conectadas a la unidad **j** en la capa siguiente a la que se encuentra esta y $\mathbf{Z_k}$ representa los errores calculados para las salidas de estas unidades (que fueron calculadas usando las ecuaciones **(3)** o **(4)** en dependencia de si la capa en la que se encuentran las unidades indicadas por **k** es de salida u oculta respectivamente.

Existe una función matemática para realizar esta modificación de los pesos:

$$W_{ij}(n+1) = W_{ij}(n) + t * E_j * Y_i + h * (W_{ij}(n) - W_{ij}(n-1))$$

Donde **(n+1)**, **(n)** y **(n-1)** indican el nuevo peso, el presente y el anterior respectivamente.
t es un número real que denota la velocidad de aprendizaje.
h es una constante entre 0 y 1 la cual determina el efecto de los cambios de pesos previos sobre la dirección actual de movimiento en el espacio de los pesos (*momentum*). [5]

1.4.1 *Backpropagation* con *Momentum*.

Esta modificación está basada en la observación de la última sección de la gráfica del error medio cuadrático en el proceso de convergencia típico para una red *Backpropagation*; este proceso puede verse en la figura 5 en la cual se nota la caída brusca del error en la iteración para la cual alcanza convergencia.

Figura 5. Comportamiento típico del proceso de convergencia para una red *Backpropagation*.

Este comportamiento puede causar oscilaciones no deseadas, por lo que es conveniente suavizar esta sección de la gráfica incorporando un filtro al sistema.

Este algoritmo, hace que la convergencia sea estable e incluso más rápida, además permite utilizar una rata de aprendizaje alta. [8]

1.5 Sistemas distribuidos.

La idea fundamental de un Sistema Distribuido (SD) es que constituye una combinación de computadoras y sistemas de transmisión de mensajes bajo un solo punto de vista lógico a través del cual los elementos de cómputo resuelven tareas en forma colaborativa. El sistema constituye es capaz de procesar información debido a sus características esenciales:

El sistema consiste de una cantidad de computadoras cada una de las cuales tienen su propio almacenamiento, dispositivos periféricos y potencia computacional.
Todas las computadoras están adecuadamente interconectadas.
Una variedad de componentes que incluyen tanto plataformas de cómputo como las redes de interconexión que transportan mensajes entre ellas unificadas en un solo ambiente de procesamiento.
La transparencia, como resultado de la abstracción apropiada de los componentes del sistema.
Por medio del sistema operativo adecuado, las computadoras mantienen su capacidad de procesamiento de tareas localmente, mientras constituyen elementos colaborativos de procesamiento en el ambiente distribuido. [9]

1.5.1 Ventajas y desventajas de los Sistemas Distribuidos.

Ventajas:

Procesadores más poderosos y a menos costos.
Desarrollo de Estaciones con más capacidades
Las estaciones satisfacen las necesidades de los usuarios.
Uso de nuevas interfaces.

Avances en la Tecnología de Comunicaciones.
Disponibilidad de elementos de Comunicación.

Desarrollo de nuevas técnicas.

Compartición de Recursos.
Dispositivos (Hardware).
Programas (Software).

Eficiencia y Flexibilidad.
Respuesta Rápida.
Ejecución Concurrente de procesos (En varias computadoras).
Empleo de técnicas de procesamiento distribuido.

Disponibilidad y Confiabilidad.
Sistema poco propenso a fallas (Si un componente no afecta a la disponibilidad del sistema).
Mayores servicios que elevan la funcionalidad (Monitoreo, Telecontrol, Correo Eléctrico, etc.).

Crecimiento Modular.
Es inherente al crecimiento.
Inclusión rápida de nuevos recursos.
Los recursos actuales no se afectan. [1]

Desventajas:

- Requerimientos de mayores controles de procesamiento.
- Velocidad de propagación de información muy lenta en algunos casos.
- Servicios de recopilación de datos y servicios con posibilidades de fallas.
- Mayores controles de acceso y proceso.
- Administración más compleja.
- Otro problema potencial tiene que ver con las *redes de comunicaciones*, ya que se deben considerar problemas debidos a pérdidas de mensajes, saturación en el tráfico, expansión. [1]

1.5.2 Condiciones que un problema debe cumplir para ser distribuible.

Para lograr la distribución de un problema que requiera grandes prestaciones de cómputo hay que tener en cuenta una serie de características que deben tener todo problema para encontrarle una buena solución distribuida. La característica más importante que debe tener es la posibilidad de dividir

el problema en unidades más pequeñas que puedan ser procesadas de forma individual. En este sentido existen dos formas de dividir el problema: [2]

Particionado de los datos o Descomposición del Dominio. En este particionado, el centro de atención es la partición de los datos, de ser posible estos son separados en pequeñas partes de aproximadamente el mismo tamaño. Luego, se dividen los cálculos a ser realizados, asociándolos con los datos sobre los cuales actúan.

Particionado funcional o Descomposición Funcional. La descomposición funcional es una estrategia diferente para enfrentar los problemas. El punto principal en este esquema es el cómputo a realizar en lugar de los datos manipulados. Si uno tiene éxito dividiendo las funciones del programa en grupos disjuntos, luego puede proceder a estudiar los requerimientos de datos de estos grupos funcionales. Si se da el caso de que la partición de la data resultante es también disjunta se dice que la partición funcional es completa. En este caso se procede al revés que en el caso anterior. Aquí se divide primero el cómputo y luego se mira la posibilidad de descomponer los datos.

1.5.3 Funciones más importantes de un Sistema Distribuido.

Un SD posee las siguientes funciones básicas:

- **Comunicación interprocesos.** La intención es tener comunicación entre procesos sobre la red de conexión, usando para ello las facilidades instaladas. Generalmente implica el uso de un protocolo de transporte para establecer el enlace en el ámbito de protocolo de servicios.
- **Administración y asignación de recursos.** Contempla la asignación de recursos a usuarios, toma de decisión de en d.nde deberán ser ejecutadas las peticiones, creación e instalación de nuevos recursos en la red, soporte de replicación para procesos críticos, mecanismos de control de concurrencia y sincronización.
- **Administración de nombres.** Mecanismos de asignación y mantenimiento de los nombres de los recursos, localización de servidores/usuarios, mantenimiento de directorios, etc.
- **Reinicio luego de fallas.** Implementado en varias capas de operación a lo alto de la arquitectura del sistema.
- **Funciones de protección.** Especificación de los usuarios y sus derechos de acceso, mecanismos de autentificación, políticas de acceso y contra ataques externos.

Desde el punto de vista del usuario, algunos de los requisitos que debe cumplir un SD son:

- Proveer de ayudas para la solución de problemas.

- Minimización del costo de acceso a los recursos.
- Maximización y simplificación de las facilidades de comunicación con otros usuarios o programas.

1.5.4 Modelos de computación distribuida.

Existen varios modelos fundamentales que se usan para la creación de sistemas de cómputo distribuido, los cuales se clasifican en diferentes categorías: modelo de minicomputadora, modelo de *Workstation*, modelo *Workstation-server*, modelo de pool de procesadores e híbrido. [9]

Modelo de minicomputadora.

Es simplemente la extensión del modelo de equipo centralizado. Este esquema consiste de varias minicomputadoras conectadas por una red de comunicación. Cada minicomputadora tiene su propio grupo de usuarios quienes pueden tener acceso a otros recursos no presente en su sistema a través de la red de datos. Este modelo se usa cuando existe necesidad de compartir recursos de diferentes tipos a través de acceso remoto.

Modelo de *Workstation*.

Consiste en varias estaciones de trabajo interconectadas por una red, cada usuario se autentica en su computadora de inicio o "computadora *home*" y luego desde ahí puede enviar trabajos para ejecución. Cada estación tiene su propio disco duro, y su propio sistema de archivos. La implementación mostrada tiene la desventaja de que se puede desperdiciar tiempo de procesamiento si una o más computadoras no están haciendo uso de su CPU. Si el sistema determina que la estación del usuario no posee la suficiente capacidad de proceso para una tarea, transfiere el trabajo de forma automática hacia el equipo que tiene menos actividad en ese momento y por último se regresa la salida del trabajo a la estación del usuario.

Modelo *Workstation-Server*.

Posee varias microcomputadoras y varias estaciones de trabajo (algunas de las cuales pueden ser estaciones sin disco duro) comunicadas mediante red. Ahora que existe la potencialidad de tener estaciones sin disco, el sistema de archivos a usar por estas computadoras puede ser el de una

computadora completa o alguno compartido de las diferentes minicomputadoras del sistema. Algunas otras microcomputadoras se pueden usar para proveer otros tipos de servicios, como base de datos, impresión, etc. Estas máquinas cumplen ahora nuevas tareas especializadas para proveer y administrar acceso a los recursos, en la jerga común se les llama *servidores* (servers).

Modelo de *pool* de procesadores.

Este se basa en el hecho de que los usuarios en promedio no requieren capacidad de procesamiento durante un buen rato, pero existen instantes en los que la actividad y los programas que ejecutan demandan potencia de trabajo en alto grado. A diferencia del modelo anterior en el que cada persona tiene su servidor asignado, en éste se dispone de un conjunto (*pool*) de servidores que son compartidos y asignados conforme a demanda. Cada procesador en el *pool* tiene su propia memoria y ejecuta un programa de sistema o de aplicación que le permite participar en el ambiente de cómputo distribuido.

El modelo no soporta la conexión de estaciones directamente a los servidores, sino sólo por medio de la red de interconexión. Las terminales usadas pueden ser estaciones sin disco o terminales gráficas como *X terminals.* Se tiene instalado un servidor especial llamado “de ejecución” el cual se dedica a la administración y asignación de recursos conforme a la demanda. Cuando se recibe una solicitud de una persona, este equipo asigna temporalmente el número de servers que deberán ser dedicados al trabajo de tal petición, y luego de atenderla regresan para ser liberados y quedar a disposición de la siguiente tarea. Otra diferencia importante con otros esquemas, es que aquí no existe el concepto de “computadora home”, de manera que el usuario no se da de alta en alguna máquina en particular, pero s. percibe al sistema como un todo.

Comparado con el modelo de *Workstation-Server*, el de *Pool* de *Servers* permite una mejor utilización del poder de cómputo disponible en el ambiente distribuido, dado que dicho poder computacional está disponible para todos a diferencia de *Workstation-Server*, donde varios equipos pueden estar desocupados en algún momento pero su capacidad no se puede asignar a otros. Por otra parte, esta metodología provee gran flexibilidad ya que el sistema se puede expandir agregando procesadores que operarán como servidores adicionales para soportar una carga extra de trabajo originada por incremento en el número de usuarios o por la implantación de nuevos servicios.

Modelo híbrido.

De los cuatro modelos descritos anteriormente, el de Workstation-Server es el más ampliamente utilizado para la construcción de sistemas de cómputo distribuido. Esto se debe a que un usuario promedio solamente ejecuta tareas interactivas simples como editar archivos, enviar correos electrónicos, etc. El modelo se ajusta perfectamente a especificaciones tan sencillas. Para ambientes donde hay grupos de usuarios de mayor potencia que realizan tareas masivas con alta necesidad de poder computacional el modelo Pool de Servers es más adecuado.

El modelo híbrido permite combinar las mejores características de Workstation-Server y Pool, esencialmente agregando a la red de estaciones de trabajo un Pool de servidores que pueden ser asignados dinámicamente para trabajos que son muy extensos para máquinas individuales o que necesitan varios equipos concurrentemente para una ejecución adecuada. Esta variante tiene la ventaja de garantizar el nivel de respuesta a trabajos interactivos dado que permite la ejecución en la misma computadora de la persona que lo solicita. Por otra parte, su principal desventaja estriba en que el costo de implantación se eleva puesto que requiere mayor número de componentes para su construcción.

1.6 Conclusiones.

En este capítulo se realizó una breve introducción a las Redes Neuronales Artificiales presentando sus principales características, ventajas, arquitectura, así como sus clasificaciones en diferentes aspectos. Se hizo énfasis en el perceptrón multicapa como el tipo de RNA escogida para realizar la predicción de actividad biológica de compuestos orgánicos. Se presentó también información referente a los Sistemas Distribuidos en cuanto a su concepto, características básicas, funciones principales y modelos de computación distribuida, específicamente el modelo *Workstation – Server* pues es el presente en la Plataforma de Tareas Distribuidas la cual se va a utilizar para distribuir el entrenamiento del perceptrón multicapa.

CAPÍTULO II: PROGRAMAS Y PROCEDIMIENTOS.

En este capítulo se hace referencia fundamentalmente al perceptrón multicapa que está implementado en el Weka y de Tarenal como plataforma para distribuir la fase de entrenamiento del perceptrón multicapa, así como las herramientas utilizadas para desarrollar la aplicación experimental.

2.1 Weka.

Weka (Waikato Environment for Knowledge Analysis) es una aplicación con potentes funcionalidades dedicadas a la minería de datos. Cuenta con un conjunto de bibliotecas Java para la extracción de conocimientos desde bases de datos. Es un software que fue desarrollado bajo licencia GPL lo cual ha impulsado que sea una de las *suites* más utilizadas en el área en los últimos años.

Es una implementación de esencialmente todas las técnicas de minería de datos como son: árbol de decisión, máquina de soporte vectorial, redes neuronales, clústeres. Incluye código ilustrativo e implementaciones de los métodos de aprendizaje. Ofrece claras implementaciones de las técnicas más simples diseñadas para ayudar a entender los mecanismos relacionados. También provee un banco de trabajo que incluye implementaciones funcionales en estado ideal de muchos esquemas de aprendizajes populares que pueden ser utilizados para minería de datos prácticas o investigaciones. Además contiene un *framework* que soporta la implementación de nuevos esquemas de aprendizaje.

La interfaz gráfica de usuario proporciona la carga de datos, la aplicación de algoritmos y la visualización de los modelos construidos. Una interfaz Java disponible es capaz de integrar todos los algoritmos que permite a cualquier usuario del programa utilizar sus potencialidades. [4]

2.2 Plataforma de Tareas Distribuidas (Tarenal).

La Plataforma de Tareas Distribuidas surgió como alternativa a los costosos clústeres y supercomputadores. Su objetivo principal es brindar una herramienta, que funcione como una "supercomputadora virtual", capaz de aunar y coordinar los esfuerzos entre los recursos disponibles y utilizar de esta forma, el poder computacional que ofrecen las computadoras personales generalmente de bajas prestaciones pero interconectados mediante una red local lo suficientemente rápida para entre todos lograr obtener resultados.

Ofrece una alternativa de cómputo que aglutina en un solo conglomerado a un clúster *Beowulf* y un conjunto de estaciones de trabajo. El sistema de cómputo desarrollado ha sido desplegado y utilizado

en la Facultad de Bioinformática de la Universidad de las Ciencias Informáticas y en centros del Polo Científico del Oeste de La Habana.

La Plataforma de Tareas Distribuidas está basada en el modelo Workstation – Server.

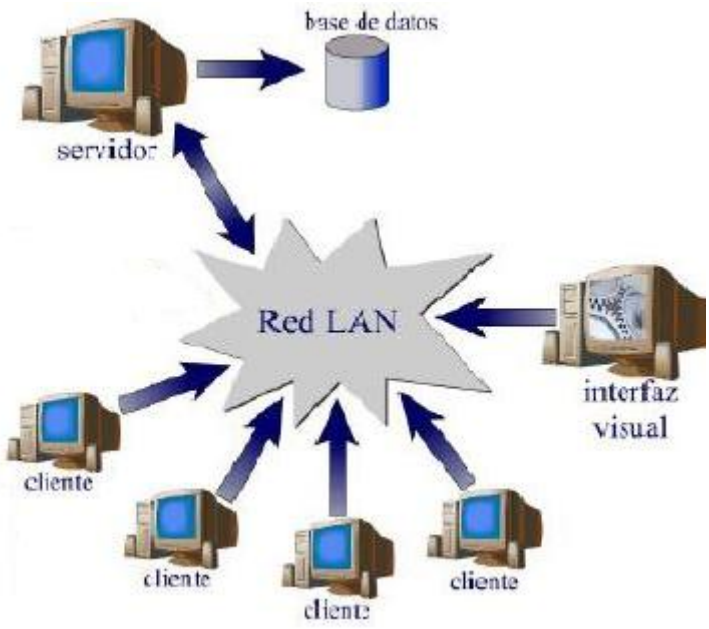


Figura 6. Vista general de la Plataforma de Tareas Distribuidas.

Está dividido en tres componentes esenciales: servidor, cliente e interfaz de administración. El módulo servidor almacena el problema (datos del problema y el algoritmo que los procesará) y divide estos en pequeños sub-problemas, llamados unidades de trabajo. El principal aspecto del servidor es repartir las diferentes unidades de trabajos entre los clientes (computadoras personales, clúster de computadoras, entre otros medios de cómputo), siempre y cuando estos estén autorizados a interactuar con el mismo; además de desarrollar una lógica de control de unidades pendientes de respuestas y otras que han expirado por algún error ocurrido.

El módulo cliente está instalado en cada una de las máquinas y conectadas al servidor mediante una red LAN. El cliente realiza una petición de una unidad de trabajo al servidor, hace el procesamiento de la misma y posteriormente retorna el resultado al servidor, y vuelve a pedir otra unidad de trabajo. Múltiples clientes pueden realizar peticiones de trabajo al servidor. El servidor colecciona el resultado de cada uno de los sub-problemas para finalmente construir el resultado del problema original. [2]

Para poder definir un cálculo distribuido sobre esta plataforma solo se requiere heredar de dos clases de Java, la clase *DataManager* y la clase *Task*, el desarrollador debe redefinir diferentes métodos de estas clases para lograr establecer su aplicación distribuida.

2.2.1 Clase *DataManager*.

La clase DataManager pertenece al servidor y su propósito es generar todas las unidades de trabajo que serán enviadas a los clientes, procesar los resultados, supervisar y ajustar el tamaño de las unidades, brindar información del estado de la ejecución del problema en un momento determinado y además se encarga de cerrar todos los recursos utilizados una vez concluido el trabajo distribuido.
La clase padre DataManager consta de una serie de métodos que deben ser redefinidos por el desarrollador para que el problema sea aceptado por el sistema. Una vez ejecutado el mismo se le asigna un directorio que podrá usar durante el cómputo.

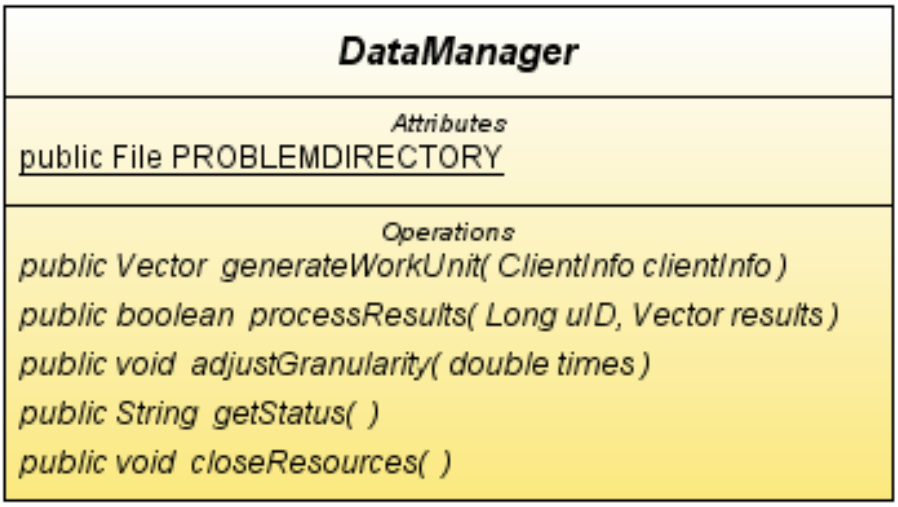


Figura 7. UML de la clase DataManager del Tarenal.

El programador debe implementar un constructor por defecto al heredar de la clase DataManager. Cualquiera de las excepciones incapturables o errores serán enviados de nuevo al servidor por medio de la cláusula throws. Un ejemplo del constructor de la clase DataManager de un problema particular se muestra a continuación. También se muestra como crear un archivo en el directorio de trabajo, asignado a la ejecución correspondiente. [2]

2.2.2 Clase Task.

La clase Task se ejecuta en el cliente y se encargará de procesar todas las unidades de trabajo generadas por el método *generateWorkUnit()* de la clase *DataManager* del servidor.
En esta clase solo se redefine el método *processUnit()*. Un algoritmo puede estar compuesto de una serie de funciones para procesar diferentes tipos de unidades de trabajo generadas por el DataManager. [2]

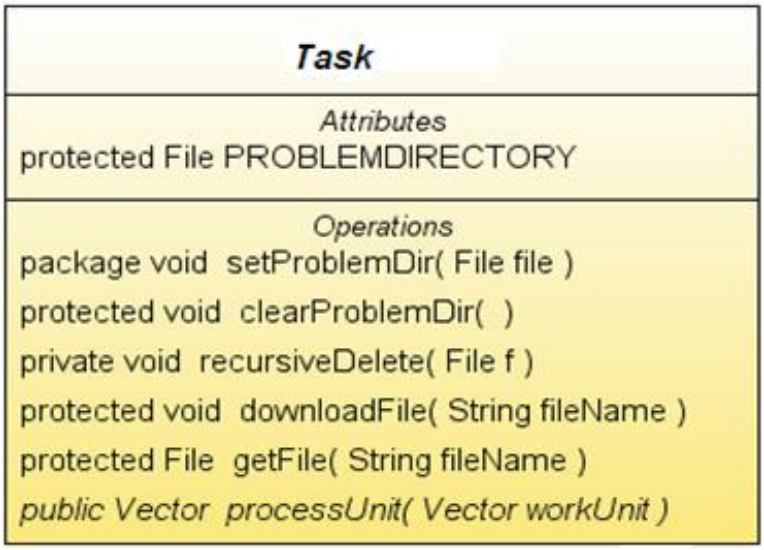


Figura 8. UML de la clase Task del Tarenal.

2.3 Java como lenguaje de programación.

Java es un lenguaje potente, orientado a objeto pues trabaja sus datos como objetos e interfaces a estos objetos. Elaborado a partir de los lenguajes C y C++ heredando sus características principales por ejemplo sus tipos de datos primitivos. Cuenta con grandes capacidades de interconexión TCP/IP por lo cual se puede acceder a la información disponible en red con mucha facilidad y seguridad. Es robusto pues chequea en busca de errores tanto en tiempo de compilación como en tiempo de ejecución, lo cual permite detectarlos lo antes posible, seguro evitando que el programador acceda de manera ilegal a la memoria. Una de sus características fundamentales es que es multiplataforma (Windows, Unix y Mac) debido a que todas sus ejecuciones y compilaciones son ejecutadas sobre una maquina virtual, por lo que no depende de la arquitectura de la maquina donde se ejecute.

2.4 Eclipse como herramienta IDE

Eclipse es un entorno de desarrollo integrado (IDE) distribuido con el código abierto y multiplataforma al igual que el lenguaje de programación java puede ejecutarse en sistemas operativos con características diversas. La arquitectura basada en *plug-ins* permite integrar varios lenguajes de programación e introducir otras aplicaciones complementarias. Conserva el registro de versiones generando y manteniendo la documentación del proyecto.

El eclipse funcionando como IDE de java presenta varias características que lo hacen muy funcional:

- Editor visual con sintaxis coloreada.
- Compilación incremental de código.
- Modifica e inspecciona valores de variables.
- Avisa de los errores cometidos mediante una ventana secundaria.
- Depura código que resida en una máquina remota.

2.5 Procedimientos.

Se han realizado diferentes estudios sobre el funcionamiento y rendimiento del perceptrón multicapa tanto en ambientes distribuidos como locales. Todos las pruebas o experimentos que se llevaron a cabo fueron realizados en computadores personales con un procesador Pentium(R) 4 HT a 3.00 GHz de frecuencia y con 1 Gb de RAM, es decir, que todos los resultados fueron obtenidos bajo esas condiciones.

2.5.1 Selección de la arquitectura del perceptrón multicapa.

La arquitectura del perceptrón multicapa se compone de: el número de neuronas en la capa de entrada el cual depende del número de componentes del vector de entrada; cantidad de capas ocultas y número de neuronas de cada una de ellas; número de neuronas en la capa de la salida el cual depende del número de componentes del vector de salida o patrón objetivo.
Puesto que tanto el vector de entrada como el de salida están definidos por el problema a resolver no se hace difícil la determinación de la cantidad de neuronas en esas capas. Respecto al criterio a tener en cuenta para la selección de las neuronas de las capas ocultas surge una interrogante, este número en general debe ser lo suficientemente grande como para que se forme una región compleja que

pueda resolver el problema, sin embargo no debe ser muy grande pues la estimación de los pesos puede ser no confiable para el conjunto de los patrones de entrada disponibles.
La mayor eficiencia del perceptrón multicapa en cuanto a su arquitectura se logra principalmente con una mayor experiencia su diseñador, aunque también existen varios criterios para realizar esta selección.

Para determinar la cantidad de capas ocultas:

Sin capa oculta: Sólo capaz de representar funciones linealmente separables o decisiones.
Una capa oculta: Pueden aproximar cualquier función continua desde menos infinito a más infinito.
Dos capas ocultas: Puede representar todas las funciones matemáticas conocidas hasta el momento.
Tres capas ocultas o más: No se ha encontrado justificación matemática para su uso.

Decidir la cantidad de capas ocultas es una pequeña parte del problema, también se tiene que determinar la cantidad de neuronas por capa oculta. Es muy importante en la arquitectura final de la red neuronal porque a pesar de no interactuar directamente con el ambiente externo tienen gran influencia en la salida final de la red. Ambos números la cantidad de capas y la cantidad de neuronas en esas capas deben ser cuidadosamente escogidos.

Para determinar la cantidad de neuronas en las capas ocultas:

Existen varios criterios para determinar la cantidad de neuronas ocultas adecuada:

1. La cantidad de neuronas ocultas debe estar entre la cantidad de neuronas de entrada y de la cantidad de neuronas de salidas.
2. La cantidad de neuronas ocultas debe ser las dos terceras partes de la cantidad de neuronas de entrada más la cantidad de neuronas de salida.
3. La cantidad de neuronas ocultas debe ser menor que dos veces la cantidad de neuronas de entrada.

Esas tres reglas se pueden tomar como punto de partida para considerar este importante aspecto de la arquitectura de las redes neuronales artificiales. [7]

2.6 Conclusiones.

En este capítulo se hizo un estudio sobre las herramientas que se usaron para realizar la investigación, presentando sus principales características. Se presentó además el proceder de la investigación haciendo énfasis en la selección de cantidad de capas ocultas y de la cantidad de neuronas ocultas del perceptrón multicapa debido a la gran influencia que tiene este aspecto en los resultados finales de la red neuronal. Puesto que el problema es representable con una función continua se utiliza una sola capa oculta y debido a que la muestra a utilizada tiene 576 características y solo se obtiene una salida, basándose en los criterios 1 y 3 de selección de la cantidad de neuronas en la capa oculta la arquitectura de la red queda conformada de la siguiente forma:

Tabla 2 Arquitectura 1 de la red neuronal.

Capa de Entrada	Capa Oculta	Capa de Salida
576	193	1

La configuración anterior constituye la arquitectura del perceptrón multicapa para realización de algunos experimentos pero también se usó la siguiente arquitectura basándose en los tres criterios de selección de la cantidad de neuronas en la capa oculta:

Tabla 3 Arquitectura 2 de la red neuronal.

Capa de Entrada	Capa Oculta	Capa de Salida
576	385	1

CAPÍTULO III: RESULTADOS Y DISCUSIÓN.

En este capítulo se muestran los resultados experimentales obtenidos, tanto en los experimentos en una sola computadora personal como en los experimentos realizados en la plataforma de tareas distribuidas Tarenal, además de una comparación de estos resultados en cuanto a tiempo de ejecución del entrenamiento y el porciento de error resultante al realizar la validación.

3.1 Breve explicación sobre el entrenamiento del perceptrón multicapa.

La aplicación genera aleatoriamente los pesos sinápticos. Se define la arquitectura de la red, la velocidad de aprendizaje y el *momentum*, además de la cantidad de patrones de entrada con que se va a realizar el entrenamiento.

Cada patrón de entrada se hace pasar a través de la estructura activando cada neurona y generando salidas en estas, dichas salidas son multiplicadas por los pesos sinápticos y constituyen la entrada de las neuronas de la capa siguiente, así sucesivamente hasta llegar a la capa de salida donde el resultado final es comparado con el resultado esperado generando un error el cual es propagado por toda la red hasta llegar al origen corrigiendo los valores sinápticos. Así sucede con cada patrón hasta que todos hayan pasado a través de la red, esto constituye una iteración o *epoch*.

Se pueden tener en cuenta varios criterios para establecer la condición de parada para el entrenamiento.

Cantidad de iteraciones o *epochs*: El investigador define la cantidad de iteraciones que considere necesaria para obtener una red neuronal lo suficientemente entrenada para procesar sus datos de manera satisfactoria.

Establecer un porciento de error máximo: El investigador define un porciento de error máximo permitido para que se detenga el entrenamiento cuando el porciento de error en varias iteraciones consecutivas se mantenga por debajo de este límite.

3.2 Resultados experimentales al ejecutar el percentrón multicapa en una sola computadora personal.

Se analizaron y compararon resultados de varias ejecuciones del perceptrón multicapa en un procesador variando algunos parámetros como la cantidad de patrones de entrada, y la arquitectura de la red. Para realizar la comparación de estos resultados se tuvo en cuenta principalmente el tiempo de entrenamiento de cada ejecución. Se hicieron varias ejecuciones del perceptrón manteniendo los mismos parámetros debido a que la inicialización de los pesos sinápticos es aleatoria. Los tiempos mostrados en los resultados experimentales es el promedio de los tiempos de las ejecuciones con esos parámetros y su formato es hh:mm:ss y los errores medios de validación mostrados también son el promedio de los porciento de error de validación resultante de los entrenamientos.

Resultado experimental número 1.
Se utilizó una muestra con 1000 patrones de entrada. Se introdujeron los siguientes parámetros:

Arquitectura:
Neuronas de entrada: 576
Neuronas ocultas: 193
Neuronas de salida: 1

Parámetros de entrenamiento:
Velocidad de aprendizaje: 0.03
Momentum: 0.01

Condición de parada:
Por ciento de error menor que: 10

Resultados:
Tiempo promedio de entrenamiento: 00:12:06
Porciento de error medio de validación: 6.5449109

Resultado experimental número 2.
Se utilizó una muestra con 3000 patrones de entrada. Se introdujeron los siguientes parámetros:

Arquitectura:
Neuronas de entrada: 576
Neuronas ocultas: 193
Neuronas de salida: 1

Parámetros de entrenamiento:
Velocidad de aprendizaje: 0.03
Momentum: 0.01

Condición de parada:
Por ciento de error menor que: 10

Resultados:
Tiempo promedio de entrenamiento: 00:54:01
Porciento de error medio de validación: 4.32759206

Estos experimentos demuestran que con muestras pequeñas no tiene sentido su distribución, debido a que los tiempos promedios de respuestas son de pocos minutos, además se puede apreciar el elevado porciento de error medio de validación lo cual indica la baja calidad del perceptrón multicapa resultante del entrenamiento debido a la escasez de patrones de entrada.

Resultado experimental número 3.
Se utilizó una muestra con 5000 patrones de entrada. Se introdujeron los siguientes parámetros:

Arquitectura:
Neuronas de entrada: 576
Neuronas ocultas: 193
Neuronas de salida: 1

Parámetros de entrenamiento:
Velocidad de aprendizaje: 0.03
Momentum: 0.01

Condición de parada:
Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 01:27:04

Porciento de error medio de validación: 3.79735167

Resultado experimental número 4.

Se utilizó una muestra con 10000 patrones de entrada. Se introdujeron los siguientes parámetros:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 193

Neuronas de salida: 1

Parámetros de entrenamiento:

Velocidad de aprendizaje: 0.03

Momentum: 0.01

Condición de parada:

Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 02:31:19

Porciento de error medio de validación: 2.93568411

En este experimento hay que tener en cuenta que se hizo también un pequeño aumento de la cantidad de patrones de entrada y se mantuvieron el resto de los parámetros y los tiempos de respuestas siguen aumentando considerablemente, pero también se puede observar una disminución del porciento de error medio de validación. Ya a partir de este punto se debe considerar la idea de distribución del algoritmo para optimizar los tiempos de entrenamiento.

Resultado experimental número 5.

Se utilizó una muestra con 15000 patrones de entrada. Se introdujeron los siguientes parámetros:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 193

Neuronas de salida: 1

Parámetros de entrenamiento:

Velocidad de aprendizaje: 0.03

Momentum: 0.01

Condición de parada:

Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 04:14:09

Porciento de error medio de validación: 1.73538914

Resultado experimental número 6.

Se utilizó una muestra con 30000 patrones de entrada. Se introdujeron los siguientes parámetros:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 193

Neuronas de salida: 1

Parámetros de entrenamiento:

Velocidad de aprendizaje: 0.03

Momentum: 0.01

Condición de parada:

Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 07:35:08

Porciento de error medio de validación: 0.92546999

Resultado experimental número 7.

Se utilizó una muestra con 60000 patrones de entrada. Se introdujeron los siguientes parámetros:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 193

Neuronas de salida: 1

Parámetros de entrenamiento:

Velocidad de aprendizaje: 0.03

Momentum: 0.01

Condición de parada:

Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 17:32:44

Porciento de error medio de validación: 0.5971254

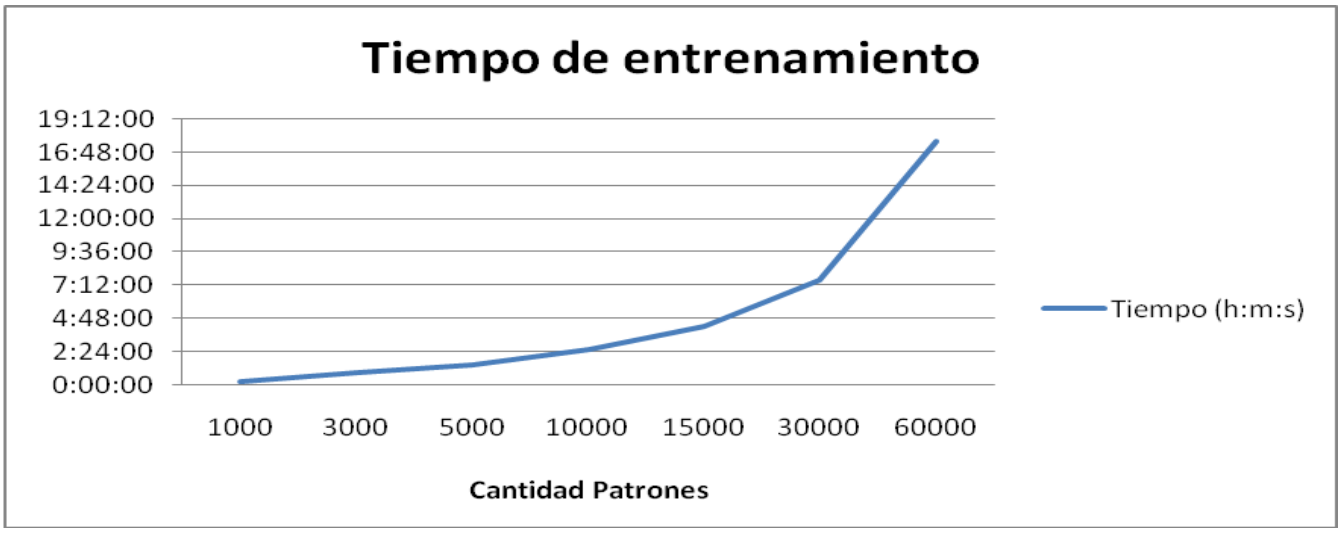
En este experimento hay que tener en cuenta que se hizo un aumento de la cantidad de patrones de entrada manteniendo el resto de los parámetros, los tiempos de respuestas aumentaron aún más y por el contrario los errores de validación disminuyeron significativamente con respecto a los experimentos anteriores.

A continuación se muestra una tabla donde se pueden observar las diferentes pruebas realizadas de manera más resumida.

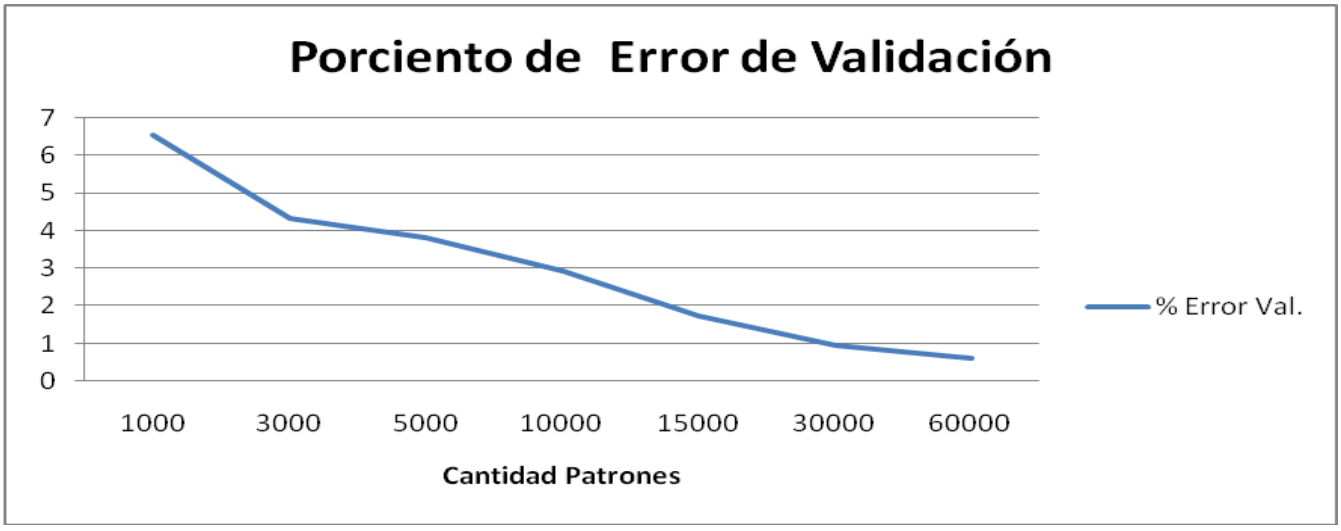
Tabla 3. Resultados experimentales variando la cantidad de patrones de entrada.

No	Arquit	Veloc Apr	Moment	% err Esp	Patrs	% error Val	Tiempo Entr
1	576;193;1	0.03	0.01	10	1000	6.5449109	00:13:50
2	576;193;1	0.03	0.01	10	3000	4.32759206	00:54:01
3	576;193;1	0.03	0.01	10	5000	3.79735167	01:27:04
4	576;193;1	0.03	0.01	10	10000	2.93568411	02:31:19
5	576;193;1	0.03	0.01	10	15000	1.73538914	04:14:09
6	576;193;1	0.03	0.01	10	30000	0.92546999	07:35:08
7	576;193;1	0.03	0.01	10	60000	0.5971254	17:32:44

Se muestra una gráfica de tiempo respecto a la cantidad de patrones de entrada y otra de porciento de error de validación con respecto a la cantidad de patrones donde se observa claramente que para pocos datos de entrada no es necesario hacer uso de técnicas de distribución, pero a medida que los datos de entrada van en aumento se hace necesario el uso de alguna técnica de programación distribuida y que el error de validación disminuye al aumentar la cantidad de patrones lo que hace a la red resultante de mayor calidad.



Gráfica 1. Comportamiento del tiempo a medida de que la cantidad de patrones aumenta.



Gráfica 2. Comportamiento del porciento de error de validación a medida de que la cantidad de patrones aumenta.

En los siguientes experimentos es importante señalar que se utiliza una arquitectura diferente a la de los anteriores, en cuanto a que se usa una mayor cantidad de neuronas en la capa oculta lo que provoca que se creen más conexiones sinápticas.

Resultado experimental número 8.
Se utilizó una muestra con 1000 patrones de entrada. Se introdujeron los siguientes parámetros:

Arquitectura:
Neuronas de entrada: 576
Neuronas ocultas: 385
Neuronas de salida: 1

Parámetros de entrenamiento:
Velocidad de aprendizaje: 0.03
Momentum: 0.01

Condición de parada:
Por ciento de error menor que: 10

Resultados:
Tiempo promedio de entrenamiento: 00:27:43
Porciento de error medio de validación: 4.939979723

Si se compara este experimento con el número 1 que cuenta con los mismos parámetros y solo difieren en la cantidad de neuronas en la capa oculta, se puede observar cómo la cantidad de neuronas ocultas influye en el tiempo de entrenamiento aumentándolo considerablemente lo que se refleja positivamente en la calidad del perceptrón resultante del entrenamiento al disminuir el porciento de error medio de validación.

Resultado experimental número 9.
Se utilizó una muestra con 5000 patrones de entrada. Se introdujeron los siguientes parámetros:

Arquitectura:
Neuronas de entrada: 576

Neuronas ocultas: 385
Neuronas de salida: 1

Parámetros de entrenamiento:
Velocidad de aprendizaje: 0.03
Momentum: 0.01

Condición de parada:
Por ciento de error menor que: 10

Resultados:
Tiempo promedio de entrenamiento: 02:46:48
Porciento de error medio de validación: 2.778651768

En este experimento se aumentó la cantidad de patrones de entrada con respecto al experimento anterior y se pudo observar un aumento en los tiempos de entrenamiento y una disminución de los errores medios de validación lo que representa mayor calidad de la red entrenada. Comparándolo con el experimento número 3 también se puede apreciar cómo con esta nueva arquitectura aumenta el tiempo de entrenamiento pero a su vez disminuye el porciento de error medio de validación.

Resultado experimental número 10.
Se utilizó una muestra con 15000 patrones de entrada. Se introdujeron los siguientes parámetros:

Arquitectura:
Neuronas de entrada: 576
Neuronas ocultas: 385
Neuronas de salida: 1

Parámetros de entrenamiento:
Velocidad de aprendizaje: 0.03
Momentum: 0.01

Condición de parada:
Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 08:03:47

Porciento de error medio de validación: 1.215752671

Resultado experimental número 11.

Se utilizó una muestra con 30000 patrones de entrada. Se introdujeron los siguientes parámetros:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 385

Neuronas de salida: 1

Parámetros de entrenamiento:

Velocidad de aprendizaje: 0.03

Momentum: 0.01

Condición de parada:

Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 18:08:55

Porciento de error medio de validación: 0.598360725

Resultado experimental número 12.

Se utilizó una muestra con 60000 patrones de entrada. Se introdujeron los siguientes parámetros:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 385

Neuronas de salida: 1

Parámetros de entrenamiento:

Velocidad de aprendizaje: 0.03

Momentum: 0.01

Condición de parada:

Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 39:23:42

Porciento de error medio de validación: 0.384556452

En estos experimentos también se puede evidenciar que al aumentar los patrones de entrada aumentan los tiempos promedios de ejecución de los entrenamientos y un aumento también de la calidad de las redes obtenidas. Comparándolos con experimentos números 5, 6 y 7 respectivamente se puede notar que al aumentar la cantidad de neuronas en la capa oculta aumentan también los tiempos de entrenamiento repercutiendo positivamente al disminuir los errores medios de validación. Esta es la muestra más grande con la que se cuenta, pero se cree que es suficiente para demostrar la necesidad de distribuir los entrenamientos del perceptrón multicapa debido a que los tiempos de entrenamiento son excesivamente altos, pero es necesario destacar que la calidad de la red es superior.

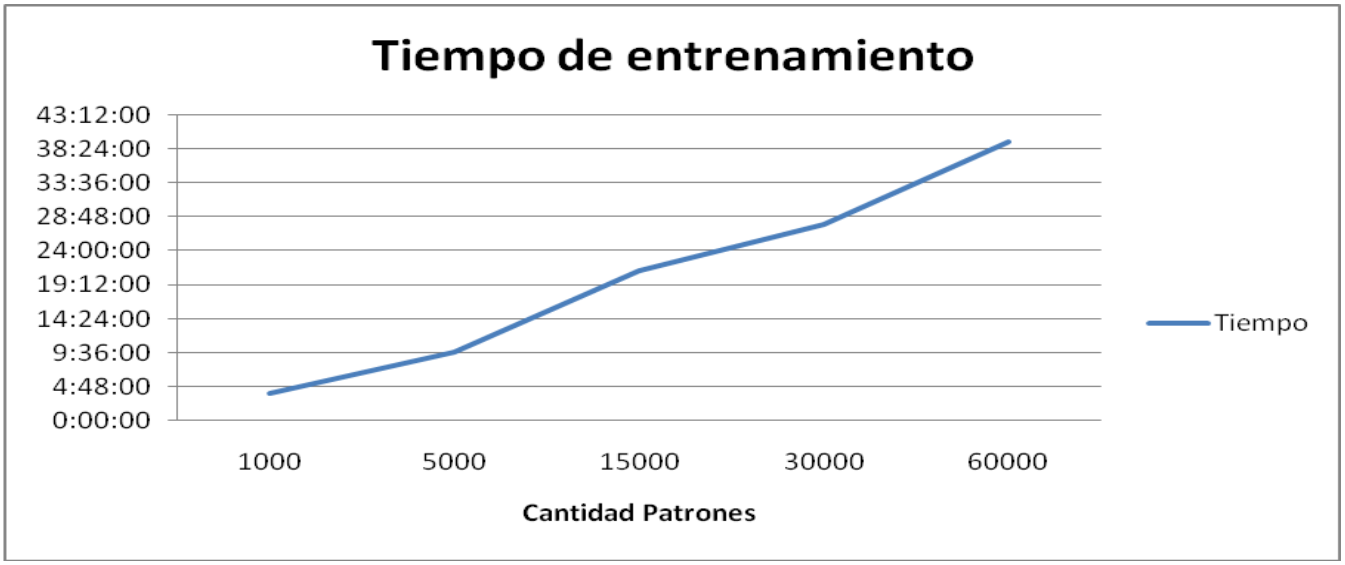
A continuación se presenta una tabla donde se muestran los resultados de los experimentos anteriores.

Tabla 4. Resultados experimentales variando la cantidad de patrones de entrada.

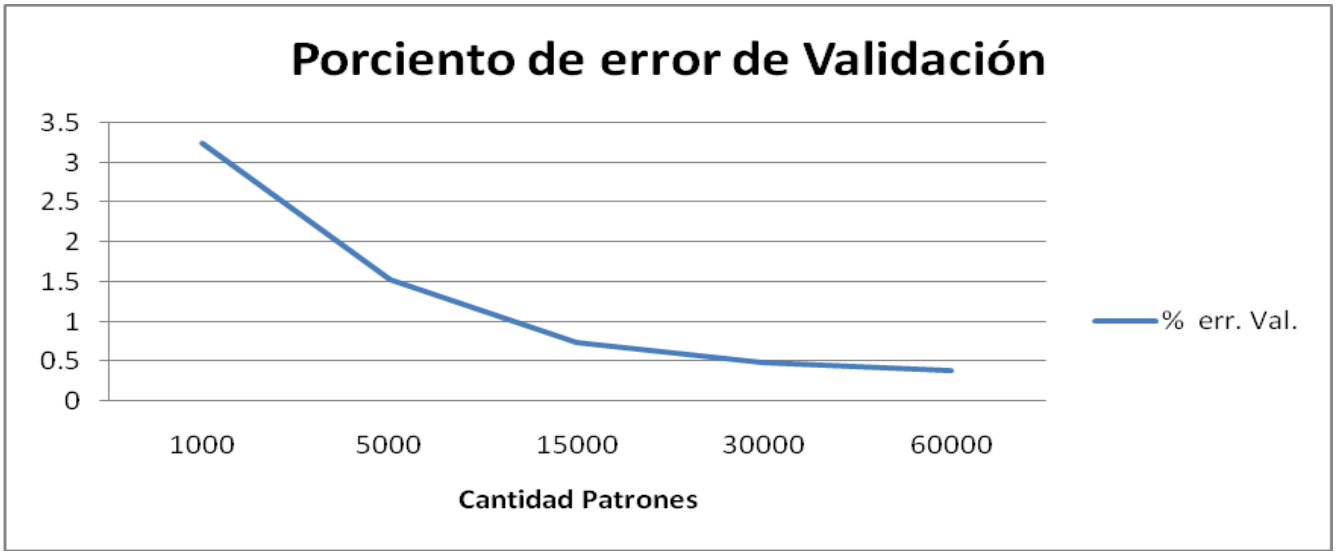
No	Arquit	Veloc Apr	Moment	% err Esp	Patrs	% Error Val	Tiempo Entr
8	576;385;1	0.03	0.01	10	1000	4.939979723	00:27:43
9	576;385;1	0.03	0.01	10	5000	2.778651768	02:46:48
11	576;385;1	0.03	0.01	10	15000	1.215752671	08:03:47
12	576;385;1	0.03	0.01	10	30000	0.598360725	18:08:55
13	576;385;1	0.03	0.01	10	60000	0.384556452	39:23:42

Las siguiente gráficas ilustran el comportamiento del tiempo de entrenamiento en cuanto a la cantidad de patrones de entrada y del porciento de error medio de validación con respecto a la cantidad de patrones de entrada respectivamente en los experimentos anteriores donde se puede observar que

con el aumento de la cantidad de patrones entrada aumenta el tiempo de entrenamiento y disminuye el porciento de error de validación lo que implica un aumento de la calidad de la red entrenada.



Gráfica 3. Comportamiento del tiempo a medida de que la cantidad de patrones aumenta.

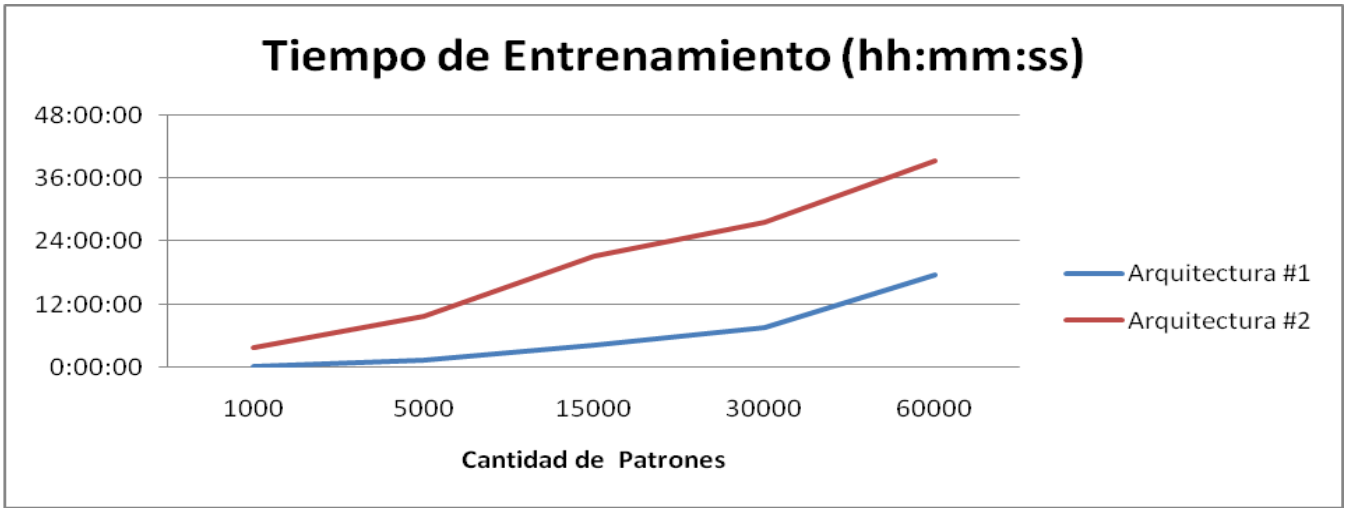


Gráfica 4. Comportamiento del porciento de error de validación a medida de que la cantidad de patrones aumenta.

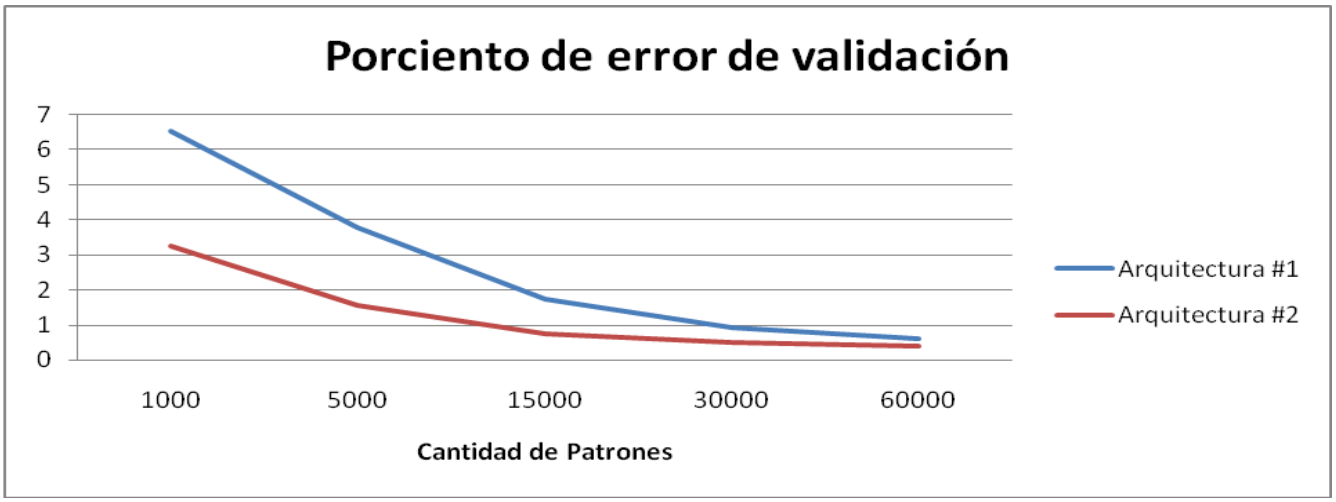
3.3 Conclusiones Parciales.

Observándose los resultados anteriores se puede notar varios aspectos importantes del entrenamiento del perceptrón multicapa. Al aumentar la cantidad de patrones de entrada o la cantidad de neuronas de la capa oculta de la red (este aumento no puede ser realizado arbitrariamente, se debe hacer basándose en los criterios de selección de la cantidad de neuronas oculta expuesto en el epígrafe 2.5.1 o basándose en la experiencia del investigador) provoca en los resultados un aumento considerable en los tiempos de ejecución del entrenamiento y una disminución del porciento de error medio de validación lo que implica un aumento de la calidad del perceptrón multicapa resultante de los entrenamientos.

A continuación se muestran gráficas donde se puede observar una comparación entre las 2 arquitecturas en cuanto a tiempo de entrenamiento y porciento de error medio de validación.



Gráfica 5. Comportamiento del tiempo de entrenamiento con las arquitecturas 1 y 2.



Gráfica 6. Comportamiento del porciento de error medio de validación con las arquitecturas 1 y 2.

Por tanto para obtener un perceptrón multicapa lo mejor entrenado posible se escogió la arquitectura 2 (576, 385, 1).

Basándose en el gran número de computadoras conectadas entre sí mediante una red con la cual se cuenta en la Universidad se piensa que la plataforma de tareas distribuidas Tarenal es una potente alternativa para lograr que los tiempos de entrenamiento del perceptrón multicapa sean reducidos.
La ejecución del entrenamiento del perceptrón multicapa es secuencial puesto que entrena con cada patrón de entrada de uno en uno y el actual se ejecuta sobre una red que ya ha sido entrenada con los anteriores. Esto impide que el algoritmo pueda ser totalmente distribuido por tanto la distribución debe realizarse entrenando varios perceptrones independientemente con la muestra dividida en partes iguales y representativas de la muestra total. Para lograr que los fragmentos de las muestras sean partes representativas de la muestra total es necesario que la totalidad de los datos sean escalados, que no es más que reorganizar los datos de la muestra con la cual se realizará el entrenamiento de forma tal que al dividirla cada fragmento sea una parte representativa de su total. La principal ventaja de esta operación es evitar que los atributos en mayores rangos numéricos dominen a esos de menores rangos, otra ventaja es que se evitan dificultades numéricas durante el cálculo.

3.4 Algoritmo de distribución.

La plataforma de tareas distribuidas Tarenal funciona de la siguiente forma: mediante una interfaz se copian al servidor los datos con los cuales trabajará y los algoritmos necesarios para la distribución y el

procesamiento de los mismos, los ordenadores en los cuales está instalado el cliente del Tarenal se reportan haciendo una petición de una unidad de trabajo, el servidor a su vez se encarga de distribuir las unidades de trabajo entre los clientes reportado. Una vez que los clientes hayan terminado el trabajo asignado devuelven los resultados al servidor, el cual se encarga de empaquetarlos para permitir su posterior descarga desde la interfaz de usuario.

El algoritmo utilizado para distribuir el entrenamiento del perceptrón multicapa funciona con una muestra previamente escalada para garantizar que los fragmentos de la muestra sean partes representativas de la muestra total, una vez fragmentada la muestra en el servidor de la plataforma de tareas distribuidas se crea el perceptrón multicapa cuyos pesos sinápticos son inicializados con valores aleatorios, posterior a esto el servidor envía hacia los clientes que hayan hecho una petición de trabajo una copia del perceptrón inicializado y un fragmento de los datos, en los clientes se lleva a cabo la ejecución del entrenamiento con el fragmento y terminando esta acción van enviando al servidor los perceptrones resultantes de los entrenamientos, por su parte el servidor promedia las matrices de los pesos sinápticos proveniente de cada cliente, obteniendo una nueva matriz de pesos sinápticos la cual formará parte del perceptrón que será devuelto a la interfaz de la plataforma de tareas distribuidas, dicho perceptrón está entrenado y listo para ser validado para uso posterior.

3.5 Resultados experimentales obtenidos al ejecutar el algoritmo de distribución del perceptrón multicapa.

En este epígrafe se muestran y se analizan los resultados obtenidos al realizar entrenamientos del perceptrón multicapa en Tarenal, variando algunos parámetros con el objetivo de probar y validar la factibilidad del uso de técnicas de distribución en entrenamiento del perceptrón multicapa. Los tiempos mostrados en los resultados son el promedio de los tiempos de entrenamiento y su formato es hh:mm:ss y el porciento de error medio de validación mostrado es el promedio de los errores cometidos por el perceptrón multicapa al ejecutarse con los datos de validación.

Resultado experimental número 13.

Se utilizó una muestra previamente escalada de 5000 patrones de entrada la cual fue dividida en 2 fragmentos y los parámetros con que se inicializó el perceptrón fueron los siguientes:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 385
Neuronas de salida: 1

Parámetros de entrenamiento:
Velocidad de aprendizaje: 0.03
Momentum: 0.01

Condición de parada:
Por ciento de error menor que: 10

Resultados:
Tiempo promedio de entrenamiento: 00:57:08
Porciento de error medio de validación: 2.879433371

Resultado experimental número 14.
Se utilizó una muestra previamente escalada de 5000 patrones de entrada la cual fue dividida en 5 fragmentos y los parámetros con que se inicializó el perceptrón fueron los siguientes:

Arquitectura:
Neuronas de entrada: 576
Neuronas ocultas: 385
Neuronas de salida: 1

Parámetros de entrenamiento:
Velocidad de aprendizaje: 0.03
Momentum: 0.01

Condición de parada:
Por ciento de error menor que: 10

Resultados:
Tiempo promedio de entrenamiento: 00:22:49
Porciento de error medio de validación: 2.945226232

Resultado experimental número 15.

Se utilizó una muestra previamente escalada de 5000 patrones de entrada la cual fue dividida en 10 fragmentos y los parámetros con que se inicializó el perceptrón fueron los siguientes:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 385

Neuronas de salida: 1

Parámetros de entrenamiento:

Velocidad de aprendizaje: 0.03

Momentum: 0.01

Condición de parada:

Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 00:10:13

Porciento de error medio de validación: 3.243803298

Resultado experimental número 16.

Se utilizó una muestra previamente escalada de 15000 patrones de entrada la cual fue dividida en 2 fragmentos y los parámetros con que se inicializó el perceptrón fueron los siguientes:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 385

Neuronas de salida: 1

Parámetros de entrenamiento:

Velocidad de aprendizaje: 0.03

Momentum: 0.01

Condición de parada:

Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 03:40:12

Porciento de error medio de validación: 1.708156529

Resultado experimental número 17.

Se utilizó una muestra previamente escalada de 15000 patrones de entrada la cual fue dividida en 5 fragmentos y los parámetros con que se inicializó el perceptrón fueron los siguientes:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 385

Neuronas de salida: 1

Parámetros de entrenamiento:

Velocidad de aprendizaje: 0.03

Momentum: 0.01

Condición de parada:

Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 01:07:11

Porciento de error medio de validación: 2.126186774

Resultado experimental número 18.

Se utilizó una muestra previamente escalada de 15000 patrones de entrada la cual fue dividida en 10 fragmentos y los parámetros con que se inicializó el perceptrón fueron los siguientes:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 385

Neuronas de salida: 1

Parámetros de entrenamiento:
Velocidad de aprendizaje: 0.03
Momentum: 0.01

Condición de parada:
Por ciento de error menor que: 10

Resultados:
Tiempo promedio de entrenamiento: 00:32:37
Porciento de error medio de validación: 2.37568895

En los experimentos anteriores se mantuvieron los mismos parámetros que en las ejecuciones locales, demostrando que al hacer uso de las técnicas de distribución los tiempos de entrenamientos disminuyen considerablemente pero disminuye a su vez la calidad de la red obtenida, debido a que al fragmentar los datos, las muestras resultantes son más pequeñas.

Resultado experimental número 19.
Se utilizó una muestra previamente escalada de 30000 patrones de entrada la cual fue dividida en 2 fragmentos y los parámetros con que se inicializó el perceptrón fueron los siguientes:

Arquitectura:
Neuronas de entrada: 576
Neuronas ocultas: 385
Neuronas de salida: 1

Parámetros de entrenamiento:
Velocidad de aprendizaje: 0.03
Momentum: 0.01

Condición de parada:
Por ciento de error menor que: 10

Resultados:
Tiempo promedio de entrenamiento: 06:06:13
Porciento de error medio de validación: 1.12626251

Resultado experimental número 20.

Se utilizó una muestra previamente escalada de 30000 patrones de entrada la cual fue dividida en 5 fragmentos y los parámetros con que se inicializó el perceptrón fueron los siguientes:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 385

Neuronas de salida: 1

Parámetros de entrenamiento:

Velocidad de aprendizaje: 0.03

Momentum: 0.01

Condición de parada:

Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 02:47:49

Porciento de error medio de validación: 1.570070622

Resultado experimental número 21.

Se utilizó una muestra previamente escalada de 30000 patrones de entrada la cual fue dividida en 10 fragmentos y los parámetros con que se inicializó el perceptrón fueron los siguientes:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 385

Neuronas de salida: 1

Parámetros de entrenamiento:

Velocidad de aprendizaje: 0.03

Momentum: 0.01

Condición de parada:

Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 01:07:16

Porciento de error medio de validación: 1.718555274

Resultado experimental número 22.

Se utilizó una muestra previamente escalada de 60000 patrones de entrada la cual fue dividida en 2 fragmentos y los parámetros con que se inicializó el perceptrón fueron los siguientes:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 385

Neuronas de salida: 1

Parámetros de entrenamiento:

Velocidad de aprendizaje: 0.03

Momentum: 0.01

Condición de parada:

Por ciento de error menor que: 10

Resultados:

Tiempo promedio de entrenamiento: 14:00:35

Porciento de error medio de validación: 0.861061283

Resultado experimental número 23.

Se utilizó una muestra previamente escalada de 60000 patrones de entrada la cual fue dividida en 5 fragmentos y los parámetros con que se inicializó el perceptrón fueron los siguientes:

Arquitectura:

Neuronas de entrada: 576

Neuronas ocultas: 385

Neuronas de salida: 1

Parámetros de entrenamiento:
Velocidad de aprendizaje: 0.03
Momentum: 0.01

Condición de parada:
Por ciento de error menor que: 10

Resultados:
Tiempo promedio de entrenamiento: 04:38:49
Porciento de error medio de validación: 1.003166686

Resultado experimental número 24.
Se utilizó una muestra previamente escalada de 60000 patrones de entrada la cual fue dividida en 10 fragmentos y los parámetros con que se inicializó el perceptrón fueron los siguientes:

Arquitectura:
Neuronas de entrada: 576
Neuronas ocultas: 385
Neuronas de salida: 1

Parámetros de entrenamiento:
Velocidad de aprendizaje: 0.03
Momentum: 0.01

Condición de parada:
Por ciento de error menor que: 10

Resultados:
Tiempo promedio de entrenamiento:
Porciento de error medio de validación:

Con los anteriores experimentos queda demostrado que a medida que aumenta la cantidad de procesadores en el entrenamiento de una red, el tiempo de entrenamiento disminuye en gran medida, pero a su vez la calidad de la misma se hace menor pues al dividir la muestra en mayor cantidad de

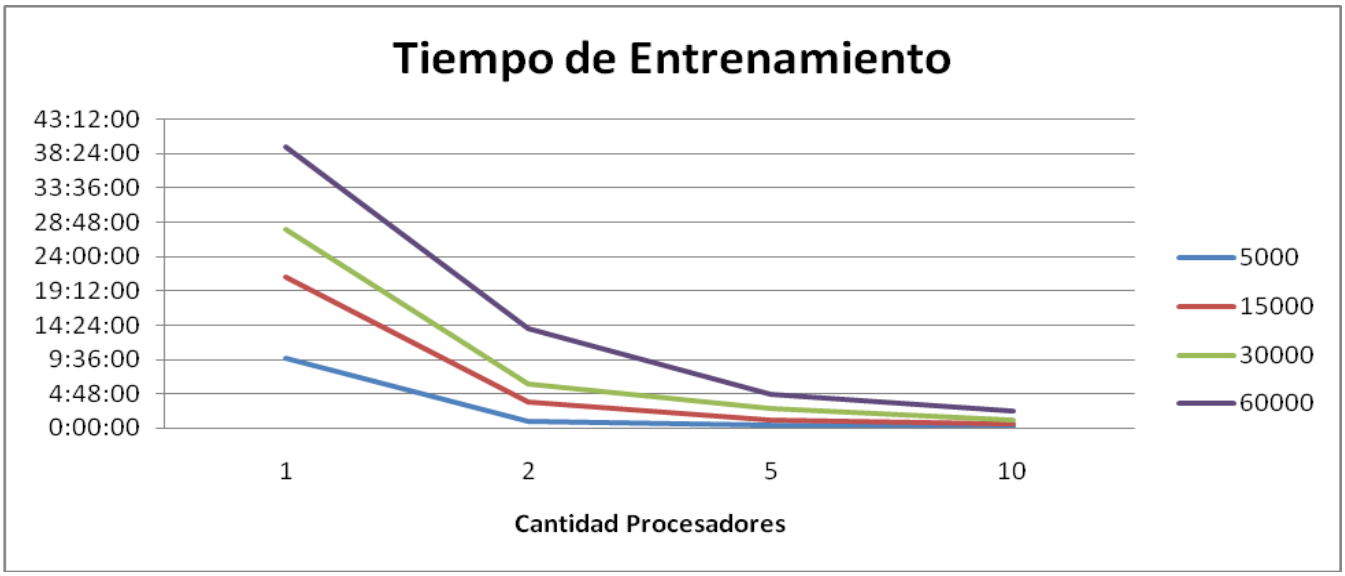
fragmentos el entrenamiento en cada estación de trabajo se realiza con menos patrones de entrada, lo que hace aumentar el porciento de error de validación.

A continuación se muestra una tabla donde se puede apreciar el comportamiento del perceptrón multicapa en un ambiente distribuido.

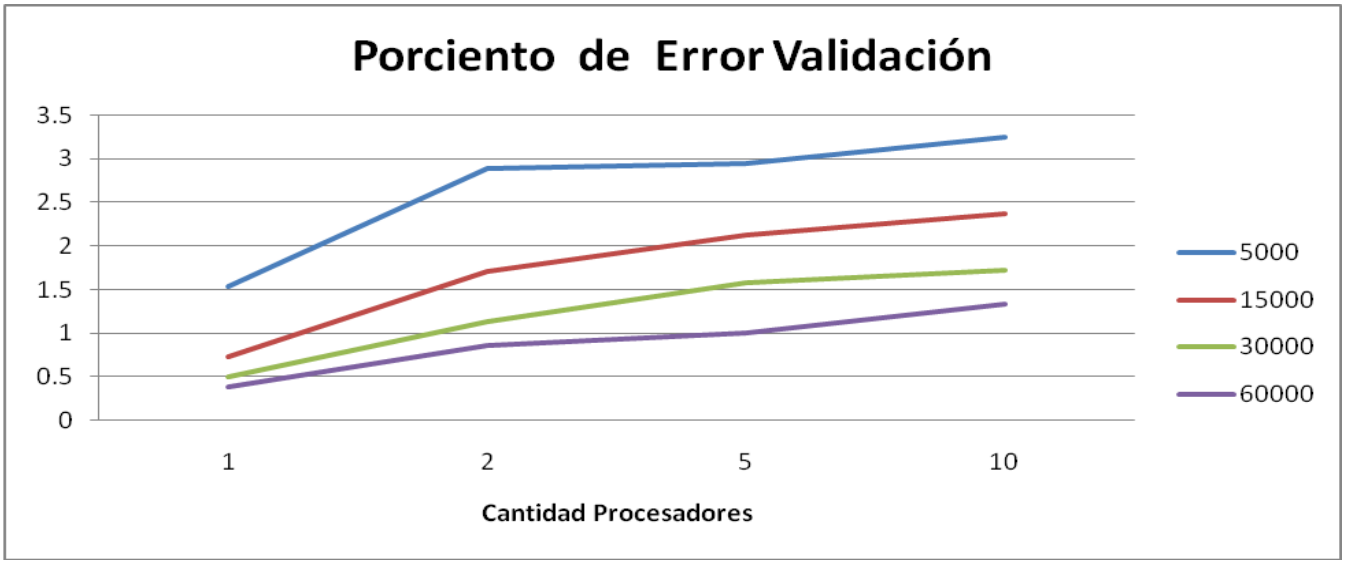
Tabla 5. Resultados experimentales variando la cantidad de patrones de entrada y cantidad de fragmentos en que se divide la muestra.

No	Arquit	Veloc Apr	Moment	% err Esp	Patrs	Frag	% error Val	Tiempo Entr
13	576;385;1	0.03	0.01	10	5000	2	2.8794333	00:57:08
14	576;385;1	0.03	0.01	10	5000	5	2.9452262	00:22:49
15	576;385;1	0.03	0.01	10	5000	10	3.2438032	00:10:13
16	576;385;1	0.03	0.01	10	15000	2	1.7081565	03:40:12
17	576;385;1	0.03	0.01	10	15000	5	2.1261867	01:10:07
18	576;385;1	0.03	0.01	10	15000	10	2.3756889	00:32:37
19	576;385;1	0.03	0.01	10	30000	2	1.1262625	06:06:13
20	576;385;1	0.03	0.01	10	30000	5	1.5700706	02:47:49
21	576;385;1	0.03	0.01	10	30000	10	1.7185552	01:07:16
22	576;385;1	0.03	0.01	10	60000	2	0.8610612	14:00:35
23	576;385;1	0.03	0.01	10	60000	5	1.0031666	04:42:31
24	576;385;1	0.03	0.01	10	60000	10	1.3357890	02:24:41

Las siguientes gráficas muestran el comportamiento del perceptrón multicapa en cuanto a tiempo de entrenamiento y el porciento de error de validación con respecto a la cantidad de procesadores y al tamaño de la muestra usados en el entrenamiento distribuido.



Gráfica 7. Comportamiento del tiempo de entrenamiento con varios procesadores.



Gráfica 8. Comportamiento del porciento de error de validación con varios procesadores.

3.6 Conclusiones.

En este capítulo se expusieron los resultados de varios experimentos tanto locales como distribuidos con los cuales se puede demostrar que la cantidad de patrones de entrada tiene una relación directa con el tiempo de entrenamiento y la calidad de la red neuronal entrenada. Luego de realizar varias

pruebas en un ambiente distribuido se puede concluir que para grandes volúmenes de datos es muy eficaz el uso del algoritmo de distribución implementado, pues disminuye en gran medida los tiempos de entrenamiento sin sacrificar su calidad. Es necesario establecer un equilibrio entre la cantidad de procesadores a utilizar y la calidad deseada para la red resultante del entrenamiento distribuido, pues un aumento excesivo en la cantidad de unidades de procesamiento se refleja en un deficiente entrenamiento de la red.

CONCLUSIONES

1. Se definió una arquitectura del perceptrón multicapa adecuada, basándose en los criterios de selección expuestos y en la experiencia adquirida mediante experimentos.

2. El algoritmo de distribución implementado reduce en gran medida los tiempos de ejecución del entrenamiento del perceptrón multicapa, manteniendo dentro de un rango aceptable la calidad de la red resultante.

3. Se demostró la eficiencia del algoritmo a través de la realización de varias pruebas con datos de validación.

RECOMENDACIONES

- Investigar la factibilidad de este método de distribución para el entrenamiento de otros tipos de redes neuronales artificiales.
- Escalar los datos antes de realizar cualquier entrenamiento del perceptrón multicapa.
- Establecer un equilibrio entre la cantidad de procesadores y la calidad de la red neuronal entrenada en el ambiente distribuido.

REFERENCIAS BIBLIOGRÁFICAS.

[1]. **Araujo, Alfonso.** Sistemas Distribuidos Concepto y Características. [En línea.] [Citado el 17 mayo de 2009] http://mx.geocities.com/alfonsoaraujocardenas/sistemasdistribuidos.html.

[2]. **Bioinformática.** *Manual del Desarrollador - Plataforma de Tareas Distribuidas.* Ciudad de la Habana: s.n., 2008.

[3]. **Carrasco, Ramón y Escalona, J. C.** Facultad de Química de la Universidad de la Habana. *Facultad de Química.* [En línea] [Citado el: 15 de enero de 2009.] http://www.fq.uh.cu/investig/lqct/imagenes2/diseño.pdf.

[4]. **Dimov, Rossen.** *Weka: Practical Machine Learning Tools and Techiniques with Java Implementations.* 23 de abril de 2007.

[5]. **Dr. Bello Pérez, Rafael.** *Curso introductorio a las redes neuronales artificiales.* Departamento de Ciencia de la Computación, Universidad Central de Las Villas : s.n., 1993.

[6]. **Gutiérrez, Elizabeth.** Redes Neuronales Ventajas y Desventajas. [En línea] [Citado el 23 de abril de 2009] http://egkafati.bligoo.com/content/view/184582/Redes-neuronales-ventajas-y-desventajas.html.

[7]. **Heaton, Jeff.** Heaton Research. *Introduction to Neural Networks for Java, 2nd Edition.* [En línea] 2005. [Citado el: 22 de abril de 2009.] http://www.heatonresearch.com/book/programming-neural-networks-java-2.html. 1604390085.

[8]. **Ing. Acosta, Maria Isabel, Ing. Salazar, Harold y Ing. Zuluaga, Camilo A.** Universidad Tecnológica de Pereira. [En línea] [Citado el: 22 de enero de 2009.] http://ohm.utp.edu.co/neuronales/Capitulo2/Backpropagation/MomentumB.htm.

[9]. **Morales, L.** Instituto Nacional de Astrofísica Óptica y Electrónica. INAOE - Ciencias Computacionales. [En línea] [Citado el: 20 de febrero de 2009.] http://ccc.inaoep.mx/~lamorales/distribuidos/Sistemas_Distribuidos.pdf.

[10]. **Mortiz, J.** Lenguajes y Ciencias de la Computación. *Universidad de Malaga.* [En línea] [Citado el: 2 de abril de 2009.] http://www.lcc.uma.es/~jmortiz/archivos/Tema4.pdf.

[11]. **Soria, Emilio y Blanco, Antonio.** Redes Neuronales Artificiales. *Autores Científico-Técnicos y Académicos (ACTA).* [En línea] [Citado el: 1 de noviembre de 2008.] http://www.acta.es/articulos_mf/19023.PDF.

[12]. **Villanueva, María del Rosario.** Visión Panorámica de las Redes Neuronales Artificiales. *Facultad de Ciencias Matemáticas - UNMSM* [En línea.] [Citado el: 14 de diciembre de 2008] http://matematicas.unmsm.edu.pe/weboperativa/congreso/ASSETS/DOCS/2210/VISION_P_DE__RN.PPT

BIBLIOGRAFÍA

1. **Araujo, Alfonso.** Sistemas Distribuidos Concepto y Características. [En línea.] [Citado el 17 mayo de 2009] http://mx.geocities.com/alfonsoaraujocardenas/sistemasdistribuidos.html.

2. **BARRO, M. J**. *Computación Neuronal.* Universidad Santiago de Compostela, 1995.

3. **BioInformatica.** *Manual del Desarrollador - Plataforma de Tareas Distribuidas.* Ciudad de la Habana : s.n., 2008.

4. **Carrasco, Ramón y Escalona, J. C.** Facultad de Química de la Universidad de la Habana. *Facultad de Química.* [En línea] [Citado el: 15 de enero de 2009.] http://www.fq.uh.cu/investig/lqct/imagenes2/diseño.pdf.

5. **DARPA**. Neural network study. *Defense Advanced Research Projects Agency (DARPA)*, 1988.

6. **Dimov, Rossen.** *Weka: Practical Machine Learning Tools and Techiniques with Java Implementations.* 23 de abril de 2007.

7. **Dr. Bello Pérez, Rafael.** *Curso introductorio a las redes neuronales artificiales.* Departamento de Ciencia de la Computación, Universidad Central de Las Villas : s.n., 1993.

8. **Gutiérrez, Elizabeth.** Redes Neuronales Ventajas y Desventajas. [En línea] [Citado el 23 de abril de 2009] http://egkafati.bligoo.com/content/view/184582/Redes-neuronales-ventajas-y-desventajas.html.

9. **HAYKIN, S**. *Neural Networks. A Comprehensive Foundation. .* Second Edition ed. Prentice Hall, 1999.

10. **HAYKIN**. *Neural networks: a comprehensive foundation* New York: Maxwell Macmillan, 1994.

11. **Heaton, Jeff.** Heaton Research. *Introduction to Neural Networks for Java, 2nd Edition.* [En línea] 2005. [Citado el: 22 de abril de 2009.] http://www.heatonresearch.com/book/programming-neural-networks-java-2.html. 1604390085.

12. **Ing. Acosta, Maria Isabel, Ing. Salazar, Harold y Ing. Zuluaga, Camilo A.** Universidad Tecnológica de Pereira. [En línea] [Citado el: 22 de enero de 2009.] http://ohm.utp.edu.co/neuronales/Capitulo2/Backpropagation/MomentumB.htm

13. **PÉREZ, J. M**. *Introducción a la Neurocomputación* España: Universidad de Málaga, [En línea] [Citado el: 9 de febrero de 2009] http://www.lcc.uma.es/~munozp/.

14. **Morales, L.** Instituto Nacional de Astrofísica Óptica y Electrónica. INAOE - Ciencias Computacionales. [En línea] [Citado el: 20 de febrero de 2009.] http://ccc.inaoep.mx/~lamorales/distribuidos/Sistemas_Distribuidos.pdf.

15. **Mortiz, J.** Lenguajes y Ciencias de la Computación. *Universidad de Malaga.* [En línea] [Citado el: 2 de abril de 2009.] http://www.lcc.uma.es/~jmortiz/archivos/Tema4.pdf.

16. **SKAPURA, D. M.** *Building Neural Networks.* 1996.

17. **SKAPURA, J. A. F. D. M.** *Neural Networks: Algorithms, Applications, and Programming Techniques* Addison-Wesley, 1991. ISBN 0201513765.

18. **Soria, Emilio y Blanco, Antonio.** Redes Neuronales Artificiales. *Autores Científico-Técnicos y Académicos (ACTA).* [En línea] [Citado el: 1 de noviembre de 2008.] http://www.acta.es/articulos_mf/19023.PDF.

19. **Villanueva, María del Rosario.** Visión Panorámica de las Redes Neuronales Artificiales. *Facultad de Ciencias Matemáticas - UNMSM* [En línea.] [Citado el: 14 de diciembre de 2008] http://matematicas.unmsm.edu.pe/weboperativa/congreso/ASSETS/DOCS/2210/VISION_P_DE__RN.PPT.

ANEXOS

1. Clases Implementadas.

A continuación se muestra la implementación de las clases necesarias para ejecutar el algoritmo diseñado para distribuir el entrenamiento del perceptrón multicapa sobre la Plataforma de Tareas Distribuidas Tarenal.

DataManager:

```
package jds;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.Vector;

import mlp.MultiLayerPerceptron;

public class Distributor extends DataManager
{
 private MultiLayerPerceptron Network;
 private ArrayList<File> Fragments;
 private double [][][][] PromNet;
 private String [] IPs;
 private int cantPfrag;
 private int fragcant;
 private int generated;
 private int processed;
 private int ins;
 private int outs;
 private double lrate;
 private double moment;
 private int cantcases;
 private double percent;
 private int cicles;


 public Distributor() throws Throwable
 {
  File flcfg = new File(PROBLEMDIRECTORY, "network.cfg");
  File fldat = new File(PROBLEMDIRECTORY, "trainning.dat");

  FileReader flrddat = new FileReader(fldat);
```

```
 Scanner cin = new Scanner(flcfg);
 BufferedReader cdat = new BufferedReader(flrddat);


 int layers = cin.nextInt();
 int [] neuronlayers = new int[layers];
 ins = cin.nextInt();
 for (int i = 0; i < neuronlayers.length; i++)
 {
  neuronlayers[i] = cin.nextInt();
 }
 outs = cin.nextInt();
 lrate = Double.parseDouble(cin.next());
 moment = Double.parseDouble(cin.next());
 cantcases = cin.nextInt();
 percent = Double.parseDouble(cin.next());
 cicles = cin.nextInt();
 cantPfrag = cin.nextInt();
 processed = 0;
 generated = 0;
 int frnum = 0;
 BufferedWriter cfrag = null;
 Fragments = new ArrayList<File>();
 for (int i = 0; i < cantcases; i++)
 {
  String readed = cdat.readLine();
  if(readed != null)
  {
   if (i % cantPfrag == 0)
   {
    File temp = new File(PROBLEMDIRECTORY, "frag" + frnum + ".dat");
    FileWriter flwfrag = new FileWriter(temp);
    if(cfrag != null)
    {
     cfrag.close();
    }
    cfrag = new BufferedWriter(flwfrag);
    Fragments.add(temp);
    frnum++;
   }
   cfrag.write(readed + "\n");
  }
 }
 if(cfrag != null)
 {
  cfrag.close();
 }
 fragcant = frnum;
 PromNet = new double [fragcant][][][];
 IPs = new String [fragcant];
 Network = new MultiLayerPerceptron(ins,outs,neuronlayers);
}
```

```
public void adjustGranularity(double times) throws Throwable
{

}

public void closeResources() throws Throwable
{

}

@SuppressWarnings("unchecked")
public Vector generateWorkUnit(ClientInfo clientInfo) throws Throwable
{
 if (generated == fragcant)
 {
  return null;
 }
 Vector send = new Vector();
 send.add(Network.clone());
 send.add(Fragments.get(generated));
 send.add(Integer.valueOf(ins));
 send.add(Integer.valueOf(outs));
 send.add(Double.valueOf(lrate));
 send.add(Double.valueOf(moment));
 send.add(Integer.valueOf(cantPfrag));
 send.add(Double.valueOf(percent));
 send.add(Integer.valueOf(cicles));
 send.add(Integer.valueOf(generated));
 IPs[generated] = clientInfo.getIP();
 generated++;
 return send;
}

public String getStatus() throws Throwable
{
 String status = "";
 for (int i = 0; i < generated; i++)
 {
  status += Fragments.get(i).getName() + " --- " + IPs[i] + "\n";
 }
 return status;
}

@SuppressWarnings("unchecked")
public boolean processResults(Long uid, Vector results) throws Throwable
{
 int compl = ((Integer)results.get(0)).intValue();
 PromNet[compl] = ((MultiLayerPerceptron)results.get(1)).getStructure();
 processed++;
 if (processed == fragcant)
 {
  double [][][] weights = Network.getStructure();
  for (int i = 0; i < weights.length; i++)
  {
   for (int j = 0; j < weights[i].length; j++)
```

```
    {
     for (int k = 0; k < weights[i][j].length; k++)
     {
      weights[i][j][k] = 0;
      for (int m = 0; m < fragcant; m++)
      {
       weights[i][j][k] += PromNet[m][i][j][k];
      }
      weights[i][j][k] /= fragcant;
     }
    }
   }
   Network.setStructure(weights);
   File flnet = new File(PROBLEMDIRECTORY, "trainned.mlp");
   FileOutputStream flosnet = new FileOutputStream(flnet);
   ObjectOutputStream cnet = new ObjectOutputStream(flosnet);
   cnet.writeObject(Network);
   cnet.close();
   return true;
  }
  else
  {
   return false;
  }
 }
}
```

Task:

```
package jds;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.PrintWriter;
import java.util.NoSuchElementException;
import java.util.StringTokenizer;
import java.util.Vector;

import mlp.DataSet;
import mlp.MultiLayerPerceptron;

public class Calculator extends Task
{
 @SuppressWarnings("unchecked")
 public Vector processUnit( Vector workUnit ) throws Throwable
 {
  Vector result = new Vector();
  MultiLayerPerceptron net = ((MultiLayerPerceptron)workUnit.get(0));
  File fldat = ((File)workUnit.get(1));
  int ins = ((Integer)workUnit.get(2)).intValue();
```

```
  int outs = ((Integer)workUnit.get(3)).intValue();
  double lrate = ((Double)workUnit.get(4)).doubleValue();
  double moment = ((Double)workUnit.get(5)).doubleValue();
  int cantcases = ((Integer)workUnit.get(6)).intValue();
  double percent = ((Double)workUnit.get(7)).doubleValue();
  int cicles = ((Integer)workUnit.get(8)).intValue();

  FileReader flrddat = new FileReader(fldat);
  BufferedReader cdat = new BufferedReader(flrddat);

  double[][] inputs = new double[cantcases][ins];
  double[][] outputs = new double[cantcases][outs];

  for (int i = 0; i < cantcases; i++)
  {
   String line = cdat.readLine();
   StringTokenizer st = new StringTokenizer(line," \t\n\r\f:");
   for (int j = 0 ; j < outs; j++)
   {
    outputs[i][j] = Double.parseDouble(st.nextToken());
   }

   int num = Integer.parseInt(st.nextToken());
   double value = Double.parseDouble(st.nextToken());
   boolean end = false;

   for (int j = 0 ; j < ins; j++)
   {
    if (j + 1 < num || end)
    {
     inputs[i][j] = 0;
    }
    else
    {
     inputs[i][j] = value;
     try
     {
      num = Integer.parseInt(st.nextToken());
      value = Double.parseDouble(st.nextToken());
     }
     catch (NoSuchElementException e)
     {
      end =true;
     }

    }
   }
  }
  DataSet teacher = new DataSet(inputs,outputs,lrate,moment);

  File flout = new File(PROBLEMDIRECTORY, "result" + (Integer)workUnit.get(9) +
".out");
  FileOutputStream flosout = new FileOutputStream(flout);
  PrintWriter cout = new PrintWriter(flosout);
```

```
  int iter = 0;
  int good = 0;
  double globalerror = 0;
  long clock = System.currentTimeMillis();
  do
  {
   double error = teacher.train(net);
   System.out.println(error);
   cout.println(error);
   globalerror += error;
   if (error < percent)
   {
    good++;
   }
   else
   {
    good = 0;
   }
   iter++;
   cicles--;
  }
  while(good < 10 && cicles > 0);
  clock -= System.currentTimeMillis();
  clock = Math.abs(clock);
  long [] time = new long[3];
  clock /= 1000;
  time[2] = clock % 60;
  clock /= 60;
  time[1] = clock % 60;
  clock /= 60;
  time[0] = clock;
  globalerror /= iter;
  result.add(workUnit.get(9));
  cout.println(globalerror);
  cout.println("Time: " + time[0] + ":" + time[1] + ":" + time[2]);
  cout.close();
  result.add(net);
  return result;
 }
}
```