35
29.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

CAMPUS ARAGÓN

"RED NEURONAL DISTRIBUIDA IMPLEMENTADA EN LENGUAJE JAVA"

T E S I S

QUE PARA OBTENER EL TITULO DE

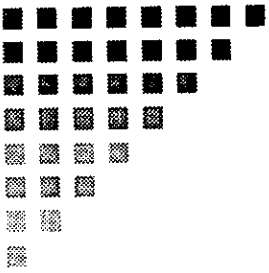
INGENIERO EN COMPUTACIÓN

P R E S E N T A:

LILIANA LOPEZ GUZMAN

ASESOR: FIS. FERNANDO ANGELES URIBE

MÉXICO 1998



Mejor es la sabiduría que la fuerza... Las palabras del sabio escuchadas en quietud, son mejores que el clamor del señor entre los necios. Mejor es la sabiduría que las armas de guerra. Ec9.16.

AGRADECIMIENTOS

A DIOS:

Por estar a mi lado en todo momento. Por darme las fuerzas para librar los obstáculos y vencer las adversidades. Por realizar mis sueños en compañía de aquellos que quiero.

A MI PADRE:

J. Francisco López Quezada por ser esa persona que siempre me apoya, que siempre me brinda su respaldo y cariño. Por que ha impulsado con su gran amor y carácter todos mis sueños y aspiraciones. Por la responsabilidad que inculco en mí, y empeño de hacer las cosas bien. Por mostrarme que de los errores se aprende para ser mejor. Por su esfuerzo y paciencia y por todo lo que me ha dado, solo puede decirle ¡Muchas gracias papá!

A MI MADRE:

Lilia Guzmán Hernández por esa lucha, esfuerzo y entrega. Por la alegría y momentos difíciles que hemos pasado juntas. Por los desvelos y sacrificios, pero en especial por todas las cosas que me enseño, al inculcar en mí el ejemplo de trabajo y generosidad. A ella mi más sincero agradecimiento porque sin ella este y muchos otros logros no tendrían sentido. Por darme la vida y enseñarme a vivirla con amor y empeño, para luchar por lo que se quiere y se ama. Por esa entrega total de ella y de mi padre, quiero que sepan que su esfuerzo y sacrificio no ha sido en vano. Y sobre todo ¡Muchas gracias por ser mi madre!

A MI ASESOR:

Muy en especial quiero agradecer al Fis. Fernando Angeles U. por su paciencia, inteligencia, apoyo y dedicación para realizar el presente trabajo. Por darme su tiempo incondicionalmente. Por esos consejos que contribuyeron a mi formación profesional. y sobre todo por ser más que un profesor un buen amigo.

A LA UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO:

En especial agradezco a nuestra Máxima Casa de Estudios por albergarnos en sus aulas y habernos formado en esta hermosa profesión. Y ala Escuela Nacional de Estudios Profesionales Aragón por darme el privilegio de formar parte de su comunidad para hacer posible mi formación profesional. Al Instituto de Astronomía por darme la oportunidad de desarrollar este trabajo de Tesis en sus instalaciones proporcionándome su apoyo académico.

A MIS PROFESORES:

Por contribuir a mi formación académica, por sus enseñanzas y consejos de superación. Por inyectar en mi el entusiasmo para concluir mis estudios profesionales. Por ese ejemplo de entrega total a la profesión.

ING. ARMANDO BOLAÑOS G.:

Por estar conmigo en los momentos más difíciles, por compartir mis triunfos y fracasos. Por su motivación y trabajo de equipo, por ese amor que impulsa a romper barreras y alcanzar sueños. Por ser amigo y pareja. Y por creer en mí. Por llegar a mi vida y compartir la suya conmigo. Te amo.

A MIS HERMANOS:

Quiero hacer partícipes de este trabajo de tesis a mis hermanos, Adri, Carmen y Javier, y agradecerles su amor y apoyo que me han brindado al compartir conmigo gran parte de su vida, deseo que sepan lo orgullosa que me siento de ser su hermana y les deseo todo el éxito del mundo. Sepan que siempre pueden contar conmigo.

A MIS AMIGOS:

Esther, Elia, Bertha .... por brindarme el apoyo incondicional haciéndome ver mis defectos y virtudes, quienes han estado conmigo en las buenas y las malas, por esos momentos que hemos pasado juntos.

A MIS ABUELITAS:

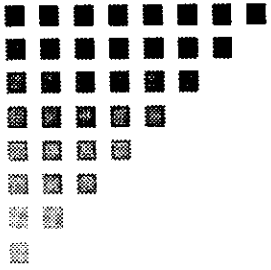
A Guadalupe Quezada, por enriquecerme con su experiencia y constante cariño. Y a Guadalupe Hernández, por su dedicación hacia el trabajo, por su esfuerzo de seguir adelante, aun en contra de las enfermedades y adversidades, por ese entusiasmo de disfrutar la vida, y compartir con aquellos que padecen de soledad y angustia.

A MIS TIOS:

Por que en el comienzo de mi adolescencia, estuvieron cerca de mí, alimentándome con palabras de apoyo, me impulsaron a alcanzar mis objetivos, por poner a mi alcance sus conocimientos. En especial a mi tía Amparo por su ejemplo de tenacidad hacia la vida, por esa fuerza de corazón que no se rinde y a mi tío Salvador por darme la oportunidad de tener un desarrollo económico que auxilio en mis estudios.

En general:

A todas aquellas personas que me alentaron de alguna u otra manera a lo largo de mi formación profesional.



INTRODUCCIÓN

"¡Supera la barrera que existe en tu mente! Cuando el hombre no vuelve a intentarlo, es como una paralisis temporal o permanente. Es una actitud pasiva hacia si mismo y su medio" A.L.C

Una experiencia de vital importancia para un profesionista es la realización de un proyecto que refleje los conocimientos adquiridos durante los años de estudios, así como a su vez el aprendizaje que permite el desarrollo de una tesis, y que refleja la satisfacción personal.

La satisfacción personal de un profesionista radica en gran parte en realizar proyectos de utilidad, para facilitar o resolver problemas en distintas áreas, la elección de este tema de tesis surge como propuesta para solucionar la problemática que implica el disminuir la deformación en imágenes observacionales de los cuerpos celestes.

A lo largo de muchos años los astrónomos han utilizado diferentes formas de mejorar la resolución en las imágenes, la cual esta limita por los efectos de la *turbulencia atmosférica*, algunos de los cuerpos celestes que se encuentran en el espacio emiten luz, en forma de onda esférica a lo que se denomina *frente de onda*, estos cuerpos se encuentras a miles de kilómetros lejos de la tierra, por lo que al alcanzarnos, este frente de onda es plano[1], al pasar por las diferentes capas atmosféricas, aunado a ello los cambios de temperatura del aire, el frente de onda plano se deforma, distorsionando la imagen, la cual se observar como un punto que titila en el cielo, aun cuando se observe por el mejor telescopio esta imagen se ve como una mancha borrosa.

Algunos creen que la mejor manera de tener imágenes de alta calidad, es el construir telescopios con espejos primarios más grandes, otros astrónomos afirmaban que para evitar la turbulencia atmosférica se tendría que construir un telescopio que se encontrara fuera de la atmósfera, estas teorías han traspaso barreras, un ejemplo claro de ello, es el Telescopio Espacial Hubble que se puso en órbita 1990, por la NASA[2], este telescopio se encuentra fuera de la atmósfera de la tierra lo cual elimina la turbulencia atmosférica y en consecuencia produce imágenes más nítidas de los cuerpos celestes.

Hoy en día existen telescopios terrestres que mejoran la calidad de las imágenes, tomadas por el telescopio Espacial Hubble, ya que la ciencia avanza aceleradamente y para hacer modificaciones al telescopio Hubble se requiere de costos muy elevados, por lo que no se le hacen las adaptaciones necesarias para continuar a la vanguardia astronómica, por ello el Instituto de Astronomía de la UNAM realiza investigaciones para la construcción de un telescopio (TIM)[3] el cual estaría formado por un espejo primario que se divide en 19 espejos hexagonales, para poder corregir el error en la imagen provocado por la

[1] Por lo que se le llama frente de onda plano.
[2] ASA (National Aeronautics and Space Administration)
[3] TIM (Telescopio Infrarrojo Mexicano)

turbulencia atmosférica, esta corrección esta delimitada por el movimiento de 19 actuadores que se encuentran en cada subespejo hexagonal, cada uno de estos subespejos se mueve de la forma que la computadora lo indica, para observar la imagen sin turbulencia.

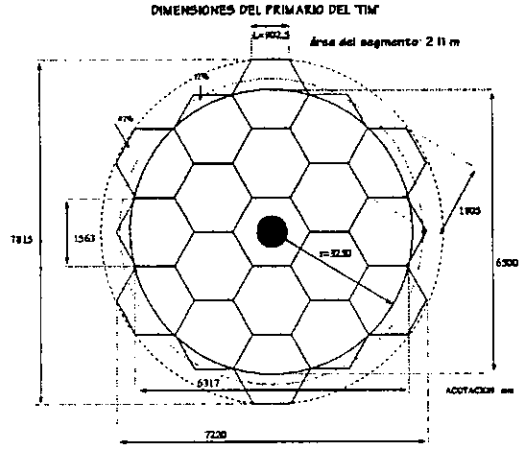


Figura Espejo Primario para el Telescopio TIM, con 19 subespejos hexagonales.

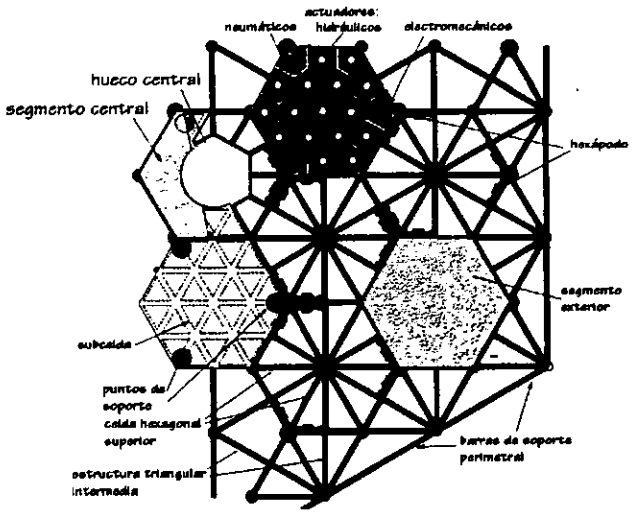


Figura Mecanismo del espejo primario.

El frente de onda del cuerpo celeste es distorsionado al pasar por la atmósfera, este entra al telescopio siendo captado por el espejo primario, el cual refleja el muestreo del frente de onda al espejo secundario para que este posteriormente sea captado por un grupo de sensores, que transmitirá la información a la computadora, la cual procesara la información recibida reconstruyendo el frente de onda y determinando el movimiento de cada uno de los actuadores que se encuentran en los subespejos, este movimiento debe ser cada 10 milisegundos, para tener una imagen más nítida y de calidad.

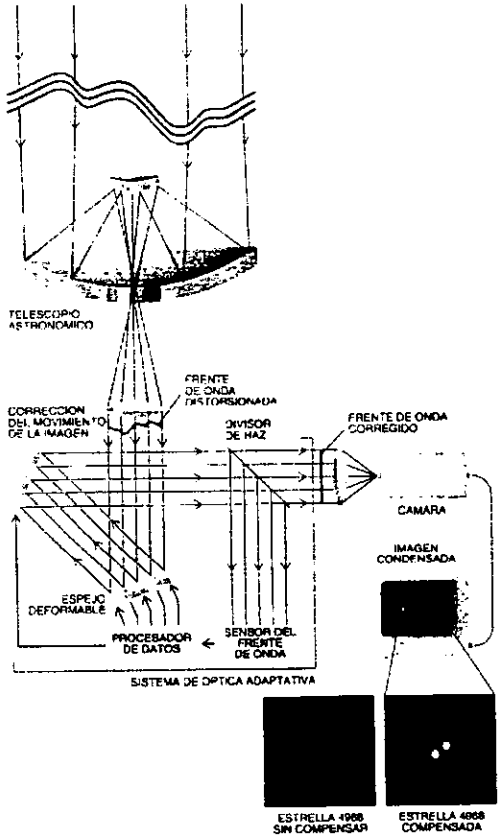


Figura Del sistema adaptado al Telescopio TIM.

Quizá la pregunta ahora sea ¿dónde aplicamos Redes Neuronales? Como se puede observar en lo anterior la modificación de los actuadores debe ser cada 10 milisegundos aproximadamente, por lo que reconstruir el frente de onda, implica la resolución de ecuaciones Laplacianas, dicho de otra manera esto requiere de una numerosa serie de operaciones de punto flotante para el procesador, lo cual llevaría demasiado tiempo, en este caso se requiere de un procesador lo suficientemente rápido para realizar dichas operaciones, con el consecuente costo elevado. Por lo que se propone usar una Red Neuronal que se encuentre distribuida, para acelerar él cálculo del resultado, la Red Neuronal se compondrá de tres facetas, las fases de entrenamiento y prueba en las cuales se utilizaran imágenes distorsionadas e imágenes deseadas, a través de la simulación de frentes de onda, con lo que la red habrá aprendido a reconocer una imagen distorsionada, en la tercera etapa de trabajo la cual servirá para reconstruir la imagen nítida del cuerpo celeste, de esta manera calculará el movimiento de cada uno de los actuadores.

El Capítulo I muestra una expectativa general de Redes Neuronales, así como los problemas que pueden resolver. Ya que los sistemas Neuronales Computacionales son el reflejo de numerosos estudios, que conllevan a la imitación del funcionamiento cerebral, y con ello la inmensidad de problemas que nos podemos resolver con dichos algoritmos.

El Capítulo II muestra las ventajas de los sistemas de Computo Concurrente así como los sistemas de Computo Distribuido, los dos primeros Capítulos permiten tener una visión clara del análisis subsecuente de esta tesis.

El siguiente Capítulo muestra las ventajas del ¿por qué? Hablamos de Computo Concurrente y Distribuido al aplicarlo a una red neuronal.

El Capítulo IV que muestra la Programación Orientada a Objetos (P.O.O.), que tiene inicio en los años 70's, a partir de la necesidad de mejorar la creación de procesos, mantenimiento y empleo de software. Pero es en la década de los 90's cuando empieza a tener mayor auge, ya que las características de la mayoría de los sistemas de los 90's son complejidad, diversidad, e interconectividad. Históricamente, la creación de programas implicaba la definición de procesos que actuaban sobre un conjunto independiente de datos, la P.O.O cambia el centro de atención de los procesos de programación a los objetos, módulos autocontenidos que incluyen tanto los datos como los procedimientos que actúan sobre los datos. Como resultado del avance que ha tenido la P.O.O se encuentra la facilidad de ampliar los programas, la facilidad de mantener dichos programas sin mayor tiempo-costo, fácil de utilizar y flexible, así como el de realizar programas para aplicaciones más complejas, incrementar

la funcionalidad que puede incorporarse a las aplicaciones. Los métodos de P.O.O posibilitan el que los usuarios puedan ascender a los actuales tipos de datos por medio de plataformas de computación heterogéneas.

Bajo el enfoque de P.O.O, se puede controlar la complejidad de los programas y convertir en creciente entorno de sistemas en aplicaciones más útiles, modulares e intercambiable, dicho enfoque rompe con los métodos tradicionales que aplican procedimientos estructurados, los objetos pueden ser distribuidos y reutilizados, con ello un programa complejo se simplifica poniendo su atención en objetos y no en procesos para diferentes datos. La ventaja de utilizar lenguajes Orientados a Objetos importar y exportar objetos así como el procesamiento distribuido.

Este Capítulo da la pauta a la elección del lenguaje a elegir para el Diseño de Redes Neuronales, no bastaba con un lenguaje de programación de redes neuronales, se requería de algo más, ya que las redes neuronales contienen cierto número de elementos o neuronas, que no se encuentran preestablecidos y en determinado problema crecen dinámicamente, por ello se requería un lenguaje que soportara el entorno de la P.O.O, así mismo, que permitiera utilizar Sistemas Distribuido y Concurrentes.

¿Qué lenguaje decidir? Actualmente existen varios lenguajes de programación que soportan el entorno de la P.O.O, pero por su complejidad en la abstracción de la orientación a objetos, no cualquiera podría ser el destinado a solucionar problemas de Redes Neuronales, de ahí el decir utilizar el lenguaje Java, que se analiza en el Capítulo V, y que facilita el diseño de Sistemas Distribuidos y Concurrentes.

Con ello no quiere decir que Java sea el único, pero si el mejor, que es económico y esta a nuestro alcance a comparación de otros. Dichas características permiten fácilmente desarrollar un Red Neuronal bajo la implementación en Java.

Los dos Capítulos subsecuentes reflejan el análisis del lenguaje en la aplicación así como las pruebas de desempeño de la implementación. El siguiente Capítulo muestra las conclusiones a las que llegue de acuerdo al objetivo.

El objetivo de este trabajo aunado a los que ya se mencionaron es el de aplicar la importancia que tiene una Red Neuronal Distribuida así como la facilidad y ventajas de implementar dicha Red Neuronal en lenguaje Java.

La interrelación que puede existir entre una Red Neuronal y el computo distribuido concurrente es reflejo de la importancia de la ingeniería en resolución de problemas.

La relación que existe de una Red Neuronal programada en Java, permite ampliar la facilidad de procesos simultáneos, para la solución de problemas. Esto como apoyo de las multitareas y tiempo para su realización.

Este proyecto surge de la necesidad de elegir la temática a desarrollar que refleje los conocimientos obtenidos durante el proceso de estudios de la carrera de Ingeniería en Computación, lo cual me lleva a la conclusión que lo importante es satisfacer las inquietudes personales, así como el adquirir conocimientos en el desarrollo de este proyecto, y complementar mi desarrollo profesional.

OBJETIVO

a) Aplicar enfoque de Programación Orientada a Objetos a Redes Neuronales.

b) Realizar una red Neuronal en ambiente Cliente-Servidor, es decir, que se encuentre distribuida, para reducir tiempo de reconocimiento

c) Usar lenguaje Java para la Distribución de la Red Neuronal, a partir del enfoque de la Programación Orientada a Objetos.

HIPÓTESIS

Bajo el enfoque de la Programación Orientada a Objetos se desarrollan con gran facilidad Redes Neuronales que permiten varía su crecimiento sin tener que reprogramar nuevamente el código, y si a esto aumentamos que la Red se ejecute en un ambiente Cliente-Servidor, el tiempo de reconocimiento se divide en él número de servidores que se tengan, logrando resultados eficientes.

OBJETIVO

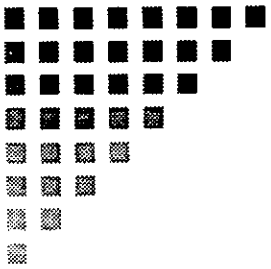
a) Aplicar enfoque de Programación Orientada a Objetos a Redes Neuronales.

b) Realizar una red Neuronal en ambiente Cliente-Servidor, es decir, que se encuentre distribuida, para reducir tiempo de reconocimiento

c) Usar lenguaje Java para la Distribución de la Red Neuronal, a partir del enfoque de la Programación Orientada a Objetos.

HIPÓTESIS

Bajo el enfoque de la Programación Orientada a Objetos se desarrollan con gran facilidad Redes Neuronales que permiten varía su crecimiento sin tener que reprogramar nuevamente el código, y si a esto aumentamos que la Red se ejecute en un ambiente Cliente-Servidor, el tiempo de reconocimiento se divide en él número de servidores que se tengan, logrando resultados eficientes.



CAPÍTULO I

INTRODUCCIÓN A REDES NEURONALES

"Un hombre que no sueña está vacío, no tiene ilusiones, ideales, retos ni objetivos, ha dejado de pensar, de desear, sólo vive de instintos y de mensajes del medio. A esas personas se les clasifica como mediocres, masificados socialmente y algunos los llaman cobardes..." A.L.C.

I.1 Definición de Red Neuronal

Si bien las computadoras han ayudado a acelerar cálculos y manejos de información más bien repetitivos y en esos campos han superado la eficiencia de los humanos, hay tareas donde es requerido un análisis diferente al algorítmico al cual están sujetas. Particularmente problemas que pueden parecernos tan sencillos como el reconocimiento de imágenes, voz, toma de decisiones, etc. no han sido reproducidos por una computadora a un nivel tan eficiente como lo hace el cerebro. Claramente, el resolver este tipo de problemas nos acerca cada vez más a lo que llamaríamos un programa "inteligente". El método que usa nuestro cerebro más que algorítmico es heurístico, lo que implica, entre otras cosas, que:

1. Siempre existirá una respuesta aun cuando una entrada no haya sido considerada con anterioridad.

2. El sistema tiene la flexibilidad suficiente para tomar decisiones por sí mismo.

Para poder emular este comportamiento, es necesario que analicemos la estructura física del cerebro, sus constituyentes mínimos y comportamiento de los mismos.

La materia gris del cerebro, responsable principal de los procesos intelectuales, está compuesta por células llamadas neuronas, las cuales se encargan de realizar el proceso de información en el mismo. En la figura I.1.1, se muestra la estructura de las neuronas. La membrana de la neurona separa el plasma intracélular del fluido extracélular o también llamado intersticial, que se encuentra fuera de la célula. La membrana actúa de tal manera que mantiene una diferencia de potencial entre el fluido intracélular y el fluido extracélular. En la figura se muestra el axón con una cubierta que se denomina vaina de mielina. Esta capa aislante es interrumpida en varios puntos por los nodos de Ranvier.

Las sinapsis es el efecto de conectar a la neurona con otras neuronas. La conexión se produce en distintos lugares de la célula. Los impulsos nerviosos que pasan a través de las neuronas que están conectadas, pueden dar lugar a cambios locales en el potencial del cuerpo de la célula receptora. Estos potenciales se denominan también potenciales de entrada que pueden propagarse por el cuerpo principal de la célula. Pueden ser excitatorios (que hacen disminuir la polarización de la célula) o bien inhibitorios (que incrementa la polarización de la célula). Los potenciales de entrada se suman en el montículo del axón. Si la suma es mayor que un cierto umbral, se genera un potencial de acción, este potencial viaja de lo largo del axón, para transmitir información a otras células, dando lugar a la comunicación entre ellas, como resultado de la liberación de unas sustancias llamadas neurotransmisores.

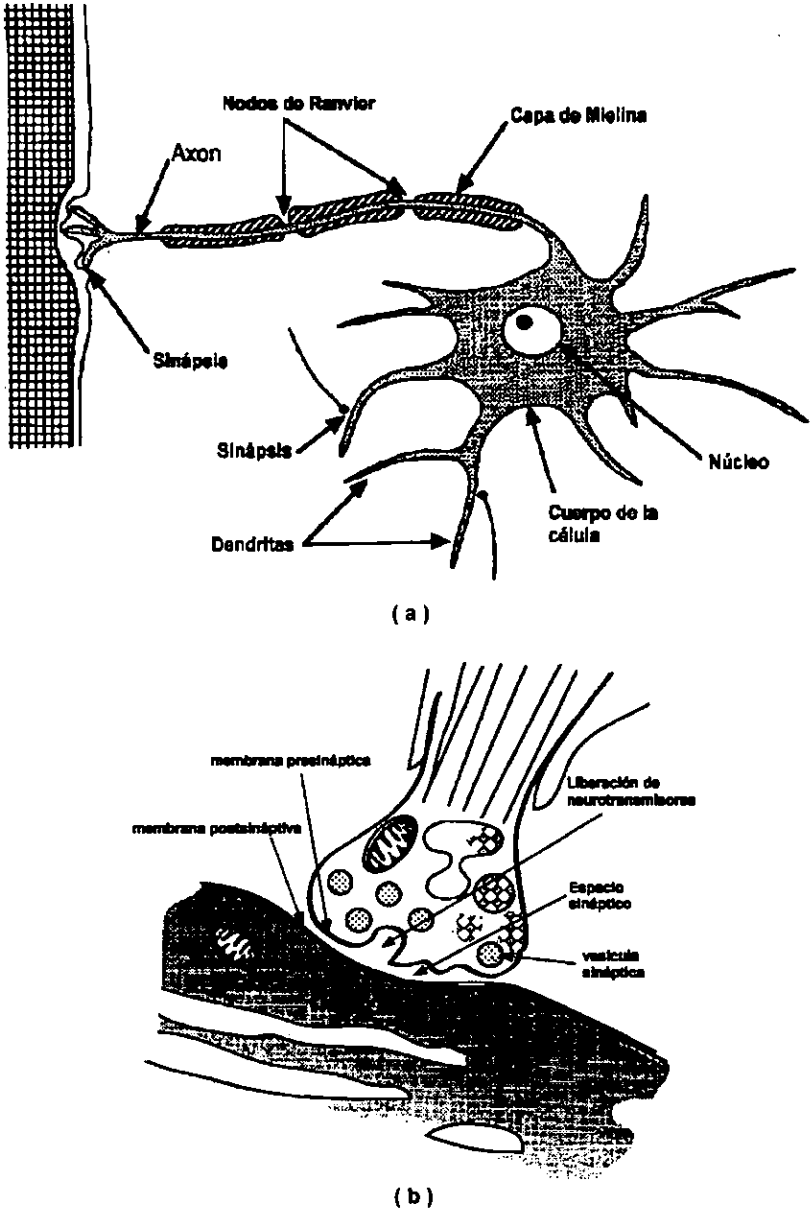


Figura I.1.1 (a) Muestra la estructura de una célula nerviosa. (b) Sinapsis

¿Cómo se lleva a cabo el proceso de información? Las sinapsis controlan el paso de los impulsos de una célula a otra. Se ha demostrado experimentalmente que el aprendizaje esta fuertemente correlacionado con la modificación de esas sinapsis, por lo que el cerebro representa en sí una estructura flexible que puede automodificarse para aceptar nueva información. Si el funcionamiento de los elementos cerebrales es tan sencillo (relativamente) ¿qué nos impide el simular la totalidad de éste órgano? El truco del cerebro es el número de estos elementos: cerca de 12,000 millones de neuronas, cada una recibe de 10,000 a 80,000 conexiones[1], que se encargan de realizar la función que llamamos "pensar", procesando información independientemente unas de otras y en paralelo, y como es fácil comprender, no existe sistema de cómputo en este planeta de igualar tal hazaña, por lo menos hasta este momento.

Sin embargo, es posible emular algunas de las funciones de reconocimiento modelando el comportamiento de las neuronas y definiendo una topología de conexiones usando herramientas computacionales o electrónicas. Estos modelos han recibido el nombre de sistemas neuronales artificiales o simplemente redes neuronales.

En la figura I.1.2 se encuentra la forma generalizada de una neurona artificial, que recibe frecuentemente el nombre de nodo o elemento de procesamiento, donde las conexiones de entrada se representan en forma de flechas procedentes de otros elementos de procesamiento. Cada conexión de entrada tiene asociada una cantidad W_{ij} que se denomina *peso*. Hay un único valor de salida que se puede aplicar a otras unidades.

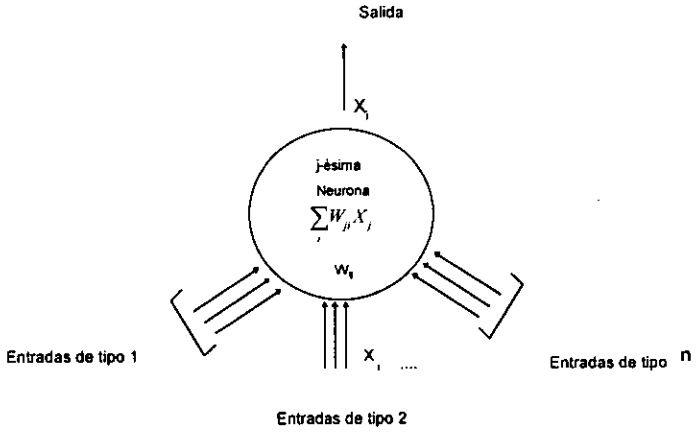


Figura I.1.2 Forma generalizada de una red neuronal.

[1] Estos valores son aproximados, ya que en el campo científico aun no se encuentra determinado el valor de neuronas y conexiones dendriticas, por lo que muchos autores manejan diferentes valores.

Las redes neuronales artificiales relativamente simples consisten de una serie de elementos de entrada, denominados *capa de entrada*, una capa de neuronas intermedia (u oculta) y una capa de neuronas de salida. Como se muestra en la figura I.1.3 existen interconexiones entre todos los elementos de una capa y todos los elementos de la siguiente. Estas interconexiones tienen un cierto *peso* que es caracterizado por un valor numérico; si el valor es positivo se considera como una conexión excitatoria, si el valor es negativo se considera como una conexión inhibitoria.

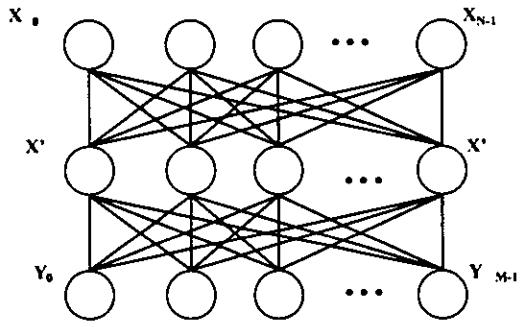


Figura I.1.3 Interconexiones entre elementos de las capas.

La base de la mayoría de los modelos de redes neuronales fue tomada partiendo del ejemplo de los circuitos básicos que se encuentran en el sistema nervioso central, donde se muestran los principios de convergencia y divergencia en la circuitería neuronal, cada neurona envía impulsos a muchas neuronas (divergencia) y recibe impulsos procedentes de muchas neuronas (convergencia).

Los sistemas neuronales biológicos no nacen con todo el conocimiento y la capacidad que llegan a tener eventualmente. El proceso de aprendizaje tiene lugar a lo largo de un cierto período de tiempo, durante el cual se modifica de alguna manera la red para incluir la nueva información, de la misma manera sucede en los sistemas de redes computacionales. El poder de aproximación de una red neuronal no consiste necesariamente en la elegancia de una cierta solución, sino más bien en la generalidad de la red para ser capaz de hallar sus propias soluciones a problemas concretos, alimentándola únicamente con ejemplos de comportamiento deseado. El proceso de entrenamiento de la red es simplemente una forma de codificar información acerca del problema que hay que resolver. Es decir, dicho proceso es cuestión de modificar los pesos que codifican las relaciones de entrada-salida deseadas. El concepto bajo el cual fueron desarrollados los métodos de aprendizaje, para las redes neuronales, es el *principio de difusión mínima* (minimal disturbance principle), el cual sugiere que durante el entrenamiento es aconsejable introducir nueva información.

I.2 Tipos de Redes Neuronales

Las arquitecturas de redes neuronales han recibido el nombre de *arquitecturas conexionistas*. Estas se han concebido para imitar el funcionamiento del cerebro humano, ya que se caracterizan por tener un gran número de elementos muy simples que procesan la información de modo similar a las neuronas nerviosas y conexiones entre ellos; como anteriormente se menciono los pesos en las conexiones codifican el conocimiento de una red.

1950 y 1960 fueron años importantes en el desarrollo de las redes neuronales con el auge del Perceptrón y Adaline.

Este trabajo no tiene el fin concreto de estudiar detalladamente todos los modelos conexionistas, por lo que sólo se mencionaran algunos como el perceptrón, red Hopfield y Retropropagación (backpropagation).

Perceptrón.

Frank Rosenblatt fue uno de los primeros en realizar estudios que harían gran aportación dentro del campo de las redes neuronales, entre sus aportaciones se encuentra el desarrollo de un clasificador de redes neuronales artificiales llamado *Perceptrón*, en el cual se componía de unidades de procesamiento que fueron organizadas en capas de neuronas con conexiones entre una capa y la siguiente, las interconexiones se encuentran determinadas por pesos fijos asociados a las neuronas. El perceptrón aprende usando un ajuste de pesos en las conexiones, dando una respuesta la cual determina a que clase pertenece la entrada.

El perceptrón fue el resultado de un primer intento de simular la computación neuronal para llevar a cabo tareas complejas.

Rosenblatt pudo demostrar que el perceptrón era capaz de clasificar tramas[2] correctamente, en el cual cada clase estaba formada por tramas que eran similares unas con otras en algún sentido. Los trabajos realizados por Rosenblatt dieron lugar a que se demostrase un importante resultado conocido con el nombre de Teorema de convergencia del perceptrón, el cual afirma que "si la clasificación puede ser aprendida por el perceptrón, entonces el procedimiento será aprendido en número finito de ciclos de entrenamiento para un perceptrón con una unidad de respuesta que está aprendiendo a diferenciar tramas de dos clases diferentes." (12)

[2] Se define como trama a un conjunto de puntos o una área cuadricular que contienen una información, donde si observáramos alguno de los puntos individualmente, no comprenderíamos que a que información contiene, pero en conjunto muestran un objeto.

El tipo más simple de perceptrones de una sola capa sirvió para probar la convergencia de un algoritmo de aprendizaje, es decir, el modo de cambiar los pesos interactivamente hasta encontrar el conjunto de pesos deseados. Mucha gente expresó un gran entusiasmo con dicho modelo. El diagrama del perceptrón idealizado por Rosenblatt aparece en la figura I.2.1.

En 1969 surge un libro llamado Perceptrones : *An Introduction to Computational Geometry*, que había sido escrito por Marvin Minsky y Seymour Papert, donde se presento un análisis detallado del perceptrón, en términos de sus capacidades y limitaciones. Una de los temas del libro trata sobre las restricciones que se tiene en algunas clases de problemas para los cuales es inadecuado el perceptrón, es decir, los perceptrones de dos capas sólo pueden distinguir tramas si las tramas son linealmente separables. Hay muchos problemas de clasificación que no son linealmente separables, por lo que esta condición impone unos límites bastantes restrictivos al aplicar el perceptrón. (12)

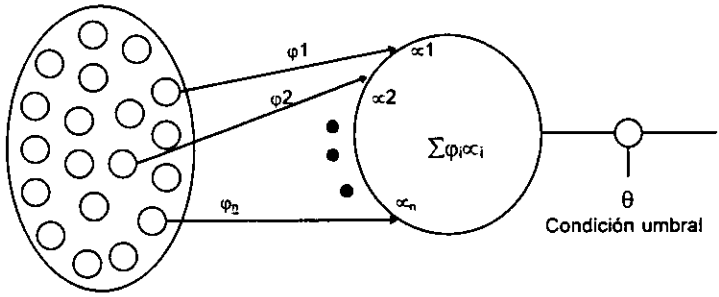


Figura I.2.1 Diagrama del Perceptrón

Debido a estas limitaciones de los perceptrones, se dio lugar a una considerable controversia que tuvo como resultado que el campo de las redes neuronales artificiales fuese abandonado casi por 30 años, salvo por algunos pocos investigadores, que continuaron los estudios en este campo.

Uno de los ejemplos más sencillos de problemas que no se pueden resolver mediante un perceptrón es el problema XOR. Este problema se ilustra en la figura I.2.2

La función de salida de la unidad de salida que se muestra en la figura es una función umbral.

$$f(total) = \begin{cases} 1 \text{ si total} >= \theta \\ 0 \text{ si total} < \theta \end{cases}$$

En donde θ es el valor *umbral*[3]. Este tipo de nodo se denomina unidad de umbral lineal.

La activación del nodo de salida es :

$$total = W_1X_1 + W_2X_2$$

por lo tanto el valor de la salida es :

$$o = f(total) = \begin{cases} 1 \text{ si } W_1X_1 + W_2X_2 >= \theta \\ 0 \text{ si } W_1X_1 + W_2X_2 < \theta \end{cases}$$

X_1	X_2	Salida
0	0	0
0	1	1
1	0	1
1	1	0

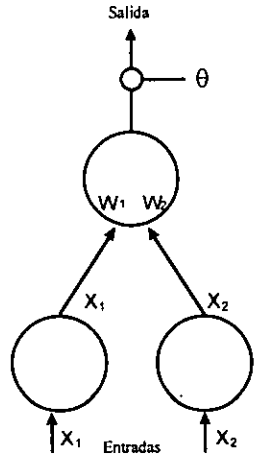


Figura I.2.2. Esta red de dos capas tiene dos nodos en la capa de entrada, con unos valores de entradas X_1 y X_2 que pueden adoptar los valores 0 ó 1. Desearíamos que la red fuera capaz de responder a las entradas de tal modo que las salidas sean la función XOR de las entradas, como se indica en la tabla.

El principal objetivo de este problema consiste en seleccionar, valores para los pesos, de tal manera que todos los valores de los pesos de las entradas den lugar a los valores de salida de acuerdo a la tabla de la función XOR. Pero esto no es posible; ya que la ecuación $\theta = W_1X_1 + W_2X_2$ determina una línea en el plano X_1, X_2.

En la figura I.2.3 se ilustra este plano, junto con cuatros puntos que son las posibles entradas a la red.

[3] Umbral : valor mínimo de un estímulo para producir una reacción.

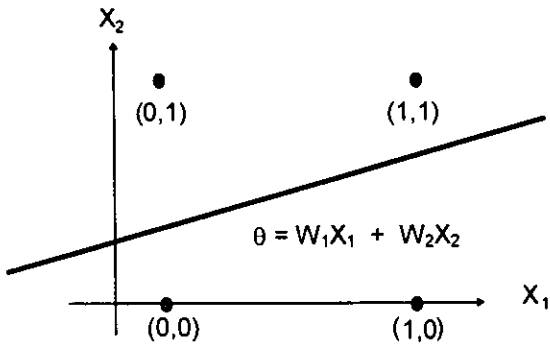


Figura I.2.3. Plano X_1, X_2, con los cuatro puntos que forman los vectores de entrada para el problema XOR.

En la figura podemos ver que la ecuación $\theta = W_1X_1 + W_2X_2$, determina una línea que parte el plano en dos regiones distintas como máximo. De acuerdo a la función XOR se pueden clasificar los puntos en dos clases : una región como perteneciente a la clase que posee una salida de uno, y los de la otra región como pertenecientes a la clase que posee una salida nula; sin embargo no hay ninguna forma de precisar la posición de la línea de manera que los dos puntos pertenecientes a cada clase se encuentren en la misma región separados por la línea. Por lo que el perceptrón no es capaz de resolver correctamente la función XOR.

Este problema es resuelto por otro tipo de redes que pueden aprender la diferencia entre varias clases, en un espacio de n-dimensiones denominados *hiperplanos*. Los hiperplanos son objetos de (n -1) dimensiones. (Los espacios n_dimensionales suelen recibir el nombre de *hiperespacios*).

La adición de dos capa oculta ó capa intermedia, dan a la red la flexibilidad necesaria para resolver el problema. De hecho, la existencia de esta capa oculta nos da la capacidad de construir redes que puedan resolver problemas mayor complejidad. Este sencillo ejemplo no implica que todas las criticas del perceptrón podrán tener respuesta añadiendo capas ocultas a la estructura.

En la figura I.2.4 se muestran diferentes arquitecturas y diferentes regiones de decisión, que pueden ser formados por perceptrones de una o dos capas. En la primer columna se muestra las diferentes arquitecturas, la segunda indica las diferentes regiones que pueden ser formadas con diferentes redes. Las siguientes columnas están formadas por el problema de la OR Exclusiva y problemas con diferentes tipos de regiones.

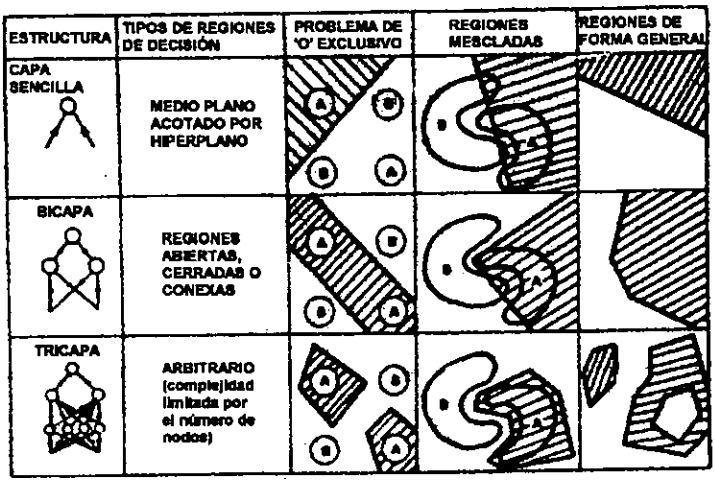


Figura I.2.4 Tipos de regiones de decisión que pueden ser formadas por perceptrones de una o dos capas.

ADALINE

Posteriormente Bernanrd Widrow, desarrolla una regla de aprendizaje denominada mínimos cuadrados ó regla delta, la cual fue muy popular como regla de aprendizaje, esta regla es parecida al perceptrón pero a diferencia de él, la regla delta ajusta los pesos hasta reducir la diferencia entre la red de entrada y la salida deseada. Matemáticamente prueba que, él error que existe entre la respuesta deseada y la obtenida debe encontrar un mínimo local bajo ciertas condiciones. Esta regla también, recibió el nombre de *Adaline (Adaptative Linear Element).* La figura siguiente ejemplifica la regla delta.

La fórmula de regla delta se encuentra definida por:

$$W_{new} = W_{old} + \frac{\beta \varepsilon x}{(x)^2}$$

donde : β es la constante de aprendizaje
$\varepsilon = I_o - Y$ (I_o es la señal de supervisión)
W_{old} = Vector de pesos Nuevo.
W_{new} = Vector de pesos Anterior

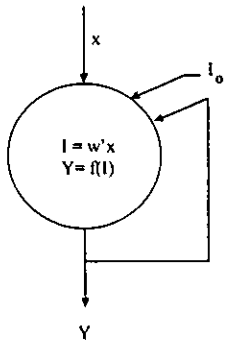


Figura 1.2.5. Representación del modo de aprendizaje de la Regla Delta con una entrada "x" y una salida "y"

La *constante de aprendizaje* β es una medida de la velocidad de la convergencia de los vectores de peso de la neurona, hacia su punto ideal. El valor que se le dé a β depende de que tan buenos son los patrones de entrada en el entrenamiento. Sí son parecidos conviene un dar un valor grande para β, para que de esta manera aprenda más rápido, sí los patrones son dispersos es necesario un valor pequeño para β.

Anteriormente, se diseñaban filtros analógicos RLC para el procesamiento de señales, empleando circuitos RLC (resistencias, inductores y capacitores), para eliminar el ruido de las señales empleadas en la comunicación. En la actualidad, el procesamiento de señales se ha transformado en una tecnología de múltiples facetas, ha pasado de la construcción de circuitos a los procesadores digitales de señales, que pueden llevar a cabo los mismos tipos de aplicaciones de filtrado, a través de filtros realizados mediante programas.

Las redes adaline son un ejemplo claro del tratamiento de señales. El tratamiento de señales es una rama de la ingeniería que consta fundamentalmente de la realización de filtros, para eliminar o reducir componentes frecuenciales no deseados de una señal portadora de información. Los sistemas adaline se utilizan frecuentemente como filtros.

RED HOPFIELD.

En la 1960, se suspendieron las investigaciones dentro del campo de redes neuronales no habiendo muchos adelantos en esta rama. En 1980 se desarrollan nuevos modelos para el diseño de redes neuronales, reanudándose el entusiasmo dentro del campo de las redes neuronales. En esta misma década Hopfield propuso una red neuronal como una nueva teoría de la memoria; la red elige aleatoriamente una unidad, si alguna de sus vecinas está activada, la unidad calcula la suma de los pesos

en las conexiones de esas unidades, si la suma es positiva la unidad se activa y si ocurre de modo contrario la unidad se desactiva, entonces, se elige otra unidad aleatoriamente y se repite el proceso hasta que la red alcance un estado estable, es decir, hasta que no quede una unidad que pueda cambiar de estado. El principal problema que se encontró en las *redes de Hopfield*, es que convergen hacia mínimos locales.

Una red de Hopfield tiene las siguientes características: *representación distribuida* (se define como una memoria que se almacena como un patrón de activaciones a través de un conjunto de elementos de proceso), *control asincróno y distribuido* (cada elemento de proceso toma decisiones basadas únicamente en su situación local), *memoria direccionable por contenido* (se pueden almacenar un determinado número de patrones en una red), *tolerancia a fallos* (aunque algunos elementos de la red fallen la red continuara funcionando adecuadamente).

En la figura I.2.6. se muestra el diseño de la red de Hopfield, donde las unidades en negro están activas, y las unidades en blanco están inactivas. Las unidades están conectadas unas con otras por conexiones simétricas y con pesos.

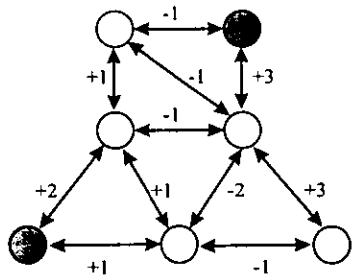


Figura I.2.6. Ejemplo de una red de Hopfield

Una conexión con peso positivo indica que las dos unidades tienden a activase la una a la otra. Una conexión con peso negativo permite que una unidad activa desactive a su unidad vecina.

El funcionamiento de la red consiste en elegir una unidad aleatoriamente, si alguna de sus vecinas está activada, la unidad calcula la suma de los pesos en las conexiones de esas unidades. Como anteriormente se ha mencionado, sí la suma es positiva, la unidad se activa y si es negativa se desactiva. Posteriormente se elige otra unidad aleatoriamente y se repite el proceso hasta que la red alcanza un estado estable, es decir, hasta que no quede ninguna unidad que pueda cambiar de estado. Este proceso se denomina *relajación paralela.*

Esta red tiene únicamente cuatro estados estables distintos, que se muestran en la figura I.2.7

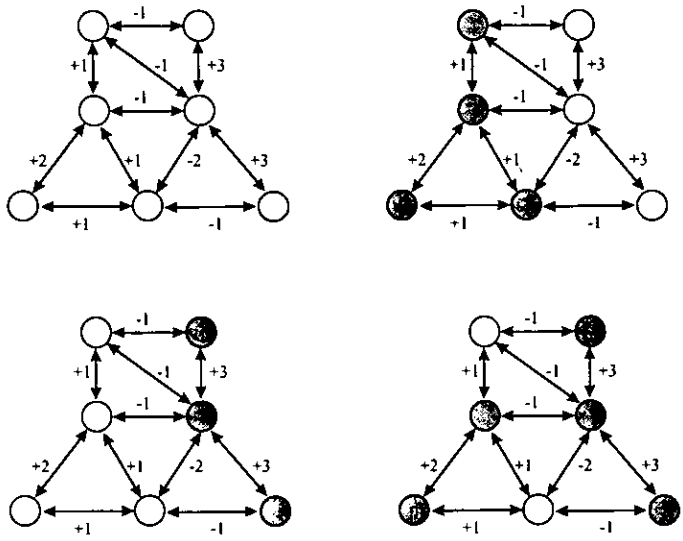


Figura 1.2.7. Los cuatros estados estables de una red de Hopfield concreta.

La red de Hopfield añadió una gran contribución al campo de las redes neuronales, esta consistía en mostrar que dado un conjunto de pesos y un estado inicial, el algoritmo de relajación paralela en algún momento llevará la red hacia un estado estable. La relajación paralela consiste en una búsqueda de un estado mínimo local.

Algo que parecía ser un problema es este tipo de redes es el hecho de suponer que una unidad fallará de repente haciéndose activa o inactiva cuando no debe, pero no representaba un gran problema, ya que las unidades que la rodean rápidamente volverán a ponerla en el camino correcto.

Las redes de Hopfield, además de servir como memorias direccionables por contenido, pueden resolver varios problemas de verificación de restricciones. La idea es de considerar cada unidad como una hipótesis, y situar pesos positivos entre las unidades que se representen hipótesis compatibles ó que se apoyen mutuamente, y pesos negativos en las unidades que ocurra lo contrario. Como se mencionó el principal problema de las redes de Hopfield es que convergen hacia mínimos locales, ya que durante el proceso de relajación paralela se asigna un valor de certeza ó falsedad a las distintas hipótesis, y al violar el número de restricciones, converge hacia un estado estable.

Las redes de Hopfield no pueden encontrar soluciones globales porque se sitúan en los estados estables por medio de un algoritmo completamente distribuido. Si la red alcanza un estado estable en determinado tiempo, ninguna unidad querrá cambiar su estado para moverse hacia una solución óptima.

En la década de los 80's se integra al campo de redes neuronales el algoritmo Backpropagation, con este algoritmo se podían resolver muchos problemas relacionados con las limitaciones de linealidad que presentaban los perceptrones. Dos fueron las razones del abandono de estudios en el campo de las redes neuronales. Dichas razones fueron el fracaso del perceptrón de una sola capa que no fue capaz de resolver simples problemas como la función XOR y la falta de un método general de entrenamiento de una red multicapas. *Backpropagation ó Retropropagación,* es la propagación de la información de error de acuerdo a la salida deseada hacia atrás. Este método fue descubierto en la década anterior pero en ese entonces no se le dio gran importancia.

RETROPROPAGACIÓN

Existe una serie de aplicaciones computacionales que resultan complejas de realizar mediante procesos secuenciales en el reconocimiento de tramas. Uno de los sistemas de redes neuronales que ha resultado muy útil para resolver problemas que requieren el reconocimiento de tramas complejas y la realización de funciones de correspondencia no trivial es la red de Retropropagación (Backpropagation).

El algoritmo de retropropagación emplea una arquitectura de tres o más unidades de procesamiento. La primer capa o unidad de procesamiento es la de entrada (las únicas unidades en la red que reciben entrada externa); la siguiente es la capa intermedia, en la que las unidades de procesamiento están interconectadas tanto a la capa anterior como a la siguiente; la siguiente capa es la capa de salida. Cada unidad de procesamiento recibe señales de todos los nodos de la capa anterior y envía conexiones a todos los nodos de la capa siguiente, sin embargo, se debe notar que los nodos de la misma capa no se encuentras conectados entre sí.

El procedimiento de entrenamiento de la red de retropropagación se realiza con una técnica llamada *aprendizaje supervisado*, en la que a la red se le presenta una serie de pares de patrones y cada par consta de un patrón de entrada y uno de salida; cada patrón es un vector de números reales. Los pesos son ajustados para decrementar la diferencia entre la salida de la red y la salida deseada. El patrón de salida es la respuesta deseada al patrón de entrada y es usado para determinar el error en la red cuando se realiza el ajuste de pesos. A continuación se hará un seguimiento más detallado de esta red.

El algoritmo de aprendizaje de retropropagación se conforma de dos pasos llevando a cabo un paso de propagación hacia adelante y uno de propagación hacia atrás. Ambos pasos se realizan para cada presentación del patrón durante la etapa de aprendizaje.

Propagación hacia adelante.

La propagación hacia adelante comienza con la presentación de un patrón de entrada (X_0 ... X_{N-1}) como estimulo para la primera capa de la red y continua con el cálculo del nivel de activación que se propaga hacia las capas superiores. En cada capa, las unidades de procesamiento ó nodos, suman sus entradas y aplican una función no lineal (es decir, una función sigmoidal figura I.2.8.), para calcular la salida. Las unidades de la capa de salida producen la salida de la red como se muestra en la figura I.2.9.

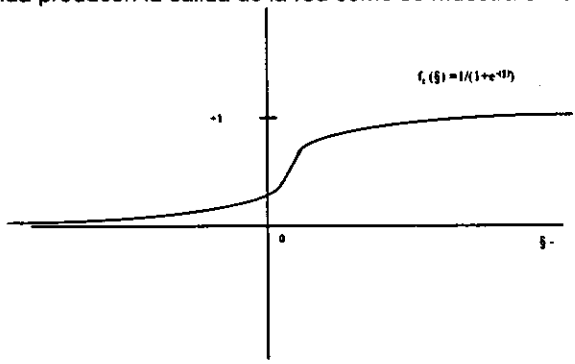


Figura I.2.8. Esta gráfica muestra la forma de una "S" característica de la función sigmoidal.

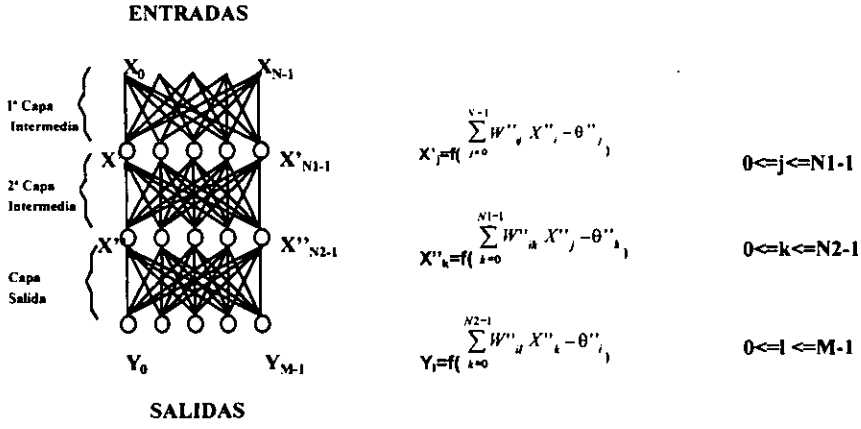


Figura I.2.9. Red Backpropagation con tres capas

La figura muestra el esquema de una red backpropagation con tres capas con N valores de entrada, M salidas y dos capas intermedias. La regla de decisión es para seleccionar que clase corresponde de la salida del nodo con la salida deseada. En la fórmula, X'_j y X''_k son las salidas de los nodos en la primera y segunda capa oculta, θ es el umbral para cada capa, W son los pesos de conexión entre las capas.

El conjunto inicial de valores de pesos representa una primera aproximación de los pesos correctos para el problema. El cambio de pesos para diferentes tipos de redes se puede hacer mediante la regla delta.

$$W_{ij}(t+1) = W_{ij}(t) + \eta\, \delta_j X'_j$$

donde : η es una constante (constante de aprendizaje).
X'_j es la componente del vector j de entrada al nodo.
δ es el factor de error

El factor η es el parámetro de velocidad de aprendizaje (que comúnmente se encuentra entre 0.25 y 0.75). Valores muy grandes en este parámetro pueden llevar a la red a la inestabilidad (que la red no caiga dentro del mínimo global) y a un mal aprendizaje. Por otro lado, los valores muy pequeños pueden hacer que el aprendizaje sea muy lento. Algunas veces puede ser conveniente que el factor de aprendizaje sea variable para producir un aprendizaje más eficiente en la red, por ejemplo: permitiendo que el valor de η comience en un valor grande y decrementandolo gradualmente durante el aprendizaje.

La regla Delta generalizada, que se utiliza como algoritmo de aprendizaje para la red de propagación hacia atrás, lleva a cabo un descenso mediante el gradiente por una superficie. Es posible según se observa a veces en la práctica, que el sistema acabe en un mínimo local. Cuando el sistema acaba en un mínimo local produce un efecto es que la red parece dejar de aprender, esto es, cuando el error no disminuye aplicando un entrenamiento adicional. El determinar que esta situación sea aceptable ó no, depende del valor del error cuando se alcanza el mínimo. Si el error es aceptable, entonces no importa si el mínimo es global ó no. Si el error no es aceptable suele ser posible remediar el problema entrenando nuevamente la red con parámetros distintos. En el caso de la propagación hacia atrás, se ve que es deseable hallar el mínimo global, pero en muchos casos los mínimos locales son soportables. En la figura I.2.10 se muestra un ejemplo del descenso de error, donde es fácil observar que cuando el error se detiene en el punto "A" se encuentra en un mínimo local, y cuando llega al punto "B" alcanza un mínimo global.

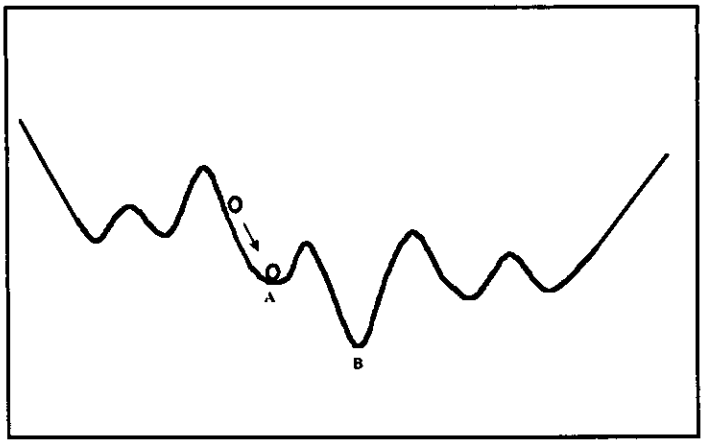


Figura I.2.10. Gráfica de Mínimos

En la figura I.2.11 podemos observar una superficie hipotética del espacio de pesos que da una idea de la complejidad de esta superficie, donde los pesos debería encontrar el mínimo global.

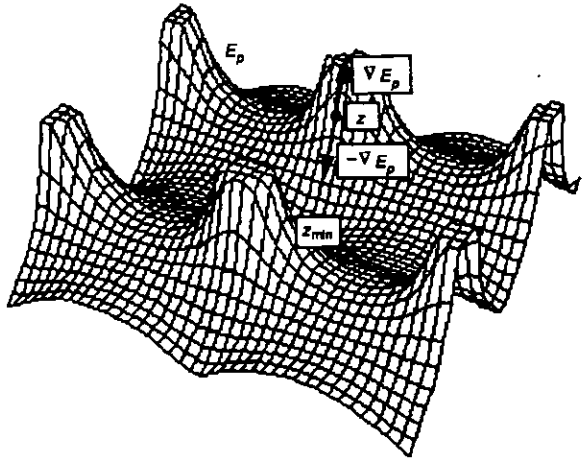


Figura I.2.11. Superficie hipotética de pesos

Propagación hacia atrás.

La propagación hacia atrás inicia con la comparación del patrón de salida de la red con el patrón deseado, dando como resultado una diferencia ó error. La propagación hacia atrás calcula el valor de error para los nodos de las capas escondidas y cambia los pesos comenzando con la capa de salida y moviéndose hacia atrás a través de las capas ocultas sucesivas.

Él cálculo de error para la capa de salida esta determinado por la fórmula:

$$\delta_j = Y_j(1 - Y_j)(d_j - Y_j)$$

donde : d_j es la salida deseada en el nodo j y Y_j es la salida actual.

Él cálculo de error para la capa ó capas ocultas se define por:

$$\delta_j = X'_j(1 - X'_j)\sum_k \delta_k W_{jk}$$

donde : k es el índice de una capa que se encuentra sobre la capa con índice j.

Las señales de error se transmiten entonces hacia atrás, partiendo de la capa de salida, hacia los nodos de la capa intermedia que contribuyen directamente a la salida, las unidades de la capa intermedia sólo reciben una fracción de la señal total de error. Este proceso se repite, capa por capa, hasta que todos los nodos de la red hayan recibido una señal de error que describa su contribución relativa al error total. Posteriormente la red actualiza sus pesos de conexiones de cada unidad, de tal forma que el error observado se decremente en la siguiente etapa del entrenamiento.

Después del entrenamiento, cuando se presenta a la red una trama arbitraria de entrada que contenga ruido o éste incompleta, las unidades de las capas ocultas de la red responderán con una salida activa, si la nueva entrada contiene una trama que se asemeje a aquella características que las unidades individuales hayan aprendido a reconocer durante su entrenamiento.

Hay varios investigadores que han demostrado que, durante el entrenamiento, las redes retropropagación tienden a desarrollar relaciones internas entre nodos con el fin de organizar los datos de entrenamiento en clases de tramas. (12)

Lo importante es que la red encuentra una representación interna que le permite generar las salidas deseadas cuando se le han dado las entradas de entrenamiento. Esta misma representación interna se puede aplicar a entradas que no fueron utilizadas durante el entrenamiento. La red de retropropación clasificará las entradas que no había visto hasta el momento, según las características que compartan con los ejemplos de entrenamiento.

Si la función de salida es una sigmoide, entonces será preciso aplicar una escala a los vectores de entrada, por la forma de la función sigmoideal la red no puede alcanzar el cero ni el uno. Por tanto hay que usar valores como 0.1 y 0.9, para determinar un rango de valores .

Las preguntas que muchas veces nos hacemos son ¿cuántos nodos se necesitan, exactamente para resolver un problema concreto?, ¿Basta siempre con tres capas o cuántas son necesarias?, Pero no hay una respuesta concreta para este tipo de preguntas, algunos autores mencionan que tres capas son suficientes, hay veces que según la magnitud del problema son necesarias más de una capa oculta. En lo que se refiere a los nodos la primera capa (capa de entrada), viene determinada por la naturaleza de la aplicación. En la mayor parte de las ocasiones es posible determinar el número de nodos de salida decidiendo si se desean valores analógicos o valores binarios en las unidades de salida. El determinar el número de nodos que debe de contener la capa o capas ocultas no suele ser tan fácil. La idea principal consiste en utilizar el menor número posible de nodos, ya que cada unidad supone una carga para el procesador durante las simulaciones e irlo variando de acuerdo a la experiencia o el mejor desempeño que demuestre la red.

I.3 ¿Para qué sirve una Red Neuronal ?

Lo importante de estos modelos es que todos ellos muestran un comportamiento útil al aprender, reconocer y aplicar relaciones entre objetos y trama propios del mundo real.

Las redes neuronales son útiles en una gran variedad de campos en los sistemas computacionales, tales como los ejemplo que se muestran a continuación:

EL PROBLEMA DEL VIAJANTE

Este es un ejemplo claro de la red de Hopfield, ya que es de la clase de problemas que se conocen con el nombre de problemas de optimización, es decir, estos problemas suelen plantearse en términos de hallar la mejor manera de llegar a una solución . La mejor solución suele definirse mediante un criterio especifico. Por ejemplo, podría haber un coste asociado a cada posible solución, y la mejor solución es la que minimice el coste, satisfaciendo todos los requisitos necesarios para ser una solución aceptable. De hecho en muchos casos los problemas de optimización se describen en términos de una función de coste.

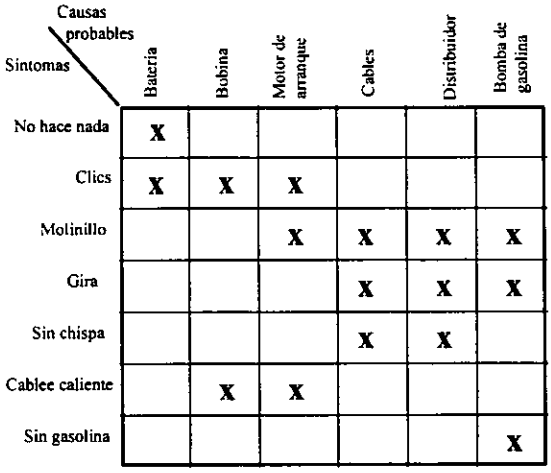
Uno de estos problemas de optimización es el problema del viajante. En su forma más sencilla, un viajante tiene que realizar una ruta que pasa por un determinado números de ciudades, visitándolas todas una sola vez, y minimizando la distancia total recorrida. El problema es hallar la secuencia correcta en que hay que visitar las ciudades. Las limitaciones son: visitar todas las ciudades, visitar cada una sólo una vez, y que el viajante vuelva al punto inicial al final del viaje. La función de coste que hay que minimizar es la distancia total recorrida durante el viaje.

Las cantidades de tiempo de cálculo que requiere una computadora digital para resolver este problema crece exponencialmente con el número de ciudades. El problema pertenece a una clase de problemas que se conocen con el nombre de problemas *NP-Completos*[4]. Como consecuencia de la carga de cálculo, en los problemas de optimización suele suceder que una solución buena que se encuentre rápidamente sea preferible que la mejor solución, cuando esta última se encuentra demasiado tarde para resultar útil. La red Hopfield es muy adecuada para este tipo de problemas.

[4] NP-Completos : son la clase de problemas para la que existe un algoritmo no determinista y cuyo tiempo de ejecución es polinómico respecto al tamaño de los datos de la entrada.

SINTOMAS DE DIAGNOSTICOS

Considerando un ejemplo concreto, es la aplicación de diagnóstico para automóviles, donde se diagnostica por qué no funciona un coche. Primero se definen los distintos síntomas que hay que considerar. A continuación, se describen las posibles causas del problema, basándose en los síntomas. Aunque nuestra lista no es una representación completa de todos los posibles problemas, los síntomas podrían indicar alguno de estos problemas, o aun combinación de ellos. Posteriormente se construye una matriz que indica la correspondencia de los síntomas con sus causas probables. La matriz se ha ilustrado en la figura I.3.1.



Síntomas \ Causas probables	Batería	Bobina	Motor de arranque	Cables	Distribuidor	Bomba de gasolina
No hace nada	X					
Clics	X	X	X			
Molinillo			X	X	X	X
Gira				X	X	X
Sin chispa				X	X	
Cablee caliente		X	X			
Sin gasolina						X

Figura I.3.1. Matriz de síntomas

Al inspeccionar la matriz se aprecia la variedad de problemas que puede indicar cualquier síntoma. Nuestro sistema automatizado debe ser capaz de correlacionar muchos síntomas distintos, y en el caso de que alguno de los síntomas no se aprecie o esté ausente, debe ser capaz de llenar los espacios en blanco, del problema basándose solamente en los síntomas indicados.

Puesto que se desea introducir a la red, los síntomas observados y hacer que esta responda con la ó las causas probables, un buen candidato de arquitectura es el que consiste en hacer corresponder directamente cada síntoma con una unidad de procesamiento individual de entrada, y una de las causas probables con una unidad de salida, por lo que es conveniente usar la arquitectura de Red Boltzmann de entrada-salida.

ADALINE EN LA TELEFONÍA

El fenómeno del eco que es producido en las conversaciones telefónicas, es decir, las palabras que uno pronuncia en el micrófono se oye en fracción de segundo después a través del auricular del teléfono, el eco resulta especialmente apreciable en llamadas de larga distancia sobre todo en aquellas que se realizan mediante enlace vía satélite, en las cuales los retrasos de transmisión pueden llegar a ser fracciones significativas de segundo. Los circuitos telefónicos contienen dispositivos llamados híbridos que tienen la misión de aislar las señales salientes, evitando por tanto el efecto de eco. Desafortunadamente estos circuitos no tienen un rendimiento del 100%, dando lugar a que se produzca un eco que se le devuelve al que habla. Aunque la señal de eco haya sufrido una atenuación, puede seguir siendo audible importunando al que habla.

Para resolver el problema de eco se pueden utilizar filtros adaptivos (ADALINE) para eliminar el efecto de eco sin el entrecortamiento que producen circuitos de eliminación de ruido.

El siguiente ejemplo es un diagrama que muestra un filtro adaptivo que se utiliza para predecir los valores de una señal. (Figura I.3.2.) La señal de entrada que se emplea para entrenar la red es un valor retardado de la señal actual; basándose en una entrada que es el valor de la señal en algún instante anterior, se puede utilizar directamente la señal actual como entrada sin el retardo. Entonces el filtro realizará una predicción del valor futuro.

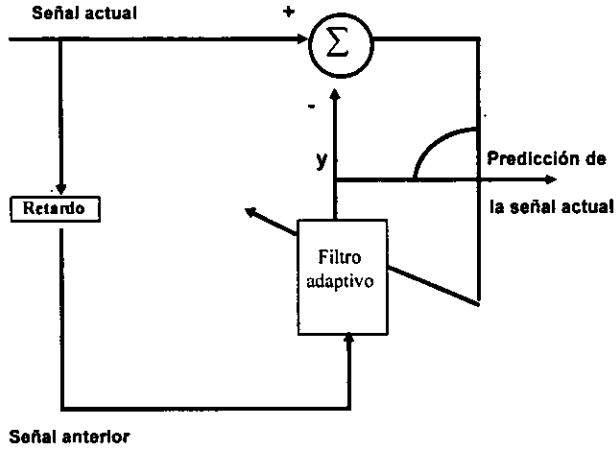


Figura I.3.2. Este diagrama muestra un filtro adaptivo que se utiliza para predecir los valores de una señal.

INSPECCION DE CALIDAD DE PINTURA

La inspección visual de superficies pintadas, como puede ser la carrocería de un automóvil, es en la actualidad un proceso muy lento y muy costoso en términos de mano de obra. Para reducir la cantidad de tiempo que se requiere para llevar a cabo esta inspección, Se proyecta un haz láser en la superficie pintada y se examina el haz reflejado en una pantalla. Dado que la fuente de luz es un haz coherente, la cantidad de dispersión que se observa en la imagen reflejada del láser proporciona una indicación de la calidad de acabado de la pintura del coche. Un trabajo de pintura mal hecho contiene ondulaciones, tiene aspecto de piel de naranja o carece de brillo. Una haz láser que se refleja en una superficie mal acabada será por lo tanto, relativamente difuso. Por lo contrario, un buen acabado de la pintura será relativamente suave y poseerá un brillo considerable. La luz láser que se refleja en una superficie pintada con un buen acabado tendrá para el observador un aspecto prácticamente uniforme en toda su extensión. (12)

Para la solución de este problema se crea un sistema de redes neuronales con el método de retropropagación, para automatizar el proceso de inspección de pintura, se utiliza un sistema de vídeo, que examine y califique la calidad del acabado dada la entrada del vídeo. La imagen de la entrada procedente de vídeo se hace pasar por un grabador de cuadros para registrar la imagen láser reflejada. La imagen contiene un tamaño en pixeles. La salida deseada se determina como una calificación numérica en el intervalo de 1 a 20. La red retropropagación se construye de manera que tenga una sola salida: ésta unidad de salida produce una salida líneas que se interpretaba como calificación a escala de la pintura.

Después de haber entrenado a la red, se toma como muestra 10 imágenes. En la figura I.3.3 se muestra el diagrama del sistema de la red retropropagación para llevar a cabo una estimación de la calidad de acabado.

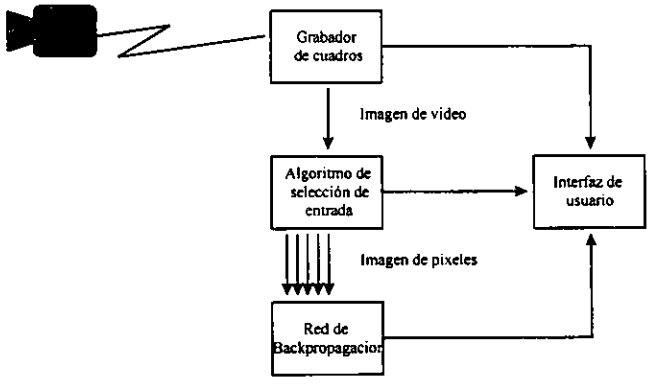


Figura I.3.3. Diagrama para la estimación de calidad de pintura

Existen otras tareas que se han resuelto con éxito mediante redes neuronales, entre las que se encuentran: aprender a jugar bacgammon, clasificar señales de sonar, comprimir imágenes y conducir un vehículo por carretera. Aunque existen otras técnicas para poder resolver estos problemas, los sistemas conexionistas basados en el aprendizaje se pueden construir frecuentemente con más rapidez y con menos conocimiento que los métodos tradicionales. (8)

I.4 Red Neuronal Para esta aplicación

Para crear una red neuronal se debe tener una arquitectura y un algoritmo para entrenar la red y para garantizar que funcione de manera adecuada, también se requiere de un teorema de convergencia que garantice la obtención de una solución final, con el resultado correcto.

Consideremos ahora la cantidad de software adicional (y, por lo tanto, de tiempo de procesador) que se debe añadir al algoritmo de la tabla de búsqueda para mejorar la capacidad de la computadora y con ello saber a que carácter debería haber correspondido la imagen con ruido. Los errores de un sólo bit son bastantes fáciles de hallar y corregir. Los errores de muchos bits se vuelven cada vez más difíciles a medida que crece el número de bits erróneos. Para complicar todavía más las cosas, ¿cómo podría nuestro software compensar el ruido de la imagen si el ruido llegase a hacer que una "O" tuviese el aspecto de una "Q", o si hiciera que una "E" pareciese a una "F"?, Si el sistema de conversión de caracteres tuviese que producir salidas precisas en todo momento, se invertirá una cantidad disparada de tiempo de procesador, eliminando el ruido de la trama de entrada, antes de enviarla como vector.

Una solución para este problema consiste en aprovechar la naturaleza paralela de las redes neuronales para reducir el tiempo requerido por un procesador secuencial para determinar la correspondencia. El tiempo de desarrollo del sistema se puede reducir como consecuencia, ya que la red puede aprender a través de un algoritmo correcto sin que alguien tenga que deducir por anticipado lo que tiene que hacer.

Existen muchos algoritmos de entrenamiento y diferentes tipos de arquitecturas, pero para este proyecto se utilizará la arquitectura de redes de **Retropropagación** (backpropagation), debido a que puede resolver problemas de clasificación de patrones complejos y desarrollar funciones de mapeo altamente no lineales.

Una de las formas de utilizar sistemas de redes neuronales artificiales es en problemas de reconocimiento de caracteres. En este ejemplo, la aplicación de una capa de entrada puede dar lugar a que se activen muchas de las unidades de la segunda capa (o capa oculta). La actividad de la capa oculta debería dar lugar entonces a que se activasen una y sólo una de las unidades de salida, es decir, la que este asociada a la trama que se este identificando. También debería tenerse en cuenta el elevado número de conexiones que se necesitan para esta red.

Este ejemplo es especialmente interesante por dos razones:

- Aunque se puede definir un conjunto de caracteres de forma rigurosa, todos tendemos a personalizar la forma de escribir. Esta sutil variación de estilos es difícil de adoptar cuando se utiliza una aproximación algorítmica del reconocimiento de tramas, porque incrementa la forma combinatoria del número de entradas que hay que examinar.

- La aproximación de redes neuronales para resolver el problema no sólo proporciona una solución factible, sino que además se puede utilizar para obtener información interna acerca de la naturaleza del problema.

Tomando como referencia un diagrama típico de red, se representa esquemáticamente cada elemento de procesamiento (o unidad) de la red como un nodo, indicando las conexiones entre nodos mediante arcos. Se indicará la dirección del flujo de información dentro de la red mediante el uso de puntas de flechas en las conexiones. Figura I.4.1.

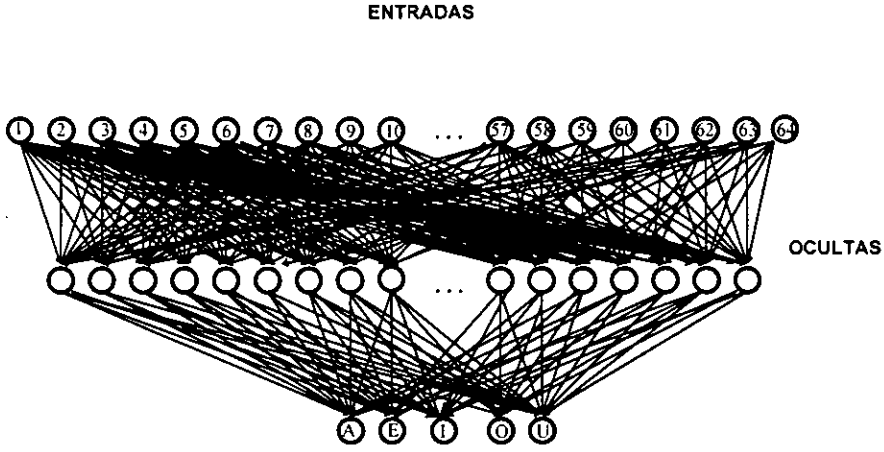


Figura I.4.1 Diagrama de la red retropropagación para el reconocimiento de caracteres.

Dado que el objetivo es que la red determine cual de las cinco vocales (A, E, I, O, U) es el carácter escrito, se creo una estructura para la red, que tuviera 5 unidades de salida, una para cada carácter que haya que identificar. Esta estrategia simplifica la función de discriminación de caracteres de la red, puesto que nos permite utilizar una red que contiene unidades binarias de la capa de salida, por ejemplo para cada trama de entrada que se introduzca, la red debería activar una y sólo una de las 5 unidades de salida indicando así a cual de los 5 caracteres que se intenta reconocer se parece más la entrada.

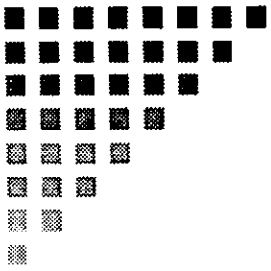
De manera similar es preciso determinar como modelar la entrada de caracteres para la red, si se modela los datos de entrada como un vector que contiene elementos binarios, lo cual permite utilizar una red con un sólo tipo de unidad de procesamiento. Para crear este tipo de entrada, se toma una idea del mundo del vídeo y pixelizando el

carácter. Se toma arbitrariamente una matriz de 8x8 pixeles como imagen, empleando 1 para denotar que un pixel esta encendido, y un 0 para denotar que esta apagado. Más aun, se puede descomponer esta matriz en un conjunto de vectores fila, que se pueden concatenar en un sólo vector de dimensión 1x64. De esta forma se ha definido la dimensión del vector de entrada.

Posteriormente se determina el número de unidades de procesamiento (denominadas unidades ocultas) que deben ser utilizadas internamente, para conectarlas con las unidades de entrada y de salida ya definidas empleando conexiones poderosas y entrenar la red mediante el ejemplo de pares de datos.

Como ilustra este ejemplo, un sistema de redes neuronales es robusto, en el sentido de que responderá con alguna salida incluso en el caso de que se le presente entradas que no haya visto nunca, tales como tramas que contengan ruido, si el ruido introducido no ha destruido la imagen del carácter, la red producirá una buena suposición empleando aquellas partes de la imagen que no hayan quedado ocultas y la información que ha almacenado acerca del aspecto que se suponen que tienen que tener los caracteres. La capacidad de enfrentarse a tramas ruidosas o distorsionadas es una de las propiedades de los sistemas de redes neuronales.

Este ejemplo también ilustra el proverbio de las redes neuronales que se menciona en el libro NEURONAL NETWORKS, de James A. Freeman / David m.skapura: " la potencia de la aproximación de los sistemas de red neuronal no consiste necesariamente en la elegancia de una cierta solución, sino más bien en la generalidad de la red que es capaz de hallar sus propias soluciones para problemas concretos, dándosele únicamente ejemplos del comportamiento deseado".



CAPÍTULO II

VENTAJAS GENERALES DE COMPUTO DISTRIBUIDO Y CONCURRENTE

"El ser humano ha luchado se ha sacrificado por muchos siglos para librarse de las trabas externas que le impedían hacer y pensar por sí mismo. Ahora que lo ha logrado, no sabe qué hacer con su libertad. ¡No ha sido capaz de comprometerse con su existir! A.L.C.

II.1 Sistemas de Computo Concurrente

Antes hablar de procesos concurrente, es bueno saber que relación tiene con las redes neuronales ¿ Para qué nos sirve saber que son los procesos concurrentes? Las redes neuronales comúnmente necesitan compartir recursos sobre todo de CPU, pero lo principal, es que para caracterizar una red neuronal distribuida primero tenemos que evaluar la red de forma concurrente.

Hablamos de procesos concurrentes cuando dos o más procesos se ejecutan simultáneamente, es decir, al mismo tiempo, pero en realidad están compartiendo recursos de tiempo de procesador. Los procesos pueden funcionar independientemente unos de otros, o pueden estar relacionados. Cuando un proceso requiere de compartir recursos con otros procesos, necesitan de una coordinación, para asegurar que funcionaran correctamente.

En el caso de que los procesos tengan conocimiento uno del otro, ya sea cuando tienen acceso a un objeto en común, como por ejemplo a la memoria principal de una computadora, o a un archivo, los procesos cooperan para no entrar en conflicto.

Cuando existe uno o más procesos y no tienen conocimiento uno del otro, estos comienzan a competir por usar el mismo recurso, un ejemplo es: el tiempo de procesador, acceso a dispositivos de entra y salida; la competencia da lugar a la existencia de conflicto entre ellos.

En el caso de que los procesos se encuentren en competencia, deben enfrentarse a problemas de control:

A) *Exclusión mutua*: Cuando existe un grupo de procesos concurrentes, sólo uno pude accesar a determinado recurso o ejecutar una función dada en cualquier momento, a lo cual se le denomina exclusión mutua; durante la ejecución cada proceso envía comandos al recurso compartido, y recibe información del estado en que se encuentra este, a este recurso se le llama *recurso critico;* y la parte del programa que lo uso se le conoce como *sección critica* del programa.

Un ejemplo de ello, lo vemos reflejando cuando tenemos un grupo de procesos, y uno de ellos quiere imprimir un archivo, por lo que espera tener control de la impresora hasta que termine de usar este recurso, en el caso de que no existiera exclusión mutua entre los procesos, estos al encontrarse en competencia por accesar un recurso al mismo tiempo, en este caso la impresora, cada proceso mandaría su archivo de impresión, por lo cual, las líneas que se imprimieran serian intercaladas unas con otras de cada proceso. Por ello se requiere que cuando un proceso entre en una sección critica, libere lo más pronto posible este recurso (Figura II.1.1). Se pueden usar diferentes técnicas de exclusión mutua para resolver los conflictos que se presentan entre procesos al competir por los recursos.

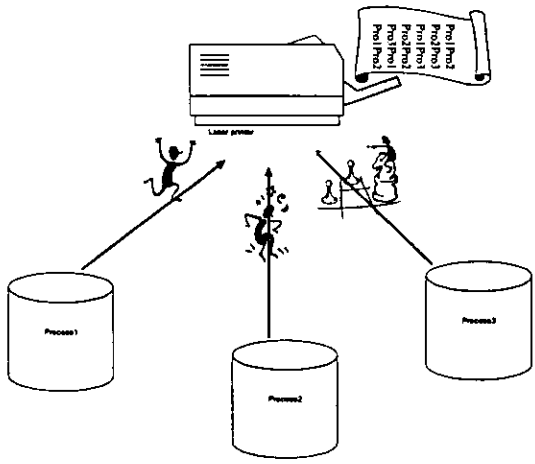


Figura II.1.1 Problemas de Exclusión Mutua

Se han desarrollado varios algoritmos de programación para proporcionar exclusión mutua, de los cuales el más conocido es el algoritmo de Dekker. El algoritmo de Dekker resuelve el problema de exclusión mutua pero con un programa bastante complejo que es difícil de seguir y cuya validez se comprueba en forma engañosa. Peterson proporciona una solución simple: basado en *banderas* que indica la posición de cada proceso con respecto a la exclusión mutua y una variable global "turno" que resuelve los conflictos de manera simultánea. (16)

El primer avance notable al tratar con los problemas de procesos concurrentes tuvo su presentación 1985 con el tratado de Dijkstra, sobre procesos secuenciales cooperativos: "dos o más procesos pueden cooperar por medio de señales simples, de manera que se puede obligar a un proceso a detenerse en un lugar especifico hasta que reciba una señal especifica. Cualquier requisito de coordinación compleja puede satisfacer la estructura apropiada de señales. Para señalizar se usan variables especiales llamadas semáforos, si la señal correspondiente todavía no se transmite, el proceso se suspende hasta que ocurre la transmisión". (17)

Existen muchas técnicas para el control de exclusión mutua: los *semáforos* y las instalaciones de *mensajes*. Los *semáforos* se usan para señalizar entre procesos y pueden utilizarse con rapidez para obligar una exclusión mutua. Los *mensajes* son útiles para la ejecución de la exclusión mutua y también proporcionan un medio de comunicación entre procesos.

Los mensajes y la memoria compartida proporcionan medios de comunicación de procesos, mientras que los semáforos y las señales se usan para activar acciones de otros procesos; los mensajes tienen la ventaja adicional de prestarse a la

implementación en sistemas distribuidos y en sistemas de un sólo procesador y multiprocesadores de memoria compartida.

B) *Interbloqueo (deadlock).* El interbloqueo ocurre cuando un proceso se encuentra en esperar de un recurso que se le va asignar. El interbloqueo puede involucrar recursos reutilizables o recursos consumibles. Un recurso *consumible* es el que se destruye cuando lo adquiere un proceso. Un recurso *reutilizable* es uno que no se agota o destruye por el uso.

Por ejemplo consideraremos dos procesos, P1 y P2 y dos recursos críticos R1 y R2. Suponiendo que cada proceso necesita acceso a los dos recursos para ejecutar parte de su función, entonces se presenta la siguiente situación: el sistema operativo asigna R1 a P2 y R2 a P1; cada proceso espera uno de los dos recursos; pero ninguno libera el recurso que ya le pertenece, sino hasta que haya adquirido el otro y ejecutado su sección critica. Por lo que se dice que ambos procesos están interbloqueados.

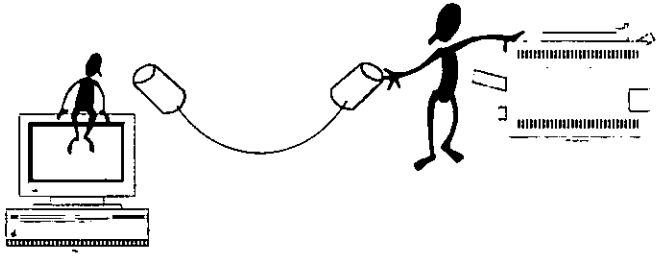


Figura II.1.2 Interbloqueo entre procesos

Otro ejemplo de interbloqueo es la solicitud de escritura y lectura, es decir se crea un tubo o *pipe*, para escritura que tiene un determinado tamaño en bytes, cuando un proceso intenta escribir en un tubo, la solicitud de escribir se ejecuta de inmediato, si hay espacio suficiente; de otra manera, el proceso se bloquea, lo mismo sucede en un proceso de lectura es decir se bloquea si se intenta leer más bytes que los que están en ese momento en el tubo; el sistema que controla los procesos impone la exclusión mutua: esto es, sólo un proceso a la vez tiene acceso a un tubo.

Para enfrentar los problemas de interbloqueo tenemos que tomar en cuenta tres enfoques importantes: prevención, detección y anulación. La prevención y detección garantiza que el interbloqueo entre procesos no ocurra, asegurándose que una sección critica sea liberada para ser usada por otro proceso. La *anulación* del interbloqueo involucra el análisis de cada nueva solicitud de recursos para determinar si algún

proceso podría conducir a interbloqueo, y de esta manera asignar la solicitud sólo si existe la seguridad de que el interbloqueo no se presente.

C) *Inanición (starvation)*. La inanición se presenta cuando un proceso esta en espera de un recurso, pero nunca tiene acceso a él por prioridad. Por ejemplo suponiendo que de tres procesos, P1, P2 y P3, cada uno requiere acceso periódico al recurso R; si consideramos la situación en la cual P1 esta en posesión del recurso y P2 y P3 están retrasados esperando ese recurso, cuando P1 sale de su sección critica, a P2 o P3 se le permite tener acceso a R, suponiendo que a P3 se le permite el acceso al recurso y antes de que complete su sección crítica, P1 requiere acceso otra vez, si a P1 se le permite el acceso después de que P3 ha terminado, por lo que P1 y P3 se les esta concediendo un acceso alternado, entonces a P2 se le puede negar el acceso al recurso en forma indefinida, aun cuando no exista situación de interbloqueo.

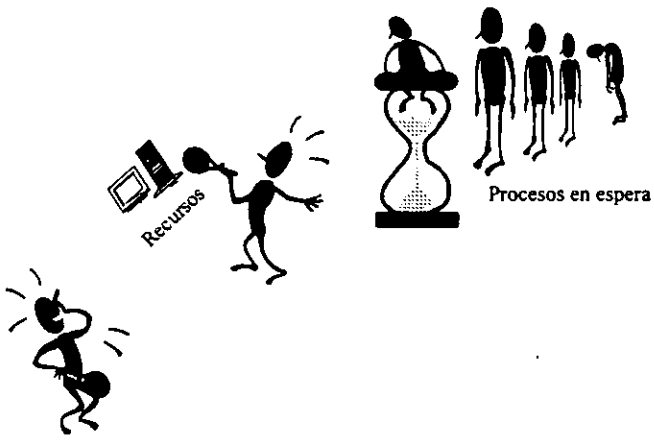


Figura II.1.3 Inanición de procesos

Cuando hacemos uso de la concurrencia, debemos tomar muy en cuenta las secciones criticas y hacer cumplir la exclusión mutua para que dicho sistema funcione correctamente. Por lo que la realización de cualquier sistema debe considerar los siguientes requisitos:

•La exclusión mutua debe reforzarse, es decir, sólo se permite un proceso a la vez dentro de su sección critica, entre todos los procesos que tienen secciones criticas para el mismo recurso u objeto compartido.

•Un proceso que se detiene en su sección critica no debe hacerlo sin informar a los otros procesos.

•No debe ser posible que un proceso que requiere acceso a una sección critica se retrase en forma indefinida; no pueden permitirse el interbloqueo o la inanición.
•Un proceso permanece dentro de su sección critica sólo durante un tiempo finito.

Existen varios lenguajes de programación que proporcionan características para procesamiento concurrente, lo cual significa que el compilador se encarga de que se lleve a cabo la concurrencia de procesos, un ejemplo de ello son el manejo de Hilos o *Threads* en programas como Java, que no requiere el diseño detallado de primitivas para implementar exclusión mutua, ya que están implementadas sus aplicaciones, en librerías de clase.

Muchas veces hemos oído hablar de procesos concurrente y paralelos pero, ¿ Qué diferencia existe entre procesos concurrentes y paralelos? En una computadora que sólo pose un procesador la ejecución de los procesos se intercala, compartiendo tiempo de procesador y es a lo que denominamos concurrencia, en el caso de tener una computadora con más de un procesador, la ejecución de los procesos puede traslaparse, es decir ser simultánea, ya que las tareas se dividen en cada procesador; estas dos formas se muestran en la Figura II.1.2.

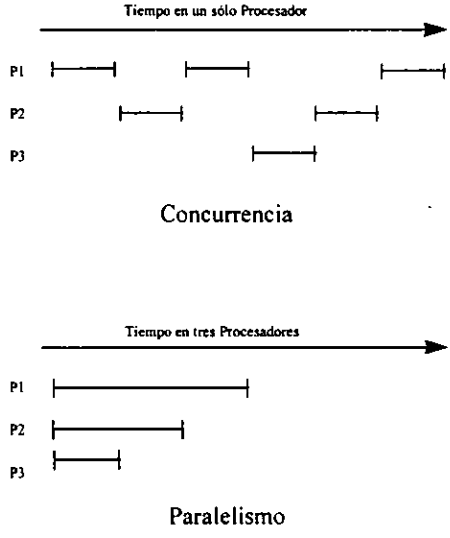


Figura II.1.4 Diferencia entre Concurrencia y Paralelismo

La concurrencia de procesos puede existir como resultado de la multiprogramación. En un ambiente de multiprogramación, la concurrencia puede aparecer en tres contextos diferentes:

- Múltiples aplicaciones: La multiprogramación se invento para permitir que el tiempo de procesamiento de la computadora se intercambie en forma dinámica entre varios trabajos o aplicaciones activas.
- Aplicación estructurada: Como una extensión de los principios de diseño modular y programación estructurada, algunas aplicaciones pueden implementarse, en forma correcta, como un conjunto de procesos concurrentes.

- Estructura del sistema operativo: Algunos sistemas operativos se implementan como un conjunto de procesos por lo que se involucra la concurrencia.

En la figura II.1.1 se muestra el ejemplo de la diferencia que existe entre un procedimiento tradicional y un proceso concúrrete. En un procedimiento tradicional, existe una relación amo/esclavo entre el procedimiento llamado y el que llama. El procedimiento que llama puede ejecutar una llamada desde cualquier punto; entonces, el procedimiento llamado se inicia en su punto de entrada y regresa al procedimiento que llama en el punto de llamada.

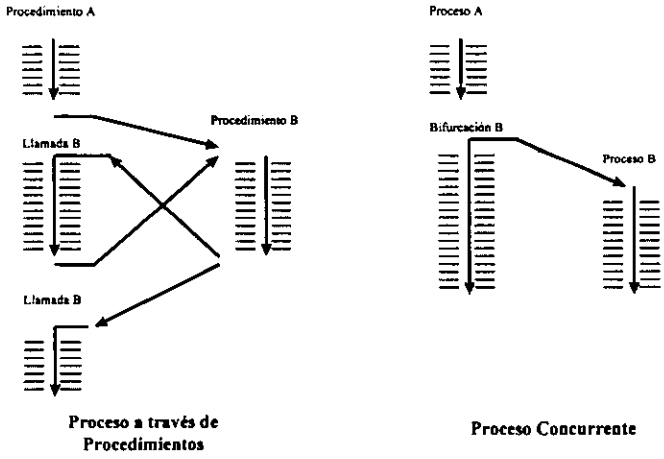


Figura II.1.5 Diferencia de un Proceso Tradicional y un Proceso Concurrente

II.2 Sistemas de Computo Distribuido

Hemos hablado de los procesos concurrentes, pero para la realización de este proyecto de tesis no es suficiente, también necesitamos conocer que son los procesos distribuidos. Se dice que un sistema es distribuido, cuando tenemos un conjunto de subsistemas informáticos repartidos, con distinta o igual capacidad y rapidez de procesamiento, que se conectan de manera apropiada para que desde sitios diferentes puedan interactuar, optimizando y en algunos casos, compartiendo recursos. Con ello se resuelven necesidad de tratamiento de grandes cantidades de datos. La P.O.O nos da la facilidad de impulsar los procesos distribuidos, de la misma forma hacer más fácil la conceptualización y programación, pero más adelante en otro capítulo hablaremos de la P.O.O.

El uso de procesamiento de datos distribuidos, permite que los procesos, datos y otros aspectos de los sistemas de procesamiento de datos se dispersen dentro de una organización, esta dispersión proporciona que un sistema tenga mayor capacidad de respuesta para satisfacer las necesidades del usuario, aparte de proporcionar mejor tiempo de respuesta, y minimizar el costo de comunicaciones en comparación con un enfoque centralizado.

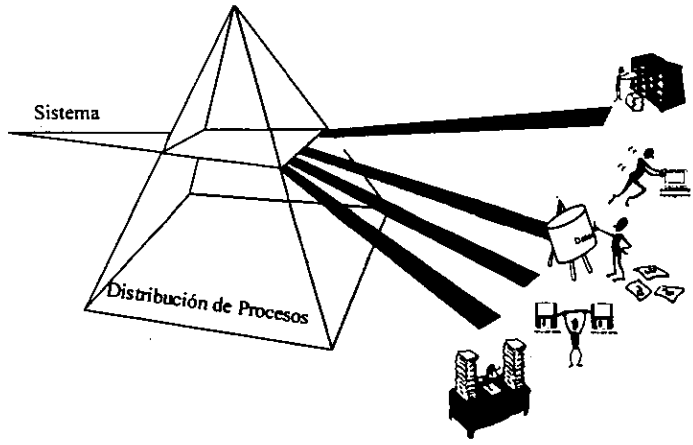


Figura II.2.1 Diferencia de un Proceso Tradicional y un Proceso Concurrente

Una manera sencilla de comprender como funciona la distribución de procesos se muestra en la Figura II.2.1, donde en Sistema o proyecto pasa por un prisma como si fuese un haz de luz, a partir de este los procesos se dividen en diferentes áreas donde cada uno tiene una función especifica.

Frecuentemente en los sistemas de procesamiento distribuido, las computadoras no comparten memoria principal; como en el caso de este proyecto de tesis, cada una es un sistema computacional aislado. Por lo tanto, no funcionan las técnicas de se basan en memoria compartida, tales como semáforos y el uso de un área de memoria común, por lo que se usan técnicas que se basan en paso de mensajes.

¿ Qué relación tiene una red de computadoras con los sistemas distribuidos? Un sistema de red son computadoras que interactúan por medio de comunicación de software y hardware. Los usuarios saben en que computadora están atendiendo sus tareas. Los sistemas distribuidos son sistema de software que distribuye las tareas de los usuarios sobre una red, obteniendo con ello la ventaja de tener una red más flexible, homogénea y estándar. Con un sistema distribuido diseñado apropiadamente las empresas tienen la capacidad de accesar cualquier información a través de cualquier sistema (o usuario) que tenga la autorización apropiada.

El uso de procesos distribuidos sirve de apoyo para aplicaciones tales como correo electrónico, transferencia de archivos y acceso desde terminales remotas, que involucran el intercambio de datos entre sistemas computacionales. Las computadoras tienen una identidad distinta para el usuario y para las aplicaciones, que deben comunicarse con otras computadoras por referencia explícita.

En la Figura II.2.2 se muestra el modelo que comúnmente es usado para pasar mensajes distribuidos, el cual se denomina cliente-servidor.

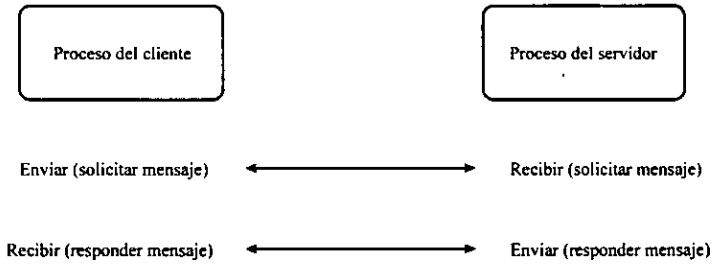


Figura II.2.2 Paso de Mensajes Distribuidos

Un dispositivo de paso de mensajes confiable es el que garantiza la entrega de información cuando se requiere del uso de este, tal dispositivo usa un protocolo de comunicación que transporta el mensaje, para ejecutar la verificación de errores, reconocimiento y retransmisión de mensajes.

Para el desarrollo de este proyecto en el cual, se realiza la distribución de una red neuronal, se hará uso de la invocación de métodos remotos. Una variación en el modelo de paso básico de mensajes es la llamada de *procesamientos remotos*[1], la cual es un método común y eficaz para encapsular la comunicación en un sistema distribuido, es decir, con esta técnica se permite que programas en diferentes máquinas interactuen usando un enfoque simple de llamada/regreso de procedimientos, de la misma manera que si los programas estuvieran en la misma máquina.

El concepto básico para el procesamiento distribuido es la concurrencia. Cuando múltiples procesos ejecutan en forma concurrente aparecen aspectos de cooperación y solución de conflictos, mismos, que dan inicio a una distribución de procesos. Todos los aspectos a los cuales se enfrenta un sistema concurrente, como exclusión mutua, interbloqueo e inanición, también se presentan en los sistemas distribuidos.

Un algoritmo distribuido se caracteriza por tener las siguientes propiedades:

- Todos los nodos tienen una cantidad igual de información, en promedio.
- Cada nodo tiene sólo una imagen parcial del sistema total y debe decidir en base a esta información.
- Todos los nodos tienen igual responsabilidad para la decisión final
- Todos los nodos gastan en promedio igual esfuerzo al efectuar una decisión final.
- La falla de un nodo, en general, no da como resultado la falla en el sistema total.

En algunos sistemas distribuidos se requiere que toda la información sea conocida por cualquier nodo y que se comuniquen entre sí.

En los sistemas distribuidos, los recursos se reparten en varios sitios y el acceso a ellos lo regulan procesos de control que no tienen el conocimiento del estado global del sistema, y por consiguiente deben decidir basándose en información local. La principal dificultad que presenta la detección de interbloqueo distribuido es que cada sitio conoce sólo sus propios recursos, mientras que un interbloqueo puede involucrar recursos distribuidos.

Cada uno de los procesos que conforma un sistema distribuido, se comunica a través de mensajes, suponiendo que alguno de ellos espera un mensaje de otro miembro del grupo y no existe el mensaje en transito, el proceso se interbloquea. Otra manera en la cual puede ocurrir el interbloqueo en un sistema de paso de mensajes, se debe a la asignación de buffers para el almacenamiento de mensajes en transito.

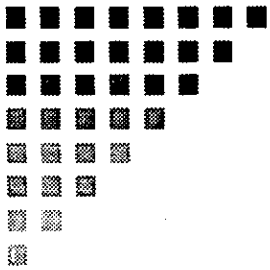
[1] Procesamiento Remoto: la interacción de dos programas en máquinas diferentes.

Los sistemas distribuido se encuentran compuestos por un conjunto de módulos separados, a estos módulos se les llama a menudo agentes, ya que se espera que cada módulo actúe como una entidad resolutoria de problemas por derecho propio estos sistemas presentan diferentes ventajas como:

- Modularidad del sistema: Es más fácil construir y mantener un conjunto de módulos casi independientes que uno enorme.
- Arquitecturas de computación rápida: A medida que los sistemas se hacen más complejos, también necesitan ciclos más complejos para resolver dichos problemas. Aunque la tecnología crece aceleradamente y se crean máquinas más rápidas, en problemas muy complejos, los verdaderos aumentos de velocidad se dan en un simple grupo de procesadores independiente, los sistemas distribuidos están más capacitados para utilizar dichas arquitecturas.
- Perspectivas múltiples: Los sistemas distribuidos facilitan el hecho de resolver problemas muy extenso que requieren de un grupo de trabajo, es decir, un conjunto de personas que resuelvan cada una de las partes del sistema.
- Problemas distribuidos: Algunos problemas son distribuidos por naturaleza. Por ejemplo puede existir diferente información disponible para el funcionamiento del sistema, en diferentes localizaciones físicas.
- Confiabilidad : Si un problema se distribuye a través de módulos de diferentes sistemas, la resolución del problema puede continuar, incluso si uno de los sistemas fallase.

En muchos sistemas se requieren que los datos o problemas no sólo se encuentren centralizados, sino que se distribuyan, con lo cual se tienen un aumento en el rendimiento, más disponibilidad de servicios, extensibilidad y confiabilidad.

El principal problema en el que nos enfrentamos al diseñar cualquier sistema distribuido, es la forma en que puede coordinarse la acción de los módulos individuales que lo componen, de manera que trabajen todos juntos, de modo eficaz.



CAPÍTULO III

VENTAJAS DE UNA RED NEURONAL EN SISTEMAS DE DISTRIBUCION Y CONCURRENCIA

"Reconoció que no debía de permanecer pasiva. Entonces sintió una imperiosa necesidad de distinguir su propia existencia, dándole sentido y orientación a sus actos, acciones y esfuerzos para lograr sus sueños y metas siempre con deseos de realización." A.L.C.

III.1 Ventajas de Computo Distribuido en un Red Neuronal

Comúnmente en el área de Sistemas Computacionales, los desarrolladores de software, nos enfrentamos a una infinidad de aplicaciones que se desearian automatizar, pero no sucede así como consecuencia de la complejidad que implica la programación de un sistema, para permitir llevar a cabo esas tareas las cuales requieren de múltiples procesos simultáneos, con ello no se quiere decir que los problemas sean irresolubles, más bien son difíciles de resolver empleado algoritmo secuenciales.

Un ejemplo de los problemas que resultan complicados de resolver con algoritmos secuenciales son aquellos que requieren de Redes Neuronales, tema principal de este proyecto, a continuación se verá ¿por qué? y ¿para qué?, programar una Red Neuronal Distribuida.

Requerimos que una Red Neuronal sea distribuida, para facilitar el tiempo de procesamiento en el caso del reconocimiento visual de tramas, por el gran número de elementos que conforman estas, los cuales contienen información, estos elementos individualmente no se reconoce a simple vista que representan, sin embargo, colectivamente, representan objetos fácilmente reconocibles para un ser humano, pero no para un sistema computacional, frecuentemente estos problemas resultan aun ser más difíciles de reconocer con un algoritmo secuencial.

Aun más difíciles son los problemas tales como el ejemplo del reconocimiento de una imagen con ruido, debido a la incompatibilidad básica que existe entre la máquina y el problema. La mayoría de los sistemas actuales de computadoras se han diseñado para llevar a cabo funciones matemáticas y lógicas con velocidades que resultan inalcanzables para un ser humano. Incluso los microcomputadores que son relativamente poco sofisticados, pueden realizar cientos de miles de comparaciones numéricas o combinaciones por segundo. Sin embargo, no sólo la destreza aritmética es lo que se necesita para reconocer tramas complejas en entornos ruidosos.

En muchas aplicaciones de la vida real se desea que los sistemas computacionales resuelvan complejos problemas de reconocimiento de tramas, dado que la programación estructurada es evidentemente poco adecuada para este tipo de problemas, ya que resulta muy lento este reconocimiento, se requiere de sistemas que simultáneamente realicen el reconocimiento de tramas.

Tal ves muchos piensen que el coste adicional de tiempo no es significativo, pero esto es así, sólo en Redes Neuronales muy pequeñas, sin embargo, para Redes Neuronales demasiado grandes, el número de ciclos de memoria adicional que se necesitan debido al gran número de conexiones de una red neuronal desbordaría rápidamente al sistema de computación empleado para la mayoría de las simulaciones. Un sistema de computo distribuido, amplia las ventajas del diseño de redes neuronales, por lo que se puede crear un ambiente de red cliente/servidor y distribuir la aplicación del diseño del sistema, para disminuir la complejidad y el tiempo de reconocimiento.

En el caso de este proyecto el tiempo que nos llevaría el realizar el reconocimiento de varias líneas de caracteres, seria demasiado alto en comparación a cuando distribuye en varias computadoras el proceso de aprendizaje y reconocimiento, con lo cual disminuye el tiempo de ejecución, así mismo se podría invocar métodos remotos, para simular un sistema de red local.

Para realizar programación distribuida se requiere del manejo de objetos, existen diferentes tecnologías de objetos distribuidos como por ejemplo *CORBA*[1] que permite a las aplicaciones comunicarse mutuamente sin importar donde están localizadas, quien las haya diseñado o que plataforma de hardware y sistema operativo este utilizando, pero la programación de procesos distribuidos requieren de un estudio profundo y tedioso, aunado a ello su costosa adquisición, otro de los lenguajes que nos permite la utilización de procesos distribuidos es Java, el cual facilita las funciones para crear un ambiente cliente-servidor de forma muy natural, se encuentra a nuestro alcance, y es orientado a objetos, dicho lenguaje se estudiara con más detalle en el Capítulo V.

Al aplicar procesos distribuidos a un proyecto de redes neuronales, se tienen varias máquinas colocadas en distintos lugares para acelerar los procesos de aprendizaje y reconocimiento.

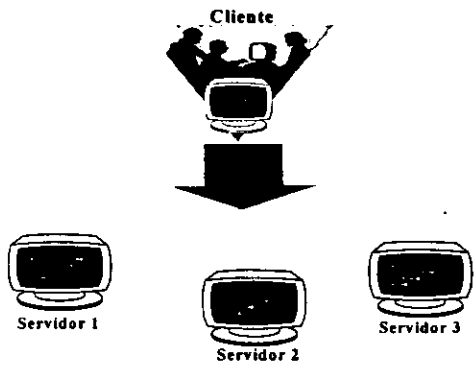


Figura III.1.1 Proceso Distribuido para una Red Neuronal

[1] CORBA : Common Object Requet Broker Architecture, Arquitectura de Agentes para Peticiones de Objetos Comunes.

III.2 Ventajas de Computo Concurrente en un Red Neuronal

Cuando se desarrolla un sistema de redes neuronales, parte del calculo de las sumas de productos, consume un tiempo importante de CPU, dado el gran número de nodos que poseen algunas redes neuronales, cada uno de los cuales podrían tener cientos o miles de entradas, que aumenta considerablemente el tiempo de CPU, ya que se deben de realizar varios millones de operaciones de punto flotante por segundo, por lo que es importante, considerar la forma en que se va a simular una red neuronal en una computadora con un único procesador.

Es necesario para mejorar el rendimiento del simulador, que en el diseño de software, se optimice la capacidad del CPU, para que comparta su tiempo entre las unidades de la Red Neuronal, es decir, cada unidad del modelo de la red neuronal compartirá el CPU durante un cierto periodo de tiempo.

Cuando requerimos desarrollar un sistema basado en la tecnología de Redes Neuronales y determinar sus beneficios prácticos, debemos de tomar en cuenta el aprovechar el tiempo de reconocimiento de determinado modelo, es decir, que pueda dividirse en tareas, para simular una ejecución simultánea, de esta manera es posible determinar con precisión la rapidez de un sistema de redes neuronales. Desafortunadamente, no tenemos acceso a un sistema diseñado específicamente para llevar a cabo un procedimiento masivamente paralelo de redes neuronales, sin embargo si tenemos acceso a herramientas de programación para llevar a cabo la simulación de un sistema de procesamiento concurrente para diseño de redes neuronales.

Las ventajas de dividir un programa en tareas son:

1. Por una parte acerca la solución del problema al mundo verdadero, en el que se realizan varias actividades concurrentes. El problema por lo tanto, es más realista que si se intenta enfocar mediante un programa secuencial.
2. Estructurar un sistema en tareas hace que el sistema sea más fácil de comprender, de construir y de mantener.
3. En todo programa secuencial hay momentos en el que el procesador esta parado esperando un suceso que puede ser una operación de Entrada/Salida, mediante la división de tareas se produce un mejor tiempo de ejecución del programa.
4. Dividir un sistema en tarea tiene como resultado la posibilidad de asociar las diferentes prioridades de cada tarea.

Considerando que el diseño de este sistema reconoce un texto de caracteres, el hecho de dividir las tareas facilita que podamos realizar concurrentemente el reconocimiento de caracteres, y así al mismo tiempo distribuir la red en diferentes máquinas.

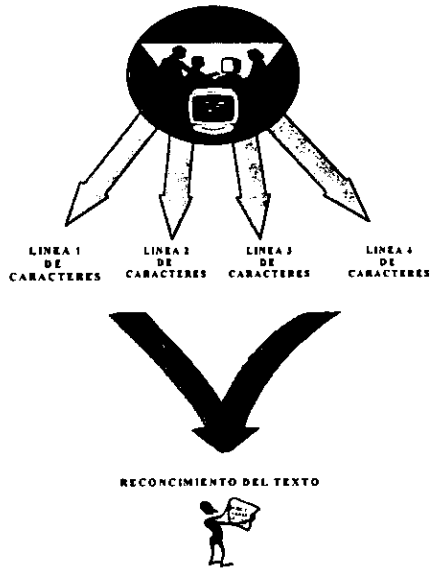
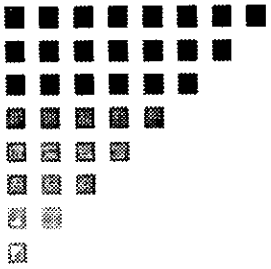


Figura III.2.1 Proceso Concurrente para una Red Neuronal

FALTA PAGINA

No. 44



CAPÍTULO IV

MODELO DE LA RED NEURONAL DISTRIBUIDA Y CONCURRENTE EN PROGRAMACION ORIENTADA A OBJETOS

"Te sientes con el derecho de exigir, juzgar y demandar más respuestas de los demás que de ti mismo, pero tú, ¿Qué estás dispuesto a dar...?"
A.L.C.

IV.1 Programación Orientada a Objetos

Las necesidades de una sociedad que crecen aceleradamente, imponen problemas que demandan sistemas de programación cada día más grande y con características más complejas que no son tan fáciles de resolver con métodos de programación convencionales o personalizados. Durante algún tiempo no se presto gran importancia a este problema, por lo que no se hacía nada para solucionarlo, con todo esto los costos para obtener un software de calidad y eficiencia han crecido aceleradamente. El software que se desarrollaba, estaba realizado por programadores que no consideraban el crecimiento a futuro, así como la idea de que sus programas fueran leídos o modificados por otras personas. Por ejemplo, cuando una compañía requería de una extensión o modificación total o parcial para un sistema, era más fácil y menos costoso volverlo a realizar que modificar el existente. Diversos autores han coincidido que las compañías en promedio gastan un 70% de su presupuesto para mantenimiento de software ya existente. (15)

En el año 1980 se realizaron investigaciones para apoyar y mejorar la calidad de programación tales como las métricas de software, las herramientas CASE, el empleo de prototipos, y conjuntamente se introdujo la reingenieria y un nuevo paradigma en la practica del desarrollo de software *el paradigma de los objetos.* En esta época se comenzó a realizar software que cumpliera con las necesidades del cliente, por lo que se aplicaba la *ingeniería de software,* que se define como la aplicación del conocimiento científico a la creación y construcción de soluciones costo-efectividad a problemas prácticos para el servicio. El costo-efectividad requiere que el producto sea fácilmente modificado y ampliado.

La crisis por la que pasa la construcción de software en gran parte se relaciona con la teoría de Fred Brooks que dice "la complejidad es una propiedad esencial (no accidental) de los sistemas de software. Esta complejidad de los sistemas de software frecuentemente excede la capacidad intelectual humana"(10). Respecto a esto el psicólogo George Miller dice "el ser humano solamente puede mantener la pista de aproximadamente siete objetos, entidades o conceptos a la vez, es decir, que la memoria recirculante inmediata necesaria para resolver problemas con múltiples elementos tiene una capacidad de 7±2 " (10); arriba de este número los errores llegan a ser numerosos, entre más complejo es un sistema, esta propenso a un mayor porcentaje de errores. La idea para simplificar esta tarea es la de construir programas modulares creando entidades independientes y fácilmente manejables. Por lo que el paradigma de la P.O.O viene a facilitar el desarrollo y mantenimiento de software cada vez más grande y complejo.

Día con día nos enfrentamos a cambios, por lo que surge la pregunta: ¿Existe la resistencia al cambio dentro de la vanguardia tecnológica? Si, los programadores, como todos los seres humanos somos resistentes al cambio. Todos estamos acostumbrados a nuestros modos tradicionales de hacer las cosas, a nuestras herramientas comunes, y a nuestras técnicas de programación conseguidas tras muchos años de tiempo y esfuerzo. El cambiar de mentalidad, el iniciar una nueva técnica por muy atractiva que esta

parezca, es un precio muy alto, tras toda una vida de trabajo. Por ello es que muchos programadores ven la P.O.O. como algo muy complejo e inútil de utilizar, y más que un apoyo en la programación lo vemos como un obstáculo abstracto, lo cual sólo forma una barrera que nos impide disfrutar de las grandes ventajas que nos ofrece la P.O.O. al facilitar el desarrollo de programas complejos, disminuyendo el tiempo de construcción.

Un cierto número de lenguajes de programación a contribuido a la evolución de los actuales lenguajes orientados a objetos, de los cuales a continuación se mencionan algunos: uno de los primeros lenguajes fue LISP que aparece en los años 50's. *LISP (LIST PROCESSING, procedimiento de listas)*, es un lenguaje de inteligencia artificial que introdujo el concepto de ligadura dinámica y las ventajas de un entorno de desarrollo interactivo en la evolución de los lenguajes orientados a objetos. *Simula*, desarrollado en los años 60's como lenguaje para programar simulaciones, contribuyo con el concepto de clase y los mecanismos de herencia (conceptos que analizaremos posteriormente en el desarrollo de este capítulo). La abstracción a datos fue introducida en los años 70's en los lenguajes *Ada* y *Modula2*. (1)

Con la aparición de *C* en los años 70's como lenguaje de programación, él cual era extremadamente popular en todas las plataformas, exigía seguir a la vanguardia en la tecnología de software, y aparece la extensión del lenguaje C para aplicaciones orientadas a objetos con el nombre de *C++*, el cual ayudo que dentro de la comunidad de programación creciera el uso del enfoque orientado a objetos, esto justifica por que se emplea con mayor frecuencia C++ que Smalltalk o Simula.

Apple Computer incorporó a *Pascal* en el año de 1985 las primeras aplicaciones o extensiones orientadas a objetos para la computadora Macintosh. Otras versiones del lenguaje Pascal orientados a objetos han sido recientemente lanzados por Microsoft Corporation y Borland International, Inc., para uso en los entornos DOS y OS/2. (15)

Posteriormente siguieron apareciendo lenguajes de P.O.O. En la figura IV.1.1 se muestra la evolución que han tenido los lenguajes hacia el concepto de la programación orientación a objetos.

Los programas de los años 90's exigen características de las arquitecturas de la P.O.O., por el apoyo que da a la construcción de programas, al simplificar y organizar el desarrollo de aplicaciones a partir de librerías de clases ya existentes.

Hubo varios puntos en contra que hizo poco atractivo el uso de la P.O.O.:

- En sus inicios los creadores de programas Empresariales y Comerciales, lo consideraban como un sistema de ventanas, restando ventajas a dicha programación.

- La necesidad de plataformas especializadas que se requería para el empleo de lenguajes basados en este enfoque, lo hacia poco atractivo económicamente.

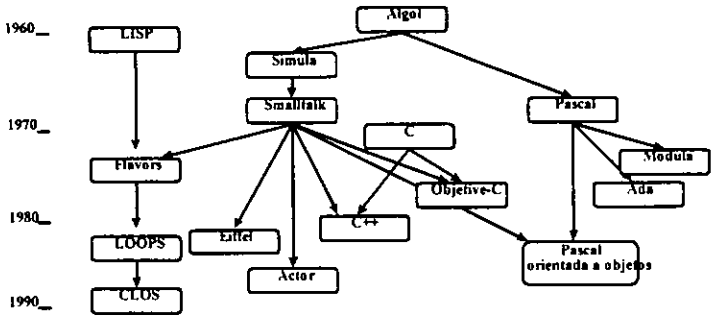


Figura IV.1.1. Evolución de los lenguajes orientados a objetos.

Los obstáculos precio/rendimiento que evitaban el empleo general de la tecnología orientada a objetos han ido desapareciendo debido a que ha sido impulsado por opiniones como: "Los programadores que utilizan las herramientas actuales de desarrollo para crear aplicaciones indican que sin estratos o capas de abstracción el desarrollo de aplicaciones se estrangula". (19)

Durante el desarrollo de este capítulo hemos visto porqué surge la P.O.O. ¿Pero, qué es la Programación Orientada a Objetos? La P.O.O es entorno que hace de la programación un modelo más natural, acercándonos al uso de funciones naturales de nuestro intelecto, con ello podemos hacer uso de herramientas para programar de una manera más rápida y sencilla.

Para desarrollar programas bajo el enfoque de la P.O.O. necesitamos comprender conceptos, que nos darán una expectativa más clara de lo que es la P.O.O, y por lo cual contribuye a la construcción de Redes Neuronales, así como a los Sistemas Distribuido y otras aplicaciones complejas.

En muchas ocasiones hemos oído hablar de *objetos,* pero, ¿qué son los objetos? Los objetos son módulos que contienen datos (propiedades) y formas de comportamiento (métodos), que actúan sobre esos datos, cada objeto tiene su propia estructura y forman parte de una organización que proporciona una función general. Figura IV.1.2.

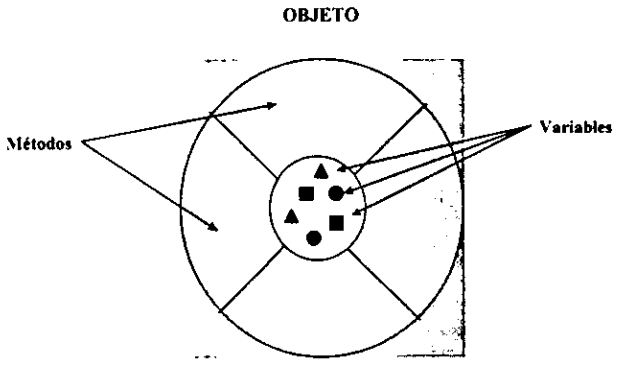


Figura IV.1.2. Representación de un objeto

Dentro de los objetos existen datos los cuales pueden ser números, matrices, arreglos, cadenas de caracteres y registros. Los objetos tienen determinadas características, y rangos que los definen, a lo cual se le denomina *propiedades de los objetos.*

Ahora, ya sabemos que son los objetos, pero, ¿ a partir de dónde surgen los objetos? Un objeto es un modelo o *instancia* de una clase, por lo que podemos decir que una clase es una plantilla de métodos, para crear objetos con características similares. Al definir una clase estamos creando código reutilizable, en lugar de escribir nuevamente el código tantas veces sea necesario, es decir, instanciamos la clase de acuerdo a nuestras necesidades.

Las variables de clase tienen valores almacenados en una clase; las *variables de instancia* tienen valores asociados únicamente con cada instancia u objeto creado a partir de una clase, las variables de instancia almacenan información ó datos locales en el objeto.

Las clases más altas en la jerarquía se denominan *superclases*, las clases de niveles inferiores heredan el comportamiento de las clases de niveles superiores. En la figura IV.1.3 vemos un ejemplo claro de los conceptos de clase. En este ejemplo un conjunto de animales forma una superclase de animales, que a su vez se compone de clases de animales, en este caso animales mamíferos y animales ovíparos. Cada animal tiene sus propias características que los hacen ser similares pero no iguales, por lo que podemos definir en este ejemplo, que un objeto podría ser el burro o la tortuga, u otro animal.

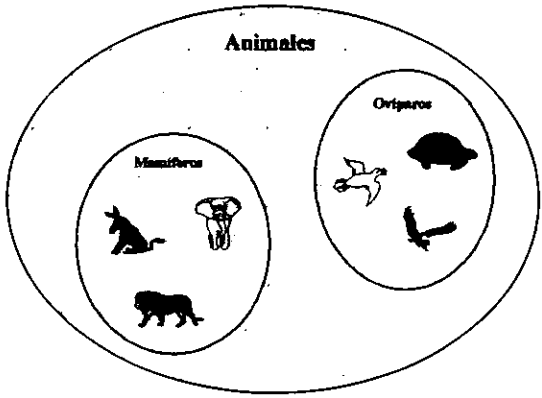


Figura IV.1.3. Ejemplo de los conceptos de clase

Como ya lo mencionamos, los objetos están capacitados para ejecutar determinadas acciones. La acción sucede cuando un objeto recibe un mensaje, es decir, una solicitud para que dicho objeto se comporte de alguna forma, esta forma de comportamiento recibe el nombre de *método*. La relación que hay entre un objeto y otro, se vincula principalmente en sus métodos.

Los métodos pueden enviar también mensajes a otros objetos solicitando acción o información. En la Figura IV.1.4 se muestra un ejemplo de como dos objetos se comunican a través de mensajes. En donde el objeto "A" envía un mensaje al objeto "B", a través de los métodos "A" y "B".

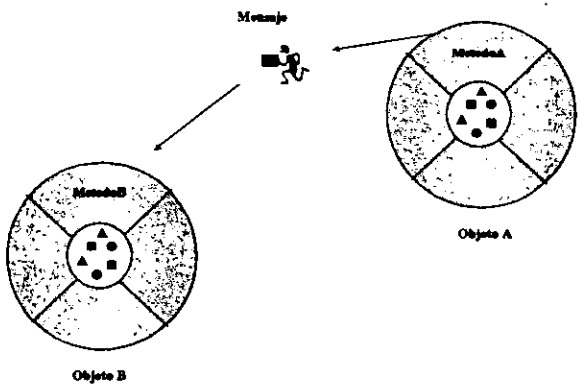


Figura IV.1.4. Envío de mensajes entre dos objetos

Solamente un método del propio objeto puede disponer de los datos del interior del mismo para su manipulación. La orientación a objetos fomenta la modularidad haciendo destacar las fronteras entre objetos, y de esta manera poder ocultar los detalles de realización y enfocarnos explícitamente a la comunicación entre los objetos.

Un ejemplo del funcionamiento de los métodos y mensajes lo vivimos nosotros mismo, por ejemplo cuando queremos caminar, correr, saltar, nuestro cerebro envía un mensaje a los nervios receptores del movimiento de nuestras piernas, para realizar determinada acción, que nos permite mover las piernas a nuestro gusto, sin controlar individualmente cada músculo.

IV.2 Beneficios de la P.O.O.

Las principales ventajas de la Programación orientada a objetos resultan de gran apoyo para hacer frente a los temas esenciales de la ingeniería de software (tales como las redes neuronales tema que es de importancia para el desarrollo de este trabajo) las cuales podemos englobar en dos grandes ventajas:

- el entendimiento de la complejidad de un sistema,
- mejora de la productividad en el proceso de desarrollo de software.

Estas ventajas nos permiten escribir código reutilizable, escribir código fácil de mantener, depurar módulos de código existentes (en ciertos casos programas que nosotros mismos no hemos creado), y compartir código con otras personas, ya que con métodos anteriores se llevaba más tiempo en entender los programas ya realizados, para adaptarlos a nuevas necesidades.

La complejidad se reduce y la productividad se mejora cuando se puede volver a utilizar el código que en un principio se creo con conciencia y calidad. Descomponer una aplicación en entidades y relaciones que sean comprensibles para los usuarios es una técnica convencional de diseño y análisis. Es más fácil diseñar y realizar aplicaciones orientadas a objetos porque los objetos de la área de aplicación, corresponden directamente con los objetos en el campo del software.

El procesamiento distribuido es una fuerza que impulsa, y guía a la P.O.O. y viceversa porque la P.O.O. facilita la realización e interconectividad de procesos distribuidos haciendo llamada a objetos y no a funciones. La P.O.O. se aplica a la mayoría de los principales componentes de software, incluyendo lenguajes, bases de datos e interfaces.

La P.O.O permite que los programadores pueden diseñar aplicaciones más complejas en parte que son modulares e intercambiables. A continuación se definen claramente las bondades de la P.O.O que lo hacen útil y atractivo.

La reutilización es la clave para aumentar la productividad ante el reto de la complejidad creciente. La encapsulación, la herencia y el polimorfismo, interactúan para lograr esa enorme ventaja dentro de la P.O.O. La reutilización de software requiere que los programadores adopten nuevos comportamientos, nuevos valores, y una nueva ética de programación como: "Debe preferirse la adopción de las clases creadas por otros a la realización de una clase nueva. La revisión del código existente para identificar oportunidades de reutilización debe tener prioridad sobre la estructura de uno nuevo. El programador debe de crear clases simples y reutilizables en vez de complejas e inestructurables" (19)

Hemos hablado mucho de la reutilización de programas, pero, ¿cómo podemos hacer programas reutilizables? la P.O.O cuenta con tres características principales que sirven de base:

A) La *herencia* es el aspecto más importante de la organización jerárquica de los objetos. Es a través de ella que podemos hablar de los objetos como estructuras organizadas. Y tener objetos que tomen características de las clases de nivel superior en la jerarquía. La herencia nos permite, compartir métodos y datos entre todos los objetos de una jerarquía de clases.

La herencia puede ser de dos tipos fundamentales: simple y múltiple. La herencia simple, se da cuando una subclase puede heredar datos y métodos de sólo una clase padre, así como añadir ó sustraer determinado comportamiento. La herencia múltiple se refiere a la posibilidad de que una subclase adquiera los datos y métodos de más de una clase, es decir, cuando una subclase hereda sus peculiaridades de más de una clase padre. La herencia múltiple es útil al construir comportamientos compuesto a partir de más de una rama de una jerarquía de clases.

El proceso de subclasificar por medio del mecanismo de herencia permite que la programación se convierta en un proceso de programar solamente las diferencias entre la subclase y superclase o clase padre. La funcionalidad de la herencia no sólo tiene efecto en la eficacia y calidad de la construcción de sistemas, sino también en la asignación y empleo de los programadores, es decir, la herencia facilita que grandes proyectos pueden ser divididos entre los miembros de un equipo de desarrollo sin importan que tan grande sea.

B) La *encapsulación* hace que la labor de desarrollo de un programa pueda ser absolutamente modular, con lo cual se afirma que todo se convierte en un mecanismo de asociar objetos y establecer la interrelación entre ellos. La encapsulación describe el conjunto de métodos y datos dentro de un objeto de forma que el acceso a los datos se permite solamente a través de los propios métodos del objeto, es decir, que permite localizar y ocultar los detalles de un objeto, y sólo se puede acceder a ellos de manera controlada. Los datos, son variables, dichas variables han de ser locales al objeto, para evitar accesos inesperados a las mismas. La encapsulación es el reflejo de la modularidad de los sistemas de P.O.O.

El mantenimiento de la coordinación entre las estructuras de datos y las funciones, consume una considerable cantidad de recursos e introduce bastantes oportunidades de error. En la P.O.O. el encapsular los datos y procedimientos de los objetos simplifica el proceso, facilita el mantenimiento y reduce la probabilidad de errores en el proceso de programación.

C) Otra de las grandes ventajas es el *polimorfismo* con el cual un usuario puede enviar un mensaje genérico y dejar los detalles exactos de la realización para el objeto receptor, el objeto receptor actúa en respuesta a los mensajes que reciben; el mismo mensaje puede originar acciones completamente diferentes al ser recibido por diferentes objetos, es decir, dos métodos con el mismo nombre pero con diferente número de parámetros pueden ejecutar acciones distintas, a este efecto se le denomina *polimorfismo*.

Asociado a ello hemos oído hablar de la *persistencia* que se refiere a la permanencia de un objeto, es decir al tiempo durante el cual se asigna espacio y permanece accesible en la memoria de la computadora. Cuando un objeto ya no es necesario, es destruido y recuperado el espacio de memoria que tenia asignado. La recuperación automática del espacio de memoria se denomina normalmente *recolección de basura*.

Los sistemas de P.O.O. son muy *extensibles*, es decir, muy susceptibles de crecer de acuerdo a las necesidades futuras. El tener encapsulada toda la información, poner un objeto ó quitarlo de un sistema no tiene por qué afectar a todo un sistema, sino a pequeños módulos del mismo. También se requiere que un sistema sea *portable*, para que pueda ser transportado sobre diferentes sistemas hardware y software.

¿Por qué usar la P.O.O. y no los métodos tradicionales?

Los métodos tradicionales aplican procedimientos activos a datos pasivos. A diferencia de los datos pasivos de los programas convencionales, los objetos pueden actuar y se activan mediante mensajes desde otros objetos. La orientación a objetos permite al desarrollador de sistemas a concentrarse en los temas más importantes e "ignorar" el resto. Grandy Booch resume la diferencia fundamental entre programación procedimental (o basada en procedimientos) y la programación orientada a objetos de la siguiente forma: "Lea las especificaciones del software que desee construir. Subraye los verbos si persigue un código procedural, o los nombres si su objetivo es un programa orientado a objetos" (19)

Algunos conceptos orientados a objetos son análogos a los métodos de programación convencional. A continuación se observan algunos contrastes entre términos y conceptos convencionales, y aquellos orientados a objetos:

- Un *método* es como un procedimiento porque ambos contienen instrucciones de procedimientos.

- Una *clase* es como un tipo abstracto de datos, aunque para los programadores orientados a objetos, el proceso de escribir los datos no se realiza fuera de la clase.

- La *herencia* no tiene una analogía inmediata en la programación convencional.

- El paso de mensajes reemplaza a las llamadas a funciones como método principal de control en los sistemas orientados a objetos. Con las llamadas a función, los valores se presentan y el control regresa a la función que efectúa la llamada. Los objetos entran en acción gracias a los mensajes, ya que el control está distribuido.

En la Tabla IV.2.1, se observa claramente la comparación de conceptos de programación tradicional y orientada a objetos.

TÉCNICAS ORIENTADAS A OBJETOS	TÉCNICAS TRADICIONALES
Métodos	Procedimientos, funciones o subrutinas
Variables de modelo	Datos
Mensajes	Llamadas a procedimientos o funciones
Clases	Tipos abstractos de datos
Herencia	(No existe técnica similar)
Llamadas bajo control del sistema	Llamadas bajo control del programador

Tabla IV.2.1. Comparación de conceptos de programación tradicional y la orientada a objetos (19)

En el método procedimental, los programadores centran su atención e los temas del lenguaje mientras que en el entorno orientado a objetos, el tema importante es cultivar una librería de clases ó un conjunto de objetos que puedan ser utilizados en circunstancias diversas. En la programación estructural las funciones son entidades aisladas que resuelven una tarea, pero no pueden ser insertadas dentro de una estructura superior a la misma, fácilmente reconocible por nuestro intelecto, los objetos se basan en las funciones más simples de nuestro intelecto como el clasificar.

La manera en que se construyen los programas orientados a objetos, consiste en considerar los programas como un conjunto de una única entidad básica, el objeto, el cual combina los datos con los métodos (procedimientos) que actúan sobre ellos. A diferencia de los programas tradicionales, que utilizan procedimientos para realizar acciones sobre un conjunto independiente de datos pasivos, los objetos reciben las peticiones e interactúan enviando mensajes a cada uno de los demás. Para poder observar claramente esta diferencia observemos la figura IV.2.1.

Como anteriormente se menciono los lenguajes orientados a objetos soportan los mecanismos de objetos, clases, métodos, mensajes y herencia. Los lenguajes tradicionales no soportan la herencia, uno de los mecanismos más potentes de los lenguajes orientados a objetos. La herencia permite al programador orientado a objetos no partir de cero. La tabla IV.2.2. muestra las diferencias fundamentales entre los lenguajes procedimentales y orientados a objetos.

En los últimos años se esta optando, desarrollar programas utilizando el entorno de la P.O.O. ya que es:

- fácil de emplear,
- fácil de mantener,
- permite el desarrollo de aplicaciones complejas,
- más fácil de utilizar
- y flexible.

Con ello permite mejorar la creación y mantenimiento de software, así como la velocidad de producción de los programadores, incrementando la funcionalidad para incorporar nuevas aplicaciones.

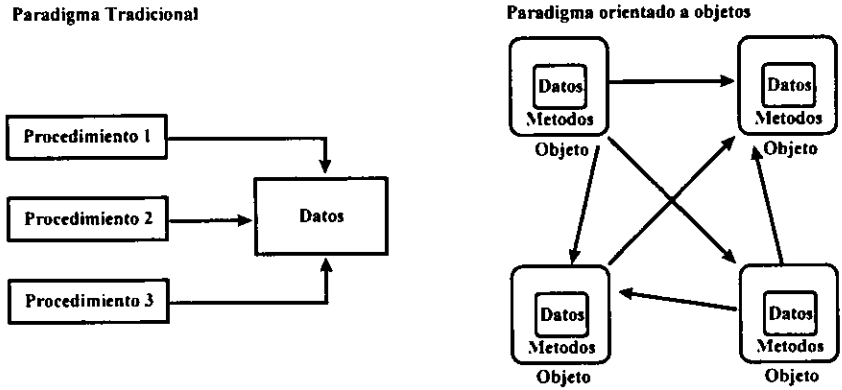


Figura IV.2.1. Comparación entre los paradigmas orientados a objetos y los tradicionales

	TRADICIONAL	ORIENTADO A OBJETOS
Realización de acciones	Los procedimientos actúan sobre los datos	Los mensajes se envían a los objetos
Abstracción de datos y funciones	Combinaciones de tipos de datos y procedimientos predefinido actuando sobre estos tipos de datos	Métodos y mensajes
Encapsulación	Componentes de la biblioteca de software	Objetos
Herencia	(No existe analogía)	Clases y subclases

Tabla IV.2.2. Diferencias fundamentales entre los lenguajes orientados a objetos y los tradicionales

¿ La P.O.O desplaza a la programación estructurada? No, los enfoques de programación estructurada son útiles para determinadas aplicaciones pero no para todas, como aquellas que requieren de un análisis profundo y que por su complejidad no son tan fáciles de desarrollar bajo este enfoque, las técnicas de los sistemas de P.O.O. tratan de simular la forma natural de pensar del hombre y, por lo tanto, son más fáciles de usar que los modelos funcionales típicos de la programación estructurada.

Los métodos orientados a objetos proporcionan una metodología de programación soportada por los nuevos lenguajes y herramientas, para mejorar la productividad, así mismo, proporcionan la consistencia y flexibilidad necesaria para hacer más fácil el empleo de las aplicaciones y su adaptación a necesidades especificas. La P.O.O. representa un cambio radical en la forma de desarrollar y utilizar el software; la reutilización del software implica que las clases pueden mezclarse y ajustarse y ser fácilmente modificadas para construir nuevas aplicaciones, la encapsulación de datos y procedimientos cambia toda la naturaleza del proceso de programación.

Las herramientas de desarrollo en la P.O.O. facilitan también el proceso de depuración y búsqueda de errores que ocurre al desarrollar componentes independientes e integrarlos en un sistema completo.

Es una controversia de la P.O.O. el decir que "los programas orientados a objetos son más fáciles de escribir", ya que los conceptos son todavía más abstractos que los métodos tradicionales, al cambiar de una metodología a la P.O.O, puede parecer complicado la comprensión al inicio, pero los objetivos del diseño pasan a modelar los objetos que existen en el mundo y sus comportamientos individuales, trayendo con ello grandes ventajas y reduciendo la complejidad en los sistemas, que con métodos tradicionales no se lograria. Hay dos puntos importantes respecto a esto que se deben aclarar: el hecho de que un programa sea escrito en bajo el enfoque orientado a objetos, no es total garantía de que el programa sea mucho más flexible para solucionar un problema, ya que depende de la correcta aplicación y entendimiento de los conceptos de la P.O.O. por lo que exige del programador un nuevo punto de vista y ética de programación. El que la P.O.O. tenga grandes ventajas no quiere decir que desplaza a la programación estructurada, porque para programar un objeto su interior debe estar correctamente estructurado.

Los objetos son relativamente fáciles de definir, realizar y mantener porque reflejan la modularidad natural de una aplicación. Con el refuerzo de la modularidad y a través del uso del mecanismo de herencia, los objetos de software pueden utilizarse de nuevo en aplicaciones futuras y así reducir substancialmente la cantidad de código nuevo que debe escribirse.

La modificación de un programa orientado a objetos no afecta a la estructura del programa. En la figura IV.2.2. se muestra un ejemplo claro de ello, al añadir un objeto no tenemos que modificar lo que ya se ha programado, simplemente se añade como una pieza más de un rompecabezas.

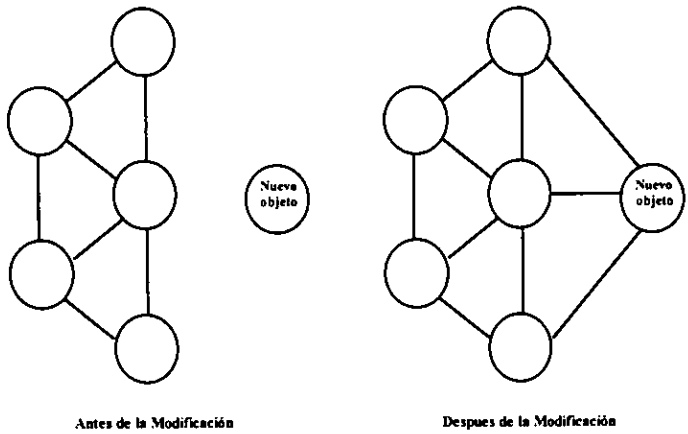
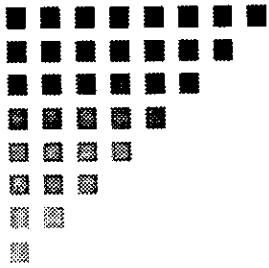


Figura IV.2.2. Modificación de un programa orientado a objetos

¿Para qué utilizar P.O.O. para la construcción de un sistema de redes neuronales artificiales? La programación orientada a objetos mejora no sólo el proceso de desarrollo del software sino también de flexibilidad y utilidad del software resultante. La realización de sistemas de Redes Neuronales distribuidas, por su arquitectura dependiendo del problema a resolver, tienden a ser modificadas dinámicamente en el número de capas y nodos que las componen, así como su comunicación, el programar una red neuronal bajo el enfoque único de la programación estructurada, involucra el hecho de escribir el mismo código así como número de capas se tenga, y más aun si posteriormente deseamos hacer crecer o disminuir el número de capas, tendríamos que reprogramar todo el sistemas para que la comunicación e interconexión de dichas capas no se vea afectada. Por ello es de gran ventaja hacer uso de la P.O.O. que nos permite modificar interactivamente un sistema, sin más problemas que la construcción de instancias a una clase tantas veces como sea necesario.



CAPÍTULO V

JUSTIFICACIÓN

"Te han hecho mucho daño...¡La libertad no se encuentra huyendo de ti mismo... ! Necesitas volver a creer en ti, en tu medio y el ser humano, confrotándote contigo mismo y tu realidad!... no detengas tu vuelo no desfes de dominar los vientos... ni las corrientes tradicionales." A.L.C.

V.1 ¿Por qué Java?

Al empezar a simular redes neuronales, frecuentemente resulta necesario diseñar el software de simulación de tal modo que el tamaño de la red se pueda determinar dinámicamente. La justificación de esta observación se basa en la idea de que no es deseable tener que reprogramar y recompilar una aplicación de redes neuronales simplemente porque se desea cambiar el tamaño de la red, por ello se requiere de un lenguaje de programación que permita la definición dinámica de una red neuronal, también es importante considerar los aspectos de transportabilidad y reutilización del código desde los primeros momentos de la realización de nuestro simulador. Existe una gran variedad de lenguajes para la construcción de redes neuronales, pero Java es un lenguaje de programación poderoso que ofrece la potencia del diseño orientado a objetos, y facilita la construcción de sistemas distribuidos, así como la construcción de redes neuronales dinámicas.

Java es poderoso por su *independencia de plataforma,* es decir, que los programas en Java pueden ejecutarse en cualquiera de las siguientes plataformas, sin necesidad de hacer cambios: Windows/95 y /NT, Power/Mac, Unix (Solaris, Silicon Graphics,...). Otros lenguajes independientes de plataformas también son sistemas *interpretados* (es decir, que son ejecutados línea por línea), como BASIC, Tcl y Perl. Estos lenguajes sufren un déficit de rendimiento casi insalvables. Si embargo Java fue diseñado para ejecutarse bien en un procesador de muy poca potencia o en una red de computo; la ejecución mediante interpretes es lenta en comparación con la de código de máquina totalmente compilado.

El compilador de Java compila el código fuente en un código de byte. El término *bytecode o código de byte*, viene por el hecho de que cada parte del programa en Java se reduce a una secuencia de bytes que representan instrucciones para una máquina virtual. Un intérprete de Java, carga el código de byte del programa e inspecciona cada byte del software, cambiando el estado del procesador virtual simulado en él interprete, para reflejar el estado del programa.

La mayoría de las metodologías y herramientas utilizadas para desarrollar sistemas cliente/servidor están basados en objetos. El Lenguaje Java, soporta el diseño de programas orientados a objetos y con ello facilita las comunicaciones cliente/servidor. Los objetos se han convertido en la clave para construir arquitecturas de informática distribuida, arquitectura que puede construirse incrementalmente para reflejar los conocimientos técnicos y de negocios, que pueden evolucionar y cambiar con el tiempo.

V.2 Java

Java, cuya denominación original fue Oak (1991), fue diseñado para programar dispositivos electrónicos de consumo y crear una red heterogénea de productos electrónicos domésticos. Para funcionar en este entorno, Java fue definido como un intérprete de tiempo real fiable y de pequeño tamaño que, sobre todo, debía ser portable y funcionar a través de canales de comunicación. La primera aplicación práctica proyectada por el equipo de Sun fue utilizar Java en decodificadores de televisión.

Después, a mediados del año 1994, los ingenieros de Sun se dieron cuenta que podían utilizar el lenguaje Java para crear un navegador de *WWW* (World Wide Web). Así apareció *HotJava*, un navegador escrito totalmente con lenguaje Java, el cual, permite acceder a Webs que contienen *applets*[1] .

Los applets, son aplicaciones pequeñas, seguras, dinámicas, multiplataforma, activas y en red. Las applets se pueden configurar y distribuir de manera segura a través de integradores de contenido de cliente con la misma facilidad de cualquier otro aspecto de *HTML* (lenguaje de marcado de hipertexto).

Lo que hace de Java un lenguaje tan atractivo principalmente frente a cualquier otro lenguaje de programación es que es independiente del hardware o plataforma, tanto en el ámbito de código fuente como en el ámbito binario.

Más importante que la obvia incompatibilidad entre PC y Mac es el tema de la longevidad y portabilidad del código. Si se escribe hoy un programa, no hay ninguna garantía de que se ejecutará mañana, incluso en la misma máquina. La evolución en la tecnología que crece aceleradamente y los cambios en los sistemas operativos y procesadores, provocan que un programa deje de funcionar correctamente. Los diseñadores de Java pusieron mayor énfasis al desarrollar él interprete, de modo que se pueda realmente escribir una vez el código, y ejecutar en cualquier sitio, en cualquier momento y para siempre. Simplemente el programador tiene que preocuparse por escribir un buen programa y Java se asegurará que funcione en Macintosh, PC, UNIX y plataformas futuras.

Java utiliza un compilador (javac) para generar los *bytecode o código de byte* (código binario), para realizar la independencia de la plataforma. Para ejecutar el programa se llama a un intérprete de bytecode denominado simplemente java, el cual depende de cada arquitectura. Este interprete está escrito en ANSI C y soporta los tipos de datos definidos por el IEEE[2], para que resulte portable a cualquier plataforma.

---

[1] Es un programa que se ejecuta en un navegador (Browser) de la red WWW.

[2] IEEE : Intitute of Electrical and Electronics Enginers

La capacidad de que un archivo binario sea ejecutable en cualquier plataforma es crucial, por ejemplo WWW, por definición, también es independiente de cualquier plataforma, así como cualquier archivo HTML puede ser leído en cualquier sistema (sea PC compatible, Macintosh, UNIX, etc.), las applets Java pueden ejecutarse en cualquier sistema que tenga un navegador que soporte Java.

La desventaja de utilizar bytecode se encuentra en la velocidad de ejecución. Los programas desarrollados para una plataforma en concreto se ejecutan más rápidamente entre 10 y 15 veces, que los bytecode de Java. Sin embargo, si el tiempo de ejecución es determinante se pueden crear programas en Java específicos para una arquitectura en concreto, sacrificando la portabilidad.

La ejecución del primer programa de Java, fue diseñada para cumplir los requisitos del mundo real sumando una lista de frases: "Simple y poderoso, Seguro, Orientado a objetos, Portable, Robusto, Interactivo, Neutral a la arquitectura, Interpretado y de Alto rendimiento, Fácil de aprender" (3) Java es un lenguaje puro ya que no es ninguna extensión de ninguna aplicación, muchos lenguajes orientados a objetos, no son aceptados en la comunidad de programadores, pero Java tiene un gran parecido a C, así que no fue tan complicado que fuera aceptado por los comunidad de programadores.

Java es un lenguaje que ha sido diseñado para producir software de alta calidad y reutilizable. A continuación se definen las características que hacen a Java sobre salir entre varios lenguajes.

Orientado a Objetos

Java es un lenguaje puro en la aplicación de programación orientada a objetos, por lo tanto soporta las tres características de este tipo de programación: encapsulación, herencia y polimorfismo, las cuales se describirán en el subcapítulo V.3.

La Simplicidad de Java

El aprender a programar en java se vuelve fácil una vez que se han comprendido los conceptos básicos de programación orientada a objetos. En java hay un número reducido de maneras de realizar una tarea dada, y no porque no se puedan programar, si lo hace asumiendo que cualquier comportamiento que pueda desear ya está encapsulado en los objetos incorporados que simplemente hay que enumerar. La simplicidad se demuestra al escribir nuestros programas en Java con un estilo sencillo y directo, es decir, no complicar el lenguaje probando usos grotescos que están de más, ¡para qué inventar el hilo negro en pleno Siglo XX!.

Java se ha diseñado para eliminar las complejidades de otros lenguajes como C y C++. Si bien Java posee una sintaxis similar a C, con el objeto de facilitar la migración de C hacia a Java, pero diferentes para el desarrollo de sistemas:

- Java no posee aritmética de apuntadores: La aritmética de apuntadores es el origen de muchos errores de programación que no se manifiestan durante la depuración y que una vez que el usuario los detecta son difíciles de resolver.
- No se necesita hacer delete: Determinar el momento en que se debe liberar el espacio ocupado por un objeto es un problema complicado de resolver correctamente. Esto también es el origen de errores que en muchas ocasiones no se detectan y solucionan tan fácilmente.
- No hay herencia múltiple: En C++ esta característica da origen a muchas situaciones en donde no se puede predecir cuál será el resultado. Por esta razón para la programación de sistemas en Java se opta por herencia simple que es mucho más simple de aprender y dominar.

Flexible y Robusto.

Uno de los problemas más comunes en los lenguajes de programación es la posibilidad de escribir programas que pueden bloquear el sistema. Algunas veces este bloqueo puede ser inmediato, pero en otras ocasiones llega a aparecer inesperadamente, por ejemplo, la aplicación accede a zonas de memoria que no estaban siendo ocupadas por otros programas hasta ese momento. Java obliga a que se declaren explícitamente los métodos, reduciendo así las posibilidades de error. Un ejemplo claro de un lenguaje no robusto es C. Al escribir código en C o C++ el programador debe hacerse cargo de la administración de memoria[3] de una forma explícita, solicitando la asignación de bloques a apuntadores y liberándolos cuando ya no son necesarios en el sistema. En Java, los apuntadores, la aritmética de apuntadores, las funciones de asignación y liberación de memoria (malloc( ) y free( ))[4] no existen. En lugar de los apuntadores se emplean referencias a objetos. De esta manera, se consigue evitar el problema de corrupción de memoria resultante de apuntadores que señalan a zonas equivocadas. El administrador de memoria de Java lleva una contabilidad de las referencias a los objetos. Cuando no existe una referencia a un objeto, éste se convierte en candidato para la recolección de residuos.

En Java los programadores no necesitan preocuparse de liberar una parte de memoria cuando ya no lo necesitan, es el "recolector de basura" el que determina cuando se puede liberar la memoria ocupada por un objeto. Un recolector de basura es un gran aporte a la productividad. Se ha estudiado en casos concretos que los programadores han dedicado un 40% del tiempo de desarrollo a determinar en qué momento se puede liberar una parte de memoria; además este porcentaje de tiempo aumenta, a medida que aumenta la complejidad del software en desarrollo. Por ejemplo es relativamente sencillo liberar correctamente la memoria en un programa de 1,000

[3] También se le denomina gestión de memoria.

[4] En el lenguaje C la función malloc() asigna memoria y free() libera memoria.

líneas. Sin embargo, es difícil hacerlo en un programa de 10,000 líneas y más aun en programas más grandes.

Un ejemplo más claro de ello es suponer que hicimos un programa de 1000 líneas hace un par de meses y ahora necesitamos hacer algunas modificaciones ; en este momento hemos olvidado gran parte de los detalles de la lógica de este programa y no es sencillo determinar si existe una referencia a un objeto que no existe, o si ya fue liberada. Peor aún, suponiendo que el programa fue hecho por otra persona, es impredecible evaluar cuan probable es cometer errores de memoria al tratar de modificarlo, así como en el caso de programas que son desarrollados dentro de un grupo de trabajo, en donde puede tomar años en terminar un sistema, ya que cada programador desarrolla un módulo que eventualmente utiliza objetos de otros módulos desarrollados por otros programadores.

Es inevitable que en la fase de prueba se deje pasar errores del manejo de memoria, que sólo serán detectados más tarde por el usuario final. Probablemente se incorporen otros errores en la fase de mantenimiento. Por lo que se puede afirmar que:

- Todo programa de 100,000 líneas que libera explícitamente la memoria tiene errores latentes.
- Sin un recolector de basura no hay verdadera modularidad.
- Un recolector de basura resuelve todos los problemas de manejo de memoria en forma trivial.

El que Java cuente con un recolector de basura llega a ser una gran ventaja sobre muchos lenguajes, muchos piensan que esto tiene un sobrecosto, pero como todo tiene un precio. El sobrecosto de la recolección de basura no es superior al 100%. es decir, si se tiene un programa que libera explícitamente la memoria y que toma X tiempo, el mismo programa modificado de modo que utilice un recolector de basura para liberar la memoria tomará un tiempo no superior a 2X. Este sobrecosto no es importante si se considera el incremento periódico en la velocidad de los procesadores. El impacto de un recolector de basura en tiempo de desarrollo y la confiabilidad del software, son muchos más importante que la poca pérdida en eficiencia.

Alto rendimiento.

Java fue diseñado para cumplir el requisito del mundo real de crear programas en red interactivos. También cuenta con características avanzadas que le permiten escribir programas que hacen muchas cosas al instante, sin perder el rastro de lo que debería suceder y cuándo.

Una de las características del lenguaje es que soporta la concurrencia. En ocasiones puede interesarnos dividir una aplicación en varios flujos de control independientes, cada uno de los cuales llevan a cabo sus funciones de manera concurrente. Los hilos múltiples o multihilos de Java permiten pensar en el comportamiento específico que se intenta codificar, sin tener que integrar ese

comportamiento en un modelo de programación global. De esta forma, un programa Java puede tener más de una hilo en ejecución. Por ejemplo, podría realizar un cálculo largo en un hilo, mientras otros hilos interactúan con el usuario. Así los usuarios no tienen que dejar de trabajar mientras los programas Java completan las operaciones más largas. La programación en un entorno multihilos suele ser difícil porque pueden producirse varios eventos al mismo tiempo; Java, sin embargo, posee características de sincronización fáciles de utilizar que simplifican la programación.

Seguro.

Actualmente es un tema novedoso la seguridad, todo mundo se encuentra preocupado por introducir el comercio a Internet. Uno de los principios de diseño claves de Java es la seguridad. Los programas de Java no pueden llamar a funciones globales y no pueden acceder a los recursos del sistema de manera incontrolada. Por este motivo se eliminó la posibilidad de manipular la memoria mediante el uso de apuntadores y la capacidad de transformación de números en direcciones de memoria (tal y como se hace en C), evitando así todo acceso ilegal a la memoria. Esto se asegura porque el compilador Java efectúa una verificación sistemática de conversiones y se puede ejercer un control sobre los programas ejecutables lo que no es posible en otros sistemas.

Una variable global es una variable que afecta a todas las partes de un programa, un apuntador es realmente la dirección de algunos datos en memoria RAM. Pasar un apuntador es equivalente a decir a las subrutinas donde esta la dirección en memoria, en vez de basarse en un conocimiento de las variables.

En muchas ocasiones nos enfrentamos a programas que tienen errores, los cuales no se determina a simple vista donde se encuentran, y a menudo esto se convierte en una situación muy complicada, en tiempo de ejecución. Dado que es imposible escribir un programa útil sin tener de alguna manera que administrar memoria o generar condiciones excepcionales, Java elimina virtualmente el problema con una administración de memoria avanzada llamada recolección de basura, y un manejo de excepciones orientado a objetos. Debido a que Java es un lenguaje muy estricto en cuanto a tipos y declaraciones, la mayoría de errores típicos se pueden descubrir durante la compilación.

Portable y Multiplataforma.

El principal objetivo de los diseñadores de Java, aunado al gran crecimiento de las redes en los últimos años, fue el de desarrollar un lenguaje cuyas aplicaciones una vez compiladas pudiesen ser inmediatamente ejecutables en cualquier máquina y sobre cualquier sistema operativo. Por ejemplo, un programa desarrollado en Java en una estación de trabajo Sun que emplea el sistema operativo Solaris, debe poderse llevar a una PC que utilice sistema operativo Windows NT.

El bytecode posee la característica de ser compacto y por tanto puede ser compilado (traducido a lenguaje máquina) muy rápidamente, en el transcurso de la propia ejecución del programa; constituye un entorno que facilita enormemente la portabilidad de un entorno o de una máquina a otra. Aparece entonces la "máquina virtual". Una *máquina virtual* es una capa lógica que hace creer al programa Java que se ejecuta en un ordenador real (con registros, memoria y procesador), cuando en realidad sólo ve una reconstrucción lógica de un ordenador. Para ejecutar un programa Java compilado (que está en bytecode), es preciso también que cuente con una implementación de la máquina virtual específica donde se desea ejecutar, la cual efectúa la transformación del bytecode en un programa comprensible para la máquina.

Java es un rico conjunto de clases de objetos poderoso que proporciona al programador abstracciones claras para muchas funciones de sistema habituales, como la administración de ventanas, de red y entrada/salida.

Java posee bibliotecas de clases estándares para:

- Manejo de archivos
- Comunicación de datos
- Acceso a la red Internet
- Acceso a bases de datos
- Interfaces gráficas

V.3 Objetos en Java

La mayor parte de los programas de computadoras que resuelven problemas del mundo real son cada vez más grandes. A lo largo del tiempo como resultado de la mayoría del software que se diseña, para resolver problemas muy extensos, se ha demostrado que la mejor manera de crear y mantener esos programas es construirlos a partir de piezas pequeñas o módulos, cada uno de los cuales es más manejable que el problema original, esto da pauta al enfoque "divide y vencerás" que hace más manejables las tareas y el desarrollo de los programas.

Un lenguaje es orientado a objetos si ofrece facilidades para definir y manipular objetos: entidades autocontenidas que tienen un estado y a las que se pueden enviar mensajes. Un lenguaje de programación orientado a objetos goza dos grandes ventajas:

- es posible escribir sistemas que se pueden modificar fácilmente, y
- proporciona un alto grado de reusabilidad.

Para que un lenguaje pueda considerarse orientado a objetos debe soportar como mínimo las características de:

- encapsulación
- herencia
- polimorfismo y
- enlace dinámico.

¡La P.O.O. requiere de un mayor esfuerzo!, ya que lo que vale la pena cuesta trabajo. Por lo cual es prudente hablar de él lado oscuro de los objetos: para programar orientado a objetos es necesario primero diseñar un conjunto de clases; la claridad, eficiencia y manutención del programa resultante dependerá principalmente de la calidad del diseño de clases. Un buen diseño de clases significará una gran economía en tiempo de desarrollo y mantenimiento. Lamentablemente se necesita mucha habilidad y experiencia para lograr diseños de clases de calidad. Un mal diseño de clases puede llevar a programas orientados a objetos a que sean de peor calidad y de más alto costo que el equivalente sistema bajo un enfoque tradicional. ¿Entonces por qué no usar otro lenguaje más simple como Visual Basic, si no necesito orientación a objetos? Porque la ventaja potencial más importante de un lenguaje de P.O.O está en las bibliotecas de clases que se pueden construir para él. Una biblioteca de clases cumple el mismo objetivo de una biblioteca de procedimientos en un lenguaje como C. Sin embargo, una biblioteca de clases es mucho más fácil de usar que una biblioteca de procedimientos, incluso para programadores sin experiencia en orientación a objetos. Esto se debe a que las clases ofrecen mecanismos de abstracción más eficaces que los procedimientos. Java es un lenguaje multiparadigma (como muchos otros lenguajes de programación). Uno no necesita hacer un diseño de clases para programar una aplicación de mil líneas.

Java es un lenguaje de programación poderoso que ofrece la potencia del diseño orientado a objetos, con una síntesis simple y familiar, en un entorno robusto y agradable de utilizar. Java fue diseñado partiendo de cero. No es un derivado directo de

un lenguaje de programación, debido a que existió la libertad de diseñar a partir de una hoja en blanco, se eligió un enfoque para los objetos limpio, utilizable y programable. La mayoría de los otros sistemas orientados a objetos han elegido tener jerarquías de objetos rígidas y difíciles de administrar, o utilizan modelos de objetos completamente abstractos que renuncian al rendimiento y facilidad de comprensión. Java consigue un equilibrio, proporcionando un mecanismo de clasificación simple (Paquetes, Clases y Objetos).

Paquetes.

Un paquete es un conjunto de clases e interfaces que tienen características o se encuentran relaciona entre sí. Los paquetes en conjunto se denominan bibliotecas de clases de Java o interfaz de programación de aplicaciones de Java (Java API). ¿ Para qué volver a inventar la rueda? El utilizar paquetes es como utilizar piezas ya existentes, y acomodar dichas piezas de acuerdo a nuestras necesidades, esto se denomina reutilización de software y es una de las ventajas más importantes de la programación orientada a objetos.

Clases.

Una clase define la forma y comportamiento de un objeto, es decir, una clase es una plantilla para un objeto, también define la estructura del mismo y su interfaz funcional, conocida como métodos. Cuando se ejecuta un programa en Java, el sistema utiliza definiciones de clase para crear instancias de la clase, que son los objetos reales. Una clase se define por la palabra "class". Cada clase se compila en un archivo separado que tiene el mismo nombre que la clase y termina con la extensión ". class"

Objetos.

Un objeto es una instancia única de una clase, que mantiene la estructura y el comportamiento definidos por la clase, como si fuera grabado por un molde con la forma de la clase. Figura V.3.1 A estos objetos se le llama a veces instancias de una clase. La estructura individual de la representación de datos de una clase esta definida por un conjunto de variables de instancia, estas variables mantienen el estado dinámico de cada instancia de clase. En Java se crea una instancia de un objeto con el operador "new".

Cada objeto tiene su propia copia de las variables de instancia de su clase, por lo que los cambios sobre las variables de instancia de un objeto no tienen efecto sobre las variables de instancia de otros objetos.

La diferencia entre clase e instancia radica en que una clase es algo que describe los atributos generales de un objeto, incluyendo los tipos de cada atributo y los métodos

que pueden operar en el objeto. Una instancia es un caso concreto de una clase de objetos.

Método.

Un método es un proceso que define el comportamiento para que el objeto realice alguna acción, la ejecución de un método depende del mensaje que se le envía.

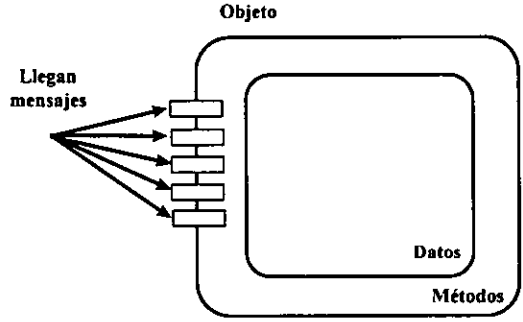


Figura V.3.1 Anatomía de un Objeto.

El formato general de una definición de método es:

```
tipo-de-valor-devuelto nombre-de-método(lista-de-parametros)
{
   declaraciones y enunciados
}
```

Los métodos se invocan en un programa escribiendo el nombre del método seguido de los argumentos de método encerrados en paréntesis. El tipo de valor devuelto indica el tipo del valor que devuelve el método invocador. Si un método no devuelve ningún valor el tipo de valor devuelto se declara como "void". El nombre de método es cualquier indicador valido. La lista de parámetros es una lista separada por comas que contienen las declaraciones de las variables que se pasaran al método. Si un método no recibe valores la lista-de-parametros se deja vacía. El cuerpo del método es el conjunto de declaraciones y enunciados que constituyen el método.

El usar métodos ya existentes para crear programas nuevos, contribuye a la reutilización de software.

Constructor.

Un constructor es un método que inicializa un objeto inmediatamente después de su creación. Se llama al método del constructor justo después de crear la instancia y antes de que "new" vuelva al punto de llamada, este debe tener el mismo nombre que la clase.

Sobrecarga de método.

Es posible y a menudo deseable crear más de un método con el mismo nombre, pero con lista de parámetros distintos (en base a los tipos, el número y el orden de los parámetros). A esto se le llama sobrecarga de método; la sobrecarga de método se utiliza para proporcionar a los programas de Java un comportamiento polimórfico. Se puede sobrecargar un método para tener dos alternativas de invocar el mismo cálculo. Cuando se invoca un método sobrecargado, el compilador selecciona el método correcto examinando los argumentos de la llamada. Por ejemplo, supóngase que se requiere de un método que realice una suma, la cual puede ser entre dos números enteros, o uno entero, flotante, y un doble, en este caso se puede construir dos métodos con el mismo nombre pero con distintos parámetros: suma(int x, int y) y suma(int x, float y, double z).

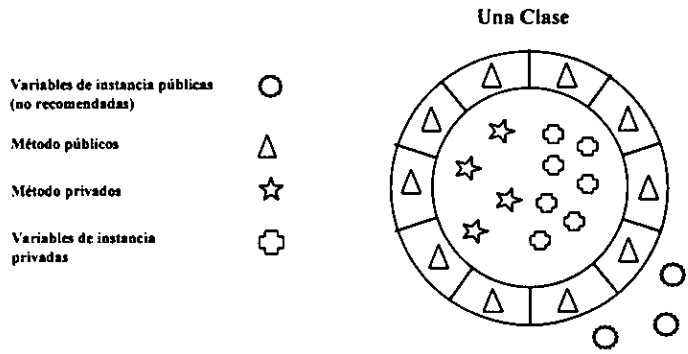
Encapsulado.

Un ejemplo claro para entender este termino es pensar que el encapsulado es como un envoltorio alrededor del código y los datos que se manipulan. El envoltorio define el comportamiento, protege el código y los datos para evitar que otro código acceda a ellos de manera arbitraria. En la figura V.3.2 se observa que los métodos o variables de una clase pueden ser públicos o privados. La interfaz publica de una clase representa todo lo que los usuarios externos de la clase pueden conocer. La interfaz privada delimita el acceso de los métodos o datos únicamente para código que se encuentre dentro de la misma clase.

Herencia.

La herencia es un concepto que relaciona clases una encima de otra de una manera jerárquica, es decir, permite definir una clase en términos de alguna ya existente. La clase recibe el nombre de subclase y la que sirve de base superclase.

Figura V.3.2. Encapsulado



Polimorfismo.

El polimorfismo significa múltiples formas. El polimorfismo y las interfaces permiten que se cree un código limpio, sensible, legible y resistente a los cambios futuros. El polimorfismo durante la ejecución, es uno de los mecanismos más poderosos que incrementa el diseño orientado a objetos para soportar la reutilización y la robustez del código.

Al número de parámetros con tipo de una secuencia especifica se le llama *signatura de tipo*. Java utiliza la signatura de tipo para decidir a qué método llamar. Es ilegal declarar dos métodos en la misma clase con el mismo nombre, compartiendo las mismas asignaturas de tipo.

Java proporciona muchos niveles de protección (clases y paquetes) para permitir un control preciso de la visibilidad de las variables y métodos. Dentro de una clase, todas las variables y métodos son visibles para todas las otras partes de la misma clase, dado que la clase es la unidad de abstracción más pequeña de la programación en Java.

V.4 Comunicación en Java

Durante las ultimas dos décadas se han producido compiladores de C para casi todas las clases de computadoras. C no es independiente del hardware, con diseño cuidadoso es posible escribir programas en C que pueden transportarse casi cualquier computadora, la versión de C orientada a objetos, recibe el nombre de C++, el cual es un lenguaje *híbrido* (es posible programar con un tipo C o con un estilo orientado a objetos o con ambos). Pero Java es un lenguaje que funciona en cualquier plataforma lo cual amplia poderse comunicar. Uno de los ejemplos claros en la comunicación es la red mundial comúnmente llamada Internet, esta se compone de sistemas: PC, Apple, Macintosh y Estaciones de trabajo UNIX. Cada uno de estos sistemas tiene entornos diferentes para trabajar en red, en la presentación de ventanas, en gráficos y en la manipulación de entrada y salida de datos. La biblioteca de clases de Java proporciona un modelo de protocolos para Internet. Los protocolos básicos para tratar con Internet están encapsulados en unas cuantas clases simples, que incluyen implementaciones ampliables de ftp (protocolo de transferencia de archivos), http (protocolo de transferencia de hipertexto), nntp (protocolo de transferencia de noticias), smtp (protocolo de transferencia de correo electrónico), junto con conectores de red de bajo nivel e interfaces. Esto permite interactuar con los servicios de red sin tener que comprender realmente los detalles de estos protocolos. Si se entiende los mínimos conceptos de bajo nivel de un protocolo de Internet, es fácil implementar un cliente o un servidor en Java.

Las clases de Java admiten especialmente los protocolos y formatos que se manejan en comunicaciones, de ahí el éxito de Java en el desarrollo de aplicaciones utilizadas en la red mundial (World Wide Web). Las clases de Java se envían a través de la red con facilidad, porque el sistema de archivos y los flujos de datos tienen una interfaz unificada, esta es la manera en que se cargan los applets en la red, también, es la manera adecuada de distribuir un programa y incluso actualizarlo sobre la marcha.

El lenguaje Java fue creado para cumplir los requisitos de un sistema en red, orientado a objetos y multihilo, desde un principio. Varios conceptos del lenguaje como la manipulación de cadenas, el manejo de multihilos y de excepciones, se encuentran implementados en las bibliotecas de clases de Java.

Por muchos años la diferencia de idiomas ha sido un problema para la comunicación entre los seres humanos. Al igual sucede en el campo de la informática, que se ha visto frenado por la variedad de lenguajes de programación, conceptos y de sistemas operativos, esto se debe a la guerra comercial e intereses propios que se presenta entre los fabricantes de software. Plantear y resolver el problema de la comunicación abriría a la comunidad enfocada al desarrollo de software la posibilidad de desarrollar aplicaciones globales, los creadores de Java dirigieron sus pasos hacia este objetivo y ¡ Hoy es una realidad!. Java es la clave para que todos puedan compartir la tecnología que inventaron.

La comunicación soluciona muchos problemas, no obstante los avances logrados en la interconexión de computadoras y el software de trabajo en grupo, los problemas de interoperabilidad entre los diferentes sistemas siguen siendo un problema que impide que la comunicación fluya y los recursos de la red puedan ser aprovechados a fondo.

Aprovechar el poder de procesamiento que se obtiene de una red fue el objetivo sobre el cual partieron los creadores de Java para desarrollar un lenguaje que, por lo tanto, tenía que poseer la cualidad de ser utilizable por cualquier tipo de sistema operativo, es decir, que fuera un lenguaje universal.

En realidad, el hecho de que haya sido diseñado para aprovechar la capacidad de procesamiento de las redes, lleva implicaciones que van mucho más allá de las posibilidades de negocio, que puede ofrecer un nuevo lenguaje de programación. Con la aparición de Java se abrió el horizonte a una nueva era en las telecomunicaciones en donde el anunciado ingreso de la computación a este campo abre posibilidades de consolidación de los elementos que faltaban para que se diera la tan esperada red global en donde el comercio electrónico, la educación personalizada y los servicios de información podrán ser una realidad al alcance de las empresas y los países dispuestos a tomar el reto de construirla.

La fórmula que desarrollaron los creadores de Java para lograr estas cualidades es, como todas las cosas valiosas del mundo, sumamente simple. Los usuarios instalan en sus máquinas un programa que juega el papel de interprete entre Java y el sistema operativo del equipo. Con este acto el usuario está ya preparado para accesar desde la red los elementos que requiere de la aplicación, es decir, en un programa diseñado en Java el usuario sólo va a emplear la porción que le es útil del programa. Lo demás está al alcance de la creatividad de la industria que ha descubierto en esta fórmula, infinidad de oportunidades de negocio; algunos autores opinan que "en un futuro no muy lejano se espera la caída de precios en el hardware con la llegada masiva de WEB Tops: máquinas computadoras especialmente diseñadas para trabajar en red sin disco duro, sin drives para disquetes y sin sistema operativo". (21)

Java es el pasaporte de acceso al mercado global. Con el enfoque universal que hace posible compartir los programas y hacer uniones de aplicaciones de tal modo que los desarrolladores puedan usar y aprovechar otros programas. Esto obviamente trae nuevos retos respecto a la forma como actualmente se registran y se conceden las licencias, pero en teoría esto es posible, lo cual permite que cada usuario (sobre todo los corporativos) puedan comprar software y adaptarlo a la medida de sus necesidades. Pero las oportunidades de negocio no sólo son para los programadores, también los actuales directores de sistemas pueden obtener beneficios de esta tecnología. De hecho ellos son quienes lo pueden hacer de inmediato, sin tener que esperar a que esta tecnología se difunda.

Uno de los grandes beneficios que obtienen las compañías, especialmente las grandes corporaciones es el poder conservar su competitividad sin tener la necesidad de

estar vigilando el costo que implica la infraestructura y administración de redes y de sistemas críticos.

La programación se ha vuelto más compleja y más exigente. Hoy día los usuarios quieren aplicaciones provistas de interfaces gráficas con el usuario final, que les permiten aprovechar las capacidades de multimedia, gráficos, imágenes, animaciones, audió e incluso vídeo. Ellos requieren aplicaciones que puedan ejecutarse en redes de computadoras con Internet y comunicarse con otras aplicaciones, así como, también aprovechar la flexibilidad y mejoras del rendimiento que son posibles con el procesamiento multihilos. Los usuarios desean aplicaciones con más amplias capacidades de procesamiento de archivos que las que ofrece el lenguaje C y C++, ya que no quieren que dichas aplicaciones estén limitadas a su escritorio o incluso a la red local de su organización si no que puedan integrar componentes de Internet.

Los lenguajes de programación generalmente ofrecen un conjunto sencillo de estructuras de control que permiten realizar sólo una acción a la vez. La concurrencia que la mayoría de los lenguajes realiza, se implementa con el uso de *primitivas*[5] a las que sólo programadores de sistemas experimentados tienen acceso. Java a diferencia de otros lenguajes, no requiere de hacer uso de primitivas para realizar programación concurrente, dentro de sus aplicaciones se encuentra el uso de multihilos.

Con la computación distribuida las operaciones de computo de una organización se distribuyen mediante el trabajo en red a los sitios en los que se lleva acabo el verdadero trabajo de la organización. El llamado a procedimientos remotos extiende el mecanismo, de llamadas a procedimientos de una sola computadora a un ambiente distribuido. Java tiene la ventaja de realizar llamadas remotas, sin complicarnos la vida, ya que cuenta con una aplicación de *RMI* (Remote Method Invocation, Invocación de Métodos Remotos).

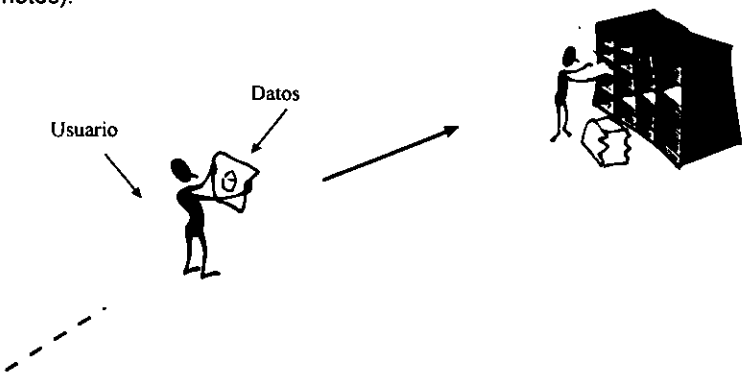


Figura V.4.1 Transferencia de información Antiguamente

[5] Una primitiva especifica la función que se va a ejecutar y los parámetros que se usan para pasar datos e información de control.

Esto hace posible distribuir la ejecución de un programa entre múltiples computadoras. Por ejemplo, anteriormente la transferencia de información de un lugar a otro se hacia mediante el traslado de una persona la cual físicamente llevara la información a su destino.

Hoy en día, no se requiere que una persona que se encuentra por decir, en el D.F., y que necesite información de su compañía que se encuentra en Toluca, se traslade físicamente para consultar dicha información o dar de alta un registro, con el uso de terminales se puede manejar información desde determinado lugar y proyectarse los resultados en otro lugar.

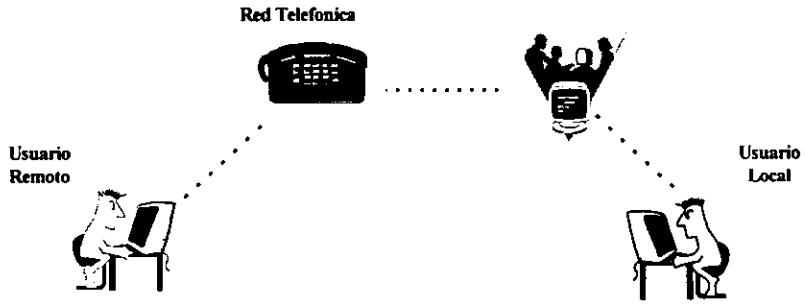


Figura V.4.2 Terminales Remotas

En la siguiente figura V.4.3 se muestra el ambiente general de RMI y como interactúan cliente y servidor. Un *cliente* es un programa que hace un llamado a un procedimiento remoto. Un *servidor* es un proceso que implementa una interfaz determinada. El proceso servidor escucha requerimientos de los clientes por alguna de las operaciones en la interfaz, cuando él recibe un requerimiento de un cliente, ejecuta el procedimiento respectivo y envía la respuesta al cliente. Los programas acceden los objetos a través de interfaces.

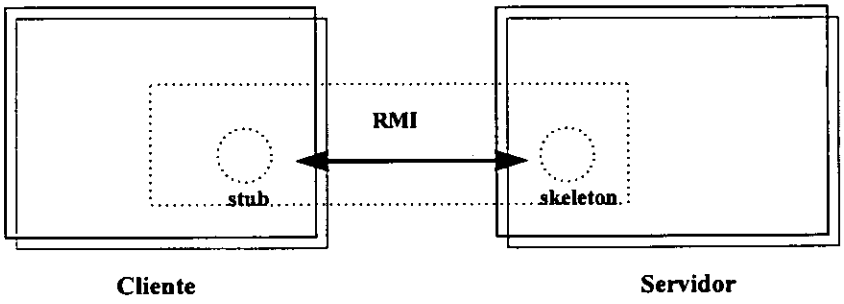


Figura V.4.3 Ambiente general

A continuación se muestra las capas del modelo RMI.

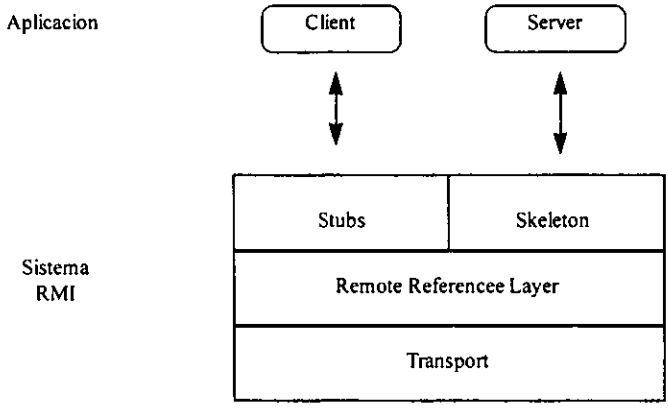


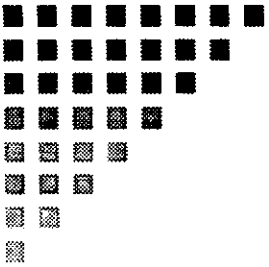
Figura V.4.4 Capas del modelo RMI

Stubs y Skeleton.

Tanto el cliente como el servidor, necesitan para comunicarse unos módulos especiales conocidos como stubs y skeleton. El stub y skeleton son los encargados de transferir invocaciones remotas y respuestas, entre los clientes y servidores como si fuesen locales.

Es importante saber que pasos se tienen que seguir para la ejecución de Métodos Remotos, los cuales son:

1.- Definir las interfaces de los objetos que son visibles remotamente
2.- Desarrollar la implementación de estas interfaces en el servidor
3.- Desarrollar el cliente, instanciando un objeto de la interfaz deseada
4.- Compilar los programas con "rmic" para generar el Stub y el Skeleton
5.- Determinar el puerto de comunicación con "rmiregistry"
6.- Ejecutar el ambiente de ejecución del RMI en el servidor
7.- Ejecutar el cliente



CAPÍTULO VI

IMPLEMENTACIÓN

"El profesional no tiene razón alguna para justificar su falta de preocupación, esfuerzo o amor por dar aquello que le es propio, porque él escogió libremente esa misión como su razón de existir" A.L.C.

El proyecto de este trabajo es la realización de una Red Neuronal Distribuida programada en lenguaje Java, por lo que analizaremos paso a paso la implementación del desarrollo para el programa de la Red Neuronal así como la parte distribuida en lenguaje Java.

Como se menciono en el capítulo I, la arquitectura que se utiliza para esta red neuronal es la arquitectura Retropropagación (Backpropagation), la cual se basa en una etapa de aprendizaje, que se dividen en dos parte: 1a. parte, aprendizaje hacia adelante, 2a. parte aprendizaje hacía atrás, posteriormente en la etapa de reconocimiento la red determinar a que carácter pertenece el vector que lee desde un archivo "Aprende.dat" donde se encuentran los vectores a reconocer, que pertenecen a los caracteres "A, E, I, O, U", en estos vectores se encuentran diferentes tipos de vocales, es decir, las cinco vocales pero que pueden estar desfasadas a la derecha o izquierda, así como contener ruido. El leer los vectores desde un archivo es con el fin de poder modificar los vectores de aprendizaje sin tener que reconstruir el programa.

La determinación del el número de capas y nodos, puede ser variable, por las ventajas que se tienen al programar en un entorno orientado a objetos. Para este proyecto se utilizaron diferentes números de capas y nodos hasta determinar cual era el más conveniente en eficiencia, por lo que se eligió que tuviera tres capas, la primer capa (capa de entrada) con 64 nodos que corresponden a la matriz de puntos 8x8 de la imagen de cada carácter, la segunda capa (capa intermedia) con 10 nodos, la tercera capa (capa de salida) con 5 nodos, y 10 entradas. Como podemos observar sólo posee una capa intermedia, esto lo veremos más claramente en el Capítulo VII.

La salida de la red es un vector de 5 bit, que corresponde a cada una de las vocales que debe aprender, como se muestra en la figura VI.1. Los valores correspondientes a cada vocal para la etapa de aprendizaje serán leídos desde un archivo "Clase.dat".

Carácter	Combinación de bits
A	10000
E	01000
I	00100
O	00010
U	00001

Figura VI.1 Tabla de correspondencia de caracteres a bits.

Para facilitar el entendimiento del desarrollo de este programa utilizaremos la siguiente nomenclatura donde:

- El nombre de las variables de las clases comienzan con una "T", pero, recordemos que el constructor de una clase debe de llevar el mismo nombre, por lo tanto, sólo comenzaran con la letra "T" las clases y su respectivo método constructor.
- El nombre de las variables y los métodos comienzan con las dos primeras consonantes minúsculas de la clase a la cual pertenecen.
- Existen variables que son de uso contador o indicador que empiezan con la letra "n".
- Se hará uso de margen de tres espacios, para poder distinguir entre cada comienzo y fin de los indicadores "{ }", como son clases, constructores, métodos, sentencias de control (if, for, etc.).
- Las instrucciones que se encuentren dentro de un comentario "*//*" son opcionales para fines prácticos de chequeo y validación.

Para iniciar el desarrollo del Proyecto es bueno recordar un enfoque que se menciono anteriormente: "Divide y vencerás", para hacer más manejables la tarea de desarrollo de los programas, en este caso dividiremos el proyecto en dos Facetas.

1ª Fase: Cuerpo de la Red Neuronal como se observa en la figura VI.2 donde se muestran el Diagramas de Clases que corresponde a la parte de aprendizaje y reconocimiento de la red neuronal.

El fin de dividir la Red en distintas clases, es el de poder determinar su tamaño dinámicamente, sin tener que modificar todo el código. Por ello es que tenemos una clase llamada TPrincipal, en donde se encuentran todos los valores que determinan el tamaño de la red, y la cual controla las dos partes de aprendizaje, y cuando reconocer un determinado vector.

Para la identificación de los objetos utilizaremos tarjetas CRC, que reciben este nombre por que en ellas se registran las Clases, Responsabilidades y los Colaboradores, dichas tarjetas son de 10x15 cms, anotando en ellas cada una de las responsabilidades de las funciones que se tiene; en la parte derecha de la tarjeta se anotará a los colaboradores. como se muestra en la figura VI.3

Clase	Colaboradores
Lista de Responsabillidades.	

Figura VI.3 Formato de la tarjeta CRC.

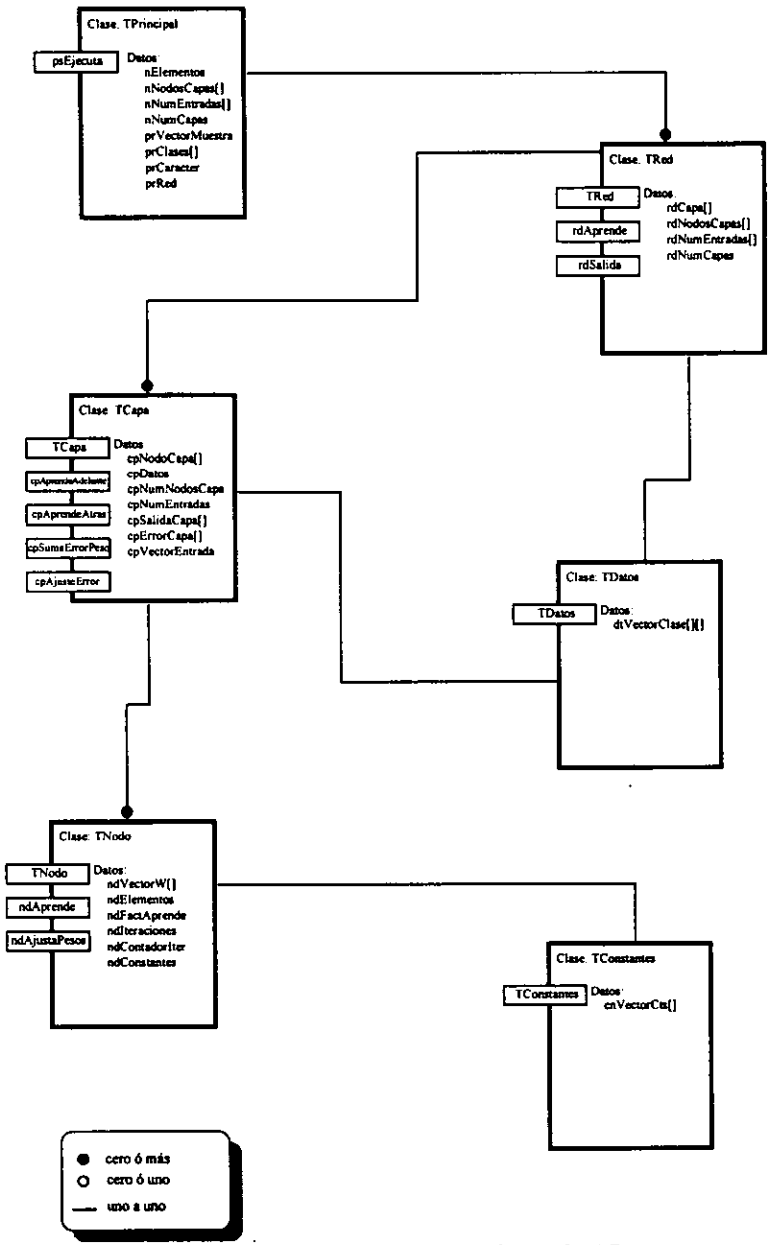


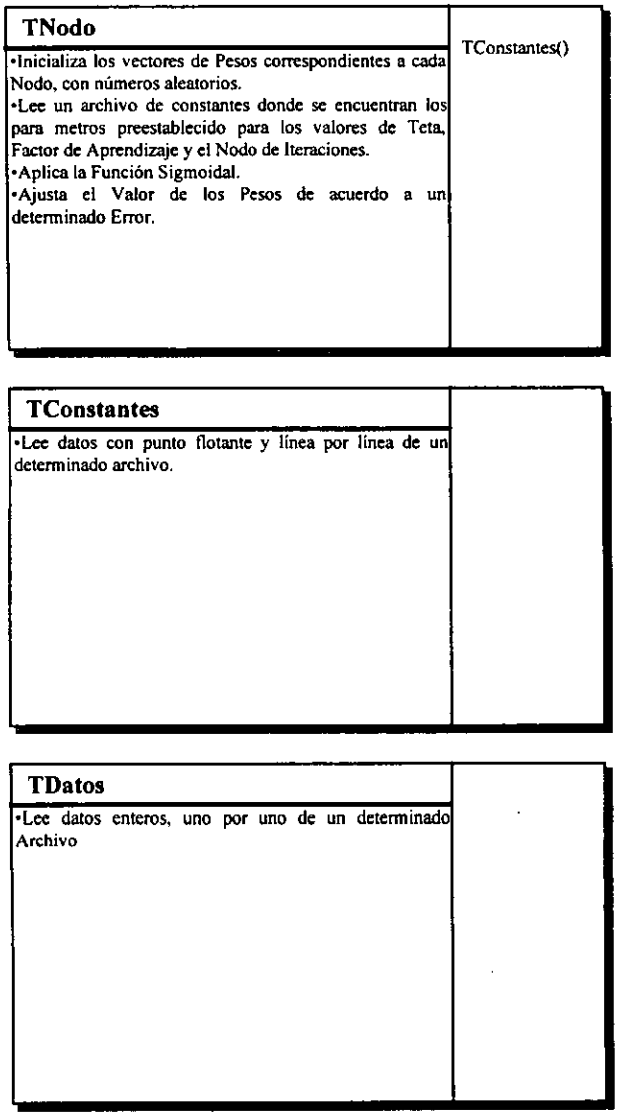
Figura VI.2 Diagrama de Clase correspondiente a la 1ª Fase

A continuación se muestra cada una de las tarjetas CRC, que corresponden a la Fase de aprendizaje y reconocimiento de la Red Neuronal.

TPrincipal	TRed()
•Inicializa variablespara el funcionamiento,así como el tamaño de la Red. •Indica a la Red cuando y que número de vectores Aprenda. •Indica a la Red que clasifique determinado vector de Entrada. •De acuerdo al valornumérico declasificación determina que carácter es.	

TRed	TDatos() TCapa()
•Inicializa instancias de acuerdo al número de capas que contiene la Red •Inicializa valor de los vectores a Aprender, de un determinado archivo. •Indica al número de Capas cuando realizar aprendizaje en su etapa de Propagación hacia adelante, así mismo como en la etapa de Propagación hacia Atrás. •Indica cuando se realice Ajuste de Pesos en las Capas. •Checa que todos los vectores de Aprendizaje sean aprendidos por la Red. •Controla la Evaluación del vector o vectores de entrada a Reconocer. •Devuelve la Clase a la cual pertenece el vector de Entrada, es decir en números enteros.	

TCapa	TDatos() TNodo()
•Inicializa Instancia de Nodos correspondientes a la capa •Inicializa vectores de Salidas deseadas correspondientes a cada Vector de Aprendizaje , leídos desde un determinado archivo. •Realiza el Aprendizaje en su capa de propagación hacía Adelante. •Realiza el Aprendizaje en su etapa de propagación hacía Atrás en la capa de Salida. •Realiza el •Aprendizaje en su etapa de propagación hacía Atrás en las capas Intermedias. • Calcula la Sumatoria de Error. •Indica que valor se ajustara a los Pesos de cada Nodo.	



Algunas de las clases que se han mencionado, se reutilizan como parte del desarrollo de la Fase cliente-servidor.

2ª Fase: Cuerpo Cliente-Servidor. En la figura VI.4 se muestra el diagrama de clase que compone la Fase distribuida del proyecto, el decir cliente-servidor.

Para la identificación de los objetos en esta etapa utilizaremos también las tarjetas CRC que corresponden a las clases del diagrama que se muestra en la figura VI.3.

TRemoteCliente	TCrearThread1() TCrearThread2()
•Ejecuta dos hilos para la invocación de métodos remotos, es decir dos procesos concurrentes	

TRemoteServer	TPrincipal()
•Detecta cuando se realizo una llamada al servidor "Tsekub", para ejecutar reconocimiento de caracteres. •Indicando al programa Principal inicializar Aprendizaje. •Indica que Vector debe reconocer la Red Neuronal.	

TRemoteServer2	TPrincipal()
•Detecta cuando se realizo una llamada al servidor "Utopía", para ejecutar reconocimiento de caracteres. •Indicando al programa Principal inicializar Aprendizaje. •Indica que Vector debe reconocer la Red Neuronal.	

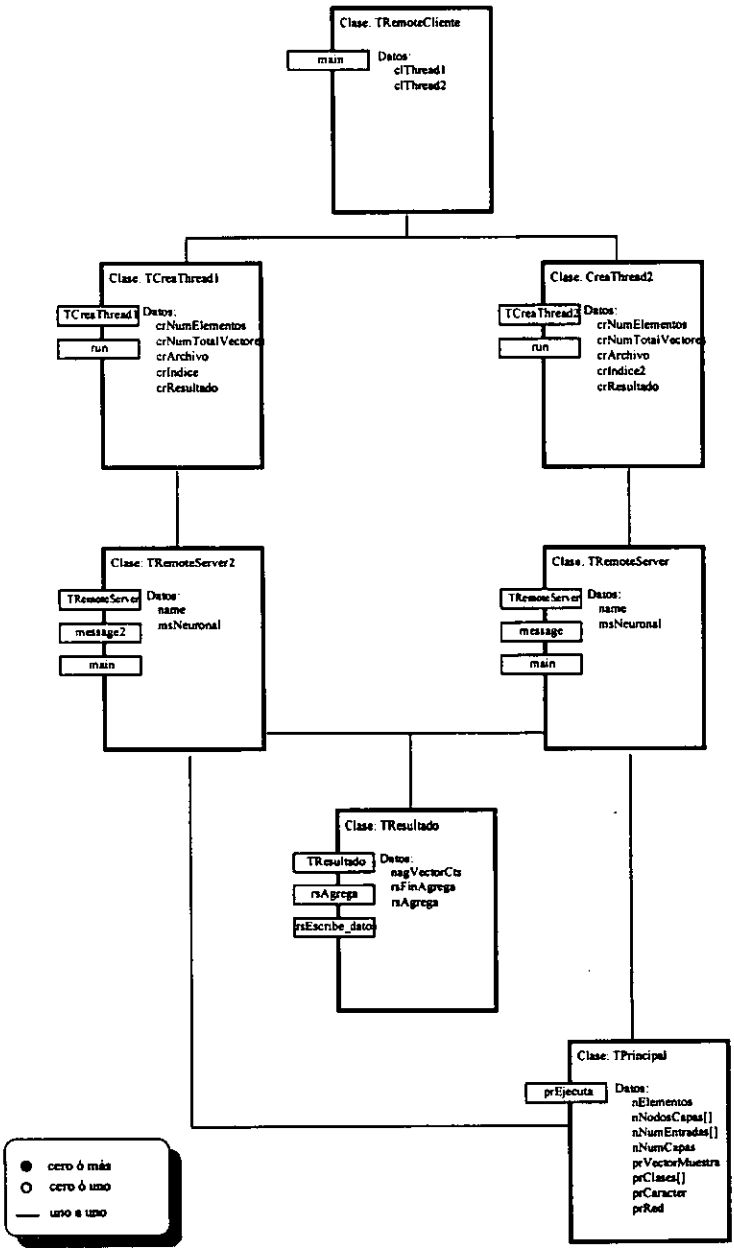


Figura VI.4 Diagrama de Clases correspondiente a la 2ª. Fase.

TCreaThread1	TDatos() TResultado()
•Lee cadena de dígitos a reconocer desde un archivo. •Inicializa la Interface Remota con el Servidor Tsekub. •Solicita servicio de reconocimiento de vector al servidor Tsekub. •Envía al objeto TResultado el carácter así como el índice que reconoció.	

TCreaThread2	TDatos() TResultado()
•Lee cadena de dígitos a reconocer desde un archivo. •Inicializa la Interface Remota con el Servidor Utopía. •Solicita servicio de reconocimiento de vector al servidor Utopía. •Envía al objeto TResultado el carácter así como el índice que reconoció.	

RemoteInterface	
• Crea la interfaz remota entre el Cliente y el Servidor	

RemoteInterface2	
• Crea la interfaz remota entre el Cliente y el Servidor	

TResultado	
•Almacena todos lo caracteres que han sido reconocido por los dos servidores Tsekub, Utopía en un arreglo. •Cual el número de caracteres es igual al número de elementos para reconocer, guarda todos lo caracteres en un Archivo "Resultado.txt"	

Hasta este momento tenemos el diagrama de clases, las fichas CRC, que contienen información de lo que realizan y requieren cada una de las clases, por lo que sólo nos falta determinar el diagrama de estados para ambas etapas, esto con el fin de saber en que momento se relacionan y permanecen los objetos.

Para ello utilizaremos los diagramas de estado que se encuentran en la figura VI.5 y figura VI.6, que corresponde a las etapas del proyecto.

Ya hemos analizado y delimitado nuestro diseño, de este proyecto, por lo que es necesario aclarar que dicho trabajo no es de etapa final para usuario, y no posee una interfaz gráfica, esto se debe a que es un modulo complementario para un diseño mayor, en el Apéndice "A" se encuentra el código correspondiente a cada una de las clases que se han diseñado en lenguaje Java.

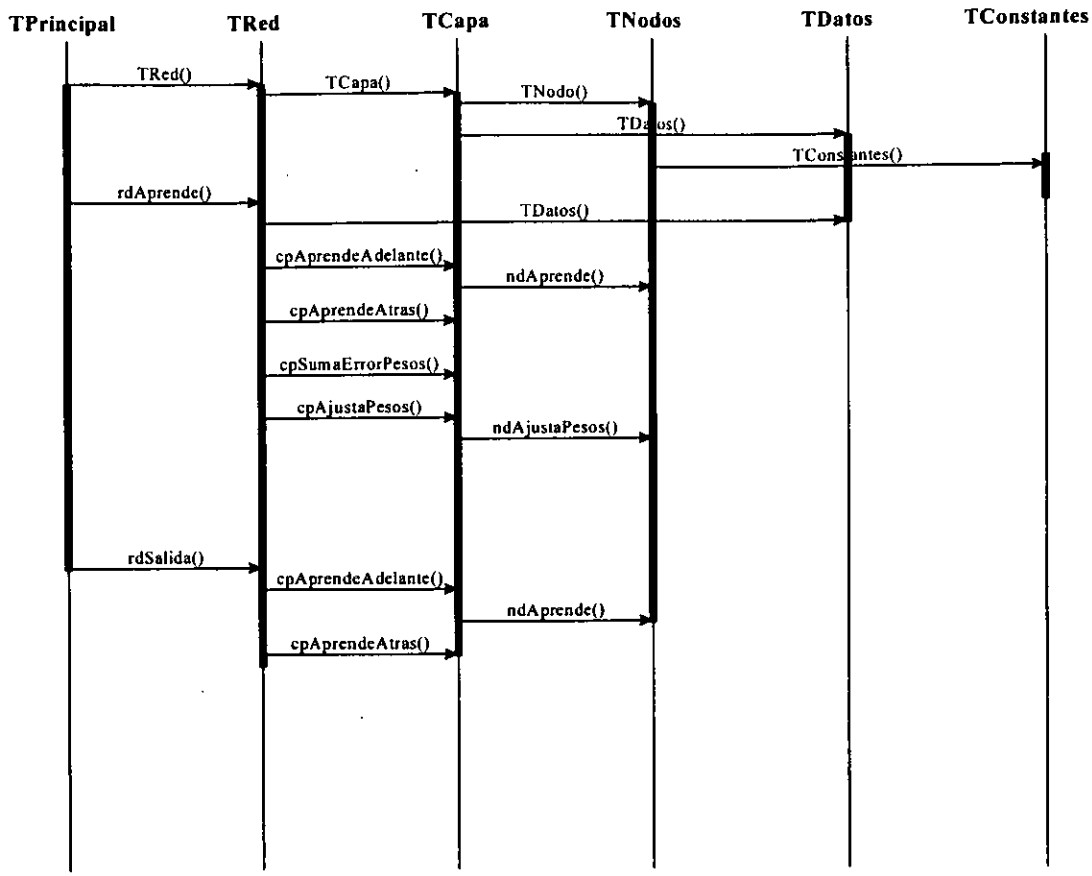


Figura VI.5 Diagrama de estados correspondiente a la 1ª Fase.

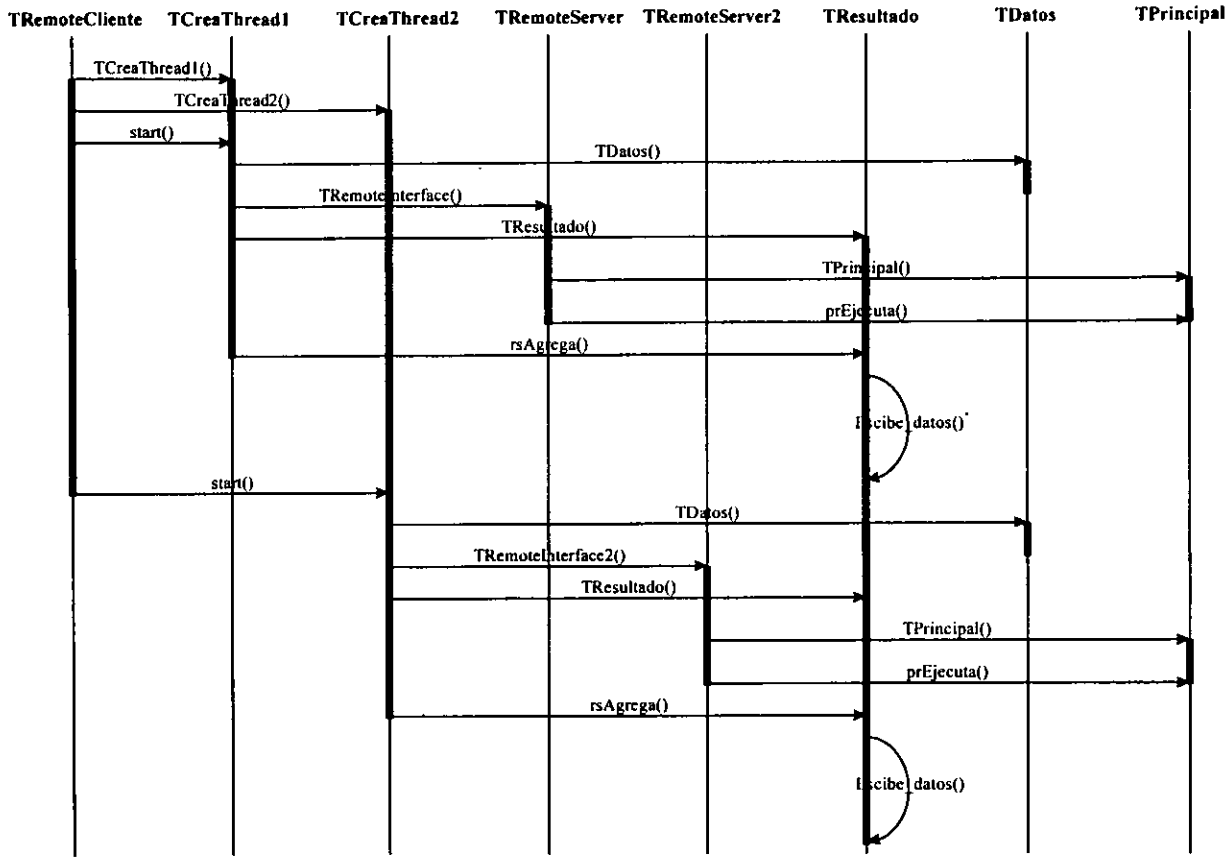
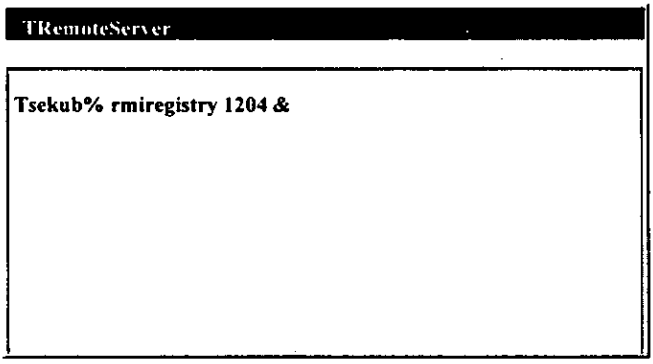


Figura VI.6 Diagrama de estados correspondiente a la 2ª Fase.

Para la Ejecución del Proyecto se requiere indicar en al sistema por cuál puerto va a escuchar el servidor, el puerto debe corresponder al que indica, en el diseño de servidor y cliente, para el TRemoteServer se utilizo:

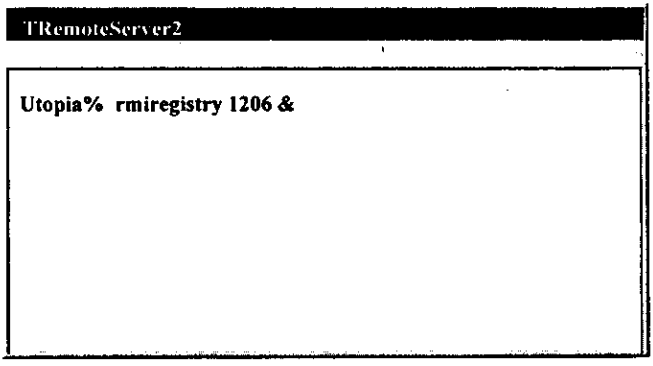
String myName1 = "//tsekub:1204/Servidor Tsekub";

Para el registro en Sistema Operativo UNIX, se utiliza como lo muestra la siguiente pantalla, para Windows se utiliza "star registry 1204"

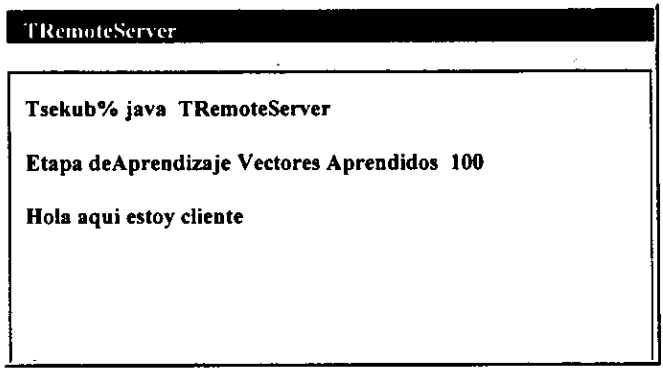


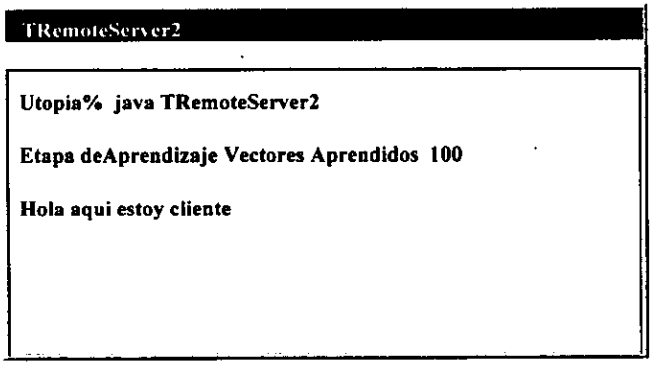
En este caso como se utilizo dos servidores, el registro se debe hacer en ambos servidores, especificando en cada uno el puerto por el cual va ha escuchar, el puerto designado para el servidor Utopia, es 1206, este puerto puede ser cualquiera arriba del puerto 1099.

String myName1 = "//utopia:1206/Servidor Utopia";

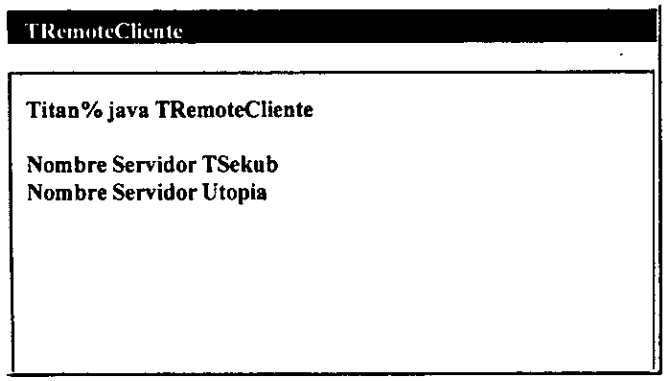


Para dar inicio al Programa de Servidor en el cual cada uno realiza su etapa de aprendizaje de la Red Neuronal, debe ejecutar el nombre del programa con "java", seguido del nombre, el cual enviara un mensaje indicando al Cliente que la ejecución fue correcta, y posteriormente indicar cuantos vectores fueron utilizados para la etapa de Aprendizaje, a continuación se muestran, las dos pantallas de los servidores, ya que esta operación se realiza simultáneamente.

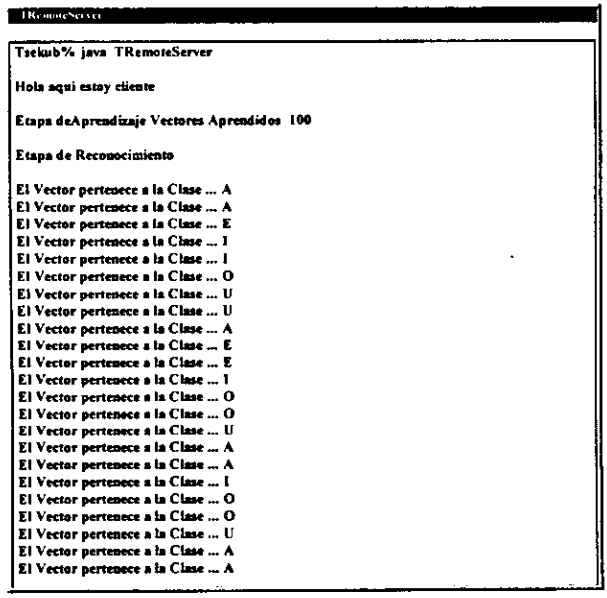


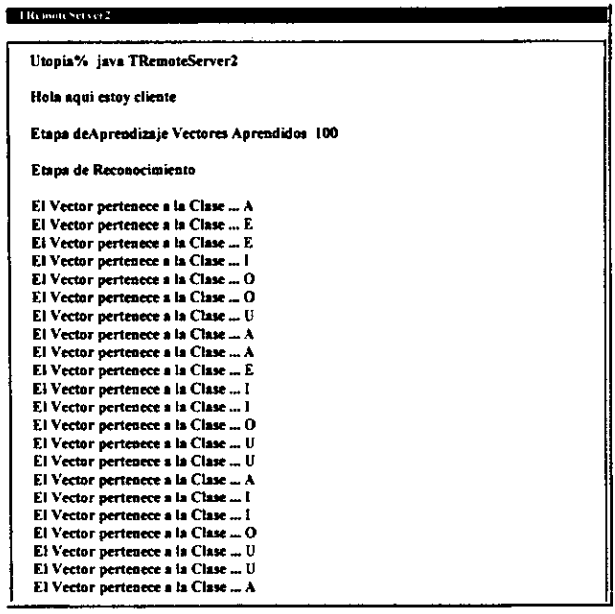


Después de haber levantado ambos servidores a utilizar, ejecutamos el programa del cliente, en este caso la ejecución se realizó en otra máquina con nombre titán, el cual realiza proceso de ejecución de dos hilo, para ambos servidores, reconociéndolos con un nombre, si alguno de los servidores no se encontrara ejecutándose enviara un mensaje de error, de no ser así, aparece la siguiente pantalla :

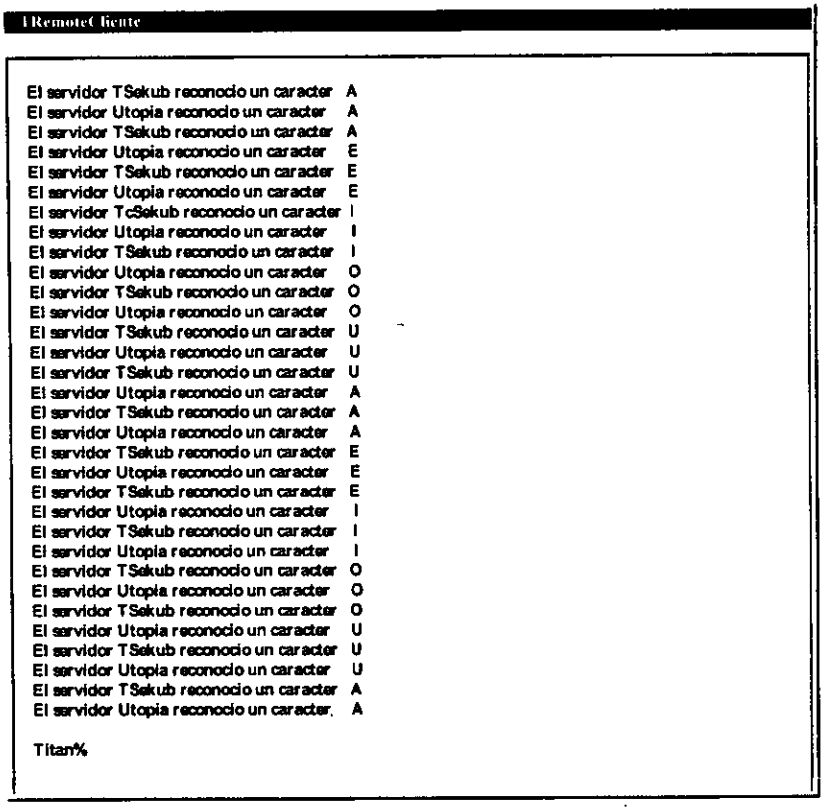


El cliente asigna a cada servidor el vector que reconocerá, y cada servidor ejecuta el reconocimiento de todos los vectores enviados, emitiendo un mensaje del vector reconocido, así mismo enviando este al cliente.





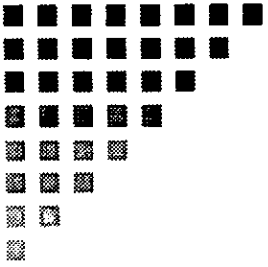
El cliente manda la impresión a pantalla de cada uno de los caracteres reconocidos así como el servidor que lo reconoció, estos pueden aparece en distinto orden, según la carga de trabajo de cada servidor.



Esto se realiza como parte de observación del buen funcionamiento del sistema, pero el resultado final se encuentra en un archivo "Texto.txt".

Texto.TXT

A
A
A
E
E
E
I
I
I
O
O
O
U
U
U
U
A
A
A
E
E
E
I
I
I
O
O
O
U
U



CAPÍTULO VII

VALIDACIÓN Y PRUEBAS DE DESEMPEÑO

"Al enriquecer el compromiso con tu propia existencia, identificarás en cada etapa de tu vida tu misión profesional, tu razón de ser y aceptarás retos y desafíos que te permitan conocer lo que eres capaz de ser y hacer." A.L.C.

Para la validación y pruebas de desempeño de la Red Neuronal Distribuida, tomaremos como indicador de aprovechamiento, la comparación de la Red Neuronal Distribuida implementada en lenguaje Java con una Red Neuronal Normal, la cual se encuentra desarrollada en "NeuroShell 2", donde a las dos se aplique las mismas características, es decir, entradas, capas intermedias, así como el valor de la constante de aprendizaje, a partir de ello iremos variando los valores de los nodos, capas intermedias, y de la constante de aprendizaje en ambos programas, para poder comprobar su desempeño.

Los vectores que se utilizan para la etapa de reconocimiento de la Red Neuronal Distribuida se encuentran en un archivo "Texto.dat" donde cada servidor reconoce un vector alternadamente.

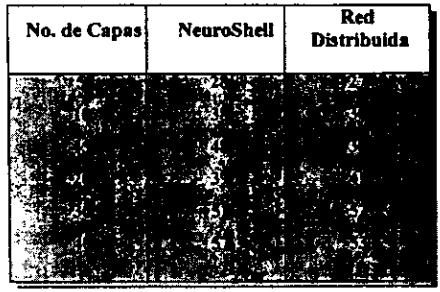
El resultado del reconocimiento de caracteres de ambos servidores se almacenara en un sólo archivo que lleve el nombre de "Resultado.dat", el cual se crea en el momento de dar respuesta a los caracteres reconocidos. La posición en la cual se almacene el resultado de cada servidor corresponde a la misma posición en la que fue leído el vector a reconocer.

Aunque para las pruebas de la Red Neuronal Distribuida utilizamos dos servidores Tsekub y Utopia, que se encuentran en distinto lugar conectados por un protocolo de comunicación, no quiere decir con ello, que no se puedan utilizar más servidores, pues pueden ser los servidores que nosotros consideremos para el reconocimiento, así mismo el tiempo de reconocimiento será menor.

Después de haber hecho nuestras respectivas pruebas, tenemos como resultado:

En la Tabla VII.1 tenemos los valores del número de épocas[1] que se realizaron durante la etapa de aprendizaje para vectores de aprendizaje que se encontraban en continuidad, es decir, de cien vectores que se tomaron para la etapa de aprendizaje, los primeros veinte eran los correspondientes al carácter "A", del 21 al 40 al carácter "E", del 41 al 60 al carácter "I", del 61 al 80 al carácter "O", y los veinte últimos al carácter "U"; En esta muestra se vario el número de capas correspondientes a ambas redes neuronales.

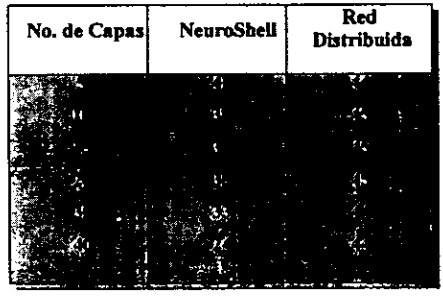
[1] Se define como época a cada ocasión en que la Red Neuronal intenta aprender todos los vectores de entrada en la etapa de aprendizaje.



No. de Capas	NeuroShell	Red Distribuida

Tabla VII.1 Número de Epocas en la etapa de aprendizaje con vectores continuos

La Tabla VII.2 corresponde a los vectores que reconocieron ambas redes, como resultado de la variación del número de nodos en la capa intermedia, estos resultados corresponden a los valores de aprendizaje de la tabla anterior.



No. de Capas	NeuroShell	Red Distribuida

Tabla VII.2 Vectores reconocidos en la etapa de trabajo usando vectores continuos en la etapa de aprendizaje

La Tabla VII.3 es el resultado del número de épocas para la etapa de aprendizaje, donde se usaron vectores dispersos, es decir, se usaron cien vectores para la etapa de aprendizaje de tres en tres, variando el número de nodos en la capa intermedia.

No. de Capas	NeuroShell	Red Distribuida

Tabla VII.3 Número de Epocas en la etapa de aprendizaje con vectores dispersos

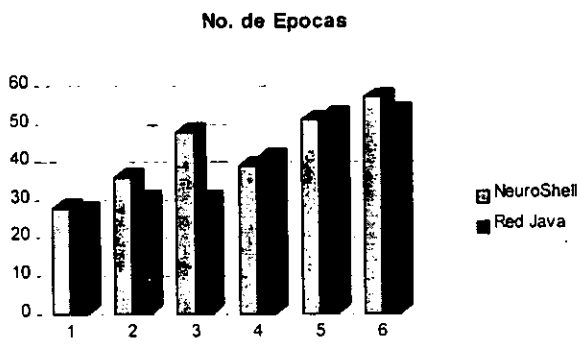
En la siguiente Tabla se muestra el número de vectores reconocidos en la etapa de trabajo usando los valores anteriores, con vectores de aprendizaje salteados

No. de Capas	NeuroShell	Red Distribuida

Tabla VII.4 Vectores reconocidos en la etapa de trabajo usando vectores dispersos en la etapa de aprendizaje

Con los resultados obtenidos de manera gráfica podemos observar claramente el desempeño de la Red Neuronal Distribuida.

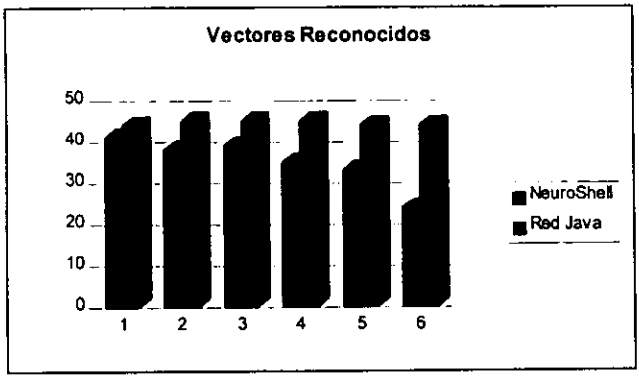
En la gráfica VII.1 que corresponde a los valores de tabla VII.1, observamos que el número de épocas en la red NeuroShell en la mayoría de los casos es siempre más alta, con ello quiere decir, que el aprendizaje de los vectores usados, han tardado más en ser asimilados por la Red NeuroShell.



Gráfica VII.1 Comparación del Número de Epocas en la etapa de aprendizaje con vectores continuos

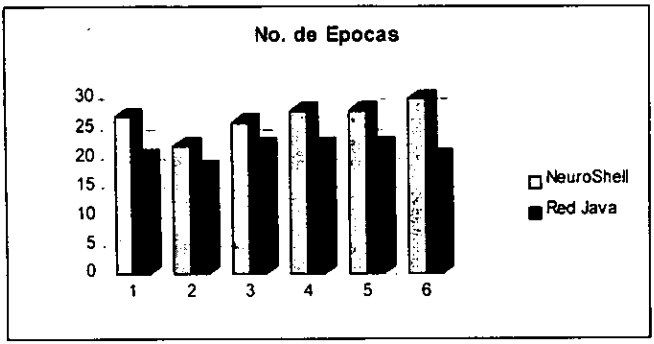
Para la comparación en cuanto al desempeño en aprendizaje, se uso el número de épocas, esto se debe a que el tiempo de aprendizaje no puede ser usado como parámetro por la variación que existe en tiempo de respuesta en una Red de Comunicación compartida, en ocasiones el número de personas conectadas al servidor de trabajo, es demasiado alto o realizan trabajos muy complicados que absorben el tiempo de desempeño, por lo que nuestra Red Neuronal en la etapa de aprendizaje variaba demasiado.

La red NeuroShell, tiene menor desempeño al aprender vectores continuos que la Red Distribuida, pero es importante observar que esta ultima no reconoció correctamente todos los vectores de trabajo.



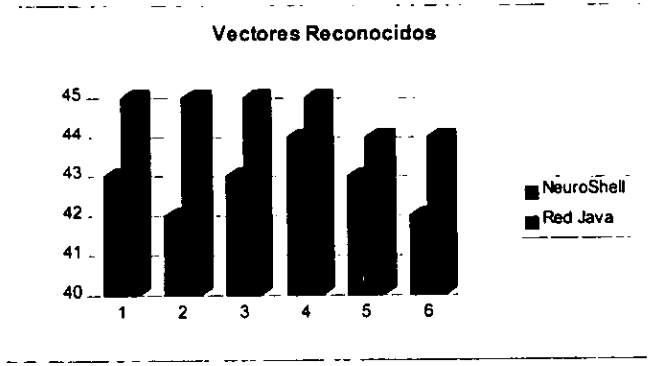
Gráfica VII.2 Comparación de Vectores reconocidos en la etapa de trabajo con vectores continuos

Para ambas redes el número de épocas disminuyo considerablemente usando vectores salteados en la etapa de aprendizaje, pero aun el número de épocas en la red NeuroShell, tienen menor desempeño que la Red Distribuida.



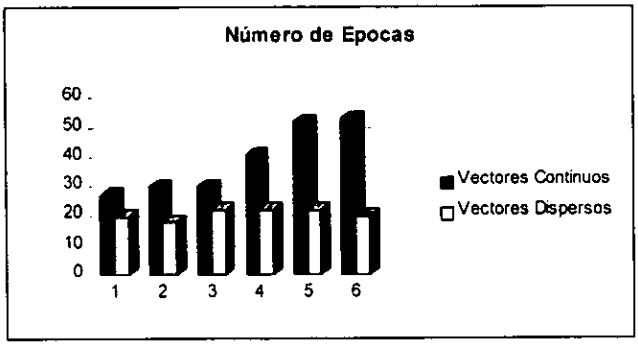
Gráfica VII.3 Comparación de Número de Epocas en la etapa de aprendizaje con vectores dispersos

En la siguiente gráfica que corresponde a la tabla VII.4, la Red Distribuida tuvo mejor desempeño que la Red NeuroShell, donde esta ultima su reconocimiento disminuye considerablemente, y su comportamiento fue muy variado, en el caso de la red Distribuida al aumentar demasiado el número de nodos en la capa intermedia, el reconocimiento disminuyo.



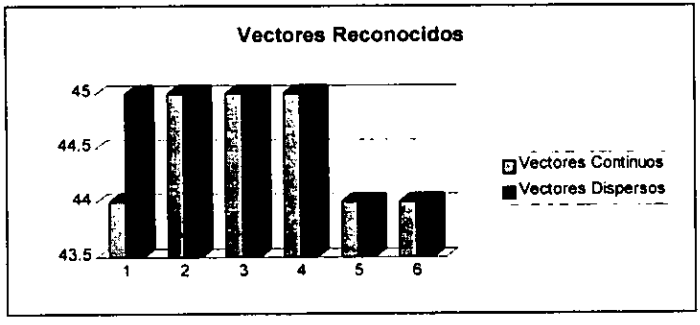
Gráfica VII.4 Comparación de Vectores Reconocidos en la etapa de trabajo con vectores dispersos

Si comparamos el desempeño entre los vectores de aprendizaje continuos o dispersos de la Red Distribuida, observamos el número de épocas es mayor en vectores continuos en comparación de los vectores dispersos en la etapa de aprendizaje de la red.

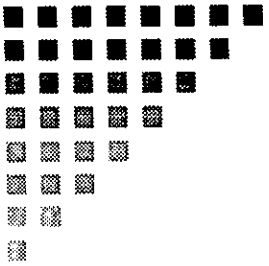


Gráfica VII.5 Comparación de Número de Epocas en la etapa de aprendizaje entre vectores continuos y vectores dispersos

En la etapa de trabajo también se nota que tienen un mejor desempeño la red Distribuida cuando se usan vectores dispersos en su etapa de aprendizaje En ambos reconocimientos el número de vectores disminuye cuando se aumenta el número de nodos en la capa intermedia.



Gráfica VII.6 Comparación de Número de Vectores Reconocidos en la etapa de trabajo entre vectores continuos y vectores dispersos



CONCLUSIONES

"¿Ves aque grupo de cangrejos que han colocado en esa fosa? Al principio, cuandeo llegan a ella, todos están entusiamados y con deseos de realizarse. Pero al ver que alguno trata de sobresalir: los demás se suben encima de él, hasta que el peso lo hace desistir, hundiéndolo de nuevo hasta el fondo." A.L.C.

Una de las razones de la simbiosis actual entre humanos y computadoras es la diferencia radical que hay entre ellos. Ya que las computadoras realizan tareas que nosotros realizamos torpemente y viceversa. Es muy difícil comprender que requerimos de una línea de investigación enfocada a desarrollar programas inteligentes como el caso de las Redes Neuronales.

Es muy complicado hacer que una red neuronal aprenda adecuadamente, ya que no existen patrones preestablecidos para hacerlo correctamente. El hecho de afirmar que una red neuronal aprenda significa hallar un conjunto de valores adecuado en los pesos, para que sea posible el reconocimiento de cualquier trama semejante a las que se utilizan durante la etapa de aprendizaje.

Durante el aprendizaje se pueden utilizar todos los datos que estén disponibles para entrenar la red, aunque quizá no sea necesario utilizarlos todos. El entrenamiento puede tener éxito, escogiendo aleatoriamente un pequeño subconjunto de los datos de entrenamiento de los que se dispone. Los datos restantes pueden emplearse para probar la red, con objeto de verificar que esta puede llevar a cabo la asociación deseada al utilizar vectores de entrada que nunca haya encontrado durante el entrenamiento.

Si se está entrenando una red para que funcione en un entorno con ruido, entonces es conveniente incluir unos cuantos vectores de entrada con ruido en el conjunto de datos de aprendizaje. Algunas veces la adición de ruido a los vectores de entrada durante el entrenamiento ayuda a la red a converger, incluso en el caso de que no se espere tal ruido. Si una red se diseña y/o se entrena de modo inadecuado, es decir, se emplea una clase concreta de vectores de entrada, por ejemplo caracteres sin ruido o en una sola posición, posteriormente la identificación de la clase para los vectores de trabajo puede ser imprecisa, o en el caso en que diseñamos una red con un numero pequeño de nodos en las capas para una clase muy compleja.

El entrenar a la red Neuronal con vectores de una clase, y pasar después con los vectores de otra clase, provoca que la red se olvide del entrenamiento anterior. Por lo que es conveniente introducir los vectores intercalados, uno por uno cada vector de cada clase. Esto depende del numero de vectores que se utilicen para la Etapa de Aprendizaje, como pudimos observar en este proyecto se utilizaron cien vectores de entrada para el aprendizaje, los cuales se introdujeron a la red de tres en tres por cada carácter.

Cuando nuestro numero de nodos en la capa intermedia disminuye es necesario aumentar la constante de aprendizaje, pero cuando aumenta no es conveniente usar una constante de aprendizaje muy alta. Es recomendable fijar el valor de la constante y disminuirlo en forma gradual cada determinado ciclo de aprendizaje.

El programar Redes Neuronales bajo el enfoque de Programación Orientado a Objetos, tiene como resultado la administración de la complejidad y el aumento de la productividad del programador, ya que los problemas complejos se dividen en módulos sencillos que ocultan los detalles de la realización recordemos aquel enfoque "divide y

vencerás". Esto no solo sucede al programar redes Neuronales sino cuando se utiliza la Programación Orientada a Objetos en cualquier aplicación.

Al cambiar de una metodología tradicional a la P.O.O, puede parecer complicado la comprensión al inicio, pero los objetivos del diseño pasan a modelar los objetos que existen en el mundo y sus comportamientos individuales, trayendo con ello grandes ventajas y reduciendo la complejidad en los sistemas, que con métodos tradicionales no se lograría. figura C.1

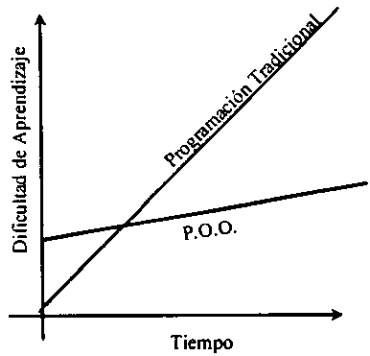


Figura C.1 Como se puede observar es menor la pendiente de la P.O.O. que de la pendiente en Programación Tradicional, por lo que en esta última al principio en muy sencilla de aprender pero conforme aumenta la complejidad de los sistemas se vuelve más complicado poder diseñar estos.

Considero importante opinar sobre dos puntos que muchas veces cuestionamos: el hecho de que un programa sea escrito en P.O.O. no es total garantía de que el programa sea mucho más flexible, y optimo, ya que depende de la correcta aplicación y entendimiento de los conceptos de la P.O.O. por lo que exige del programador un nuevo punto de vista. El segundo es referente a mencionar que la P.O.O tiene grandes ventajas como ya se observo en el capitulo IV, pero ello no quiere decir que desplaza a la programación estructurada, por que al diseñar algún objeto el programador debe de tener muy claro que los objetos en su interior deben estar correctamente estructurados.

Es válido entonces preguntarse: ¿Por qué programar en lenguajes orientados a objetos, siendo tan abstractos? Pero llegue a la conclusión de que las ventajas que proporciona la P.O.O, valen la pena intentarlas y esforzarse en aprender conceptos que nos trasladan a un nuevo punto de vista. Y no es que sea tan complicado sino, que como seres humanos siempre nos resistimos a los cambios, aun cuando estos nos faciliten las actividades.

Las características de orientación a objetos son útiles en toda etapa de vida de un programa. El polimorfismo reduce el número de procedimientos y por tanto el tamaño del programa que debe entender la persona encargada del mantenimiento. La herencia

de clase permite construir una nueva versión de un programa sin afectar a la antigua. El mecanismo de herencia documenta los cambios en un programa como subclases que representan la historia de las modificaciones realizadas en la superclase.

La P.O.O. facilita el uso de ambientes distribuidos. Pero muchos alguna vez nos hemos considerado que no es importante el tiempo de respuesta que se requiere para ciertas actividades. Pero por ejemplo en el caso de la Astronomía donde el tiempo se cuenta en milisegundos, el tener tareas distribuidas disminuye considerablemente los costos y el tiempo de respuesta.

En la actualidad existen muchos programas orientados a objetos, pero si tomamos en cuenta que requerimos cambiar nuestro punto de vista para programar, aunado a ello la abstracción que manejan la P.O.O, no todos los lenguajes de P.O.O. son eficientes, algunos nos complican más el camino, que en muchas ocasiones decepcionan al programar, pero con lenguajes como Java, donde es familiar el código, ya que muchos de nosotros hemos utilizado o conocemos el Lenguaje C, la programación y entendimiento de la orientación a objetos se vuelve menos complicado de comprender y utilizar, si a esto aumentamos la facilidad para Invocar Objetos Remotos.

Con todo lo anterior se puede concluir que los objetivos de este trabajo se alcanzaron en su totalidad observado que tan importante es tener una Red Neuronal Distribuida. En los resultados obtenidos para este diseño, como se observo en el capítulo VII, fueron satisfactorios, ya que el diseño y la implementación de este trabajo con el lenguaje Java, facilito el crecimiento de la Red Neuronal, para las pruebas de desempeño, las cuales fueron satisfactorias, por que se pudo comprobar que una Red Neuronal Distribuida, mejora el tiempo de reconocimiento que cualquier otra Red. Figura C.2.

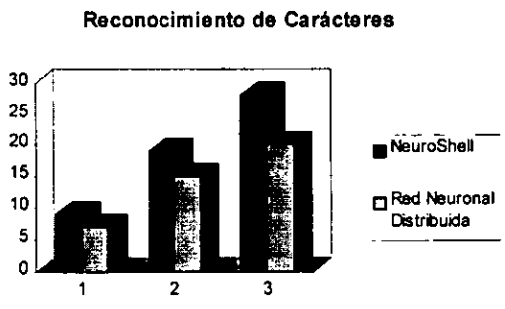
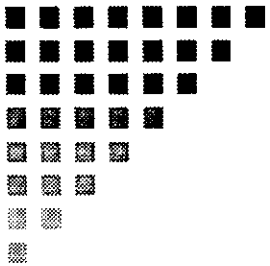


Figura C.2 Como se puede observar conforme aumenta el número de vectores para la etapa de trabajo en la Red NeuroShell es mayor el tiempo de reconocimiento, lo cual no sucede en la Red Neuronal Distribuida.



APÉNDICE

TPrincipal.java

```
import java.io.*;
import java.util.*;
import java.lang.Integer;


class TPrincipal{

  static int nElementos = 64;                    // Numero de elementos del vector de entrada
  static int nNodosCapas[]={10, 5};              // Arreglo del numero de nodos en cada capa
  static int nNumEntradas[]={64, 10};            // Arreglo de entradas en cada capa
  static int nNumCapas=2;                        // Numero de Capas
  static int prVectoresMuestra=100;              // Numero de vectores a aprender
  static String prClases[]={"A", "E", "I", "O", "U"};
  String prCaracter;

  //Inicializaciòn de Objetos
  TRed prRed=new TRed(nNumEntradas, nNumCapas, nNodosCapas);



  TPrincipal(){

     // Indica a la Red que Aprenda
    prRed.rdAprende(prVectoresMuestra);

  }

 String prEjecuta(int prVectorEntrada[]){

    int prClaseSalida[]=new int[nNodosCapas[nNumCapas-1]];

    // Evalùa la Entrada
    prClaseSalida=prRed.rdSalida(prVectorEntrada);

    for(int i=0; i<5; i++){
      System.out.print(prClaseSalida[i]);
     }

    for(int i=0; i< nNodosCapas[nNumCapas-1]; i++){

       if(prClaseSalida[i]==1){
          System.out.println(" ");
          System.out.println("El Vector pertenece a la Clase ... "+ prClases[i]);
          prCaracter=prClases[i];
       }

    }
```

```
    return prCaracter;
  }
}
```

TRed.java

```
import java.io.*;
import java.lang.*;

class TRed{

  TCapa[] rdCapa=new TCapa[10];                    // Arreglo de Capas
  int rdNodosCapas[]=new int[10];                  // Arreglo de Nodos por Capa
  int rdNumEntradas[] =new int[10];                // Arreglo de Entradas por Capa
  int rdNumCapas=0;                                // Numero de Capas


  TRed( int rdCopiaNumEntradas[], int rdCopiaNumCapas, int rdCopiaNodosCapas[]){

    // Asigna variables de copia

    rdNodosCapas=rdCopiaNodosCapas;
    rdNumEntradas =rdCopiaNumEntradas;
    rdNumCapas=rdCopiaNumCapas;

    // Inicializa el arreglo de capas

    for(int i=0; i<rdNumCapas; i++){
      rdCapa[i] = new TCapa(rdNumEntradas[i], rdNodosCapas[i]);
    }

  }


  // Realiza la rutina de aprendizaje para todos los vectores muestra

  public void rdAprende(int rdNumVectoresAprender){

    double rdVectorX[] = new double[90];           // Vector muestra
    double rdVectorSalida[][]= new double[90][90]; // Salidas de Retorno de Propaga-adelante
    double rdSumaError[]   = new double[90];       // Arreglo que contiene la suma de errores de
                                                   // la capa anterior
    boolean rdTerminaAprende=false;                // Status de Aprendizaje
    int rdContadorAprende=0;                       // Contador de Aprendizaje de Vectores
                                                   // muestra
    int rdContadorEpocas=0;                        // Contador de Epocas
```

```
// Inicializa Objetos
    TDatos rdDatos= new TDatos(rdNumEntradas[0],"Aprende.dat");


    do{

      for(int p=0; p< rdNumVectoresAprender; p++){

        // Convierte el vector leído del archivo de vectores muestra a double y asigna al VectorX
        // el vector muestra que corresponde

        for(int i=0; i< rdNumEntradas[0] ;i++){
          rdVectorX[i]=(double) rdDatos.dtVectorClase[p][i];
        }

        // Realiza propagación hacia adelante

        for(int i=0; i< rdNumCapas ;i++){

          if(i == 0){
            rdVectorSalida[i]=rdCapa[i].cpAprendeAdelante(rdVectorX);
          }else {
            rdVectorSalida[i]=rdCapa[i].cpAprendeAdelante(rdVectorSalida[i-1]);
          }

        }


        // Realiza propagación hacia atrás

        for(int i=rdNumCapas-1; i>=0 ;i--){

          if(i == (rdNumCapas-1)){              // Propagación hacia atrás en Capa de Salida
            rdTerminaAprende=rdCapa[i].cpAprendeAtras(rdNumVectoresAprender,p);
          }else {

            if(rdTerminaAprende == false ){
              rdSumaError=rdCapa[i+1].cpSumaErrorPeso();
              rdCapa[i].cpAprendeAtras(rdSumaError);

            }

          }

        }

        // Realiza ajuste de pesos en las capas
        if(rdTerminaAprende == false ){
```

```
        for(int i=0; i< rdNumCapas ;i++){
          rdCapa[i].cpAjustaPesos();
        }

      }

      // Checa si aprendió en vector correspondiente e incrementa Contador de aprendizaje
      // y continua con el siguiente vector de aprendizaje

      if(rdTerminaAprende == true){

        rdContadorAprende++;
        rdTerminaAprende=false;
      }

      rdContadorEpocas++;
      System.out.print(" "+rdContadorEpocas);
      System.out.print(" "+rdContadorAprende);

    }


  // Checa si todos los vectores muestra fueron aprendidos, si no realiza nuevamente la etapa
  // de aprendizaje

    if(rdContadorAprende == rdNumVectoresAprender){
      rdTerminaAprende=true;

    }else{
      rdTerminaAprende=false;
      rdContadorAprende=0;
    }

  }while(rdTerminaAprende == false);

  rdTerminaAprende=false;

}


// Evalúa la Salida correspondiente

public int[] rdSalida(int rdCopiaVectorEntrada[]){

  int rdSalida[]= new int[90];
  double rdVectorEntrada[]=new double[90];
  double rdVectorSalida[][]= new double[90][90];
  // Convierte el vector de entrada en double

  for(int i=0; i< rdNumEntradas[0] ; i++){
    rdVectorEntrada[i]= (double) rdCopiaVectorEntrada[i];
  }
```

```
    for(int i=0; i< rdNumCapas ;i++){

      if(i == 0){          rdVectorSalida[i]=rdCapa[i].cpAprendeAdelante(rdVectorEntrada);
      }else {
        rdVectorSalida[i]=rdCapa[i].cpAprendeAdelante(rdVectorSalida[i-1]);
      }

    }

    for(int i=0; i< (rdNodosCapas[rdNumCapas-1]) ; i++){

      if(rdVectorSalida[rdNumCapas-1][i] >= 0.5 ){
        rdSalida[i]=1;
      } else {
        rdSalida[i]=0;
      }

    }

    return  rdSalida;

  }

}
```

TCapa.java

```
import java.io.*;
import java.lang.*;

class TCapa{

  TNodo[] cpNodoCapa=new TNodo[70];                    // Arreglo de Nodos
  TDatos cpDatos;                                      // Objeto para leer la salida deseada
  int cpNumNodosCapa=0;                                // Num. de nodos
  int cpNumEntradas=0;                                 // Num. de Entradas para cada nodo
  double cpSalidaCapa[];                               // Salidas de los Nodos
  double cpErrorCapa[]= new double[90];                // Error para la capa
  double cpVectorEntrada[];                            // Vector de Entrada
  TCapa (int cpCopiaNumEntradas, int cpCopiaNodosCapa){

    cpNumNodosCapa=cpCopiaNodosCapa;
    cpNumEntradas=cpCopiaNumEntradas;


    // Inicializa los arreglos de vectores de peso para la capa
```

```
    for(int i=0; i<cpNumNodosCapa; i++){
      cpNodoCapa[i] = new TNodo(cpNumEntradas);
    }

    // Incializa objeto para leer la salida deseada

    cpDatos= new TDatos(cpNumNodosCapa,"Clases.dat");

  }

  public double[] cpAprendeAdelante(double cpCopiaVectorEntrada[]){

    cpVectorEntrada=cpCopiaVectorEntrada;
    cpSalidaCapa=new double[cpNumNodosCapa];          // Salidas de los Nodos Capa

    // Realiza el aprendizaje en su etapa de propagación hacia adelante

  for(int i=0; i < cpNumNodosCapa; i++){
      cpSalidaCapa[i]=cpNodoCapa[i].ndAprende(cpVectorEntrada);
    }
    return cpSalidaCapa;

  }

  // Realiza el aprendizaje en su etapa de propagación hacia atrás en la Capa Salida

  public boolean cpAprendeAtras(int cpNumVectoresApende, int cpNumSalidaDeseada){

    double cpVectorDeseado[]= new double[cpNumNodosCapa];
    int cpTotalNodos=0;


    // Asigna vector de salida deseada

    for(int i=0; i< cpNumNodosCapa ;i++){
      cpVectorDeseado[i]=(double) cpDatos.dtVectorClase[cpNumSalidaDeseada][i];
    }


    cpTotalNodos=0;

    for(int j=0; j < cpNumNodosCapa; j++){
      cpErrorCapa[j]=cpSalidaCapa[j]*(1-cpSalidaCapa[j])*(cpVectorDeseado[j] - cpSalidaCapa[j]);
      if (Math.abs( cpErrorCapa[j]) <= 0.1){

        cpTotalNodos++;
```

```
        if(cpTotalNodos == cpNumNodosCapa)
          return true;
      }

    }

    return false;

  }

  // Realiza el aprendizaje en su etapa de propagación hacia atrás para las Capas medias

  public boolean cpAprendeAtras(double cpSumaError[]){

    for(int i=0; i<cpNumNodosCapa; i++){
      cpErrorCapa[i]=cpSalidaCapa[i]*(1-cpSalidaCapa[i])*cpSumaError[i];
    }

    return false;

  }

 // Calcula Sumatoria de Error por Pesos

  public double[] cpSumaErrorPeso(){

    double cpSumaEW[]=new double[cpNumEntradas];

      for(int i=0; i< cpNumEntradas; i++){
        for(int j=0; j< cpNumNodosCapa;j++){
          cpSumaEW[i]=cpErrorCapa[j]*cpNodoCapa[j].ndVectorW[i]+cpSumaEW[i];
        }
      }

      return cpSumaEW;

    }

    // Ajuste de pesos

    public void cpAjustaPesos(){

  //Mult. de Error * Entrada Correspondiente

      double cpAjusteError[]= new double[cpNumEntradas];
        for(int i=0; i < cpNumNodosCapa; i++){

        for(int j=0; j < cpNumEntradas; j++){
          cpAjusteError[j]=cpErrorCapa[i]*cpVectorEntrada[j];
        }
        cpNodoCapa[i].ndAjustaPesos(cpAjusteError);
      }
```

```
  }
}
```

TNodo.java

```
import java.util.*;
import java.io.*;
import java.lang.*;


class TNodo{

  double ndVectorW[] =new double[100];          // Vector de Pesos
  int ndElementos=0;                            // Num Elementos del Vector de Peso
  double ndTeta=0;                              // Valor teta
  double ndFactAprende=0;                       // Factor de aprendizaje
  int ndIteraciones=0;                          // Variable de iteraciones
  int ndContadorIter=0;                         // Contador de iteraciones
                    // Inicializa constantes
  static TConstantes ndConstantes= new Tconstantes("Const.dat");


  TNodo(int ndNumEntradas){

    Random rand= new Random();
    ndElementos=ndNumEntradas;

    for(int ndContador=0; ndContador < ndNumEntradas; ndContador++){
    // Inicia vector W, con números aleatorios
    // dentro de un rango inicial de 0 y 1

      do{
            ndVectorW[ndContador]=rand.nextFloat();
      }while(ndVectorW[ndContador] < -0.1 || ndVectorW[ndContador] > 0.1 );
    }
// Inicializa Objetos
    //TConstantes ndConstantes= new Tconstantes("Const.dat");

    ndTeta=ndConstantes.cnVectorCts[0].doubleValue();
    ndFactAprende=ndConstantes.cnVectorCts[1].doubleValue();
    ndIteraciones=ndConstantes.cnVectorCts[2].intValue();

  }

// Realiza el aprendizaje del nodo
```

```
public double ndAprende(double ndVectorX[]){

  double ndSalida;
  double ndSumatoriaWX=0;

  // Realiza la Sumatoria y aplica la función Sigmoidal

  for(int i=0; i < ndElementos; i++){
    ndSumatoriaWX=(ndVectorW[i]*ndVectorX[i])+ndSumatoriaWX;
  }

  ndSalida= 1/(1+Math.exp(-(ndSumatoriaWX-ndTeta)));

  return ndSalida;

 }
 // Ajuste de Pesos

 void ndAjustaPesos(double ndAjusteError[]){

  for(int i=0; i < ndElementos; i++){
    ndVectorW[i]= ndVectorW[i]+ndFactAprende*ndAjusteError[i];
  }

  ndContadorIter++;
  if(ndContadorIter == ndFactAprende){

    ndContadorIter=0;
    ndFactAprende=ndFactAprende/2;

  }

 }

}
```

TConstantes.java

```
import java.io.*;
import java.lang.Double;


class TConstantes{

  Double cnVectorCts[]=new Double[20];
```

```
TConstantes(String cnArchivo){

   String cnLinea= new String();
   int contador=0;

   // Inicializacion de costantes desde un archivo

   try{

      RandomAccessFile cnArchivoEnt=new RandomAccessFile(cnArchivo,"r");
      cnLinea=cnArchivoEnt.readLine();
      while(cnLinea != null )
      {
          cnVectorCts[contador]=Double.valueOf(cnLinea);
          contador++;
          cnLinea=cnArchivoEnt.readLine();

      }

   } catch (IOException e){
      System.err.println("FileStreamsTest:"+e);
   }

 }


}
```

TDatos.java

```
import java.io.*;

class TDatos{

  int dtVectorClase[][]=new int[110][110];


  TDatos(int nElementos,String dtArchivo){

     int caracter;
     int contador=0;
     int clase=0;
     int sumacon=0;

     // Inicializacion de vectores de aprendizaje desde un archivo
```

```
    try{

     FileInputStream dtArchivoEnt=new FileInputStream(dtArchivo);
     caracter=dtArchivoEnt.read();

     while(caracter != -1){
       while( ((char)caracter)!=' '  && contador <= nElementos && caracter != -1 && caracter!=13
                                          && caracter!=10 ){
         caracter = caracter - ((byte) '0');
         dtVectorClase[clase][contador]=caracter;
         contador++;
         sumacon=0;
         caracter=dtArchivoEnt.read();
       }

       if (contador<= nElementos && sumacon == 0){
         contador=0;
         clase++;
         sumacon++;
       }

       caracter=dtArchivoEnt.read();

     }

     dtArchivoEnt.close();

} catch (IOException e){
      System.err.println("FileStreamsTest:"+e);
    }
  }


}
```

TRemoteCliente.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.Naming;

public class TRemoteCliente {

  public static void main(String args[]) {

    TCreaThread1  clThread1 = new TCreaThread1("1");
    TCreaThread2  clThread2 = new TCreaThread2("2");
```

```
    clThread2.start();
    clThread1.start();

  }

}
```

TCreaThread1.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.Naming;


class TCreaThread1 extends Thread{

   int crNumElementos=64;
   int crNumTotalVectores=12;
   String crArchivo="Texto.dat";
   int crIndice1=0;

// Inicializacion de Objetos

   TResultado crResultado=new TResultado(crNumTotalVectores);


    public TCreaThread1 (String crThread){

     super (crThread);
     System.out.println("Nombre : " + getName() );

   }


   public void run() {

     int crVectorReconoce[]=new int[90];

     // Inicializacion de Objetos

     TDatos crLeerArchivo =new TDatos(crNumElementos,crArchivo);


     // Como la 1a. llamada se hace a un servidor anterior no fue necesario escribir
     // la siguiente linea System.setSecurityManager(new RMISecurityManager());
```

```
    try {

      TRemoteInterface server = (TRemoteInterface) Naming.lookup("//tsekub:1204/Servidor
                                Tsekub");
      String serverString;

      for(int p=0; p < crNumTotalVectores; p+=2){

        // Convierte el vector leido del archivo de vectores muestra a double y
        // asigna al VectorReconoce el vector muestra que corresponde

        for(int i=0; i< crNumElementos ;i++){
          crVectorReconoce[i]= crLeerArchivo.dtVectorClase[p][i];
        }


        serverString = server.message(" Hola aqui estoy cliente ",crVectorReconoce);
        System.out.println("El servidor TSekub reconocio un caracter :\n"+"   "+serverString);
        crResultado.rsAgrega(crIndice1,serverString);

        crIndice1+=2;
      }

} catch (Exception e) {
       System.out.println("Error while performing RMI");
    }

  }


}
```

TCreaThread2.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.Naming;


class TCreaThread2 extends Thread{

  int crNumElementos=64;
  int crNumTotalVectores=12;
  String crArchivo="texto.dat";
  int crIndice2=1;

  // Inicializacion de Objetos
```

```
TResultado crResultado=new TResultado(crNumTotalVectores);


 public TCreaThread2 (String crThread){

  super (crThread);
  System.out.println("Nombre : " + getName() );

 }


 public void run() {

  int crVectorReconoce[]=new int[90];

  System.setSecurityManager(new RMISecurityManager());

  TDatos crLeerArchivo =new TDatos(crNumElementos,crArchivo);

try {

    TRemoteInterface2 server = (TRemoteInterface2) Naming.lookup("//utopia:1206/Servidor
                               Utopia");
    String serverString;

    for(int p=1; p < crNumTotalVectores; p+=2){

      // Convierte el vector leido del archivo de vectores muestra a double y
      // asigna al VectorReconoce el vector muestra que corresponde

      for(int i=0; i< crNumElementos ;i++){
        crVectorReconoce[i]= crLeerArchivo.dtVectorClase[p][i];
      }


      serverString= server.message2(" Hola aqui estoy cliente ",crVectorReconoce);
      System.out.println("El servidor Utopia reconocio un caracter :\n"+"  "+serverString);
      crResultado.rsAgrega(crIndice2,serverString);

      crIndice2+=2;

    }

  } catch (Exception e) {
     System.out.println("Error while performing RMI"+e);
  }
```

```
  }
}
```

TRemoteInterface.java

```
import java.rmi.*;

public interface TRemoteInterface extends java.rmi.Remote {
  String message (String message, int cmd[]) throws java.rmi.RemoteException;
}
```

TRemoteServer.java

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;


public class TRemoteServer extends UnicastRemoteObject implements TRemoteInterface {
  String name;


  //inicializacion de objetos
  TPrincipal msNeuronal=new TPrincipal();

  public TRemoteServer(String name) throws RemoteException {

    super();
    this.name = name;

  }

  public String message (String message, int cmd[]) throws RemoteException {

     String   msCaracter;

     // ejecucion de un objeto
     msCaracter= msNeuronal.prEjecuta(cmd);
```

```
    return msCaracter;

  }

  public static void main (String args[]) {

    System.setSecurityManager (new RMISecurityManager());

    try {

      String myName1 = "//tsekub:1204/Servidor Tsekub";
      TRemoteServer theServer = new TRemoteServer (myName1);
      Naming.rebind(myName1, theServer);

    } catch (Exception e) {
      System.out.println("An Exception occured while creating server"+e);
    }

  }
}
```

TRemoteInterface2.java

```
import java.rmi.*;


public interface TRemoteInterface2 extends java.rmi.Remote {

  String message2 (String message, int cmd[]) throws java.rmi.RemoteException;

}
```

TRemoteServer2.java

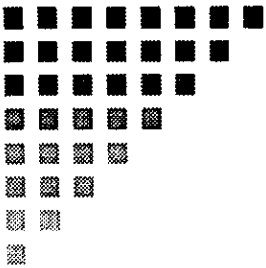
```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;

import java.*;


public class TRemoteServer2 extends UnicastRemoteObject implements TRemoteInterface2 {
```

```
String name;

//inicializacion de objetos
TPrincipal rsNeuronal=new Tprincipal();


public TRemoteServer2(String name) throws RemoteException {

  super();
  this.name = name;

}

public String message2 (String message, int cmd[]) throws RemoteException {
   String msCaracter;
// ejecucion de un objeto

   msCaracter= rsNeuronal.prEjecuta(cmd);

   System.out.println(msCaracter+"  ");

   return msCaracter ;


public static void main (String args[])

   System.setSecurityManager (new RMISecurityManager())

   try{

     String myNamee1 = "//utopia :/1206/Servidor Utopia" ;
     TRemoteServer2 theServer = new TRemoteServer2
     Naming.rebind(myName1, theServer) ;

   } catch (Exception e)

  System.out.println("An Execption ocurred while ceating server"+e);

 }
```

TResultado.java

```
import java.rmi.*;
import java.io.*;


class TResultado{
```

```
static String agVectorCts[]=new String[20];
static int rsFinAgrega=0;
static int rsAgrega=0;


 TResultado(int rsDimension){

    rsFinAgrega=rsDimension;

 }


 void rsAgrega(int agIndice, String agValorCte){
agVectorCts[agIndice]=agValorCte;
   rsAgrega++;

   if(rsAgrega == rsFinAgrega){
     rsEscribe_datos();
   }

 }


 public void rsEscribe_datos() {

   RandomAccessFile esNombreArchivo;

   try{

     esNombreArchivo= new RandomAccessFile("Resultado.txt","rw");

     for(int i=0; i< rsFinAgrega; i++){
       esNombreArchivo.writeBytes(agVectorCts[i]);
       esNombreArchivo.writeBytes("\n");
     }

     esNombreArchivo.close();

   } catch (IOException e){
     System.err.println("FileStreamsTest:"+e);
   }

 }

}
```



BIBLIOGRAFÍA

(1) AN INTRIDUCTION TO OBJET ORIENTED PROGRAMMING
Timothy Budd
Editorial Addison Wesley Publishing Company, 1991

(2) ARTIFICIAL INTELLIGENCE
Handbook Volumen II Aplications
Intrument Society of America, 1989

(3) COMO PROGRAMAR EN JAVA
H.M.Deitel / P.J.Deitel
Editorial Prentice Hall, 1998

(4) CONCURRENT COMPUTATIONS
Tewksbury / Dickinson

(5) FUNDAMENTALS OF NEURAL NETWORKS
Laurenee Fausett
Editorial Prentice Hall, 1994

(6) FUNDAMENTOS DE PROGRAMACION
Ernesto Peñaloza Romero
UNAM. ENEP-Aragón, 1994

(7) INGENIERIA ARTIFICIAL, Conceptos, Técnicas y Aplicaciones
José Mumpín Poblet
Editorial Marcombo Boixareu Editores, 1987

(8) INTELIGENCIA ARTIFICIAL
Elaine Rich / Kevin Knight
Editorial Mc Graw Hill, 1994.

(9) INGENIERIA DE SOFTWARE UN ENFOQUE PRACTICO
Roger S. Pressman
Editorial Mc.Graw Hill, 1993

(10) MANUAL DE JAVA
Patrick Naughton
Editorial Osborne McGraw Hill, 1996

(11) NEUROL COMPUTATION AND SELF-ORGANIZING MAPS.
Helge Ritter / Thomas Martinetz / Klaus Schulten
Editorial Addison Weslwy Publishing Company Inc., 1992

(12) NEUROL NETWORKS, Algoritms, Applications, and Proggramming
Ttechniques.
James A. Freeman / David M.Skapura
Editorial Addison Weslwy Publishing Company Inc., 1992

(13) NEURONAL NETWORKS
Cliffard Lau
Editorial IEEE-PRESS, 1991

(14) PROGRAMING FOR ARTIFICIAL INTELIGENCE
Worfgang Kreutser / Bruce Mckenzie
Editorial Addison Wesley, 1990.

(15) PROGRAMACION ORIENTADA A OBJETOS, Aplicaciones con Smalltalk.
Angel Morales Lazano / Francisco J.Segovia Perez
Editorial Paraninfo, 1993

(16) SISTEMAS OPERATIVOS Diseño e Implementación
Andrews S. Tanenbaum
Editorial Preentice Hall, 1988

(17) SISTEMAS OPERATIVOS Conceptos y Diseño
Milan Milenkovic
Editorial McGraw Hill, 1988

(18) SISTEMAS OPERATIVOS
William Stallings
Editorial Megabyte Noriega Editores, 1995

(19) SOFTWARE ORIENTADO A OBJETOS
Ann L.Winblad / Samuel D.Edwards / David R.King
Editorial Addison Wesley / Diaz Santos, 1993

(20) THE JAVA TUTORIAL
Mary Campione / Kathy Walrath
Editorial Addison Wesley, 1996

PAGINAS DE INTERNET

(21) http ://fciencias.ans.vabe.mc

(22) http ://www.javasoft.com

(23) http ://www.pegasus.uniandes.edu.co

(24) http ://www.pse.res.titech.ac.ip

(25) http ://www.sun.com

(26) http ://www.une.edu.ve

APUNTES

(27) APUNTES DE REDES NEURONALES
Almicar A. Monterrosa Escobar
ENEP ARAGON