INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS ESTADO DE MÉXICO

MÉTODOS PARA ACELERAR LA VELOCIDAD DE CONVERGENCIA EN REDES FEED-FORWARD MEDIANTE ALGORITMOS DE APRENDIZAJE PARALELOS

TESIS QUE PRESENTA

GUILLERMO RUEDA RUEDA

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

[redacted]

DICIEMBRE, 1997

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS ESTADO DE MÉXICO

MÉTODOS PARA ACELERAR LA VELOCIDAD DE CONVERGENCIA EN REDES FEED-FORWARD MEDIANTE ALGORITMOS DE APRENDIZAJE PARALELOS

TESIS QUE PARA OPTAR EL GRADO DE
MAESTRO EN CIENCIAS DE LA COMPUTACIÓN
PRESENTA

GUILLERMO RUEDA RUEDA

Asesores: Dr. LUIS TREJO RODRÍGUEZ
M. en C. CARLOS SANDOVAL CISNEROS

Comité de tesis: Dr. MARTÍN LOPEZ MORALES
M. en C. FRANCISCO CAMARGO SANTACRUZ

Jurado:	Dr. MARTÍN LOPEZ MORALES	Presidente
	M. en C. FRANCISCO CAMARGO SANTACRUZ	Secretario
	Dr. LUIS TREJO RODRÍGUEZ	Vocal
	M. en C. CARLOS SANDOVAL CISNEROS	Vocal

Atizapán de Zaragoza, Edo. Méx., Diciembre de 1997.

Tabla de Contenido

Capítulo 3

Máquinas Paralelas

Capítulo 4

Planteamiento del Problema

Capítulo 5

Resultados

Capítulo 6

Conclusiones

Bibliografía

Anexo A: Programas Fuente

Introducción

Objetivos del Trabajo de Investigación

El presente trabajo de investigación tiene como objetivo el estudio de diversos modelos que permitan diseñar métodos de entrenamiento en máquinas con arquitectura paralela que permitan disminuir el tiempo (tiempo físico) del proceso de aprendizaje para redes feeed-forward, usando como estrategia de entrenamiento back-propagation.

Dentro del área de redes neuronales existen múltiples trabajos para determinar métodos que permitan acelerar la velocidad de convergencia para la fase de aprendizaje en redes neuronales feed-forward, pero ninguno de ellos busca acelerar la velocidad de convergencia para la fase de aprendizaje por un método alternativo al del paralelismo puramente funcional, es decir, se basan en tres supuestos :

- Paralelismo puramente funcional.
- Paralelismo espacial.
- Paralelismo de patrones.

El objetivo es determinar métodos de paralelización de redes feed-forward, fundamentados para el análisis del mejor método en el tiempo de entrenamiento mínimo, bajo la consideración de propiedades estadísticas de valor esperado y varianza.

Como parte de este estudio se han definido tres dominios para estudiar el comportamiento de las redes neuronales feed-forward en cada uno de ellos:

- Dominio de los datos.
- Dominio del espacio de soluciones.
- Dominio aritmético.

Organización del Documento

El presente documento está dividido en cinco grandes capítulos. En el capítulo uno hace una breve descripción de los modelos computacionales y se estudian las redes neuronales como modelos alternativos de la computación en la solución de problemas complejos.

En el capítulo dos se dan los conceptos fundamentales de redes neuronales tipo feed-forward y se estudia de manera detallada la estrategia de aprendizaje mas utilizada -back-propagation- como procedimiento de aprendizaje supervisado basado en la disminución del error cuadrático total a la salida de la red.

En el capítulo tres se mencionan y explican los conceptos básicos de las máquinas paralelas. Así como los esquemas de clasificación relativos al flujo de instrucciones y al flujo de datos.

El capítulo cuatro es sin duda alguna el que constituye el soporte fundamental de este trabajo de investigación. En el se hace el planteamiento del problema, posteriormente se hacen algunas consideraciones básicas y se presentan las propuestas de solución.

En el capítulo cinco finalmente se presenta en forma estadística y gráfica los resultados que permiten visualizar en forma clara la contribución del presente trabajo, utilizando ejemplos -xor y parity-.

Finalmente se presenta el anexo donde se muestran las principales estructuras de datos usadas en la implementación y el respectivo código fuente en la implementación del programa, respectivamente documentado.

Capítulo 1

Introducción

1.1. Modelos Computacionales

La palabra computación se usa en el lenguaje científico-técnico para referirse al concepto de procesamiento automático de información es decir, al proceso que se basa en la aplicación de reglas formales (sintácticas) a representaciones abstractas de objetos concretos para la obtención de otras representaciones alternativas, o derivadas, de las representaciones originales.

El modelo matemático más elemental de un proceso computacional lo constituye la máquina de Turing que trabaja con representaciones determinadas por secuencias de símbolos de un alfabeto finito (mediante un conjunto de convenciones interpretativas). La máquina de Turing se visualiza como un autómata con un número finito de estados que "lee" una cinta semi-infinita en la que se han codificado símbolos de un alfabeto de entrada. El autómata evoluciona en un tiempo discreto, realizando transiciones entre sus diferentes estados y desplazándose en una dirección u otra a lo largo de la cinta, cuyo contenido puede ser modificado durante el proceso, según el estado del

autómata y el símbolo contenido en el registro que se esté explorando en cada ciclo de cómputo: lectura de la cinta - cambio de estado - escritura en la cinta - desplazamiento del dispositivo de lectura escritura.

En el fundamento de la teoría general de la computación está la hipótesis de Church-Turing según la cual la máquina de Turing constituye la formalización matemática del concepto intuitivo (expuesto antes) de proceso computacional.

Desde el punto de vista práctico, sin embargo, la máquina de Turing tiene muy poco valor. Uno de los objetivos centrales del estudio de la ciencia computacional ha sido la definición de sistemas físicos capaces de hacer explícitos los procesos computacionales, para lo cual la máquina de Turing no es un modelo apropiado, por lo que en su lugar se han utilizado modelos alternativos, como la máquina de Von Newman, que permiten realizar efectivamente, en mecanismos físicos, los procesos de cómputo.

La máquina de Von Newman, es el modelo computacional implementado en la mayoría de las máquinas de cómputo y se organiza, básicamente, en tres componentes: la CPU, la memoria y las interfaces de entrada-salida. A diferencia de la máquina de Turing (en realidad hay una infinidad de máquinas de Turing) la máquina de Von Newman es una máquina de propósito general. Para el proceso computacional los datos se codifican en la memoria del sistema, que es explorada por la unidad de procesamiento y control. Una máquina de Von Newman junto con un programa codificado en su memoria es matemáticamente equivalente a una máquina de Turing en el sentido de que existe una familia de homeomorfismos que "mapean" el proceso

de una máquina en el de la otra.

Durante los ciclos de cómputo de una máquina de Von Newman se transforma el contenido de la memoria, al cual se tiene acceso (desde fuera) a través de la CPU, mediante los dispositivos de entrada-salida. Las transformaciones que aplica la CPU al contenido de la memoria, en cada ciclo de cómputo, son muy sencillas e involucran el contenido de unos pocos registros, por lo que se dice que la máquina de Von Newman es una máquina secuencial. Por lo que se ve del desarrollo de las máquinas computadoras en las últimas décadas, desarrollo que se palpa en una gran variedad de aplicaciones: en sistemas de información, de procesamiento de textos, de telecomunicaciones, de control de procesos industriales, etc., la implementación del modelo de Von Newman ha sido un gran éxito científico-técnico; sin embargo, al mismo tiempo, se han hecho evidentes las limitaciones prácticas del modelo en algunos campos de aplicación, por ejemplo, las que tienen que ver con la implementación de sistemas computacionales para la emulación de la percepción y de los procesos del pensamiento.

Desde el inicio del proceso de desarrollo de la actual tecnología de cómputo, de hecho, ha habido interés en el estudio de modelos alternativos de la computación. El trabajo de McCulloch y Pitts[1] es pionero en el estudio de una familia de sistemas computacionales que, explotando una analogía con la arquitectura de los tejidos nerviosos, se definen a partir de la interconexión de numerosos elementos de

---

[1] McCulloch, W.S., and Pitts, W.E. (1943). A logical calculus of the ideas inmanent in nervous activity. Bull. Math. Biophysics, 5, pp. 115-133.

procesamiento de información en redes de procesadores, conocidas como *redes neuronales*. Una red neuronal puede describirse como un grafo dirigido con las aristas y los vértices etiquetados. Los vértices del grafo se interpretan como procesadores (neuronas), las aristas, junto con su dirección, como vías de flujo de información y las etiquetas de las aristas como características de los correspondientes canales del flujo de datos.

El desarrollo de los sistemas de redes neuronales continuó por más de veinte años desde la publicación del trabajo de McCulloch y Pitts, recibiendo contribuciones importantes de Donald Hebb en 1949[2], de F. Rosenblatt[3] en 1958, de Widrow y Hoff[4] en 1960 pero prácticamente se interrumpió con la publicación de un libro de Minsky y Papert[5] en 1968.

El interés por las redes neuronales renació en los ochentas, como consecuencia de los trabajos de Werbos[6] en 1974, Hoppfield[7] en 1982, Parker[8] en 1982, y McClelland y

---

[2] Donald O. Hebb; The organization of behaviour; Wiley, New York (1949).

[3] Rosenblatt, F.; The Perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65, pp. 386-407 (1958).

[4] Bernard Widrow and Marcian E. Hoff. Adaptative switching circuits. 1960. IRE WESCON Convention record, New York, pp. 96-104. IRE.

[5] Marvin Minsky y Seymour Papert. Perceptrons. MIT Press: Cambridge, MA, 1969.

[6] Werbos, P.J. 1974. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph.D. thesis, harvard University.

[77] John J. Hopffield. Neural networks and physical system with emergent collective computational abilities. Proc. Natl. Acad. Sci. USA. 79, pp.2254-2558, Abril de 1982. Biofísica.

Rumelhart[9] en 1986. La copiosa literatura que se ha publicado en el campo, en los últimos años, permite afirmar que ya se ha consolidado el concepto de red neuronal como alternativa interesante de arquitectura computacional para la solución de problemas complejos.

1.2. Objetivos Generales

Dentro del área de redes neuronales existen múltiples trabajos para determinar métodos que permitan acelerar la velocidad de convergencia para la fase de aprendizaje en redes neuronales feed-forward, pero ninguno de ellos busca acelerar la velocidad de convergencia para la fase de aprendizaje por un método alternativo al del paralelismo puramente funcional, es decir, se basan en tres supuestos :

- Paralelismo puramente funcional.
- Paralelismo espacial.
- Paralelismo de patrones.

El objetivo es determinar métodos de paralelización de redes feed-forward, fundamentados para el análisis del mejor método en el tiempo de entrenamiento mínimo, bajo la consideración de propiedades estadísticas de valor esperado y

---

[8] Parker, D.B. 1982. Learning Logic, Invention report S1-64, File 1. Office of technology Licensing, Stanford University.

[9] James McClelland y David Rumelhart, Parallel Distributed Processing, Vól. I-II. MIT Press. Cambridge, MA, 1986.

varianza. Para ello se pueden definir nuevos dominios para la paralelización. Como parte de este estudio se han definido tres dominios para un estudio riguroso :

- Dominio de los datos.
- Dominio del espacio de soluciones.
- Dominio aritmético.

1.3. Organización del Documento

El presente documento está dividido en seis capítulos y un anexo. En el capítulo uno se hace una introducción a la investigación definiendo los modelos computacionales y mencionando los objetivos generales y presentando las principales contribuciones de los resultados encontrados.

En el capítulo dos se dan los conceptos fundamentales de redes neuronales tipo feed-forward y se define el funcionamiento de las fases : adelante y atrás. De igual forma se defina el proceso de aprendizaje, mencionando algunas consideraciones útiles en el proceso de aprendizaje.

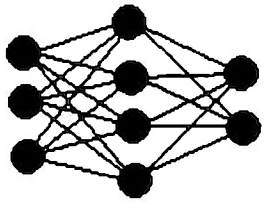
En el capítulo tres se mencionan y explican los conceptos básicos de las máquinas paralelas. En el capítulo cuatro es sin duda alguna el capítulo que constituye el soporte fundamental de este trabajo de investigación. En el se hace el planteamiento del problema, posteriormente se hacen algunas consideraciones básicas y se presentan las propuestas de solución.

En el capítulo cinco se utilizan ejemplos de xor y parity con el fin de mostrar en formar gráfica y estadística los resultados que permitan visualizar en forma clara la contribución del presente trabajo. Finalmente se presentan las conclusiones del presente trabajo de investigación y como anexo se muestran algunas de las principales estructuras de datos usadas en la implementación y el respectivo código fuente.

Capítulo 2

Redes Neuronales Feed-Forward

Existen diversos modelos de redes neuronales, entre los que se destaca, por sus diversas aplicaciones, el modelo de red neuronal feed-forward. Una red feed-forward, como su nombre lo indica, está constituida por un conjunto de elementos de procesamiento, organizados por capas C_0, Λ, C_n, de tal suerte que los elementos de procesamiento de la capa C_0 funcionan como interface de entrada y la información se transmite, entre capas consecutivas, desde la capa de entrada a la capa de salida C_n. Esquemáticamente, la arquitectura de una red neuronal feed-forward es como se muestra en la figura:



Los elementos de procesamiento (EP) de una red neuronal feed-forward (y de otro tipo de redes neuronales) se comportan como filtros analógicos: cada neurona recibe información de otras neuronas a través de las conexiones (sinápsis), información que

está codificada como un número real; esta información es "integrada" por el elemento de procesamiento para obtener una "excitación" neta, a partir de la cual se obtienen la activación y la salida del elemento de procesamiento.

$$e = \sum_{i=1}^{n} \omega_i I_i - u$$

$$a = f(e)$$

$$o_n = g(a, o_{n-1})$$

Las aristas incidentes a un elemento de procesamiento de una red neuronal feed-forward están etiquetadas con los "pesos de conexión", que son las constantes que aparecen en la expresión anterior para el cálculo de la excitación. La constante u es característica del elemento de procesamiento y se conoce como *umbral.*

La especificación completa de la naturaleza de los elementos de procesamiento de una red neuronal feed-forward se lleva a cabo mediante la especificación de las funciones f y g. Normalmente se hace $e = a$ y $o_n = S(a)$, donde S es una función con pendiente positiva en toda la recta real, que varía entre límites finitos, por ejemplo:

$$S(x) = tanh(x)$$

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

También se llega a utilizar la función $S(x) = \begin{cases} x < 0 \rightarrow 0 \\ x > 0 \rightarrow 1 \end{cases}$, convirtiendo a la neurona en

un sistema con entradas analógicas y salidas digitales.

Las redes neuronales feed-forward se han utilizado en los campos de reconocimiento de patrones, compresión de información, regresión no lineal y otros. En cualquiera de los casos se requiere un esquema de codificación de la información. Por ejemplo, para el reconocimiento de patrones, cada uno de los elementos de procesamiento de la capa de entrada admite una característica del sistema que se pretende identificar y, al propagarse la información hacia adelante, uno de los elementos de procesamiento se pone a nivel lógico alto, con los demás a nivel lógico bajo, proporcionando así un índice asociado a una categoría especifica de objetos.

La configuración de una red neuronal feed-forward para la solución de un problema, dadas las funciones de transferencia de los elementos de procesamiento, requiere de la elección adecuada de los pesos y umbrales. A diferencia de lo que ocurre con la máquina de Von Newman, no existen, o no se conocen, métodos para la elección de los pesos para aplicaciones específicas, por lo que la programación de una red neuronal se lleva a cabo mediante un proceso de adaptación, conocido como entrenamiento, en el que las características del sistema se modifican paulatinamente con un procedimiento sistemático, hasta alcanzar un sistema cuyo comportamiento sea lo más cercano posible al comportamiento deseado.

Existen fundamentalmente dos tipos de aprendizaje: *supervisado y no supervisado*. Durante el aprendizaje supervisado se presentan al sistema ejemplos del planteamiento del problema junto con la solución correspondiente; los parámetros del sistema se ajustan entonces para disminuir el valor de una función de la configuración de la red

(definida en el conjunto - ordenado - de pesos y umbrales) y de las variables de entrada y salida cuyo valor más o menos grande es una medida de la adaptación del sistema para la solución de las instancias del problema que se presentan al sistema durante el entrenamiento.

Durante el aprendizaje no supervisado, las características del sistema se modifican aplicando reglas generales heurísticas, sin tener en cuenta las características específicas del universo en que se generan los patrones.

2.1. Back-Propagation

La estrategia más utilizada para el entrenamiento de redes neuronales feed-forward es el algoritmo conocido como back-propagation (BP), que es un procedimiento de aprendizaje supervisado que se basa en la disminución del error cuadrático total a la salida de la red.

Durante el aprendizaje, se ponen patrones en la capa de entrada, propagando la información hacia la capa de salida, lo cual permite obtener una señal de error para cada elemento de procesamiento en la última capa. Las señales de error resultantes se transmiten hacia atrás, a partir de la capa de salida, capa por capa, para obtener las señales de error correspondientes a los elementos de procesamiento de las capas intermedias, hasta la capa de entrada. Con base en las señales de error, se actualizan entonces los pesos de conexión de tal suerte que, a la larga, la red converja hacia una configuración en la que estén codificados todos los patrones del conjunto de entrenamiento.

A medida que se entrena la red, las neuronas de las capas intermedias, se organizan de tal suerte que una de ellas (o un grupo de ellas) "aprende" a reconocer una característica específica del espacio de patrones de entrada.

Por otro lado, el hecho de que el algoritmo de entrenamiento se aplique en forma sistemática y reiterada, realizando adiciones de variables estocásticas, permite que BP sea aplicable al examen de patrones o datos contaminados por ruido.

2.2. Descripción de back-propagation

Sea $X = (X_{P1}, X_{P2}, \ldots, X_{Pn})$ el conjunto de patrones seleccionados para el entrenamiento de la red feed-forward. Definimos la entrada neta de la *j-ésima* unidad oculta como:

$$neta_{pj}{}^{h} = \sum_{i=1} W_{ji}{}^{h} X_{pi} + \theta_{j}{}^{h}$$

De tal suerte que $W_{ji}{}^{h}$ es el peso de la conexión procedente de la *i-ésima* unidad de entrada, $\theta_{j}{}^{h}$ es el umbral de la *j-ésima* unidad oculta y h es el índice de la capa oculta.

Dado que la activación de un nodo es igual a la entrada neta, la salida del *j-ésimo* nodo estará definida por la siguiente ecuación:

$$I_{pj} = f_{j}{}^{h}(neta_{pj}{}^{h})$$

De igual forma ocurre para cada una de las unidades de la capa de salida, descritas

por las siguientes ecuaciones:

$$neta_{pk}{}^{o} = \sum_{j=1}^{I} W_{kj}{}^{o} I_{pj} + \theta_{k}{}^{o}$$

$$O_{pk} = f_{k}{}^{o}(neta_{pk}{}^{o})$$

en donde el significado de las variables es similar al definido anteriormente, salvo que el índice utilizado caracteriza las ecuaciones como resultados de la capa de salida, es decir $W_{kj}{}^{o}$ es el peso de la conexión procedente de la *k-ésima* unidad de entrada, $\theta_{k}{}^{o}$ es el umbral de la *k-ésima* unidad y o es el índice de la capa de salida.

2.3. Procedimiento Para Entrenar la Red

i) Se aplica un patrón de entrada y se calculan los correspondientes vectores de salida.

ii) Se comparan las salidas obtenidas con las salidas deseadas, y se determina el error de salida.

iii) Se determina la dirección en que deberán cambiarse los pesos con el objeto de reducir el error.

iv) Se determina la cantidad en que es necesario modificar el error.

v) Se propagan los errores a cada una de las unidades de la red que han participado en la propagación.

vi) Repetir desde i) hasta v) con todos los patrones de entrenamiento hasta que el error para todos los patrones del conjunto de datos quede reducido a una valor

aceptable[1]

2.4. Actualización de pesos de la capa oculta

Se define $\delta_{pk} = (y_{pk} - o_{pk})$ como el error en la capa de salida (definido para patrones individuales o para el conjunto de patrones), donde p es el patrón a la entrada de la red, k es la *k-ésima* unidad de salida, y_{pk} es el valor que se obtiene al aplicar el patrón correspondiente en la entrada de la red neuronal y o_{pk} es el valor de salida esperado de la red.

Con los valores de salida de la red, -valor deseado y valor real- estamos en capacidad de calcular el error de la red, como la suma aritmética de todos los errores en las unidades de salida. Tomando en cuenta la consideración básica de back-propagation, como método de entrenamiento de redes feed-forward, se pretenderá minimizar el error medio cuadrático de la red. El error medio cuadrático está descrito por las siguientes ecuaciones:

$$E_p = \frac{1}{2}\sum_{k=1}^{M} \delta_{pk}{}^2$$

$$E_p = \frac{1}{2}\sum_{k=1}^{M} (y_{pk} - o_{pk})^2 \; ; \; o_{pk} = f_k{}^o(neta_{pk}{}^o)$$

$$\frac{\partial E_p}{\partial W_{kj}{}^o} = -(y_{pk} - o_{pk}) \frac{\partial f_k{}^o}{\partial (neta_{pk}{}^o)} \frac{\partial (neta_{pk}{}^o)}{\partial W_{kj}{}^o},$$

---

[1] Valor definido por el usuario.

aplicando la regla de la cadena se obtiene:

$$\frac{\partial(neta_{pk}{}^{0})}{\partial W_{kj}{}^{o}} = \frac{\partial}{\partial W_{kj}{}^{o}}(\sum W_{kj}{}^{o} I_{pj} + \theta_{k}{}^{o}) = I_{pj}$$

combinando las dos ecuaciones anteriores obtenemos:

$$-\frac{\partial E_{p}}{\partial W_{kj}{}^{o}} = (y_{pk} - o_{pk}) f_{k}{}^{o'}(neta_{pk}{}^{o}) I_{pj}$$

de tal forma que la ecuación para la modificación de pesos estará definida por:

$$W_{kj}{}^{o}(t+1) = W_{kj}{}^{o}(t) + \Delta_{p} W_{kj}{}^{o}(t)$$

$$\Delta_{p} W_{kj}{}^{o}(t) = \eta(y_{pk} - o_{pk}) f_{k}{}^{o'}(neta_{pk}{}^{o}) I_{pj}$$

donde η es el *coeficiente de aprendizaje o factor de convergencia* de la red. Por resultados encontrados[1] se recomiendan valores $0 < \eta < 1$. Para que tengan validez las expresiones anteriores se debe cumplir que $f_{k}{}^{o'}$ sea derivable. Como resultado de la experiencia recomienda la literatura existente en el área considerar $f_{k}{}^{o}(neta_{pk}{}^{o}) = \frac{1}{(1+e^{-neta_{pk}{}^{o}})}$, que es la función sigmoidal, que cumple las características de *función biestable y derivable.*

Derivando la función sigmoidal obtenemos:

$$f_{k}{}^{o'} = f_{k}{}^{o}(1 - f_{k}{}^{o}) = o_{pk}(1 - o_{pk}).$$

Reemplazando el resultado anterior, en la ecuación de pesos, obtenemos finalmente:

$$W_{kj}^{o}(t+1) = W_{kj}^{o}(t) + \eta(y_{pk} - o_{pk})o_{pk}(1 - o_{pk})I_{pj}$$

$$\delta_{pk}^{o} = (y_{pk} - o_{pk})f_k^{o'}(neta_{pk}^{o}) = \delta_{pk}f_k^{o'}(neta_{pk}^{o})$$

$$W_{kj}^{o}(t+1) = W_{kj}^{o}(t) + \eta\delta_{pk}^{o}I_{pj}$$

Podemos adicionar un *término de momento*, que permitirá a la red recordar los resultados de los cálculos anteriores, con el fin de mantener los cambios de peso en la misma dirección. El término de momento adicionado permitirá a la red mejorar su velocidad de convergencia, respecto al método tradicional.

$$W_{kj}^{o}(t+1) = W_{kj}^{o}(t) + \eta\delta_{pk}^{o}I_{pj} + \alpha\Delta pW_{kj}^{o}(t-1)$$

2.5. Observación Interesante

Si estuviésemos interesados en que la regla delta generalizada fuese análoga al método de los mínimos cuadrados, *no cambiaríamos ninguno de los valores de los pesos hasta que se presenten a la red todos los patrones de entrenamiento al menos una vez.* Para ello acumulamos los cambios a medida que sea procesado cada patrón, los sumamos y hacemos la actualización de pesos al final del proceso. Repetimos el procedimiento hasta que el error sea aceptable.

2.6. Actualización de Pesos de la Capa Oculta

Ya obtenido el error para cada elemento de procesamiento de la red neuronal, se deben propagar los errores a cada una de las unidades de la red para la capa oculta,

pero surge un problema, pues no tenemos forma de conocer por anticipado cuál debería ser la salida esperada de estas unidades ocultas. De acuerdo a las ecuaciones de nuestra red, tenemos definida las salidas de esta capa en la forma:

$$E_p = \frac{1}{2}\sum_{k=1}^{P}(y_{pk} - o_{pk})^2$$

$$E_p = \frac{1}{2}\sum_{k=1}^{P}(y_{pk} - f_k{}^o(neta_{pk}{}^o))^2$$

$$E_p = \frac{1}{2}\sum_{k=1}^{P}(y_{pk} - f_k{}^o(\sum_j W_{kj}{}^o I_{pj} + \theta_k{}^o))^2$$

donde I_{pj} es función de los pesos de las capas ocultas y de acuerdo a las definiciones realizadas en la primera parte :

$$neta_{pj}{}^h = \sum_{i=1}^{n} W_{ji}{}^h X_{pi} + \theta_j{}^h$$

$$I_{pj} = f_j{}^h(neta_{pj}{}^h)$$

Se obtiene:

$$\frac{\partial E_p}{\partial W_{ij}{}^h} = \frac{1}{2}\sum_k \frac{\partial}{\partial W_{ji}{}^h}(y_{pk} - o_{pk})^2 = -\sum_k (y_{pk} - o_{pk})\frac{\partial o_{pk}}{\partial(neta_{pk}{}^o)}\frac{\partial(neta_{pk}{}^o)}{\partial I_{pj}}\frac{\partial I_{pj}}{\partial(neta_{pj}{}^h)}\frac{\partial(neta_{pj}}{}$$

$$\frac{\partial E_p}{\partial W_{ij}{}^h} = -\sum_k (y_{pk} - o_{pk})\frac{\partial o_{pk}}{\partial(neta_{pk}{}^o)}\frac{\partial(neta_{pk}{}^o)}{\partial I_{pj}}\frac{\partial I_{pj}}{\partial(neta_{pj}{}^h)}\frac{\partial(neta_{pj}{}^h)}{\partial W_{ji}{}^h}$$

$$\frac{\partial E_p}{\partial W_{ij}{}^h} = -\sum_k (y_{pk} - o_{pk}) f_k{}^{o'}(neta_{pk}{}^o) W_{kj}{}^o f_j{}^{h'}(neta_{pj}{}^h) X_{pi}$$

Actualizando los pesos de la capa oculta proporcionalmente al valor negativo de la ecuación anterior obtenemos:

$$\Delta_p W_{ji}{}^h = \eta f_j{}^{h'}(neta_{pj}{}^h) X_{pi} \sum \delta_{pk}{}^o W_{kj}{}^o$$

de aquí podemos observar que todas las actualizaciones de los pesos de la capa oculta dependen de todos los términos de error, $\delta_{pk}{}^o$, de la capa de salida. Este resultado es consecuencia de la noción de la propagación hacía atrás, donde:

$$\delta_{pj}{}^h = f_j{}^{h'}(neta_{pj}{}^h) X_{pi} \sum_k \delta_{pj}{}^o W_{kj}{}^o$$

$$W_{ji}{}^h(t+1) = W_{ji}{}^h(t) + \eta \delta_{pj}{}^h X_i$$

2.7. Resumen del Algoritmo Back-Propagation

[Propagación hacía adelante]

i) Se aplica el patrón de entrada $X_p = (X_{p1}, X_{p2}, \ldots, X_{pn})$ a las unidades de entrada.

ii) Se calculan los valores netos procedentes de las entradas para las unidades de la capa oculta:

$$neta_{pj}{}^h = \sum_{i=1}^{n} W_{ji}{}^h X_{pi} + \theta_j{}^h$$

iii) Se calculan las salidas de la capa oculta:

$$I_{pj} = f_j{}^h(neta_{pj}{}^h)$$

iv) Se pasa a la capa de salida y se calculan los valores netos de las entradas para cada unidad:

$$neta_{pk}{}^{o} = \sum_{j=1}^{I} W_{kj}{}^{o} I_{pj} + \theta_{k}{}^{o}$$

v) Se calculan las salidas:

$$O_{pk} = f_{k}{}^{o}(neta_{pk}{}^{o})$$

[Propagación hacía atrás]

vi) Se calculan los términos de error para las unidades de salida:

$$\delta_{pk}{}^{o} = (y_{pk} - o_{pk}) f_{k}{}^{o'}(neta_{pk}{}^{o}) = \delta_{pk} f_{k}{}^{o'}(neta_{pk}{}^{o})$$

vii) Se calculan los términos de error para las unidades ocultas:

$$\delta_{pj}{}^{h} = f_{j}{}^{h'}(neta_{pj}{}^{h}) \sum_{k} \delta_{pj}{}^{o} W_{kj}{}^{o}$$

Es importante anotar que los términos de error se calculan antes de que hayan sido actualizados los pesos de conexión con las unidades de la capa de salida.

viii) Actualización de pesos de la capa de salida:

$$W_{kj}{}^{o}(t+1) = W_{kj}{}^{o}(t) + \eta \delta_{pk}{}^{o} I_{pj}$$

ix) Se actualizan los pesos de la capa oculta:

$$W_{ji}{}^{h}(t+1) = W_{ji}{}^{h}(t) + \eta\delta_{pj}{}^{h}X_{pi}$$

Nota: este algoritmo termina cuando el error es aceptable, es decir $E_p = \sum_{k=1}^{M} \delta_{pk}{}^{2}$ se considera un valor aceptable, pues esta es la medida del aprendizaje de la red.

2.8. Consideraciones Prácticas

Si se está probando una red en un entorno con ruido, entonces hay que incluir unos cuantos patrones de entrada contaminados en el conjunto de datos de entrada, con el fin de ayudar a la red a acelerar su velocidad de convergencia en la fase de aprendizaje.

Durante el proceso de entrenamiento, es recomendable seleccionar aleatoriamente los patrones de entrenamiento. No se deberá entrenar una red con patrones de una sola clase, y luego tomar patrones de otra clase, con el fin de que la red no olvide en cada fase de entrenamiento los datos de la fase anterior.

2.9. Dimensionamiento de la Red

Cuántas capas son necesarias, cuántas neuronas por capa se necesitan, qué valores iniciales hay que asignar a los pesos de la red y otras más, son preguntas que difícilmente tendrán una respuesta concreta. En general, podemos decir que tres capas son suficientes y el tamaño de neuronas de la capa de entrada y de la capa de salida suelen estar condicionados generalmente por el tipo de problema a analizar.

Determinar el numero de unidades de la capa oculta no es tan evidente. La idea principal consiste en utilizar el menor número de nodos o neuronas, debido a que

cada neurona implica conexiones y un aumento exponencial en la carga computacional. Podemos aplicar el método desarrollado por Rumelhart para la eliminación de neuronas innecesarias en el proceso, verificando aquellas conexiones que varían en proporciones muy pequeñas y que no contribuyan de manera sustancial en el aprendizaje de la red, utilizando para ello una revisión periódica de la red.

2.10. Pesos y Parámetros de Aprendizaje

Un factor de aprendizaje muy pequeño representa una asignación de pequeños pasos a través de la red, lo que implica que deben realizarse muchas iteraciones antes de que la red converja.

Puede ser posible aumentar el factor de aprendizaje a medida que la red converge, a medida que el error de la red disminuye. La asignación de valores muy grandes del factor de aprendizaje implica que la red en un instante de tiempo alcance un mínimo y pueda rebotar, alejándose del valor mínimo verdadero.

El coeficiente o factor de aprendizaje deberá pertenecer al intervalo $0.05 < \eta < 0.25$ para que la red converja de manera apropiada[2]. El valor de momento α es otra forma de acelerar la velocidad de convergencia, que consiste en adicionar un valor inercial, que es un porcentaje del cambio de pesos $\Delta_p W$ en la iteración anterior. Este valor tiende a mantener los cambios de peso en la misma dirección.

---

[2] Chavoya O. Conferencia Internacional de Inteligencia Artificial. Bucaramanga, Colombia.

[1] Chavoya, Oscar. Seminario Internacional de Inteligencia Artificial. Visión y reconocimiento de voz mediante redes neuronales y programación genética. Agosto 28-29 de 1995.

presentar un paralelismo funcional (descomposición de un problema en tareas distribuidas en múltiples procesadores para su ejecución simultánea), un paralelismo de datos (partición de un problema en el dominio de datos y distribuir partes de datos en múltiples procesadores para su ejecución concurrente) o una combinación de ambos.

El modelo de la máquina de Von Newman no ha cambiado, es decir, el modelo de Von Newman mantiene activo el cuello de botella entre la memoria y la CPU. Para dar solución a ello, se han planteado variantes del modelo convencional que implementan diferentes grados de paralelismo.

Las redes neuronales son por naturaleza de arquitectura paralela, es decir, podemos definir las redes neuronales como colecciones de procesadores interconectados en forma de un grafo dirigido con las aristas y los vértices etiquetados con la particularidad de efectuar algunos cálculos de manera paralela, es decir, con independencia en el flujo de datos y en forma concurrente.

3.1. Paralelismo en Sistemas Monoprocesadores[2]

Aun cuando en un sistema monoprocesador existe una sola CPU podemos hablar de paralelismo en este tipo de sistemas y alcanzar un alto grado de compartición de recursos. Se trata de planteamientos software para obtener concurrencia en sistemas monoprocesadores.

Dentro del mismo intervalo de tiempo pueden coexistir múltiples procesos activos compitiendo por los recursos de memoria, E/S y CPU (tiempo compartido).

Es importante anotar que se han desarrollado toda una serie de mecanismos de procesamiento paralelo en computadores monoprocesadores, que podemos clasificar básicamente en seis categorías:

- *Multiplicidad de unidades funcionales:* ocurre cuando un sistema computacional tiene más de una unidad funcional en su CPU y muchas de las funciones de la unidad aritmético-lógica (UAL) pueden estar distribuidas sobre múltiples unidades funcionales, independientes entre sí y que operan en forma concurrente. Un ejemplo de esta categoría de máquinas es la IBM 360/91 que poseía dos unidades de ejecución en paralelo, una para aritmética de punto fijo y otra de punto flotante. La unidad de punto flotante estaba compuesta de dos unidades funcionales: una para suma-resta y otra para multiplicación-división.

- *Paralelismo y segmentación encauzada (solapamiento) dentro de la CPU:* consiste en subdividir el proceso de entrada en una secuencia de subtareas, cada una de las cuales deberá ser ejecutada por una etapa de hardware especializada, que actúe en concurrencia con otras etapas del encauzamiento. Por ejemplo las diferentes fases de ejecución de las instrucciones se dividen en etapas entre las que se incluyen la *extracción de la instrucción, su decodificación, la extracción del operando, la ejecución aritmético-lógica y el almacenamiento del resultado.* Las etapas se conectan en cascada en forma de cauce, de donde deriva su nombre.

- *Solapamiento de las operaciones de la CPU y de entrada-salida:* ocurre cuando las operaciones de E/S pueden ejecutarse en forma simultánea con los cálculos de la

CPU, utilizando para ello canales de E/S en forma separada. El canal de acceso directo a memoria (DMA) constituye un buen ejemplo, ya que permite la transferencia directa de información entre los dispositivos de E/S y la memoria principal.

- *Uso de un sistema jerárquico de memoria:* su objetivo es disminuir el tiempo de acceso a la memoria, comparado con la duración de los ciclos de la CPU. Para ello se organiza la memoria en forma jerárquica (memoria secundaria - memoria virtual - memoria principal - caché - registros).

- *Equilibrio de los anchos de banda de los subsistemas:* es el procedimiento para solucionar de manera parcial la diferencia entre los tiempos del procesador, la memoria principal y los dispositivos de E/S. Por ejemplo, la diferencia de velocidades entre la CPU y la memoria principal, puede disminuirse insertando, entre ambas, una memoria caché.

- *Multiprogramación y tiempo compartido:* aun cuando en un sistema monoprocesador sólo existe una CPU, se puede alcanzar un alto grado de partición de recursos, entre diversos programas. La multiprogramación ocurre cuando en el mismo intervalo de tiempo pueden coexistir múltiples procesos activos, compitiendo por los recursos del sistema, como memoria, dispositivos de E/S o CPU, de forma que los procesos puedan mezclarse, siempre que se respeten las fronteras de cada tarea y los procesos del sistema. El tiempo compartido, es una extensión de la multiprogramación que asigna intervalos de tiempo (fijos o variables) a múltiples

programas, proporcionando igual oportunidad a todos los programas para el uso de la CPU.

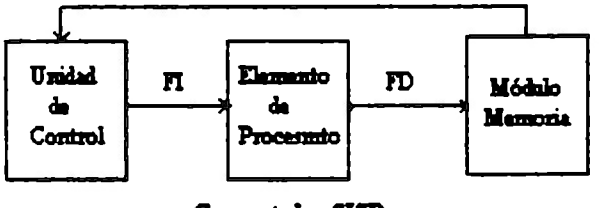
3.2. Esquemas de Clasificación de Arquitecturas

Dentro del área de máquinas paralelas encontramos la clasificación de Flynn[3] como la más conocida y aceptada, la cual se basa en la multiplicidad de los flujos de instrucciones y de datos en un computador. El proceso de computación esencial es la ejecución de una secuencia de instrucciones sobre un conjunto de datos. El término *flujo* se emplea para denotar una secuencia de elementos (instrucciones o datos) que ejecuta un procesador.

Flynn, propone cuatro esquemas para la clasificación de las arquitecturas paralelas:

3.2.1. Organización SISD (Single Instruction - Single Data)

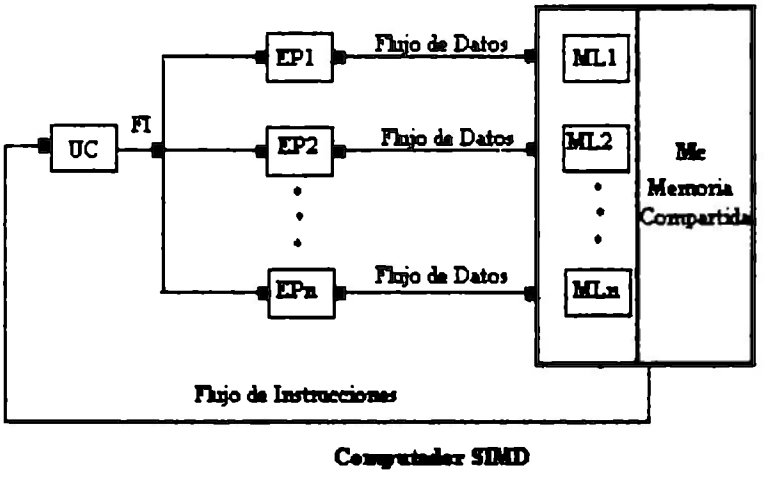
Representa a la mayoría de las computadores serie disponibles hoy día. Las instrucciones se ejecutan secuencialmente pero pueden estar solapadas en las etapas de ejecución (segmentación encauzada). Un computador con arquitectura SISD puede tener más de una unidad funcional y todas ellas están bajo la supervisión de una sola unidad de control. Cada instrucción opera sobre un elemento o dato por ciclo de reloj.



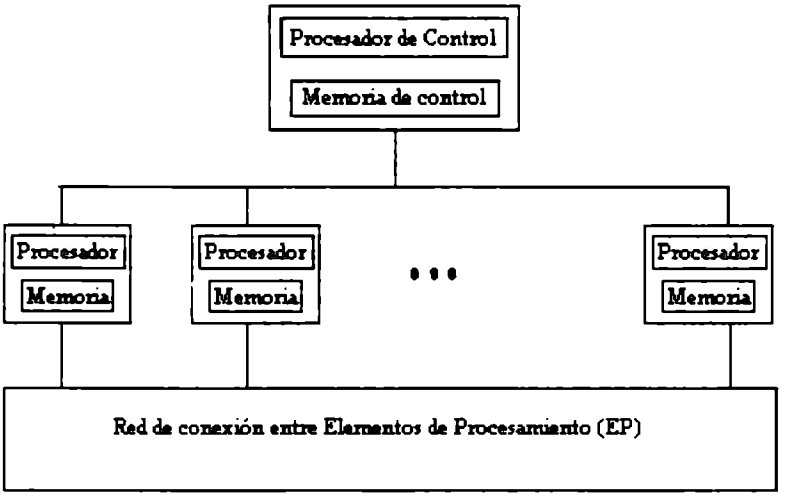
Computador SISD

3.2.2. Organización SIMD (Single Instruction - Multiple Data)

Corresponde a los procesadores matriciales, en la cual existen múltiples elementos de procesamiento (EP) que se encuentran supervisados por la misma unidad de control. Todos los EP reciben la misma instrucción emitida por la unidad de control pero operan sobre diferentes conjuntos de datos procedentes de diferentes flujos. Cada instrucción puede operar sobre más de un elemento o dato por ciclo de reloj.



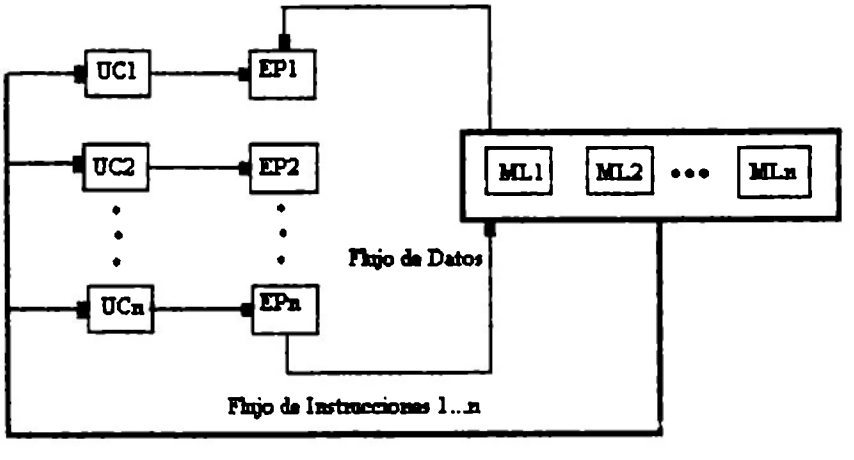
Los computadores matriciales son modelos de computadores paralelos *síncronos* con múltiples unidades aritmético-lógicas (UAL), que pueden operar en paralelo, en la modalidad "paso-bloqueo", para implementar un paralelismo espacial (procesadores vectoriales o arreglos de procesadores), como se puede ver en la figura :



Computadores Matriciales

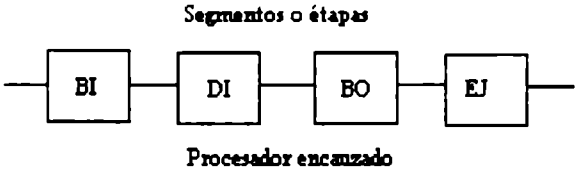
3.2.3. Organización MISD (Multiple Instruction - Single Data)

Existen *n* unidades procesadoras o elementos de procesamiento donde cada una recibe distintas instrucciones que operan sobre el mismo flujo de datos y sus derivados. Los resultados (salida) de un procesador pasan a ser la entrada (los operandos) del siguiente procesador en el *macrocauce*. Esta estructura no tiene ninguna materialización real en el mercado.



Computador MISD

La *segmentación encauzada* que aquí ocurre efectúa computaciones superpuestas (solapamiento) para explotar el paralelismo temporal. Normalmente la ejecución de una instrucción en un computador digital genera cuatro pasos básicos: *búsqueda de la instrucción desde la memoria principal (BI); decodificación de la instrucción para decodificar la operación a efectuar (DI); búsqueda del operando (BO) y ejecución de la operación aritmético-lógica decodificada (EJ)*, que en un modelo de segmentación encauzado se pueden ejecutar en modo solapado como se ilustra en las figuras:



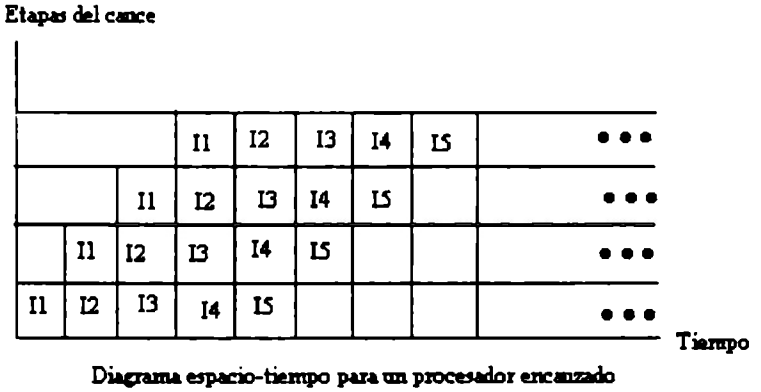


Diagrama espacio-tiempo para un procesador encauzado

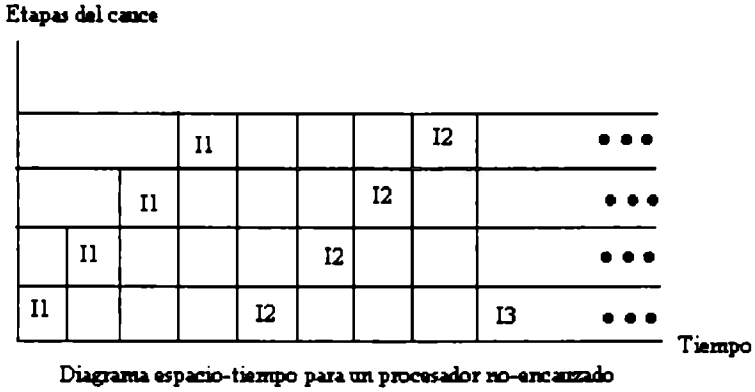
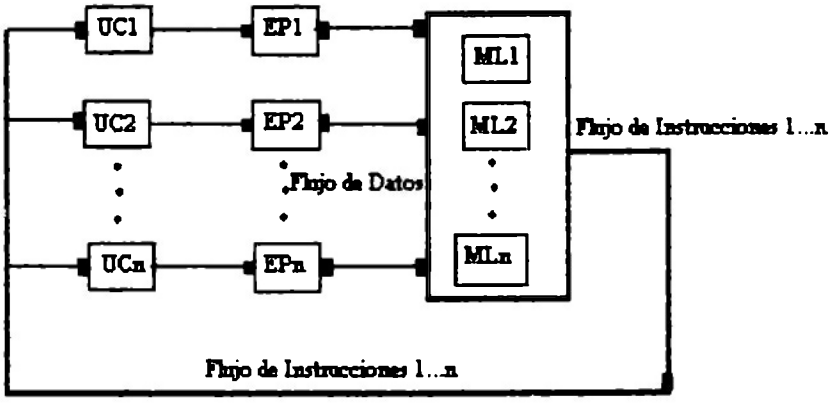


Diagrama espacio-tiempo para un procesador no-encauzado

3.2.4. Organización MIMD (Multiple Instruction - Multiple Data)

La mayoría de los sistemas multiprocesadores y de los sistemas con múltiples computadores son elementos de esta categoría. Un computador MIMD intrínseco implica interacciones entre los *n* procesadores porque todos los flujos de memoria se derivan del mismo espacio de datos compartidos por todos los procesadores. Si los *n* flujos de datos provinieran de subespacios inconexos dentro de las memorias compartidas, entonces tendríamos la llamada operación SISD múltiple (MSISD), que no es más que un conjunto de *n* sistemas monoprocesadores SISD independientes interactuando. Un computador MIMD intrínseco es *fuertemente acoplado* si el grado de interacciones entre los procesadores es elevado; caso contrario los consideramos *débilmente acoplado*. La mayoría de los computadores MIMD comerciales son débilmente acoplados.

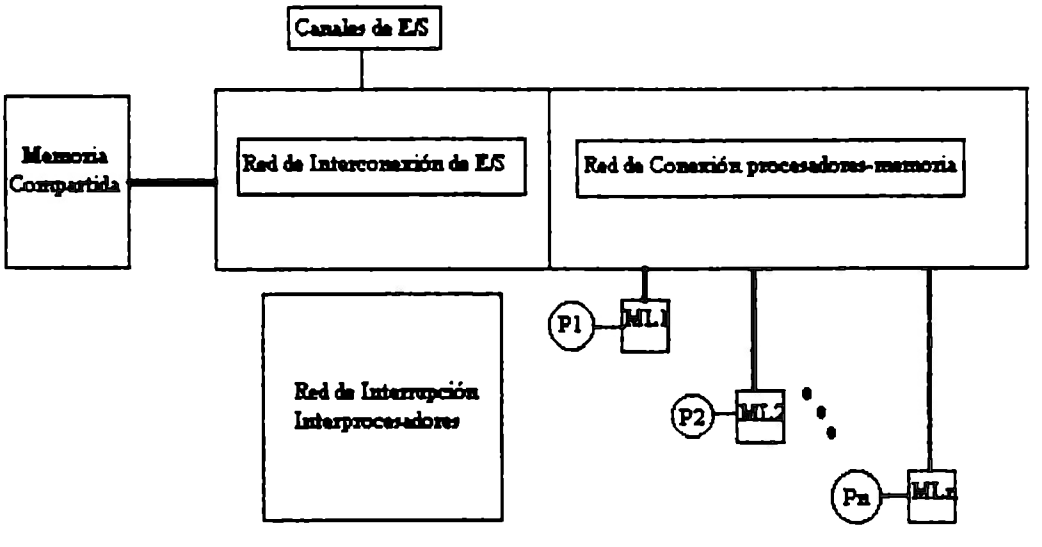


Computador MIMD

3.3. Sistemas multiprocesadores

Es un computador paralelo *asíncrono,* basado en un conjunto de procesadores interactivos que disponen de recursos compartidos. Un sistema multiprocesador tiene como objetivo mejorar la productividad de un sistema computacional, donde todos los procesadores pueden compartir acceso a grupos comunes de memoria, canales de E/S y dispositivos periféricos y/o disponer de recursos de memoria local y dispositivos de E/S independientes, donde las comunicaciones entre procesadores pueden realizarse a través de memorias compartidas o mediante una red de interrupción (paso de mensajes).

La organización de un sistema multiprocesador, está determinado básicamente por la estructura de interconexión entre las unidades de memoria y los elementos de procesamiento.



3.4. Rendimiento de los Computadores Paralelos

El incremento de velocidad que puede conseguir un computador con arquitectura paralela con n procesadores idénticos trabajando en forma concurrente en un único problema es como máximo n veces la velocidad de un sólo procesador.

Consideremos un problema de computación que pueda ser ejecutado por un sistema monoprocesador en una unidad de tiempo, $T_1 = 1$. Sea p_i la probabilidad de asignar el problema a i *procesadores* que trabajen equitativamente, con una carga promedio $d_i = \frac{1}{i}$ por procesador. Supongamos además equiprobabilidad de cada modo de operación que emplea i procesadores, es decir, $p_i = \frac{1}{n}$, para n modos de operación: $i = 1,2,\text{K},n$. El tiempo medio necesario para resolver el problema en un sistema de n procesadores está dado por la expresión:

$$T_n = \sum_{i=1}^{n} p_i . d_i = \frac{\sum_{i=1}^{n} \frac{1}{i}}{n}$$

La ganancia media de velocidad G se obtiene como razón entre T_1 y T_n, mediante la siguiente ecuación:

$$G = \frac{T_1}{T_n} = \frac{n}{\sum_{i=1}^{n} 1/i} \leq \frac{n}{\ln n}$$

3.5. Diseño y Consideraciones fundamentales

La programación paralela introduce nuevos conceptos no concernientes a la programación secuencial, tales como: ley de Amdalh, granularidad, dependencia de datos, comunicación, balance de carga, deadlocks o bloqueos entre otros.

La *ley de Amdalh* dice que la velocidad de un programa que se ejecute en N procesadores, se encuentra definida por una fracción del código que puede ser paralelizado $T_p(A_n)$ y una fracción serie $T_s(A_n)$ que permanece constante dentro del programa y que comúnmente se conoce como *Aceleración.*

$$S(A_n) = \frac{T_s(A_n)}{T_p(A_n)} = \frac{T_s(A_n)}{s.T_s(A_n) + \frac{p.T_s(A_n)}{N_p}} \leq \frac{N_p}{1+(N_p-1).s}$$

Es fácil ver de la relación anterior que a medida que el valor de s aumenta el valor de la aceleración tiende a cero $(p = 1 - s)$.

La *granularidad* está definida como el radio de variación entre los tiempos de computación y los tiempos de comunicación. Una granularidad muy fina conlleva a un overhead para sincronización y comunicación entre tareas, que puede tomar un tiempo mayor que el tiempo de computación.

La *dependencia de datos* ocurre cuando existen usos de datos dependientes de los datos en un programa, restringiendo la ejecución paralela de los programas. El siguiente fragmento de código debe ser ejecutado en serie independiente del tipo de máquina o arquitectura utilizada :

```
DO 10 I=INICIO,FINAL
A(J)=A(J-1) * factor
CONTINUE
```

El *tiempo de comunicación*, es una variable fundamental en procesos de paralelización de procesos. Para algunos problemas, incrementar el número de procesadores, genera como consecuencia disminuir el tiempo de computación; a la vez que se incrementan considerablemente las operaciones de comunicación interprocesadores.

El *balance de carga*, se refiere a la distribución de tareas en cada procesador. Si no hay una distribución balanceada es probable que los tiempos sean altos, debido a la existencia de procesos durmientes y existencia de tiempos muertos en algunos procesadores.

c'gla SP2?

3.6. Arquitectura de la IBM 9071 SP1

La IBM 9071 SP1 es una máquina paralela clasificada según el esquema de Flynn, como una máquina con arquitectura MIMD *(Multiple instructions - Multiple Data)* con las siguientes características de computo:

- La SP1 puede tener hasta 128 nodos, donde cada nodo es básicamente una máquina RS/6000 modelo 370. Este modelo tiene una velocidad reloj de 62.5 MHz y 64-KB de memoria caché -32KB para datos y 32KB para instrucciones-.

- 128 MBytes de memoria por nodo.

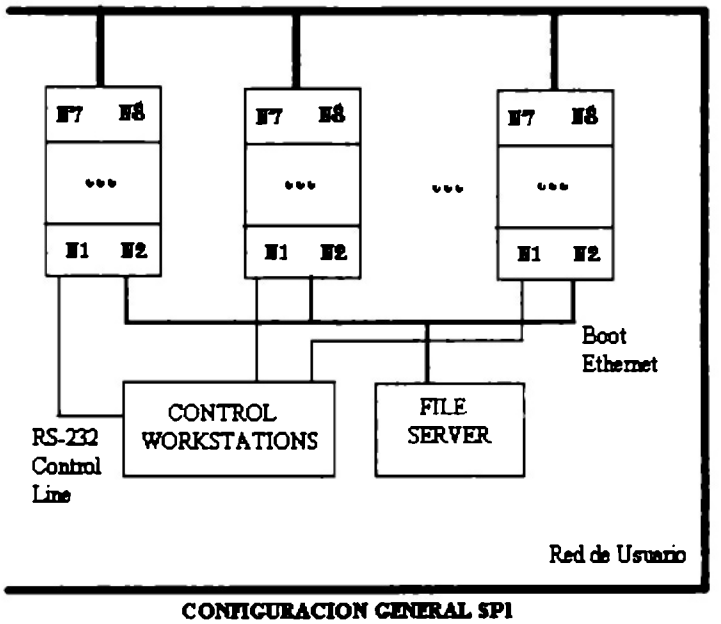
- 1 GByte de disco local en cada nodo (400 MBytes disponibles para el usuario y el restante para paginamiento)

- Uso de sistema operativo Unix en cada nodo (IBM AIX 3.2.4)

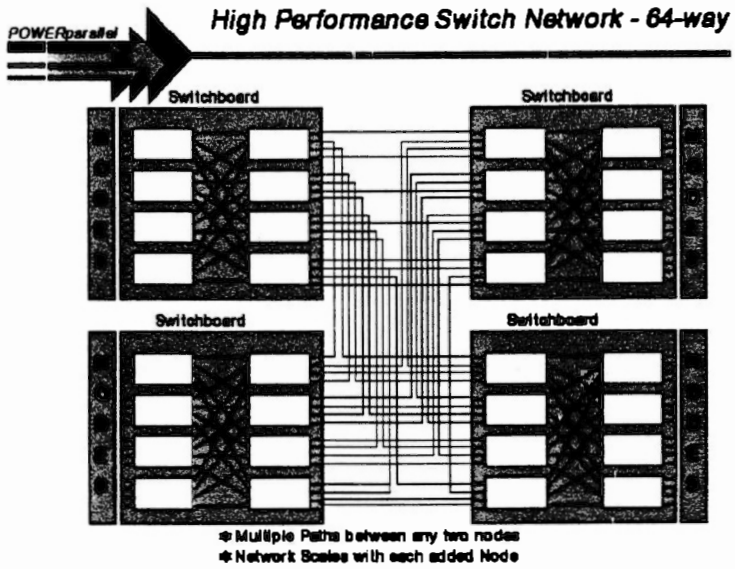
- Cada nodo es accesible vía Ethernet.

- Ejecución máxima de cada nodo de 125 Mflops.

Podemos presentar la arquitectura de la SP1 bajo los dos esquemas siguientes :



CONFIGURACION GENERAL SP1



---

[1] Ver modelos computacionales de *flujos de control* más adelante.

[2] Arquitectura de Computadoras y Procesamiento Paralelo., Hwang Kai. Briggs A. Fayé.

[3] Michael J. Flynn. 1966.

Capítulo 4

Planteamiento del Problema

Con el presente trabajo de investigación se pretenden proponer modelos que permitan diseñar métodos de entrenamiento en máquinas con arquitectura paralela que permitan disminuir el tiempo (tiempo absoluto) del proceso de aprendizaje para redes feed-forward, usando como estrategia de entrenamiento back-propagation.

Los métodos que se propongan serán comparados con Back-Propagation (BP) en su forma normal.

4.1. ¿En qué sentido podemos hablar de tiempo en relación con BP?:

Puesto que los parámetros de la red se inicializan al azar[1]*, el tiempo de entrenamiento, dado un problema fijo (es decir, un conjunto de entrenamiento y una arquitectura específica) es una* ***variable estocástica****. Por ello, la comparación de la complejidad computacional de BP con la de cualquier otro algoritmo debe basarse en la consideración de propiedades estadísticas:* ***valor esperado*** *y* ***varianza*** *(esencialmente).*

---

[1] Y los patrones se presentan al azar

Esto indica que, para la evaluación, es necesario obtener una muestra representativa de corridas, a partir de la cual sea posible *estimar la distribución de probabilidad* asociada al tiempo de entrenamiento. Haciendo esto se obtendrá una escala de tiempo asociada a back-propagation, en relación con la cual se deben computar los tiempos asociados a otras estrategias de entrenamiento.

4.2. Consideración básica.

Si existen N elementos de procesamiento, la probabilidad de que uno de ellos (arbitrario pero fijo) sea seleccionado en un instante de tiempo dado es $P_{(a)} = \frac{1}{N}$, por lo que la probabilidad de que éste no sea seleccionado en m intentos es $P_{(a)}^{\bullet} = (1 - \frac{1}{N})^m$, que se puede hacer razonablemente pequeña, tomando un valor de ***m suficientemente grande***. Si hacemos $m = N$ tenemos que la probabilidad de que el elemento a no sea seleccionado en m intentos será $P_{(a)}^{\bullet} = (1 - \frac{1}{N})^N \cong \frac{1}{e} < \frac{1}{2}$ para un ***valor moderado de N.***

La consideración anterior $P_{(a)}^{\bullet} = (1 - \frac{1}{N})^N \cong \frac{1}{e} < \frac{1}{2}$ se puede explicar de acuerdo a definiciones existentes[2].

Por definición conocemos que[2]:

---

[2] Ver demostración en Análisis Matemático de Spivack.

$$\lim_{n\to\infty}(1+\frac{1}{M})^M \equiv e$$

de esta forma podemos asegurar que se cumple para todo elemento a, que:

$$P_{(a)}^{*} = (1-\frac{1}{N})^N \cong \frac{1}{e} < \frac{1}{2}$$

fácilmente podemos encontrar un valor de N que satisfaga dicha condición para muestras pequeñas con un valor de probabilidad muy pequeña.

4.3. Algoritmo planteado.

Sea n el número máximo de iteraciones, $k = 0$ y $v = n_p N$, donde N es un entero positivo suficientemente grande tal que la probabilidad $\frac{1}{2^N}$ pueda considerarse razonablemente pequeña[3].

Este procedimiento se utilizará para el análisis estadístico del tiempo de entrenamiento y nótese que *el método propuesto es válido con independencia del algoritmo de entrenamiento*, razón que permite utilizar este algoritmo para todos los métodos de entrenamiento propuestos.

Teniendo en cuenta las consideraciones anteriores podemos expresar el segmento principal del programa en seudocódigo:

---

[3] Donde n_p es el número de patrones en el conjunto de entrenamiento y N es un entero positivo suficientemente grande para que la probabilidad $\frac{1}{2^N}$ pueda considerarse razonablemente pequeña .

```
Repetir el siguiente proceso para un número de veces igual al tamaño
deseado de la muestra
{
for (i = 0 ; i < n ; ++i)
  {
    Escoger un patrón al azar;
    Propagar la información hacia adelante;
    if( Error < ε )
        {
        if ( k == v)
                terminar escribiendo i+1
        else
                ++k;
        }
    else
        k=0;
    Aplicar el algoritmo de entrenamiento;
  }
}
```

Para continuar se trata de *evaluar modelos alternativos, sobre arquitecturas paralelas*. El más simple se basa en correr back-propagation en cada uno de los procesadores, reportando el resultado que primero se obtenga. *En el tiempo absoluto este procedimiento es mejor que BP aplicado en un solo procesador, pues es claro que, para toda variable estocástica X se cumple:*

$$\langle min\{X_1, \Lambda\ , X_k\}\rangle \leq \langle X\rangle$$

donde $X_1, \Lambda\ , X_k$ son variables estocásticas idénticamente distribuidas, con la misma distribución que la variable estocástica X. La evaluación de este modelo se puede hacer en serie repitiendo los experimentos anteriores, agrupando los resultados en grupos consecutivos y *reportando el mínimo valor de tiempo para cada grupo, como*

resultado individual [4]. ***Cualquier otra estrategia que se proponga no solamente debe ser mejor que Back-Propagation (BP), sino mejor que la anterior.***

4.4. Propuestas de Solución y Metodologías de Análisis

Se conoce que las redes neuronales feed-forward son por naturaleza altamente paralelas, ya que sus elementos de procesamiento se pueden considerar como procesadores que efectúan cálculos de manera independiente y procesos de comunicación entre ellos. Es por esto que paralelizar procesos de entrenamiento sobre redes feed-forward presenta diversas alternativas de solución, desde la más sencilla, que consiste en plantear un algoritmo de procesamiento explícito a nivel de programación, hasta lograr un procesamiento sobre dominios espaciales, aritméticos o de datos, entre otros.

Se pueden mencionar dos métodos inherentes de paralelismo para redes neuronales feed-forward. La primera es usando ***paralelismo espacial***, el cual consiste en efectuar particiones de las operaciones requeridas por cada capa en cada uno de los N procesadores existentes. El paralelismo espacial puede ser clasificado en tres niveles de acuerdo al grado[5] :

i) Particionar las neuronas en cada capa por grupos.

---

[4] Todo método propuesto deberá ser mejor que este método el cual parece muy sencillo pero sus estadísticos (media y varianza) son mejores que BP en su forma normal.

[5] Exploiting the inherent parallelism of back-propagation neural networks to design a systolic array. Jai-Hoon Chung, Hyunsoo Yoon, and Seung Ryoul Maeng. Korea Advanced Institute of Science and Technology (KAIST).

ii) Particionar las operaciones de cada neurona en sumas de productos.

iii) Particionar las sumas de productos en varios grupos.

De los trabajos existentes, se encuentra que las simulaciones de redes neuronales efectuadas sobre máquinas MIMD, explotan las alternativas de paralelismo i) y ii). La alternativa iii) no es utilizada debido a los altos costos de overhead incurridos en las comunicaciones.

El segundo tipo de paralelismo es el ***paralelismo de patrones,*** que puede ser a su vez clasificado en dos niveles :

i) Particionar el conjunto de patrones de entrenamiento y ejecución independiente de ellos en múltiples procesadores.

ii) Pipelining de los patrones de entrenamiento, permitiendo que algunos de los procesadores que finalizaron el entrenamiento de los patrones iniciales, empiecen nuevamente la fase de entrenamiento con un nuevo conjunto de patrones, mientras que otros procesadores ejecutan cálculos con los resultados anteriores.

Nuestro propósito es presentar un algoritmo back-propagation (BP) distribuido que presente las características de correr en una máquina con arquitectura paralela (múltiples procesadores), memoria distribuida y comunicación de datos a través de paso de mensajes, el cual es el caso de las máquinas paralelas comerciales más comunes, como es el caso de la SP1, SP2 o redes de Transputers.

Se pueden buscar métodos de paralelización de redes feed-forward fundamentados

para el análisis del mejor método en el tiempo de entrenamiento mínimo, bajo consideración de propiedades estadísticas como el ***valor esperado y la varianza***. Para ello se definen diferentes dominios para la paralelización del algoritmo. Como parte de este estudio se han definido tres dominios : ***el dominio de los datos, el dominio del espacio de soluciones y el dominio aritmético.***

4.4.1. Paralelización en el dominio de los datos

Es tal vez la forma más natural de paralelizar las redes neuronales, lo cual no quiere decir que sea la mejor forma de hacerlo sino la más sencilla y natural a simple vista.

El modelo a considerar es una red neuronal multicapa que contiene $C+1$ capas completamente interconectadas, tal que la capa i contiene n_i neuronas. Cada neurona de la capa i esta completamente interconectada con todas las neuronas de la capa $i+1$. Asociada con cada neurona i de la capa c tenemos un valor de activación llamado $x_i(c)$ y asociado a cada conexión de la red existe un valor de pesos denominado $w_{ji}(c,c+1)$ que representa el peso de la neurona i de la capa c con la neurona j de la capa $c+1$ considerando la existencia de p procesadores independientes.

4.4.1.1. Back-Propagation Secuencial

El algoritmo de BP secuencial puede expresarse mediante las siguientes ecuaciones:

Propagación hacía adelante:

$$O_i(c) = f_k\ (\sum_{i=1}^{n_{c-1}} W_{ij}(c-1,c).x_j(c-1)), \qquad i = 1,\ldots,n_{c-1} \qquad c = 0,\ldots,C \qquad (1)$$

Propagación hacía atrás:

$$\delta_{pk}{}^{o} = (y_{pk} - o_{pk}) f_k{}^{o'}(neta_{pk}{}^{o}) = \delta_{pk} f_k{}^{o'}(neta_{pk}{}^{o}) \qquad c = C$$

$\delta_i(c) =$

$$\delta_{pj}{}^{h} = f_j{}^{h'}(neta_{pj}{}^{h}) X_{pi} \sum_k \delta_{pj}{}^{o} W_{kj}{}^{o} \qquad c = C-1,\ldots,1 \qquad (2)$$

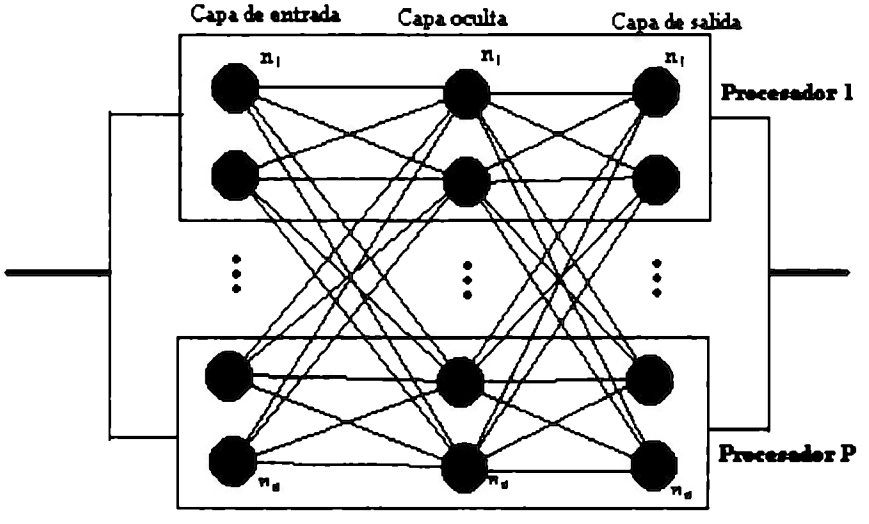
$$\Delta W_{ij}(c-1,c) = \eta \delta_i(c).O_j(c-1) \qquad (3)$$

4.4.1.2. Back-Propagation Paralelo[6]

La máquina paralela es considerada como un modelo de p procesadores (como se indica en la figura) con memoria distribuida y comunicaciones entre ellos a través de paso de mensajes. Se utiliza como estrategia de paralelización dividir la red en p partes, donde a cada procesador le corresponden n_c / p neuronas de la capa c.

---

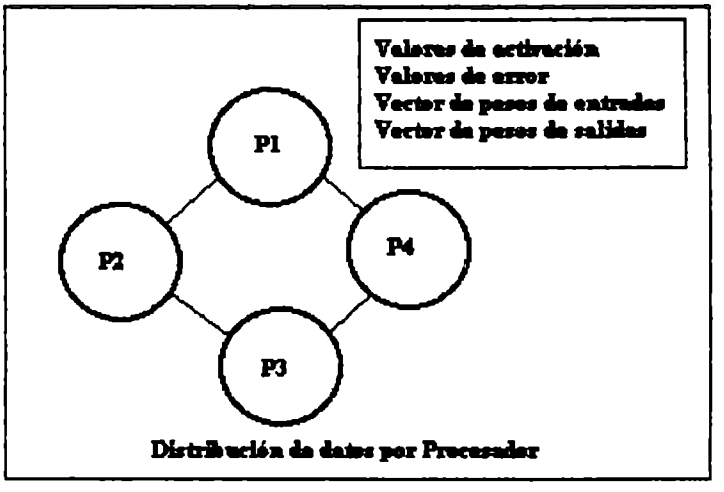
[6] A Distributed Backpropagation Algorithm of Neural Networks on Distributed-Memory Multiprocessors. Hyunsoo Yoon, Jong H. Nang, and S.R. Maeng. Center for Artificial Intelligence Research & Department of Computer Science. Korea Advanced Institute of Science and technology. hyoon@sorak.kaist.ac.kr



Cada procesador almacenará en memoria local los valores de activación, los valores de error, los vectores pesos de entrada y los vectores pesos de salida del conjunto de neuronas asignadas. Al igual que los pesos correspondientes a w_{ij} y w_{ji} de la capa c (ver figura anexa) con el fin de evitar excesos en tiempos de comunicación, los cuales generalmente son altos[7].

---

[7] Global Optimizations for Parallelism and Locality on Scalable Parallel Machines, Jennifer M. Anderson and Monica S. Lam, Computer System Laboratory, CA 94305. Proceedings of SIGPLAN'93, Albuquerque, New Mexico, 1993.



Cada uno de los p procesadores ejecuta los programas secuenciales básicos, pero algunas operaciones de comunicación se hacen necesarias para poder realizar todos los cálculos, tales como el cálculo de los valores de activación y de los errores. Para facilidad de cálculo en nuestras operaciones consideremos que $|N_t(c)| = \frac{n_c}{p}$.

Para calcular la ecuación (1) :

$$O_i(c) = f_k \ (\sum_{i=1}^{n_{c-1}} W_{ij}(c-1,c).x_j(c-1)), i = 1,\ldots,n_c \qquad c = 1,\ldots,C$$

cada uno de los p procesadores requiere conocer todos los valores $x_j(c-1)$ almacenados en los $(p-1)$ procesadores. Esto se ejecuta realizando un broadcast de estos valores de activación del procesador p al resto de procesadores, es decir, el procesador p_t hace broadcast del valor $O_j(c-1)$ para cada neurona $j \in N_t(c-1)$ y reciben los valores $O_j^{'}(c-1)$ para cada neurona $j' \notin N_t(c-1)$, este proceso es llamado un broadcast múltiple (all-to-all broadcast). Se puede representar lo

anteriormente dicho mediante el siguiente algoritmo :

```
for (c = 1; c≤ C+1; ++c)
{
  /* Broadcast y reciben O_j(c-1) */
  for (j = 1; j ∈ N_t(c-1); ++j)
    all-to-all-broadcast(O_j(c-1));
  for (i = 1; i ∈ N_t(c); ++i)
    O_i(c) = f(Σ_{j=1}^{n_{c-1}} w_ij(c-1,c).x_j(c-1);
}
```

Con el cálculo del error en el procedimiento de propagación hacia atrás sucede algo similar. Para calcular los valores $\delta_i(c)$ donde $i \in N_t(c)$, primero se deben transmitir los valores $\delta_k(c+1)$ almacenados en $p-1$ procesadores y que son requeridos para dicho cálculo. Este requerimiento puede ser satisfecho en el caso de que cada procesador efectúe un broadcast de $\delta_k(c+1)$ de cada neurona $k \in N_t(c+1)$ y recibe a su vez los valores $\partial_k'(c+1)$ de cada neurona $k' \notin N_t(c+1)$. Se puede representar lo anteriormente dicho mediante el siguiente algoritmo :

```
/* Cálculo de errores para la capa de salida */
for (i = 1; i ∈ N_t(C+1); ++i)
  ∂_i(C+1) = [y_i(C+1) - o_i(C+1)].[o_i(C+1).(1 - o_i(C+1))];

for (c = C; c = 1, --c)
{
 /* Broadcast y recibe δ_k(c+1) */
 for (k = 1; k ∈ N_t(c+1); ++k)
   all-to-all-broadcast δ_k(c+1);
 /* Cálculo de δ_i(c) */
 for (i = 1; i ∈ N_t(c); ++i)
   ∂_i(c) = [Σ_{k=1}^{n_{c+1}} ∂_k(c+1).w_ki(c,c+1).[o_i(c).(1 - o_i(c)]];
```

```
/* Modificar pesos de salida  w_ki(c,c+1) */
(for i = 1; i ∈ N_t(c) ; ++y)
 (for k = 1; k ∈ n_{c+1}; ++k)
 {
    Δw_ki(c,c+1) = η∂_k(c+1)o_i(c);
    w_ki(c,c+1) = w_ki(c,c+1) + Δw_ki(c,c+1);
 }

/* Modificar pesos de entrada */
/* Broadcast y recibe o_j(c-1) */
for (j = 1; j ∈ N_t(c-1); ++j)
 all-to-all-broadcast (o_j(c-1));
*/ Modificar pesos w_ij(c-1,c) */
for (i = 1; i ∈ N_t(c) ; ++y)
 for (j = 1; j ∈ n_{c-1}; ++j)
 {
   Δw_ki(c-1,c) = η∂_i(c)o_i(c-1);
   w_ij(c-1,c) = w_ij(c-1,c) + Δw_ij(c-1,c);
 }
}
```

4.4.1.3. Análisis de la Complejidad Computacional

El tiempo requerido por un ***solo procesador*** para ejecutar el algoritmo de back-propagation puede ser visualizado por tres ecuaciones que describen el algoritmo y corresponden a la complejidad temporal:

$$t_1 = n.\left[n.T_{ms} + T_f\right]$$

$$t_2 = n.\left[n.T_{ms}\right]$$

$$t_3 = n.\left[n.T_{ms}\right]$$

La complejidad temporal del algoritmo se puede expresar como la suma aritmética de los tiempos : $T = t_1 + t_2 + t_3 = n.\left[3n.T_{ms} + T_f\right]$. Donde T_{ms} es el tiempo para realizar las operaciones de suma y multiplicación. T_f es el tiempo para aplicar la función sigmoidal al producto de los valores de activación multiplicado por los pesos asociados y n es el número de neuronas de cada capa (asumimos que $n_c = n$).

De igual forma determinamos la complejidad temporal para nuestro algoritmo paralelo para una capa de n neuronas corriendo en un procesador p mediante las siguientes ecuaciones :

$$t_1^{\bullet} = \frac{n}{p} T_{all-all}^{(p)} + \frac{n}{p}(n.T_{ms} + T_f)$$

$$t_2^{\bullet} = \frac{n}{p}(2.T_{all-all}^{(p)} + 3.n.T_{ms})$$

La complejidad temporal de este algoritmo se expresa como la suma de $t_1^{\bullet} + t_2^{\bullet} = T^{\bullet}$, tal que $T^{\bullet} = \frac{n}{p}(3.T_{all-all}^{(p)} + 4.n.T_{ms} + T_f)$. Se deberá asumir que cada procesador sólo puede estar transmitiendo y/o recibiendo un solo mensaje en un instante de tiempo t. Luego el tiempo que necesita un procesador para hacer un broadcast será $(p-1).C$, donde C es la unidad de tiempo usada por un procesador en operaciones de manejo de información (send/receive).

Conociendo ya las ecuaciones de T y $T^{\bullet}$, podemos determinar la relación denominada ***radio de velocidad o aceleración*** la cual está definida como $R_v = \frac{T}{T^{\bullet}}$.

Ahora se debe considerar que las operaciones all-to-all broadcast y su complejidad computacional se calculan para p procesadores (considerando $C = \Psi.T_{ms}$ y $T_f = \Phi.T_{ms}$)

$$R_v(p) = \frac{T}{T^*} = \frac{n.(3.n.T_{ms} + T_f)}{\frac{n}{p}(3.T_{all-all}{}^{(p)} + 4.n.T_{ms} + T_f)} = \frac{p.(3.n.T_{ms} + \Phi.T_{ms})}{\left[3.\Psi(p-1).T_{ms} + 4.nT_{ms} + \Phi T_{ms}\right]}$$

$$R_v(p) = \frac{T}{T^*} = \frac{p.(3.n.+\Phi.)}{3.\Psi(p-1) + 4.n + \Phi}$$

Una vez que la etapa de entrenamiento ha culminado, nuestra aceleración puede expresarse de una manera mas simple:

$$R_v{}^*(p) = \frac{t_1}{t_1{}^*} = \frac{p.(n+\Phi)}{\Psi.(p-1) + n + \Phi}$$

Para redes neuronales muy grandes, es decir, que se cumpla la relación $n \to \infty$, tenemos:

$$\lim_{n\to\infty} R_v(p) = \frac{3}{4}.p$$

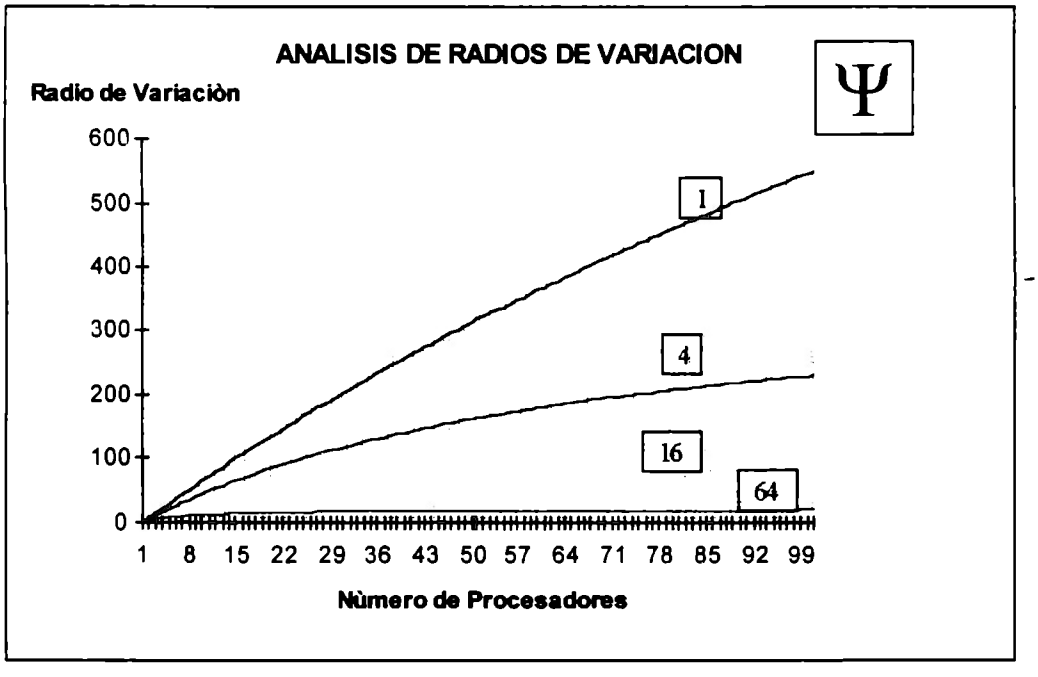
$$\lim_{n\to\infty} R_v{}^*(p) = p$$

De las ecuaciones de R_v y $R_v{}^*$ es necesario aclarar que el parámetro más importante es el valor de Ψ, que expresa el ***tiempo de comunicación/computación*** ($T_{all-all} = \Psi T_{ms}(p-1)$) utilizado para efectuar los broadcast en el algoritmo paralelo.

Se dice que Ψ oscila[8] entre $0.5 \Leftrightarrow 256$ y un valor de Φ arbitrario pero fijo.

Para interpretar estos resultados se realizará una simulación para estudiar el radio de variación como variable dependiente del número de procesadores. Aunque es claro de las ecuaciones anteriores observar que $\Phi <<< \Psi$.

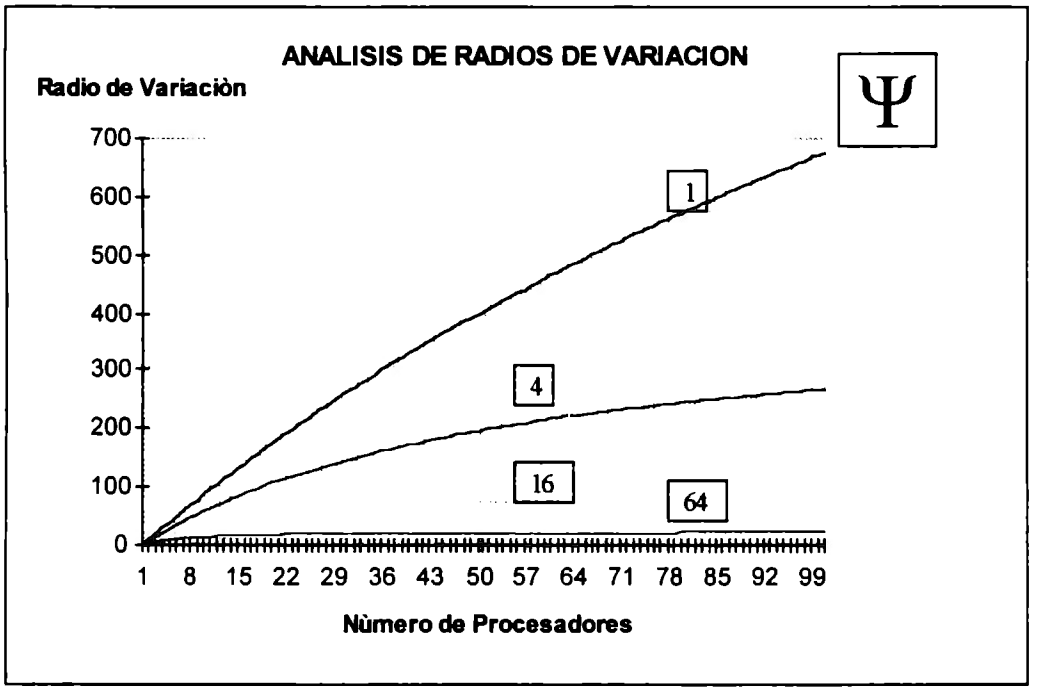
Gráfica de R_v parámetros: $1 < n < 512$; $\Phi = 40$; $\psi = 10 * \Phi ; 100 * \Phi$



Gráfica de R_v^* **para** $n = 2048$; $\Phi = 40$; ψ**=1,4,16,64**

---

[8] M. Annaratone, C. Pommerell, and R.Ruhl, "Interprocessor Communication Speed and Performance in Distributed-Memory Paralle Processors, " Proc. of the 16th Annual Int'l Symp. on Computer Architecture, pp. 315-324, 1989.



De las gráficas anteriores se pude concluir de que a pesar de que muchos procesadores fueron adicionados a la simulación , el radio de variación de velocidad no se incrementa en cifras considerables.

Podemos ver que para valores de $\Phi > \Psi$, se cumple la relación $T > T^*$, lo cual no necesariamente es cierto, ya que los tiempos de comunicación son mucho mayores que el tiempo asociado a operaciones matemáticas. Considerando $\Phi < \Psi$, sucede que $T < T^*$, generando un Rv muy pequeño. De acuerdo a las ecuaciones planteadas en el análisis de la complejidad computacional es cierto que los valores de $\Phi <<< \Psi$, debido a que los tiempos utilizados en operaciones operaciones de comunicación son mucho mayores que los tiempos empleados en operaciones aritméticas.

La segunda consideración tiene más sentido en máquinas paralelas, ya que ***los tiempos de comunicación generados por operaciones all-to-all broadcast son***

excesivamente altos y son consideradas *operaciones indeseables* en procesos de paralelización de procesos debido a los altos tiempos que consumen.

Mediante los procesos de simulación y considerando como constantes n y Φ, podemos afirmar:

- Los valores de Ψ deberán ser mucho mayores que los valores de Φ. En este caso se hacen dos consideraciones : $\Psi = 10 * \Phi$; $\Psi = 100 * \Phi$

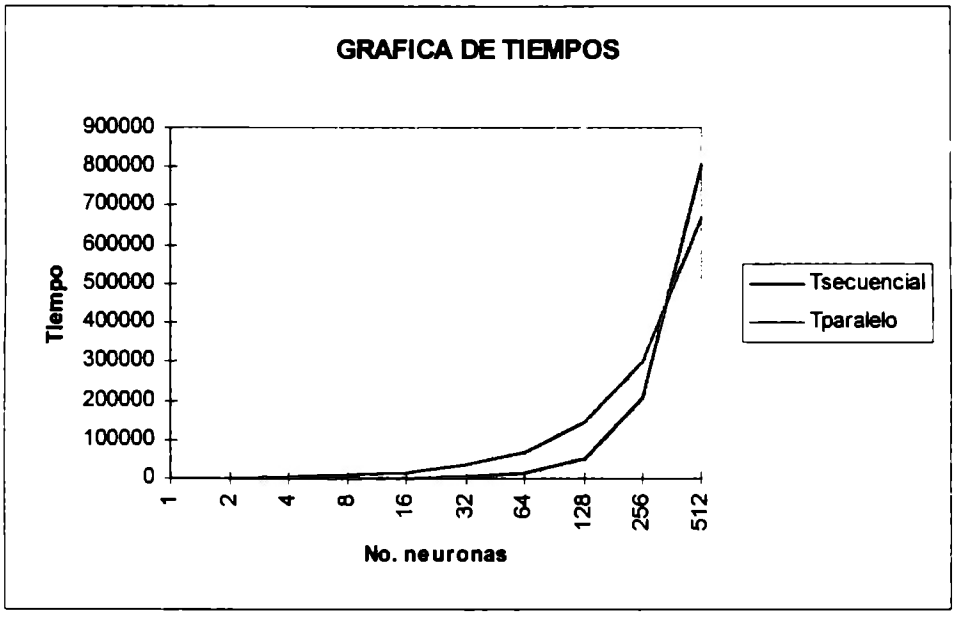
- Para valores de $\Psi >>> \Phi$, deberá cumplirse $T <<< T^*$, es decir el tiempo secuencial deberá ser mayor que el tiempo paralelo debido a los tiempos de comunicación en la red.

- Los tiempos utilizados por la máquina paralela que ha sido simulada no son mejores que los tiempos de nuestro algoritmo secuencial utilizado para el entrenamiento de redes neuronales feed-forward.

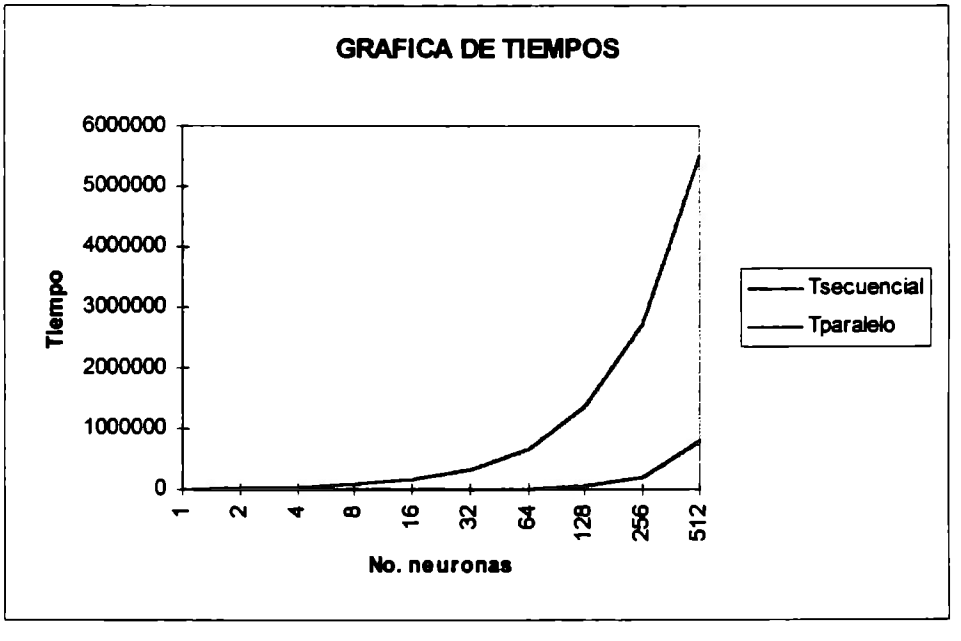
- La aceleración en la mayoría de los casos es la unidad.

Para comprender de manera inmediata estos comentarios, observamos el comportamiento que presentan los tiempos secuenciales y paralelos para cada uno de los siguientes escenarios:

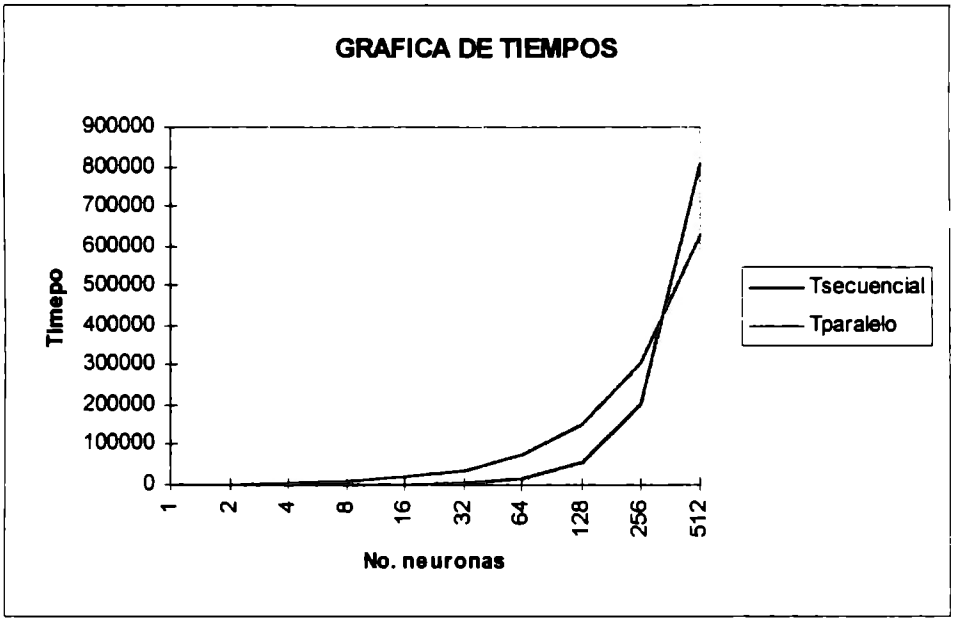
$p = 8 \quad \Phi = 40; \Psi = 10 * \Phi; 1 < n < 512$



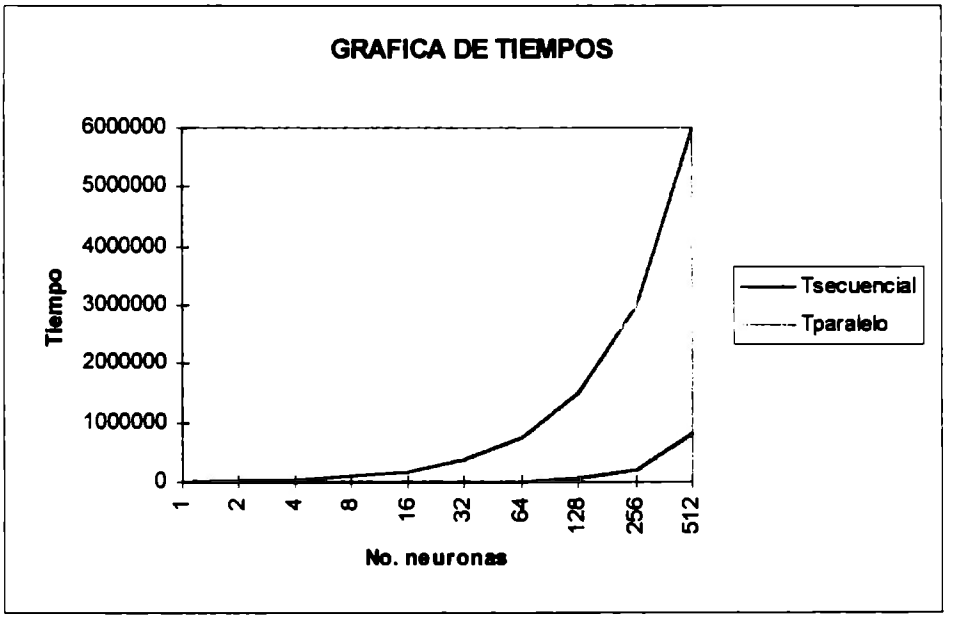
$p = 8; \Phi = 40; \Psi = 100 * \Phi; 1 < n < 512$



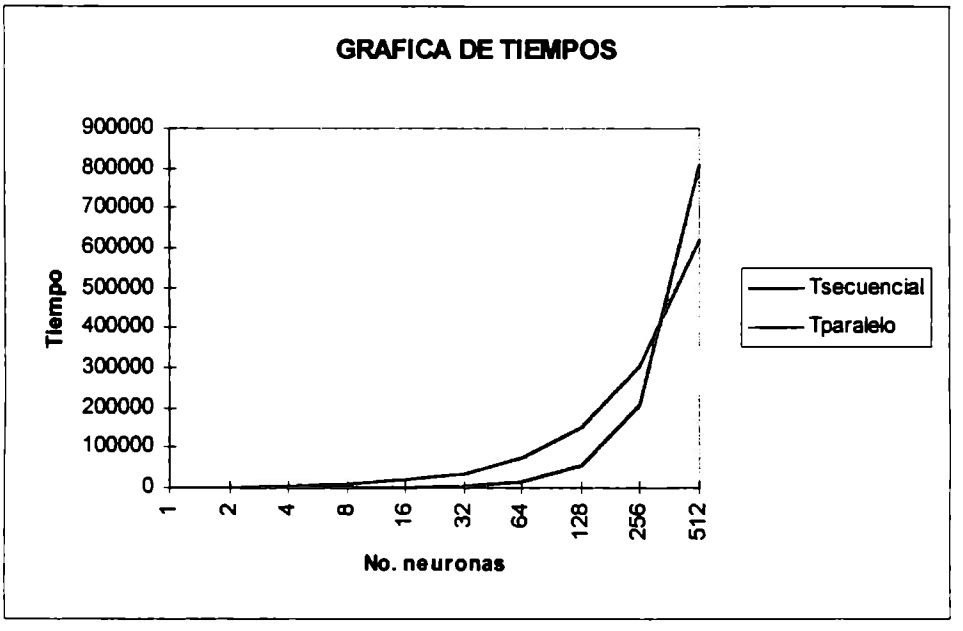
$p = 32; \Phi = 40; \Psi = 10 * \Phi; 1 < n < 512$



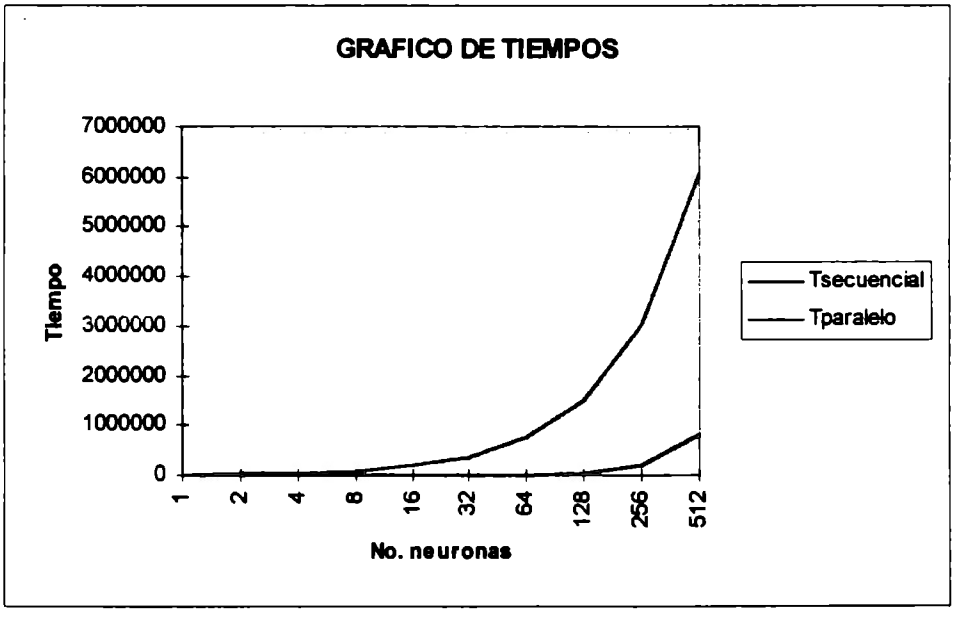
$p = 32; \Phi = 40; \Psi = 100 * \Phi; 1 < n < 512$



$p = 64; \Phi = 40; \Psi = 10 * \Phi; 1 < n < 512$



$p = 64; \Phi = 40; \Psi = 10 * \Phi; 1 < n < 512$



Los tiempos utilizados por la máquina paralela que ha sido simulada no son mejores que los tiempos de nuestro algoritmo secuencial utilizado para el entrenamiento de

redes neuronales feed-forward, debido a que la aceleración en la mayoría de los casos es la unidad.

Debido a que las operaciones all-to-all broadcast utilizadas en el algoritmo paralelo son indeseables en todos los procesos de paralelización de procesos éste método, no constituye parte del interés esencial de este estudio.

4.4.2. Paralelización en el dominio del espacio de soluciones

Este método tiene como objetivo explorar el espacio de soluciones en forma paralela con el fin de agilizar el proceso de búsqueda, utilizando para la búsqueda cada uno de los procesadores de manera independiente. Cada procesador inicia la búsqueda en el espacio de soluciones, ubicado en un punto arbitrario (con valores aleatorios) y decide iniciar la búsqueda en el menor tiempo posible, reportando resultados en el caso de mínimos locales o globales. *Es claro que el tiempo absoluto de este procedimiento es mejor que Back-Propagation aplicado en un solo procesador, pues para toda variable estocástica X siempre se cumple:*

$$\langle min\{X_1,\Lambda\ ,X_k\}\rangle \leq \langle X\rangle$$

donde $X_1,\Lambda\ ,X_k$ son variables estocásticas idénticamente distribuidas, con la misma distribución que la variable estocástica X. La evaluación de este modelo se puede hacer en serie repitiendo los experimentos de Back-Propagation, agrupando los resultados en grupos consecutivos y reportando el mínimo valor de tiempo para cada grupo, como resultado individual.

Es claro que dicho método es mejor, o en el peor de los casos igual, que Back-Propagation con un solo procesador. Esto indica que cualquier otra estrategia que se proponga no solamente debe ser mejor que Back-Propagation (BP), sino mejor que la anterior que se denominará en adelante BP-modificado.

Nuestro algoritmo se verá modificado de la siguiente forma :

```
Repetir el siguiente proceso para un número de veces igual al tamaño deseado de
la muestra
{
for ( p = 1; p < numero_procesadores; p++)
 {
   for(i = 0 ; i < n ; ++i)
        {
        Escoger un patrón al azar;
        Propagar la información hacia adelante;
        if( Error < ε )
                {
                if(k== v)
                        terminar escribiendo i+1
                else
                        ++k;
                }
        else
                k=0;
        Aplicar el algoritmo de entrenamiento;
        }
   }
   Determinación del tiempo de duración mínima ;
}
```

4.4.3. Paralelización en el dominio del espacio aritmético

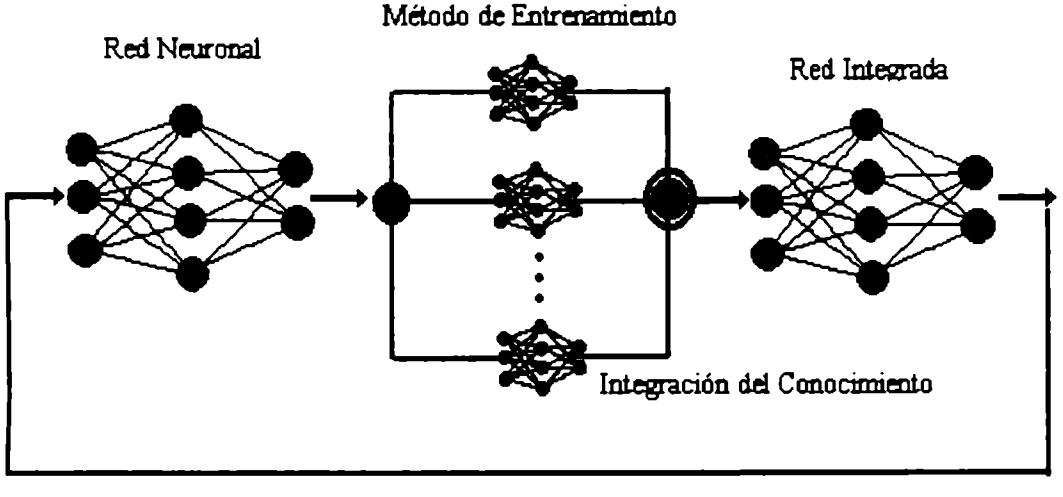
Buscar un método alterno que mejore la velocidad de convergencia en redes feed-forward implica pensar en una posición intermedia entre la regla que establece actualizar los parámetros de la red con la presentación de patrones individuales y la que establece actualizar los parámetros después de la presentación de todos los

patrones.

La base del algoritmo consiste en paralelizar[9] el algoritmo ya planteado para nuestro estudio inicial para redes feed-forward, permitiendo que cada procesador actualice los parámetros después de la presentación de k patrones m veces (nótese que es una posición intermedia entre BP y BP-modificado). Luego de esto se actualizan los parámetros del modelo según la siguiente estrategia: ***"tomar el promedio de los parámetros de todos los procesadores"***, de la forma:

$$\Delta W = \frac{\sum_{i=1}^{n\_procesadores} \Delta W_i}{n\_procesadores}$$

Que gráficamente puede ser representado de la siguiente forma :



[9] Simulamos el paralelismo de procesos, es decir, cada procesador es un programa o proceso secuencial, que interactúa con un programa maestro (Master - Slave)

4.5. Algoritmo Propuesto

De acuerdo a lo realizado hasta ahora, podemos ver el algoritmo paralelo para redes feed-forward de la siguiente forma :

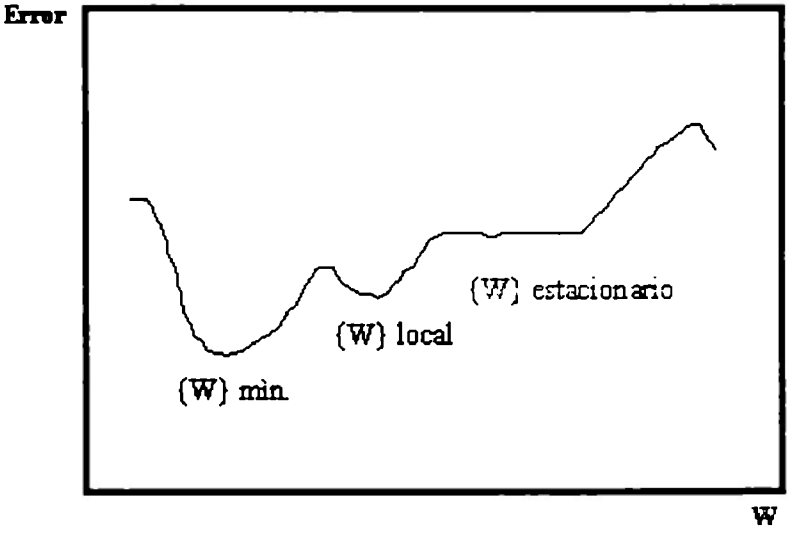
```
Repetir el siguiente proceso para un número de veces igual al tamaño
deseado de la muestra
{
for ( p = 1; p < numero_procesadores; p++)
{
  for(i = 0 ; i < n ; ++i)
      {
      /* actualizamos parámetros después de la presentación
        de k patrones m veces */
      for(j = 0; j < k;++ j)
      {
       Escoger un patrón al azar;
       Propagar la información hacia adelante;
      }
      if( Error < ε )
            {
            if(k== v)
                  terminar escribiendo i+1
            else
                  ++k;
            }
      else
            k=0;
      Aplicar el algoritmo de entrenamiento;
      Integración de conocimiento, ponderando pesos de conexión ;
      Fase de modificación de pesos
            {
              Para todos los procesadores calcular ∂red_i ;
              Calcular el valor ∂red_metodo_seleccionado ;
            }
            Modificación de pesos;
      }
  }
}
```

El método que se propone en este momento se encarga de realizar en forma

independiente por procesador el entrenamiento en la fase de propagación hacía adelante, pero tiene en cuenta el resultado de todos los procesadores para la fase de propagación hacía atrás, con el fin de acumular el conocimiento que se ha ido almacenando en cada uno de los procesadores a lo largo de un período de tiempo considerado.

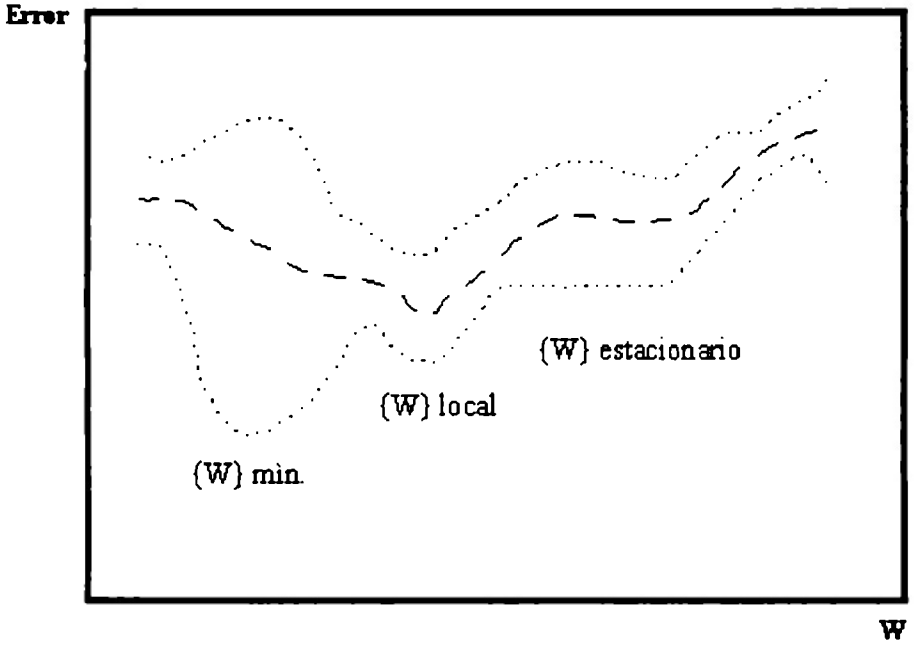
Esta ponderación del error de cada red optimiza los resultados obtenidos, ya que el aprendizaje de cada red contribuye al conocimiento global de la nueva red.

Se puede ver el comportamiento de un procesador en un instante de tiempo t, en forma gráfica de la siguiente forma :



Debido a que el nuevo método considera la existencia de varios procesadores (sistema multiprocesador, con memoria distribuida y comunicación a través de paso de mensajes), dichos valores de pesos se promediarán con el fin de ajustar el valor del error hacia un valor medio, que en la teoría de las grandes muestras sería equivalente

al valor más aconsejable. Podemos representar lo dicho anteriormente con una gráfica similar:



El método considerado en adelante se denominará Mixed Back-Propagation (MIXED-BP) y con el se harán las simulaciones en el siguiente capítulo de resultados.

Capítulo 5

Resultados

Para analizar las variables estocásticas de valor medio y varianza hemos corrido el programa propuesto (paralización en el dominio aritmético) y que denominamos MIXED-BP sobre tres conjuntos de datos : PARITY3, XOR221 y XOR231.

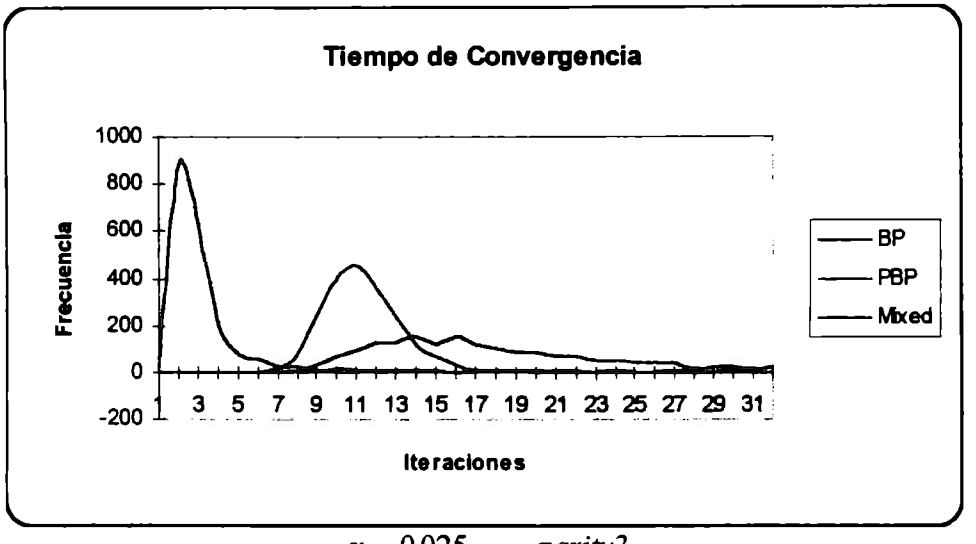
Para cada uno de estos conjuntos se han realizado ciertas consideraciones a tener en cuenta :

- Los valores de α oscilan entre [0.025 - 0.30] teniendo incrementos sucesivos de 0.025.
- Se han graficado los resultados de 2048 iteraciones agrupadas en segmentos de 64 (2048=32*64)
- La gráfica nos indica la frecuencia por segmentos en donde la red neuronal converge en forma satisfactoria.

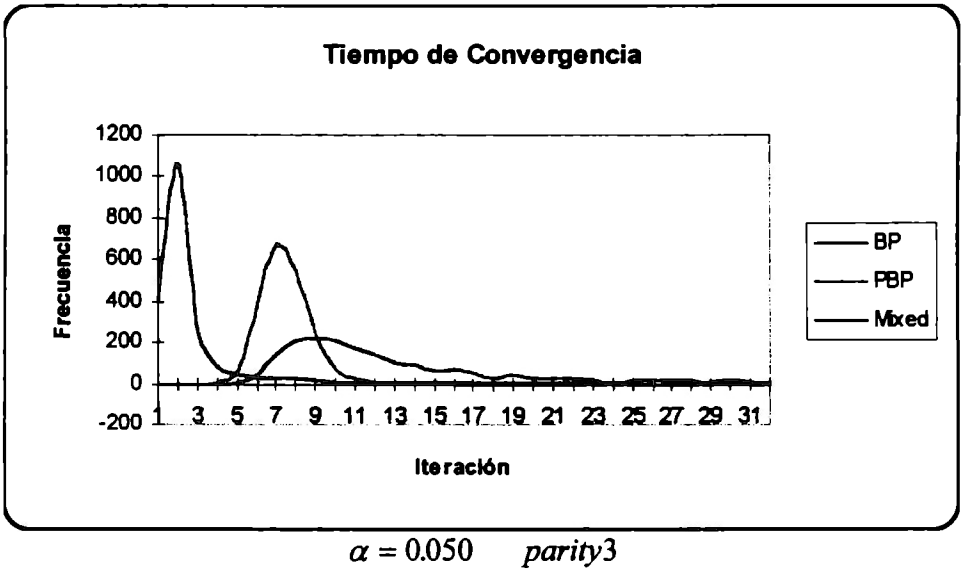
5.1. PARITY3

Datos de entrada:

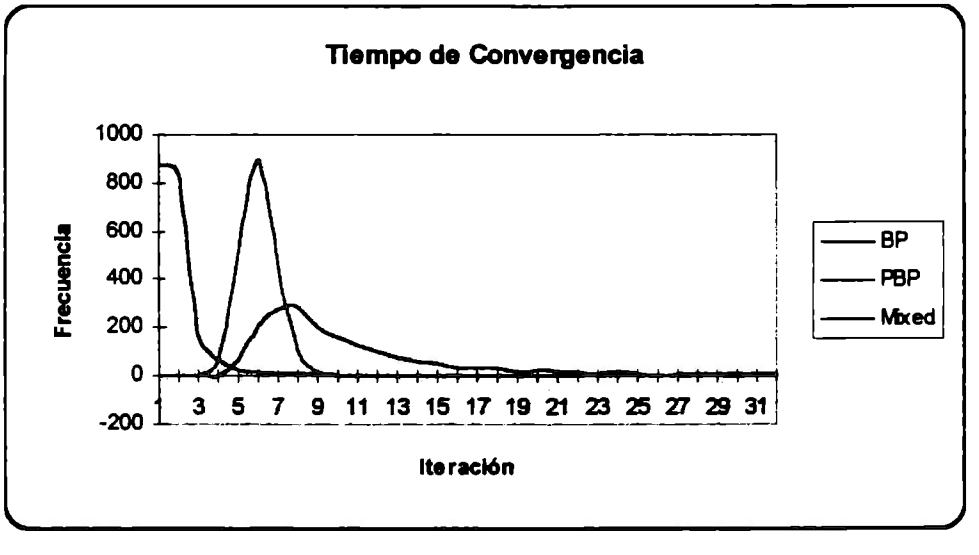
-1	-1	-1	-1
-1	-1	1	1
-1	1	-1	1
-1	1	1	-1
1	-1	-1	1
1	-1	1	-1
1	1	-1	-1
1	1	1	1



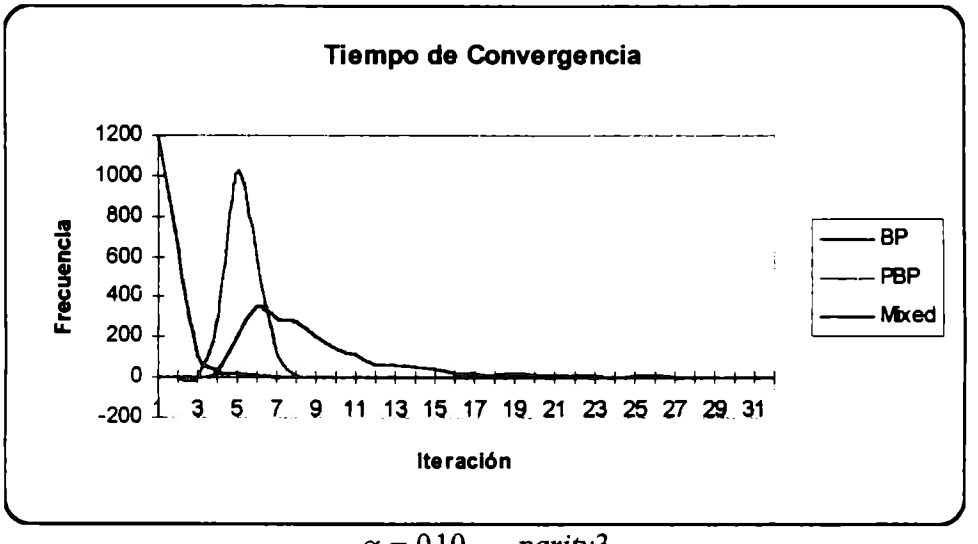
$\alpha = 0.025$ *parity*3



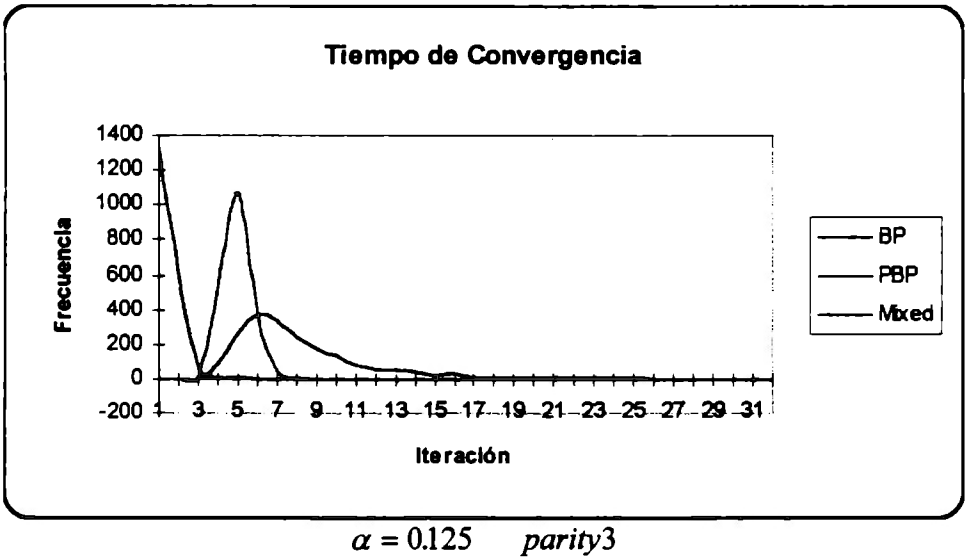
$\alpha = 0.050$ *parity*3



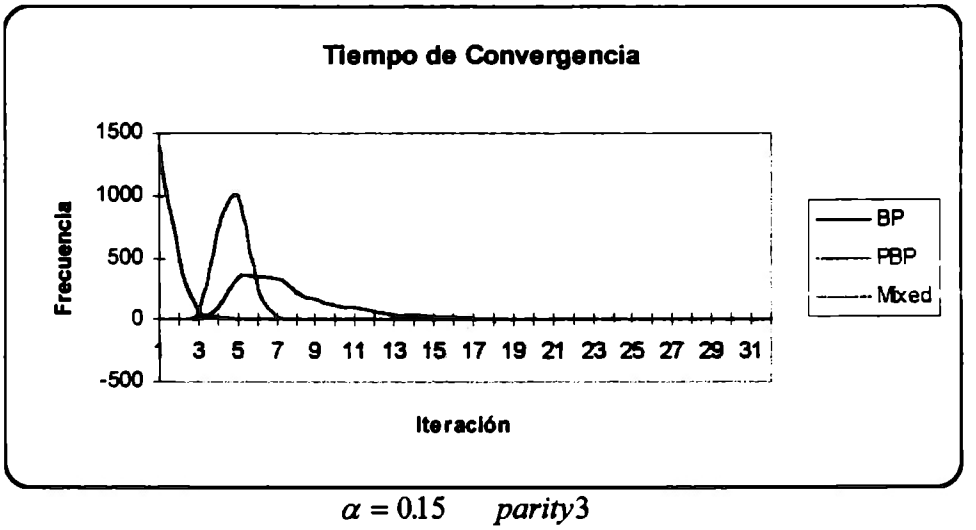
$\alpha = 0.075 \quad parity3$



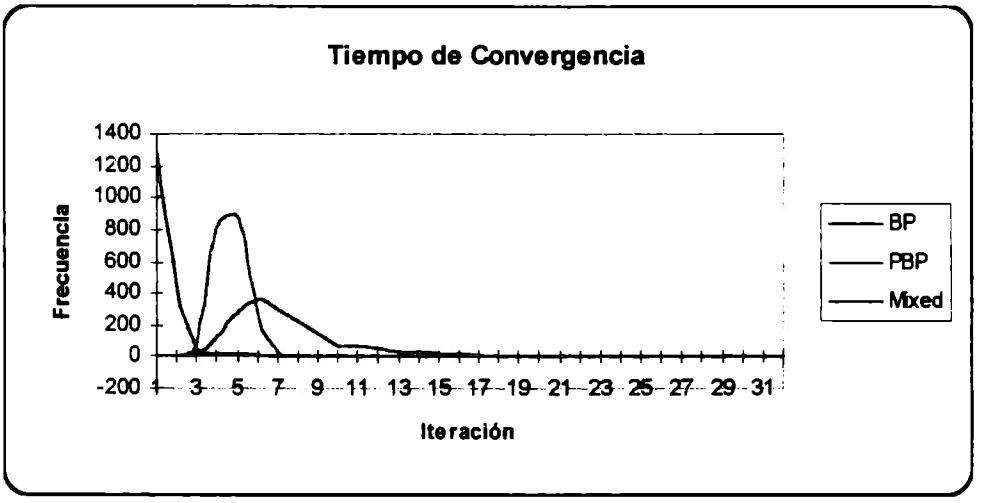
$\alpha = 0.10 \quad parity3$



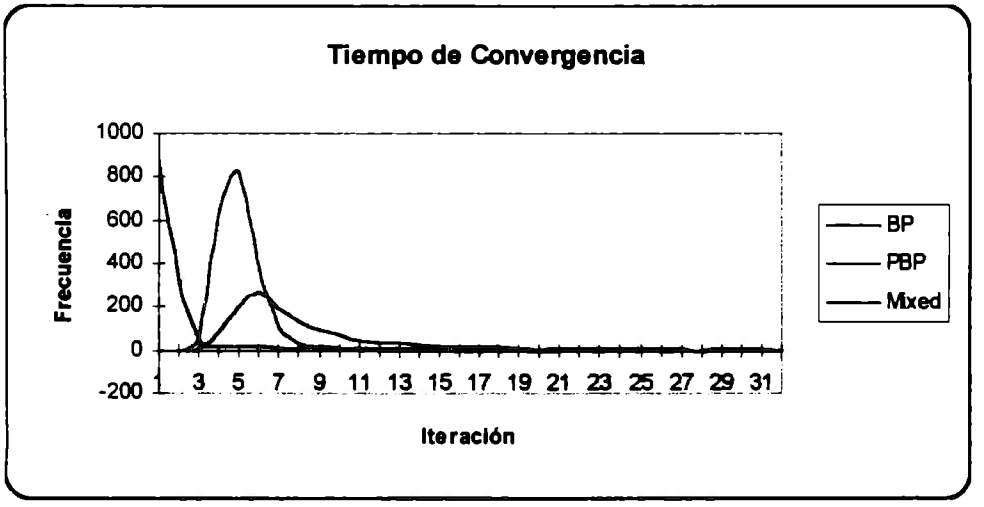
$\alpha = 0.125 \quad parity3$



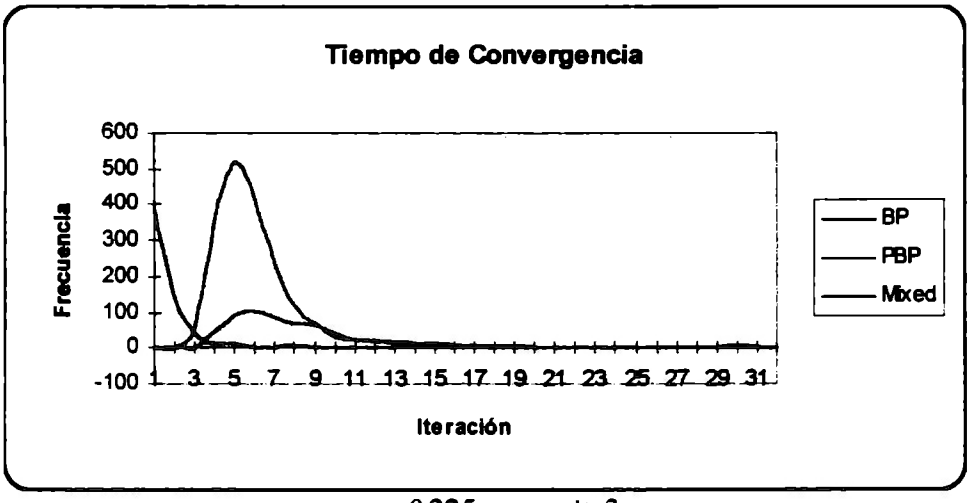
$\alpha = 0.15 \quad parity3$



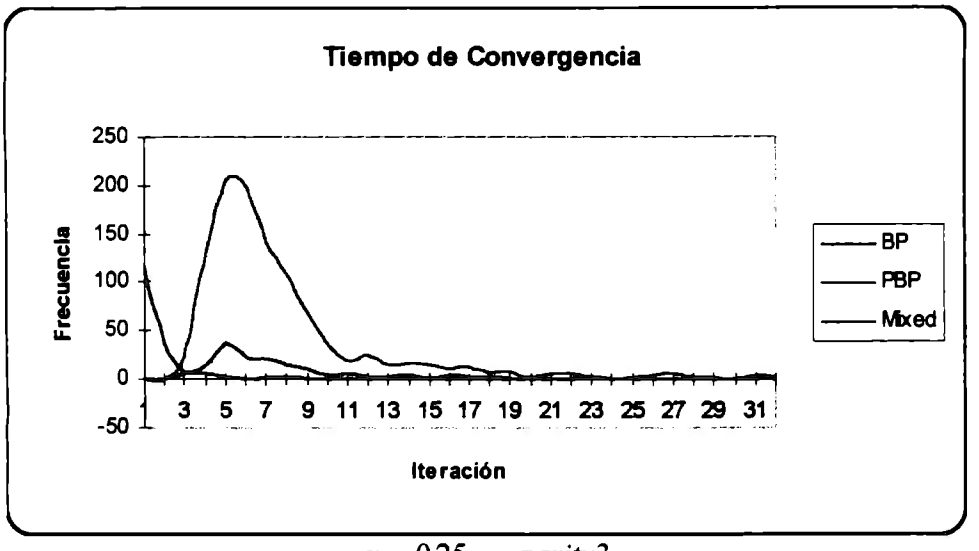
$\alpha = 0.175$ *parity*3



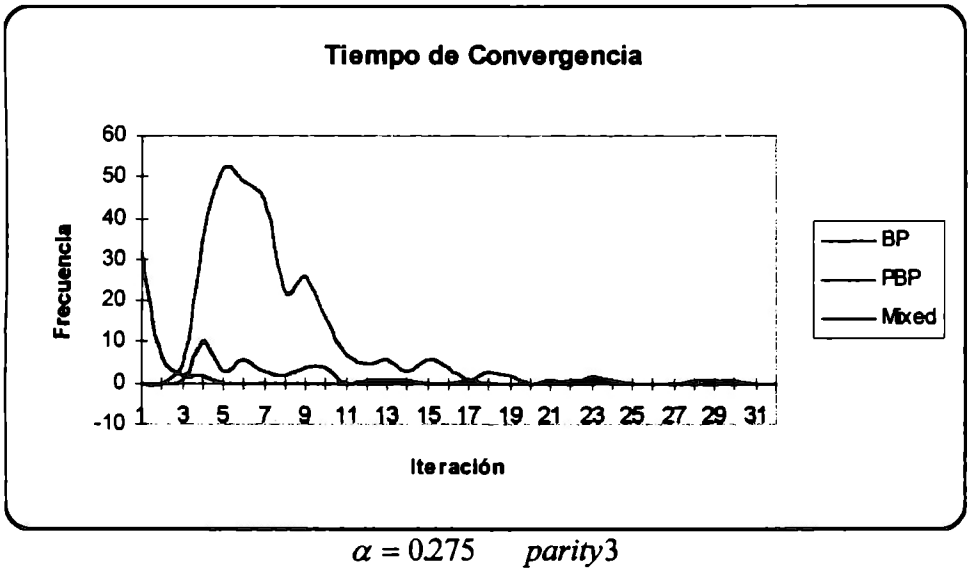
$\alpha = 0.20$ *parity*3



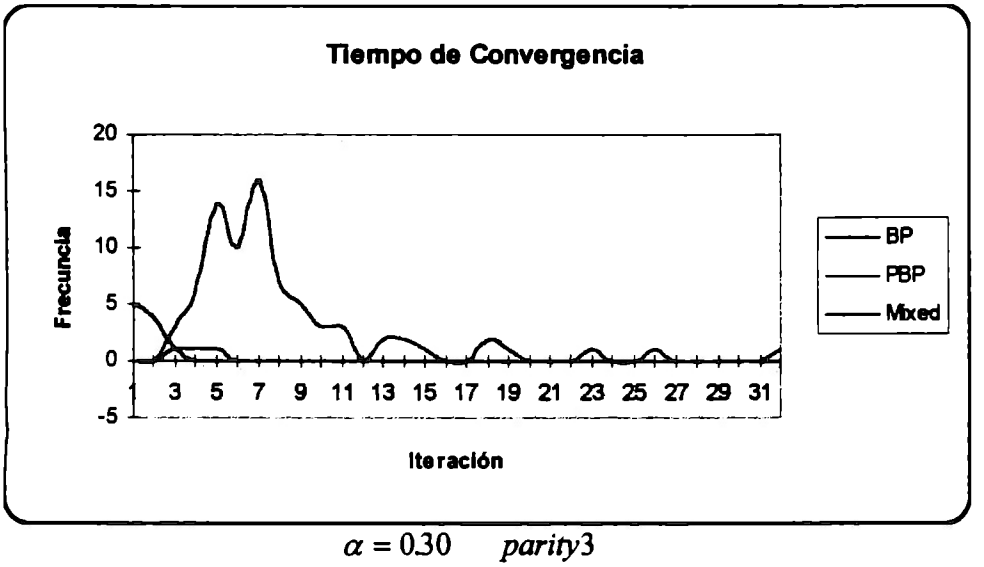
$\alpha = 0.225$ *parity*3



$\alpha = 0.25 \qquad parity3$



$\alpha = 0.275 \qquad parity3$

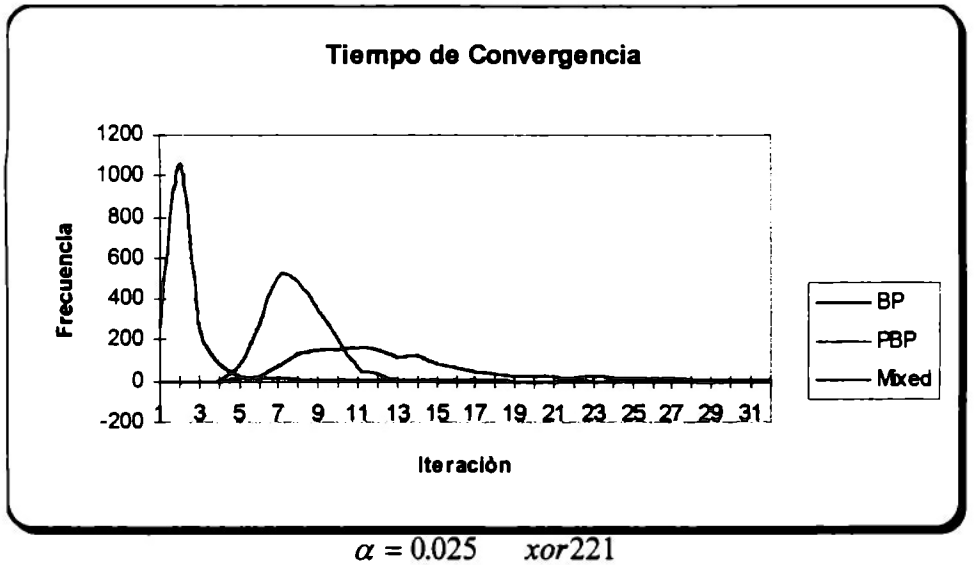


$\alpha = 0.30 \qquad parity3$

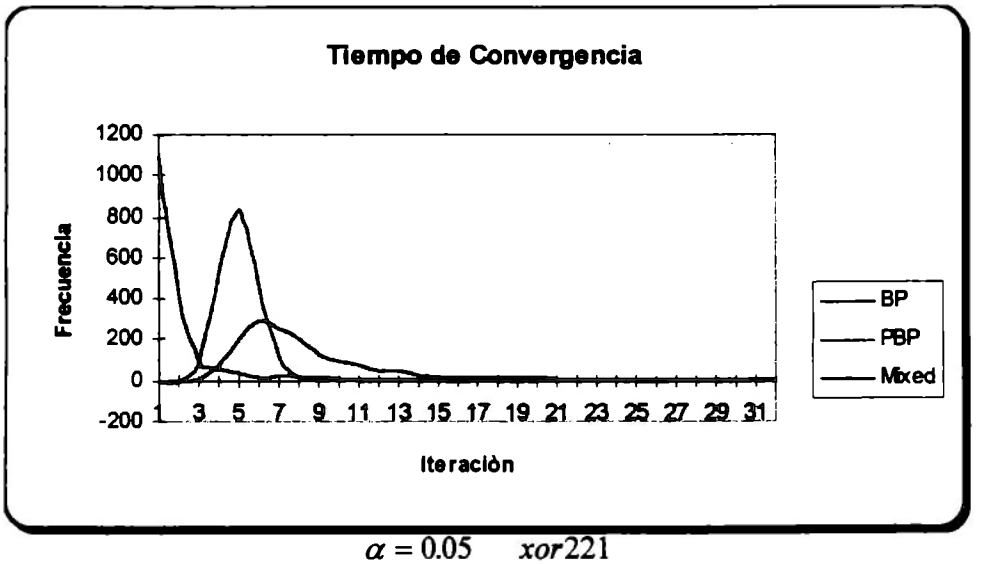
5.2. XOR221

Datos de entrada:

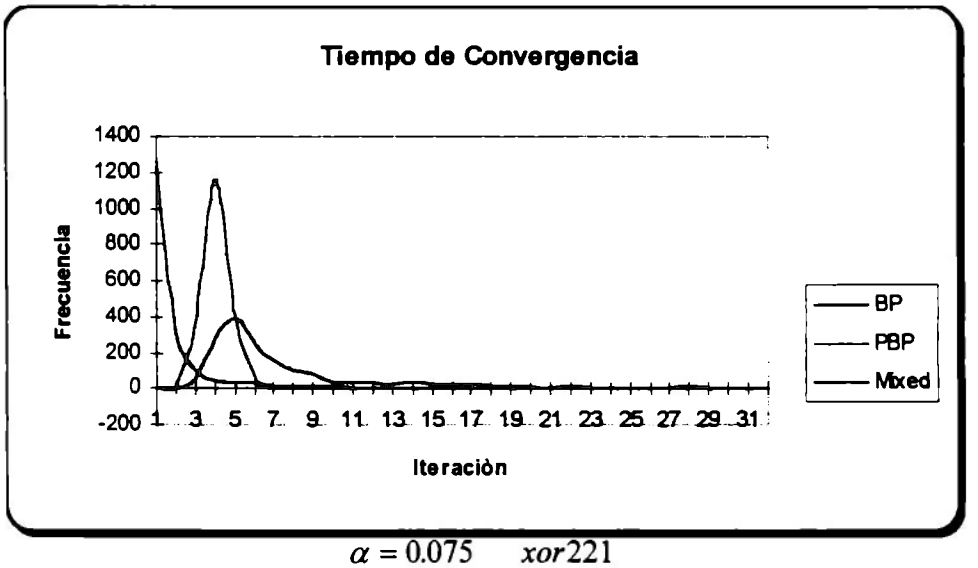
1.0	1.0	-1.0
1.0	-1.0	1.0
-1.0	1.0	1.0
-1.0	-1.0	-1.0



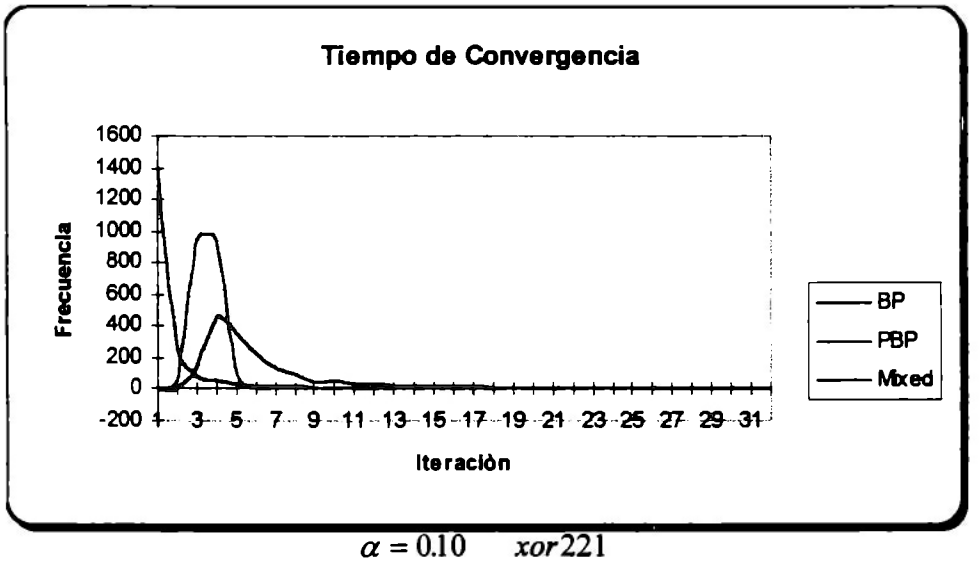
$\alpha = 0.025 \quad xor221$



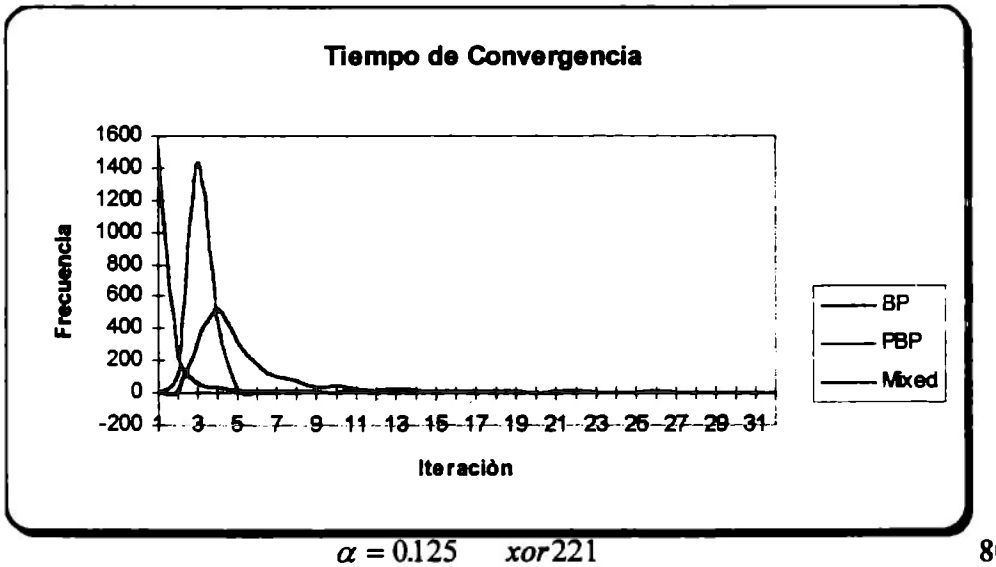
$\alpha = 0.05 \quad xor221$



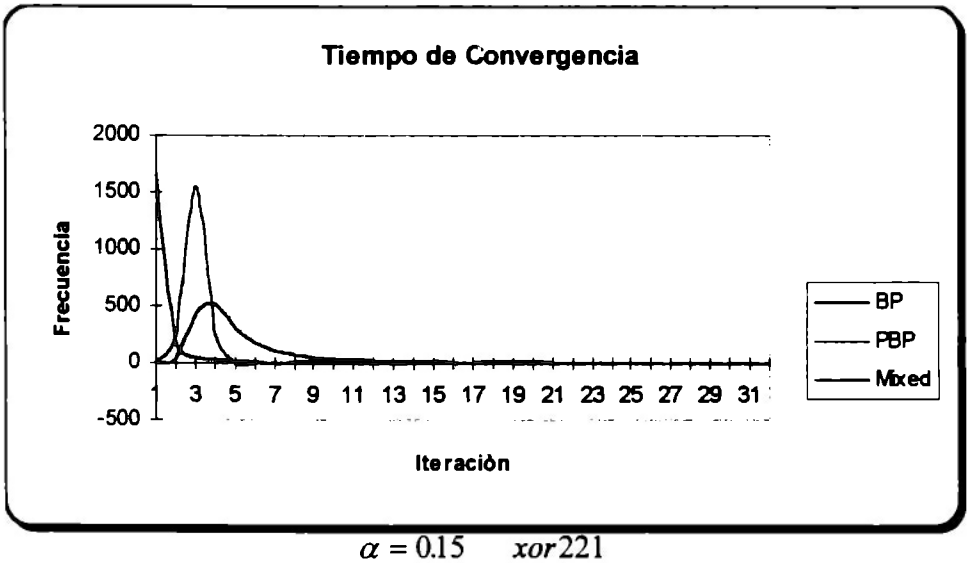
$\alpha = 0.075 \qquad xor221$



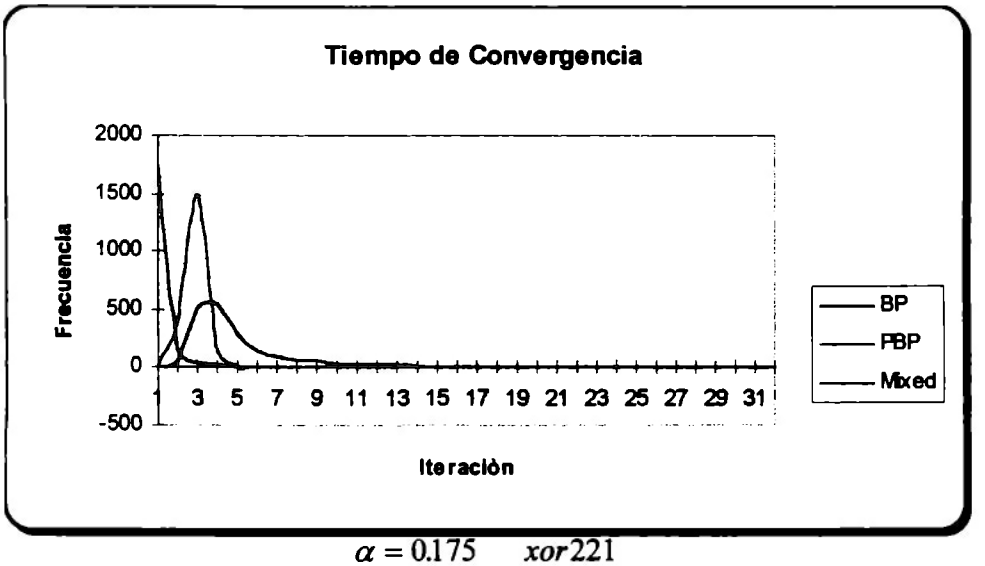
$\alpha = 0.10 \qquad xor221$



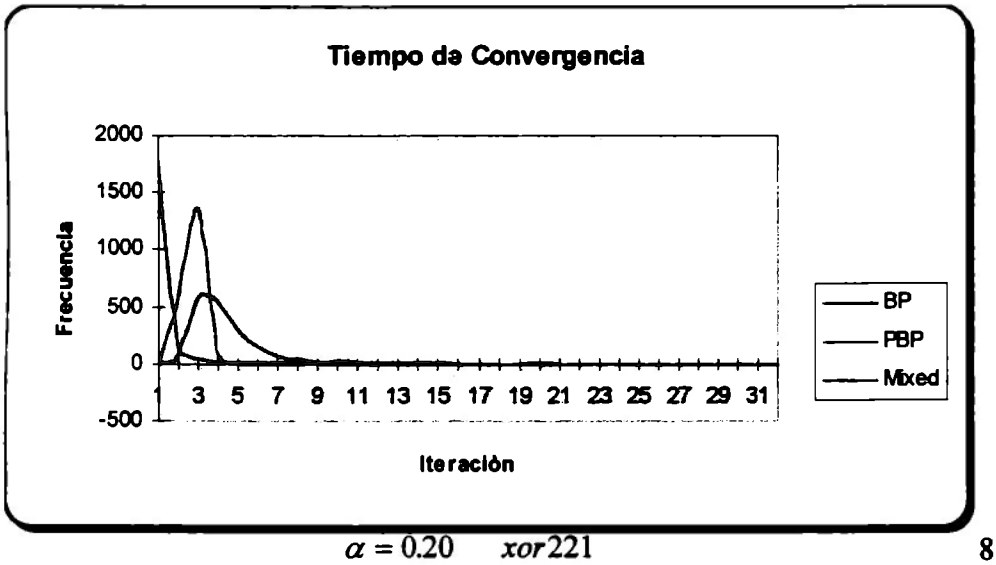
$\alpha = 0.125 \qquad xor221$



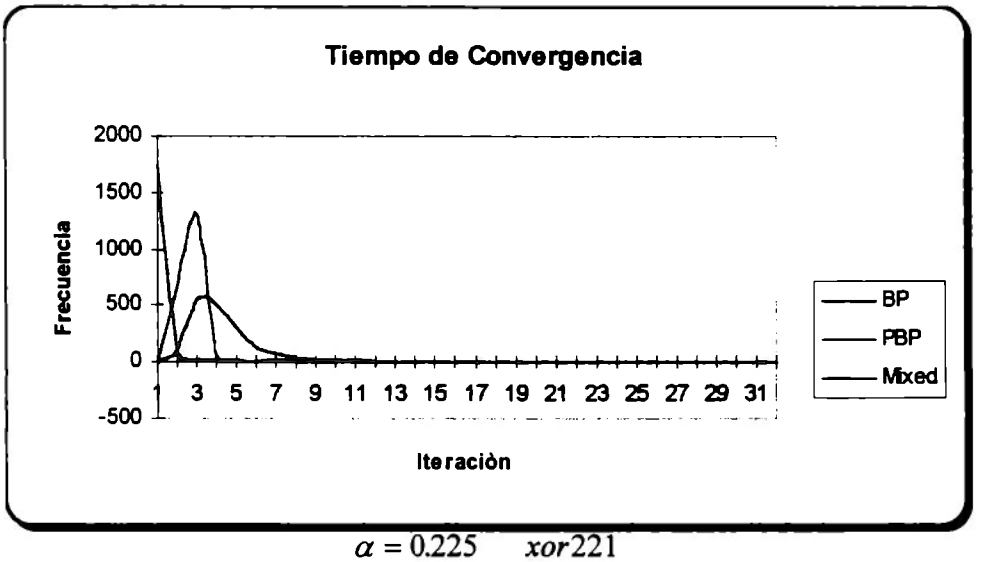
$\alpha = 0.15 \qquad xor221$



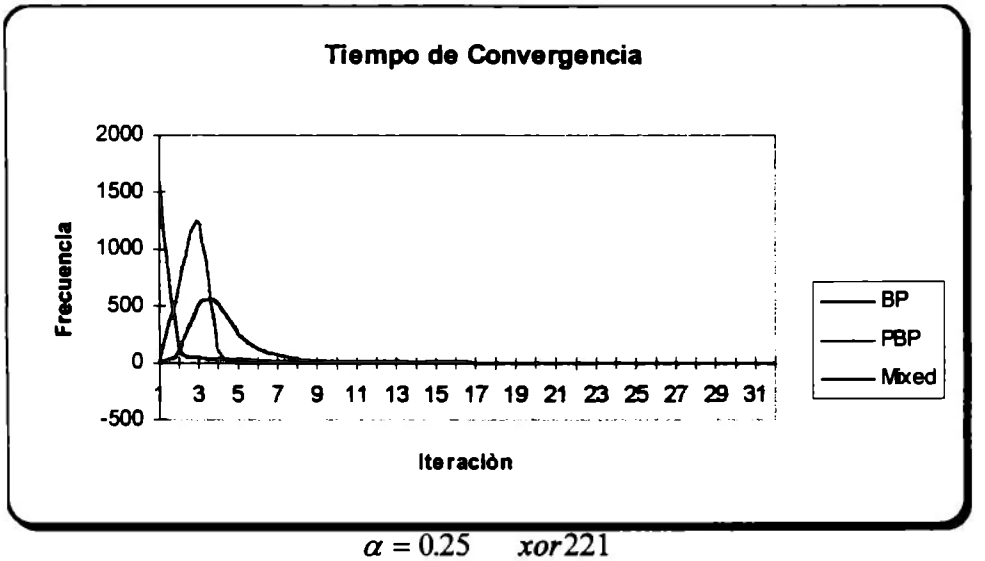
$\alpha = 0.175 \qquad xor221$



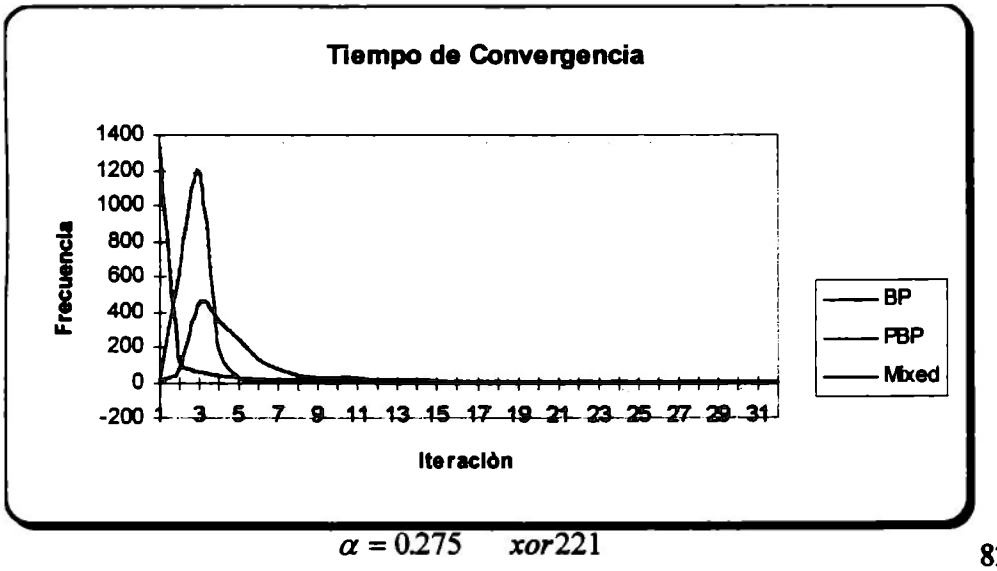
$\alpha = 0.20 \qquad xor221$



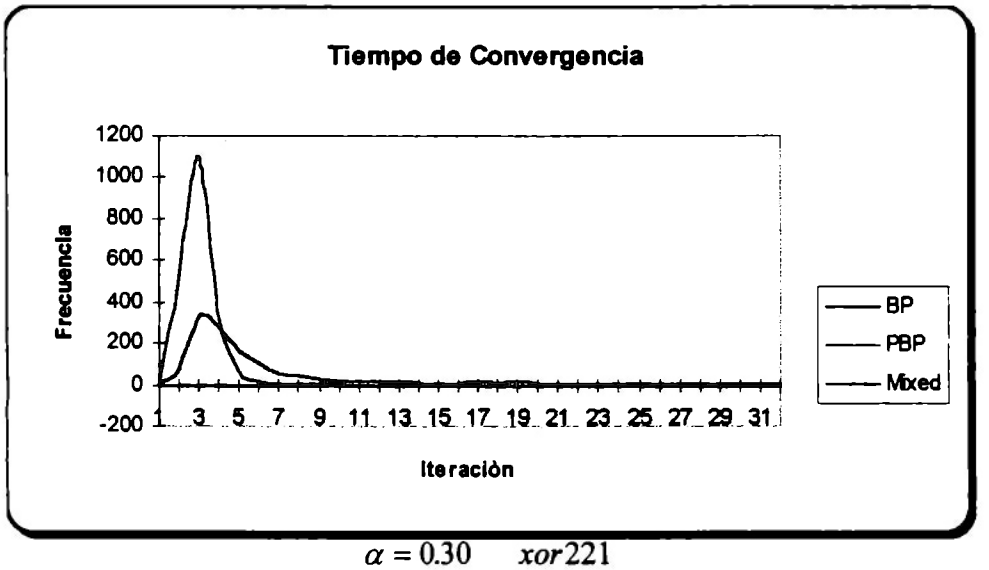
$\alpha = 0.225 \quad xor221$



$\alpha = 0.25 \quad xor221$



$\alpha = 0.275 \quad xor221$

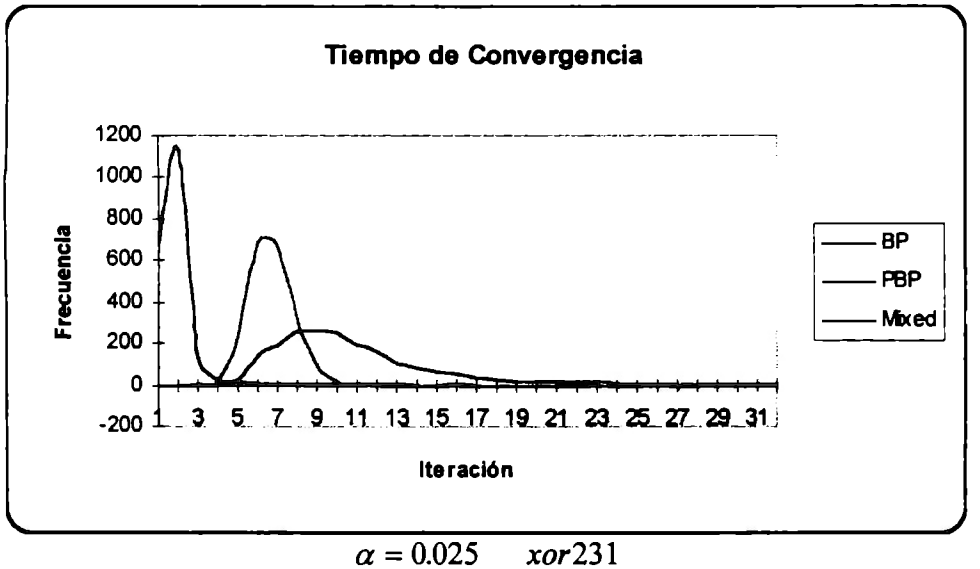


$\alpha = 0.30 \qquad xor221$

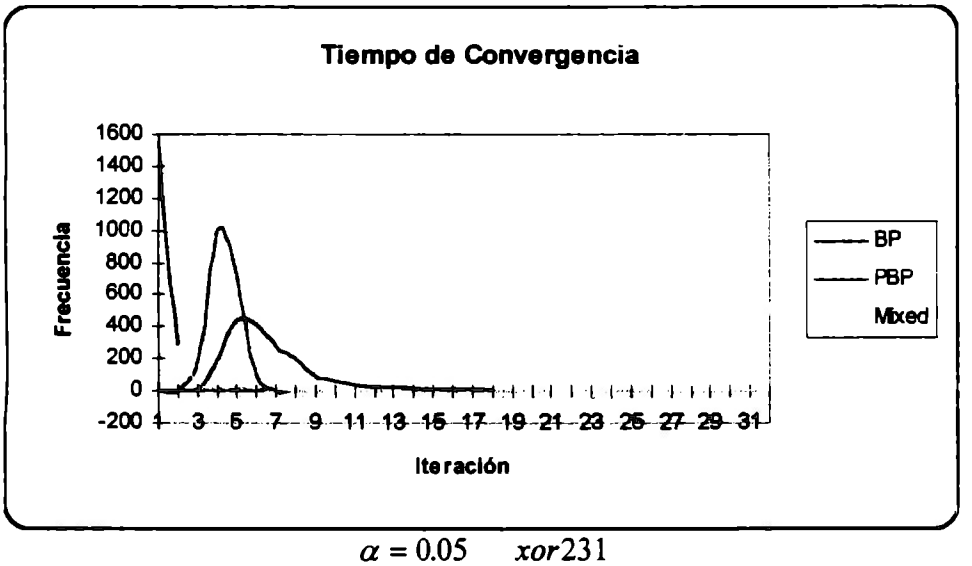
5.3. XOR231

Datos de entrada:

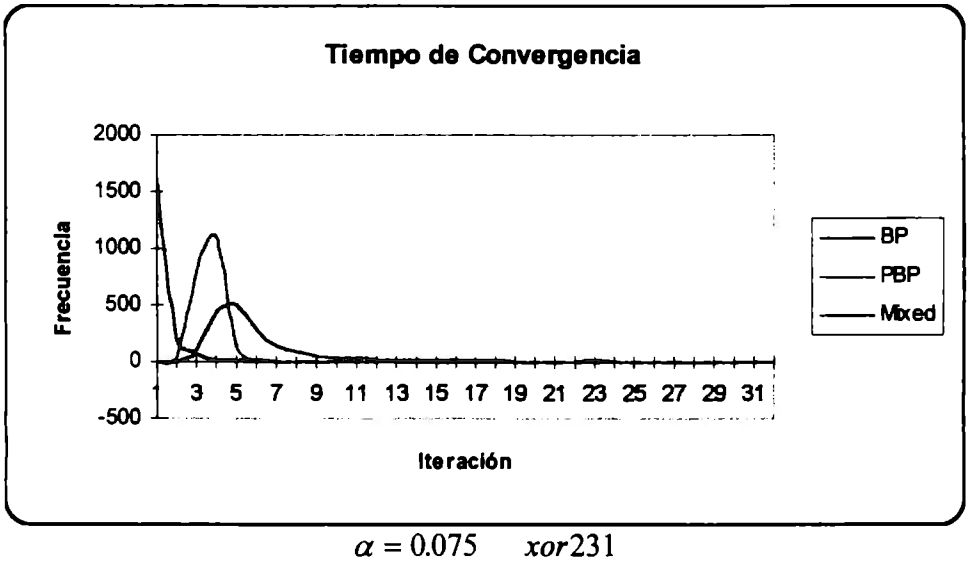
1.0	1.0	-1.0
1.0	-1.0	1.0
-1.0	1.0	1.0
-1.0	-1.0	-1.0



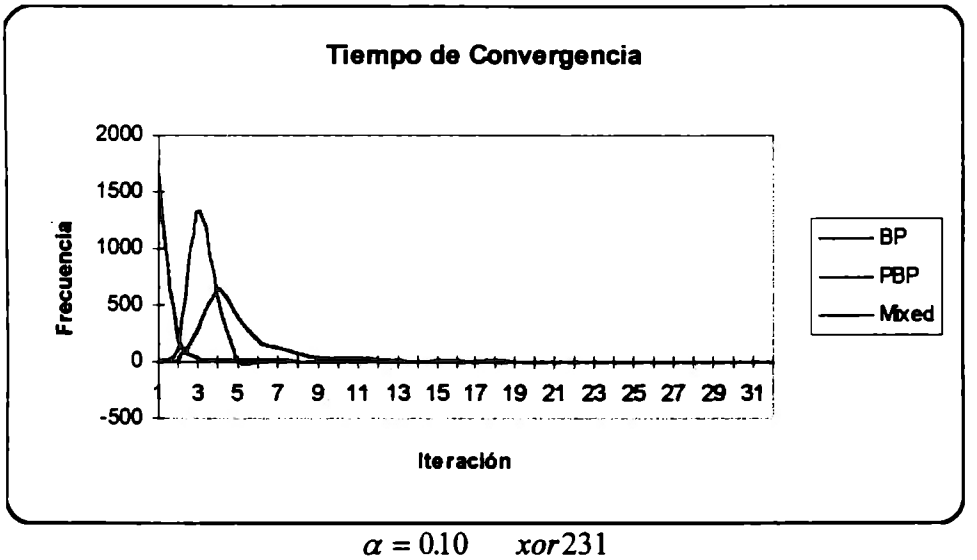
$\alpha = 0.025 \quad xor231$



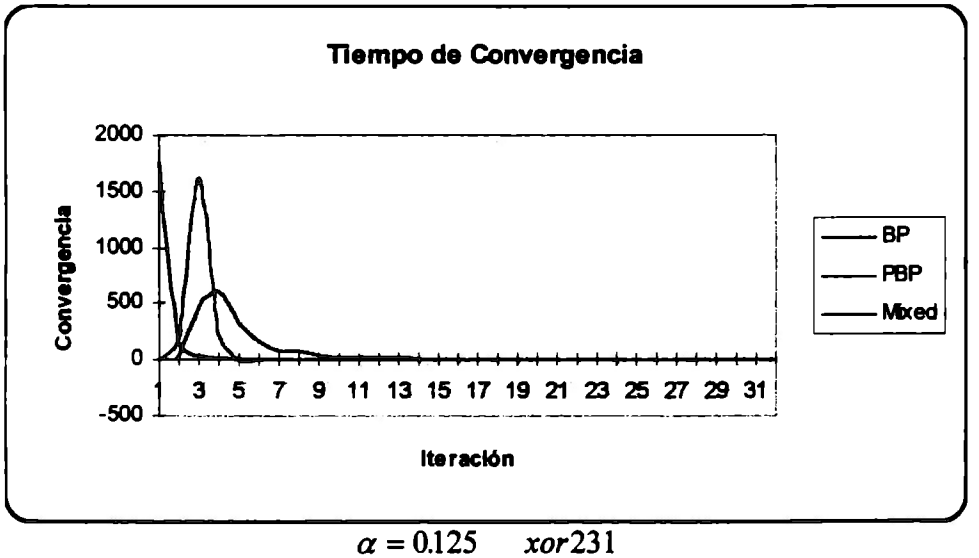
$\alpha = 0.05 \quad xor231$



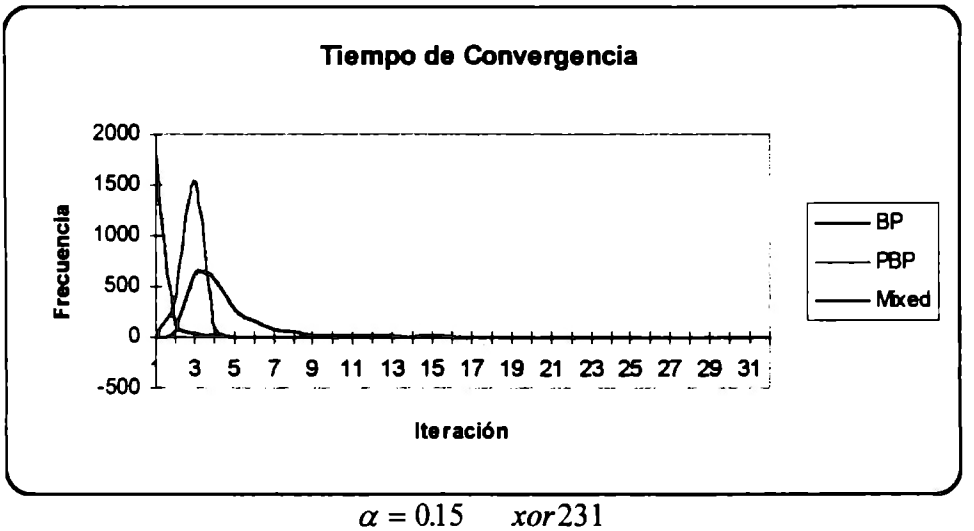
$\alpha = 0.075 \qquad xor231$



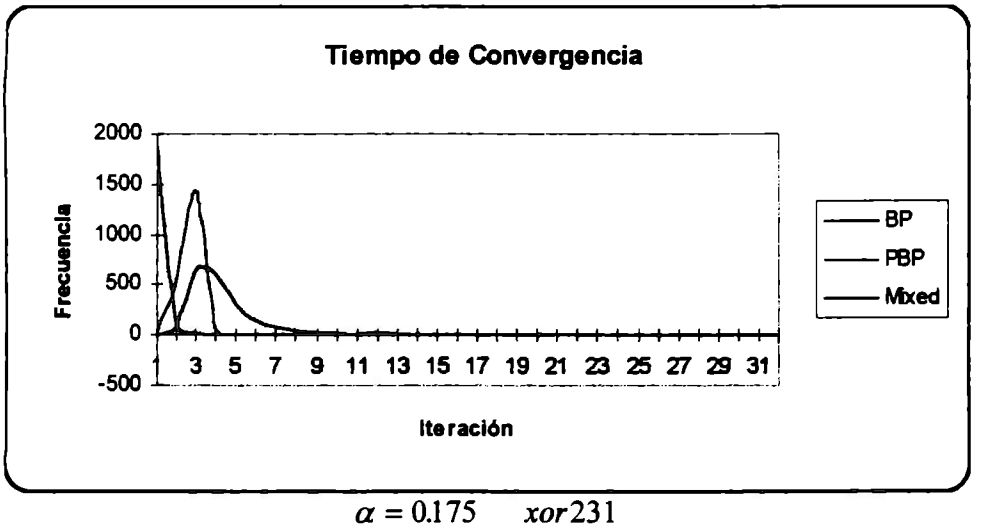
$\alpha = 0.10 \qquad xor231$



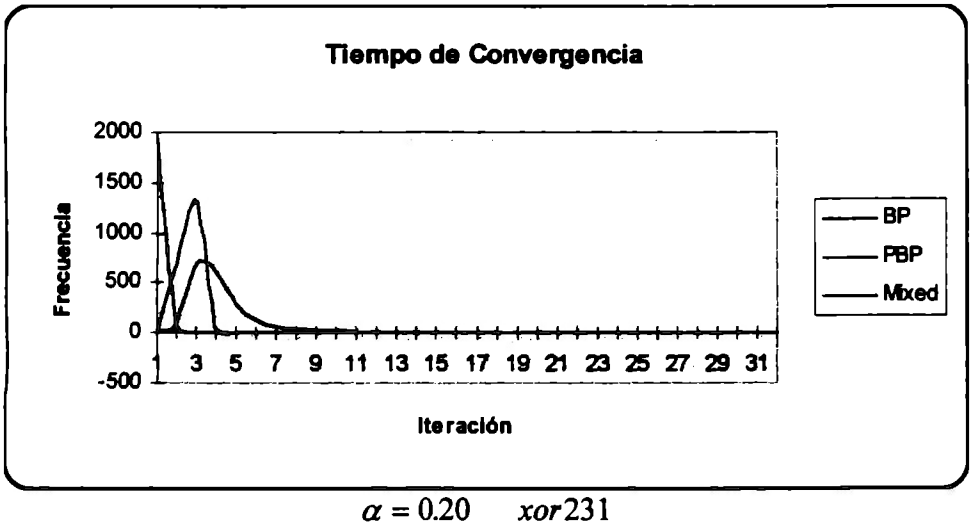
$\alpha = 0.125 \qquad xor231$



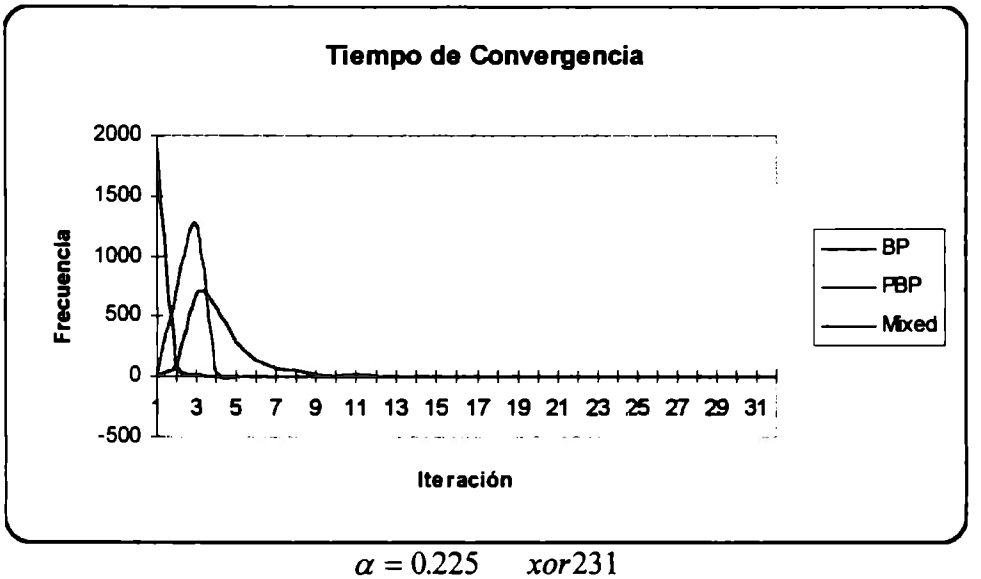
$\alpha = 0.15 \quad xor231$



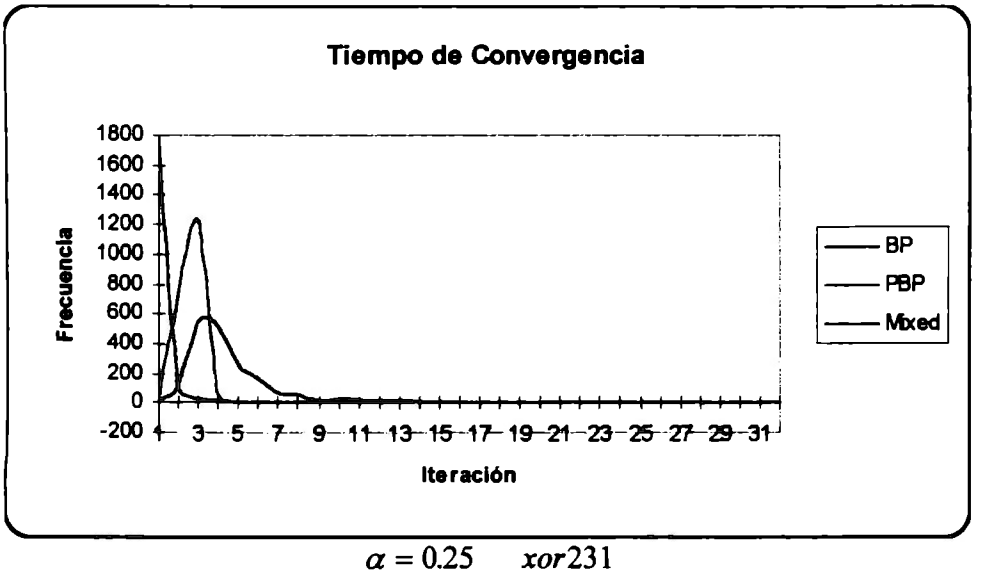
$\alpha = 0.175 \quad xor231$



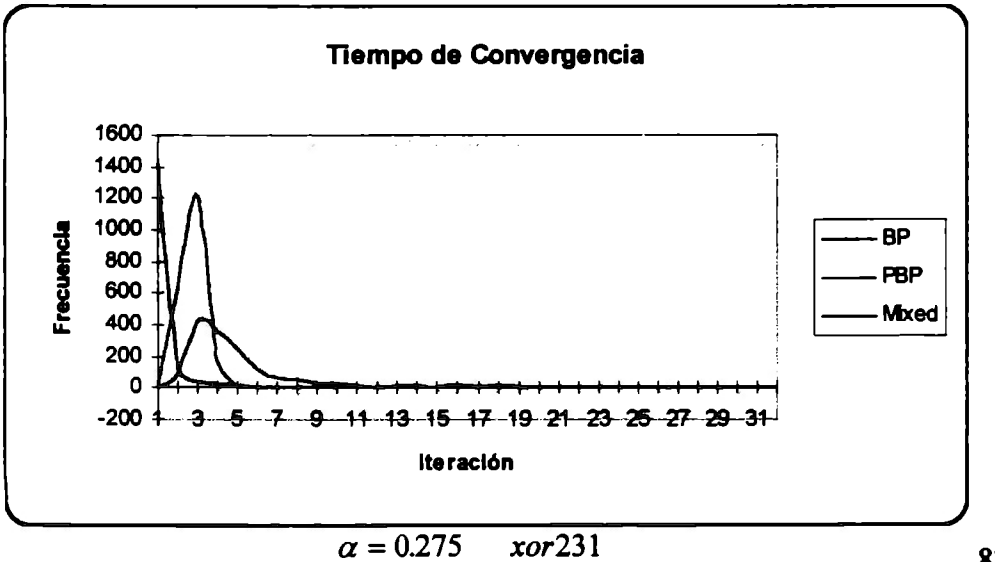
$\alpha = 0.20 \quad xor231$



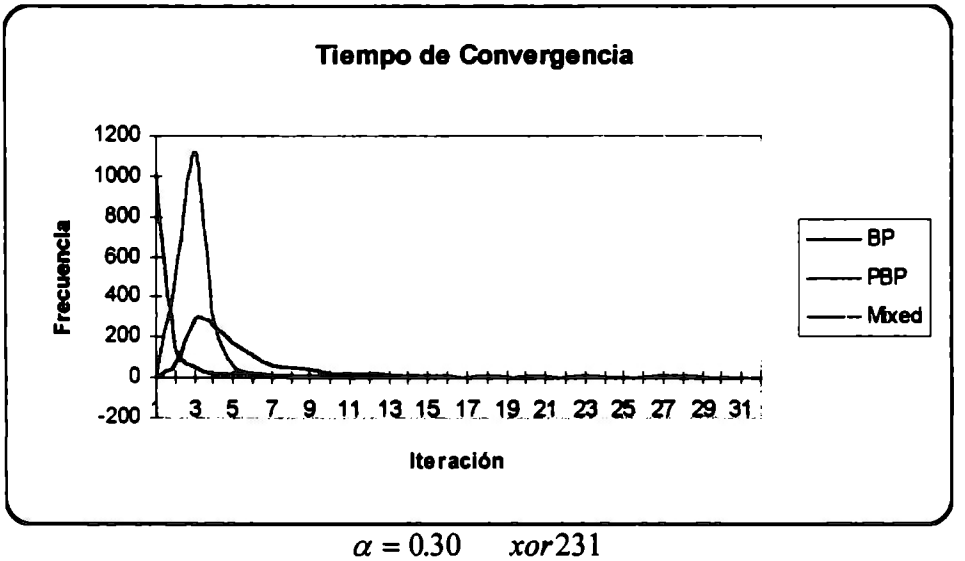
$\alpha = 0.225 \quad xor231$



$\alpha = 0.25 \quad xor231$



$\alpha = 0.275 \quad xor231$



$\alpha = 0.30 \qquad xor231$

ESTADISTICAS

PARITY3

RAZON DE APRENDIZAJE	BP	BP Modificado	Mixed BP
0.025	P = 0.85 X = 1092.042 σ = 356.08	P = 1.00 X = 690.07 σ = 122.92	P = 0.98 X = 193.78 σ = 208.54
0.050	P = 0.88 X = 767.45 σ = 344.38	P = 1.00 X = 442.42 σ = 76.82	P = 0.99 X = 148.19 σ = 199.03
0.075	P = 0.94 X = 656.03 σ = 339.76	P = 1.00 X = 354.24 σ = 56.70	P = 1.00 X = 93.40 σ = 97.39
0.100	P = 0.98 X = 555.97 σ = 282.40	P = 1.00 X = 306.77 σ = 48.88	P = 1.00 X = 69.42 σ = 62.64
0.125	P = 0.99 X = 510.59 σ = 260.82	P = 1.00 X = 285.15 σ = 45.33	P = 1.00 X = 59.45 σ = 57.00
0.150	P = 0.98 X =484.72 σ = 238.98	P = 1.00 X = 270.71 σ = 44.65	P = 1.00 X = 60.81 σ = 79.54
0.175	P = 0.89 X = 478.96 σ = 254.02	P = 1.00 X = 268.10 σ = 48.09	P = 0.93 X = 74.78 σ = 154.24
0.200	P = 0.63 X =499.54 σ = 281.80	P = 1.00 X = 290.96 σ = 78.48	P = 0.67 X = 110.28 σ = 228.63
0.225	P = 0.31 X = 529.07 σ = 330.90	P = 0.95 X = 375.54 σ = 195.11	P = 0.31 X = 102.48 σ = 212.16
0.250	P = 0.083 X = 504.21 σ = 348.78	P = 0.53 X = 468.33 σ = 298.77	P = 0.089 X = 94.25 σ = 147.72
0.275	P = 0.019 X = 471.97 σ = 295.76	P = 0.14 X = 466.77 σ = 267.91	P = 0.022 X = 90.20 σ = 221.21
0.300	P = 0.001 X = 234.67 σ = 42.84	P = 0.04 X = 496.96 σ = 324.54	P = 0.005 X = 70.4 σ = 42.45

XOR221

RAZON DE APRENDIZAJE	BP	BP Modificado	Mixed BP
0.025	P = 0.78 X = 799.80 σ = 328.72	P = 1.00 X = 474.45 σ = 100.08	P = 0.91 X = 169.43 σ = 266.75
0.050	P = 0.79 X = 546.88 σ = 335.49	P = 1.00 X = 290.71 σ = 62.97	P = 0.94 X = 132.1 σ = 270.1
0.075	P = 0.83 X = 477.37 σ = 348.25	P = 1.00 X = 228.42 σ = 46.00	P = 0.95 X = 132.1 σ = 270.52
0.100	P = 0.89 X = 434.78 σ = 350.60	P = 1.00 X = 194.25 σ = 37.21	P = 0.97 X = 115.33 σ = 252.38
0.125	P = 0.91 X = 389.04 σ = 328.53	P = 1.00 X = 173.08 σ = 32.65	P = 0.99 X = 100.26 σ = 223.80
0.150	P = 0.94 X = 394.20 σ = 285.85	P = 1.00 X = 159.63 σ = 29.38	P = 1.00 X = 80.26 σ = 164.50
0.175	P = 0.95 X = 320.66 σ = 262.66	P = 1.00 X = 149.66 σ = 28.56	P = 1.00 X = 68.34 σ = 154.82
0.200	P = 0.95 X = 284.74 σ = 218.73	P = 1.00 X = 143.07 σ = 26.97	P = 1.00 X = 73.10 σ = 175.78
0.225	P = 0.92 X = 286.10 σ = 242.13	P = 1.00 X = 139.32 σ = 26.99	P = 0.99 X = 89.92 σ = 219.10
0.250	P = 0.89 X = 309.52 σ = 288.72	P = 1.00 X = 140.98 σ = 30.77	P = 0.96 X = 102.66 σ = 224.40
0.275	P = 0.76 X = 340.80 σ = 327.36	P = 1.00 X = 146.97 σ = 40.39	P = 0.884 X = 143.21 σ = 302.52
0.300	P = 0.61 X = 382.69 σ = 378.25	P = 1.00 X = 16.79 σ = 59.15	P = 0.76 X = 152.5 σ = 200.34

XOR231

RAZON DE APRENDIZAJE	BP	BP Modificado	Mixed BP
0.025	P = 0.96 X = 651.46 σ = 255.44	P = 1.00 X = 394.29 σ = 67.76	P = 0.98 X = 91.96 σ = 110.10
0.050	P = 0.93 X = 433.15 σ = 261.19	P = 1.00 X = 247.78 σ = 44.11	P = 0.97 X = 77.04 σ = 171.41
0.075	P = 0.92 X = 392.48 σ = 283.92	P = 1.00 X = 200.93 σ = 36.05	P = 0.98 X = 86.50 σ = 209.60
0.100	P = 0.95 X = 337.65 σ = 250.40	P = 1.00 X = 175.26 σ = 30.79	P = 0.98 X = 72.60 σ = 172.65
0.125	P = 0.97 X = 318.57 σ = 252.27	P = 1.00 X = 159.62 σ = 27.43	P = 0.99 X = 63.00 σ = 142.85
0.150	P = 0.98 X = 290.17 σ = 227.25	P = 1.00 X = 147.89 σ = 27.19	P = 1.00 X = 50.99 σ = 94.03
0.175	P = 0.99 X = 272.98 σ = 203.17	P = 1.00 X = 140.83 σ = 25.19	P = 1.00 X = 39.47 σ = 38.10
0.200	P = 0.99 X = 252.82 σ = 169.54	P = 1.00 X = 136.76 σ = 25.07	P = 1.00 X = 37.16 σ = 36.40
0.225	P = 0.98 X = 258.00 σ = 184.39	P = 1.00 X = 135.90 σ = 26.63	P = 0.99 X = 43.46 σ = 81.38
0.250	P = 0.93 X = 283.39 σ =219.33	P = 1.00 X = 136.73 σ = 27.53	P = 0.96 X = 71.00 σ = 189.26
0.275	P = 0.77 X = 327.27 σ = 281.48	P = 1.00 X = 144.60 σ = 35.65	P = 0.85 X = 101.24 σ = 249.42
0.300	P = 0.61 X = 379.67 σ = 341.83	P = 0.999 X = 163.28 σ = 75.65	P = 0.67 X = 136.69 σ = 298.40

Conclusiones

Una vez realizada la siguiente investigación podemos hacer algunas afirmaciones que creemos contribuirán al desarrollo en el área de las redes neuronales:

- Dentro del área de redes neuronales existen múltiples trabajos que plantean diversos métodos para acelerar la velocidad de convergencia durante la fase de aprendizaje en redes neuronales feed-forward, pero ninguno de ellos busca acelerar la velocidad de convergencia para la fase de aprendizaje por un método alternativo al del paralelismo puramente funcional.

- Se determinaron métodos de paralelización de redes feed-forward, fundamentados para el análisis del mejor método, en el tiempo de entrenamiento mínimo bajo la consideración de propiedades estadísticas, como el valor esperado y varianza.

- Como parte de este estudio se han definido tres dominios para estudiar el comportamiento de las redes neuronales feed-forward en forma paralela. En el capítulo cuatro se presenta el estudio dentro del dominio de los datos, el dominio del espacio de soluciones y el dominio aritmético.

- Tomamos como solución del problema considerado, la paralelización en el dominio aritmético, pero haciendo la salvedad que la paralelización en el dominio de los datos es óptimo en aquellos casos en que el número de neuronas es grande (por ejemplo mayor que 512 neuronas), de tal suerte que el tiempo de comunicación entre procesadores sea equivalente o menor con el tiempo de operaciones aritméticas.

- La solución de paralelización en el espacio aritmético con la consideración de que cada procesador actualice los parámetros después de la presentación de k patrones m veces es una posición intermedia entre los algoritmos Back-propagation y Back-propagation Modificado (BP-Modificado) en donde se toma el promedio de los parámetros de todos los procesadores.

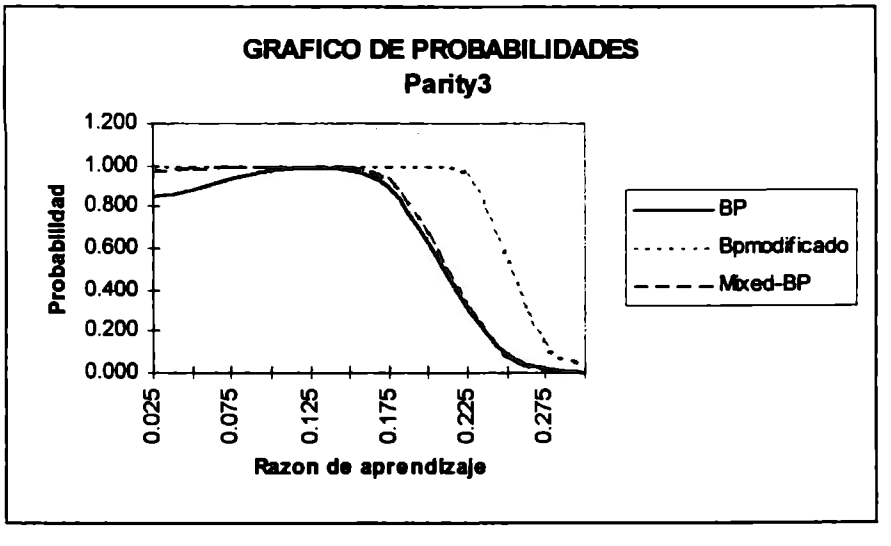
- El método de paralelización propuesto se encarga de realizar en forma independiente para cada procesador el entrenamiento en la fase de propagación hacía adelante, pero tiene en cuenta el resultado de todos los procesadores en la fase de propagación hacia atrás, con el fin de acumular el conocimiento que se ha ido acumulando en cada procesador a lo largo de un período de tiempo, optimizando de esta forma el aprendizaje global de la red neuronal.

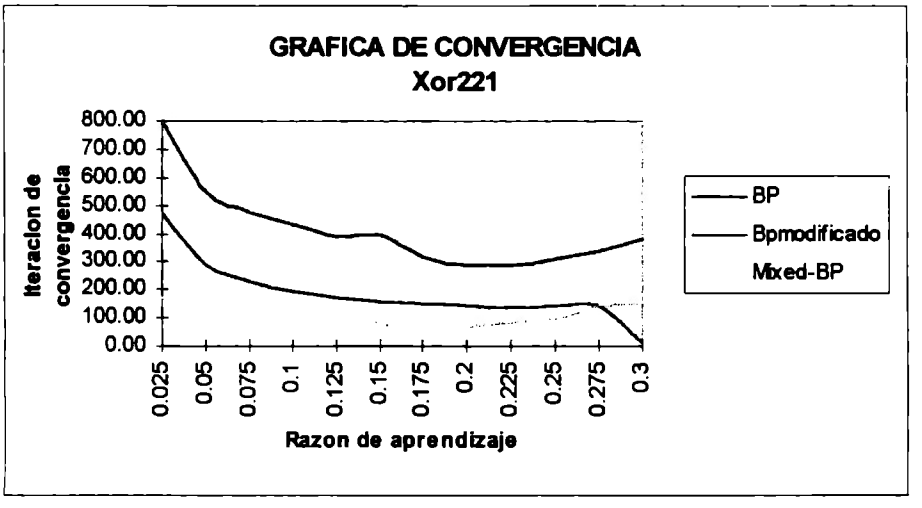
- Debido a que el método de paralelización seleccionado considera la existencia de varios procesadores (sistema multiprocesador con memoria distribuida y comunicación a través de paso de mensajes), dichos valores

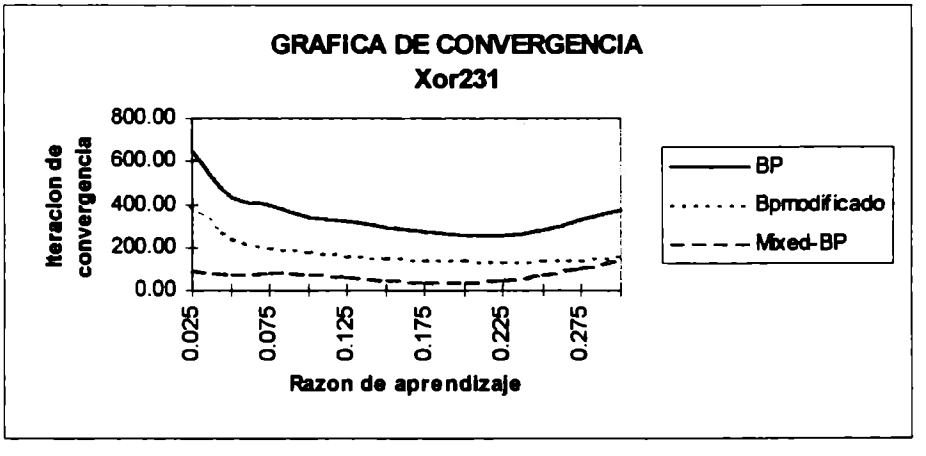
de pesos se promediaran con el fin de ajustar el valor del error hacia un valor medio, que en la teoría de las grandes muestras sería equivalente al valor más aconsejable.

- Una vez seleccionado el método y probado en tres ejemplos (Parity3 - Xor221 - Xor223) podemos concluir con base en los datos estadísticos que:

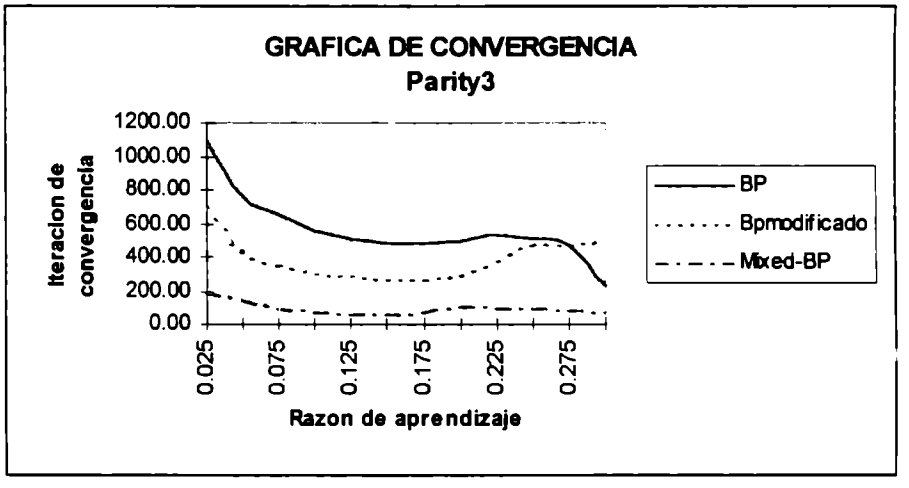
    - La probabilidad de convergencia de BP-Modificado es mejor que la de Mixed-BP y este a su vez mucho mejor que el método BP tradicional.







- Es claro que el tiempo de convergencia para todos los ejemplos seleccionados (medía aritmética) es mucho mejor para Mixed-BP, como se puede apreciar en la gráfica de tiempos de convergencia como valor medio:


GRAFICA DE CONVERGENCIA
Parity3
Iteracion de convergencia
1200.00
1000.00
800.00
600.00
400.00
200.00
0.00
0.025
0.075
0.125
0.175
0.225
0.275
Razon de aprendizaje
BP
Bpmodificado
Mixed-BP

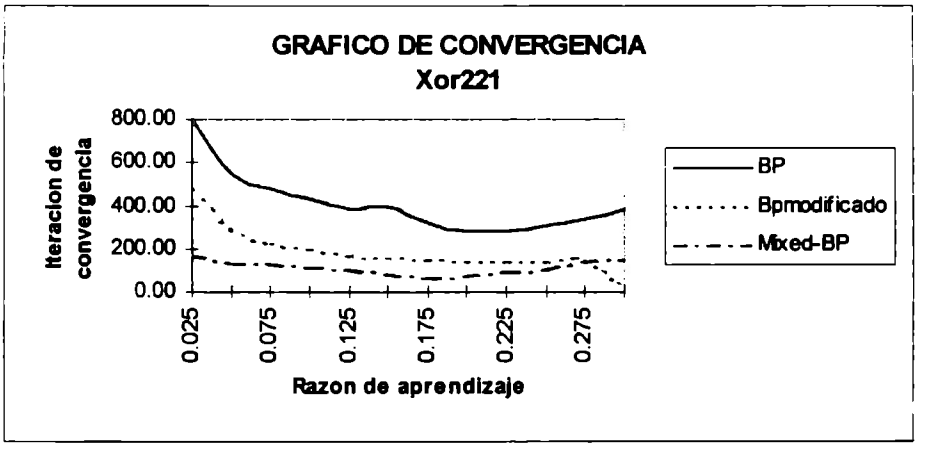


GRAFICO DE CONVERGENCIA
Xor221
Iteracion de convergencia
800.00
600.00
400.00
200.00
0.00
0.025
0.075
0.125
0.175
0.225
0.275
Razon de aprendizaje
BP
Bpmodificado
Mixed-BP

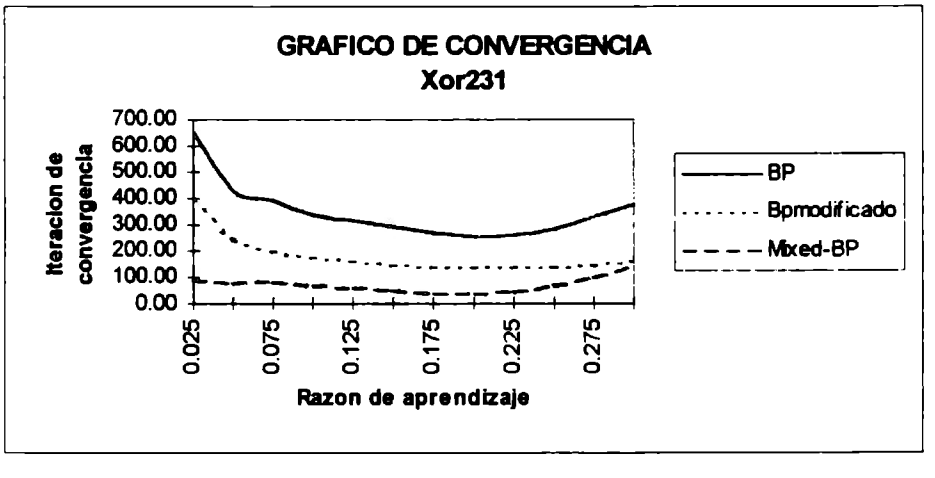


GRAFICO DE CONVERGENCIA
Xor231
Iteracion de convergencia
700.00
600.00
500.00
400.00
300.00
200.00
100.00
0.00
0.025
0.075
0.125
0.175
0.225
0.275
Razon de aprendizaje
BP
Bpmodificado
Mixed-BP

- El análisis estadístico nos lleva a concluir que el método propuesto como Mixed-BP es un método que converge en un rango [0.9-1.00] con un nivel de confianza de por lo menos 95% y el número de iteraciones necesarias para su convergencia siempre es menor que las utilizadas por BP o BP-modificado.

- Podemos hablar del método seleccionado (Mixed-BP) como una combinación de redes neuronales en las que el error medio cuadrático es el valor ponderado de cada una de ellas.

- Como continuación de esta investigación podemos hablar de estudiar diferentes métodos que sean combinaciones lineales óptimas de redes neuronales, en reemplazo de usar una sola red neuronal para el entrenamiento. La combinación lineal de redes neuronales consiste en formar sumas de las correspondientes salidas de la red y la combinación lineal es seleccionada de tal forma que se minimice el error medio cuadrático. Una vez se tenga un conjunto entrenado de redes neuronales se combinan las entradas de la red, lo cual ayuda a integrar el conocimiento y acelerar la precisión del modelo.

- Se pueden proponer métodos alternos en el cual las topologías y los parámetros de las diferentes redes neuronales sean diferentes. Al finalizar la etapa de entrenamiento se tendrá un como resultado un número finito de redes neuronales entrenadas. Sucede entonces que una de ellas es seleccionada como la mejor, basados en algún criterio de optimización,

mientras que las otras redes son descartadas. Adicional a esto podríamos pensar en conformar una superred con los mejores parámetros de cada red. Este planteamiento no es nuevo ya que Hashem y Schmeiser proponen formar una combinación lineal de las correspondientes salidas del conjunto de redes neuronales, ya que mediante su combinación se ayuda a la integración del conocimiento adquirido de manera individual y con frecuencia se producen resultados superiores en cuanto a precisión se refiere, comparado con los modelos individuales de entrenamiento de redes neuronales.

- Desde la perspectiva de las redes neuronales el combinar las salidas de las redes neuronales entrenadas, es similar a crear una gran red neuronal (HNN's) en la cual las redes neuronales son subredes operativas en paralelo de esta y la combinación de pesos es la conexión de pesos de la capa de salida.

- Las principales contribuciones del presente trabajo al área de las redes neuronales se encuentra básicamente en la solución que se le da al planteamiento del problema. El estudio de los diferentes métodos de paralelización: en el dominio de los datos, dominio del espacio de soluciones y dominio aritmético.

- Una vez se planteo el algoritmo de Back-Propagation paralelo, se estudio la complejidad computacional del algoritmo desde el punto de vista espacial y se calculo el radio de variación con respecto al algoritmo

secuencial. Se determino dejar este método como una opción, pero no la mejor, debido a los altos tiempo de comunicación incurridas en las operaciones all-to-all broadcast.

El método planteado y denominado MIXED-BP se constituye en un aporte valioso al tema de paralelización de las redes neuronales por la filosofía del mismo, que consiste en construir una super-red o red de redes neuronales, donde cada procesador trabaja en forma independiente durante un tiempo Δt y actualiza los parámetros después de la presentación de k patrones m veces (nótese que es una posición intermedia entre BP y BP-modificado). Luego de esto se actualizan los parámetros del modelo (super-red) según la siguiente estrategia: ***"tomar el promedio de los parámetros de todos los procesadores".***

- De igual forma es importante anotar que todas las pruebas fueron realizadas en máquinas PC con las siguientes características: 32MB RAM, 1.2GB DD y procesador Pentium de velocidad mayor de 100 Mhz. Los resultados obtenidos de varios procesadores actuando en forma paralela fueron simulados mediante programación y la plataforma bajo la que se desarrollaron los programas fue Borland C++ para Windows 95.

Bibliografía

[1] Alhaj, Ali M. *A Data-driven implementation of backpropagation learning algorithm.* Department of electronic Engineering. Department of Information Systems Engineering, Faculty of Engineering, Osaka University.

[2] Braun, H. and Riedmiller, M. (1992). *Rprop:a fast adaptative learning algorithm.* Proc. of the Symposium on Computer and Information Science VII.

[3] Cetin, Bedri C., Joel W. Burdick and Jacob Barhen. *Global descendent replaces gradient to avoid local minima problem in learning with artificial neural networks.* Department of Electrical Engineering, Mechanical Engineering and California Institute of Technology.

[4] Chen, J.R., and Mars, P,. *Stepsize variation methods for acceleration the backpropagation algorithm.* Proc. IJNN, Washington DC, 1990.

[5] Chung, Jai_hoon., Hyunsoo Yoon and Seung Ryoul Maeng. *Exploiting the inherent parallelisms of backpropagation neural networks to design a systolic array.* Department of Computer Science. Korea Advenced Institute of Science and Technology (KAIST).

[6] Cosnard, M., J.C. Mignot, H. Paugam-Moisy. Ecole Normale Superieure de Lyon, France. *Implementations of multilayer neural networks on parallel architectures.*

[7] Falhman, S.E. (1988). *Faster learning variations on backpropagation: An empirical study*. Conectionist Models Summer School. T.J. Sejnowski, G.E. Hinton and D.S. Touretzky.

[8] Falhman, S.E. (1991). *The recurrent cascade correlation learning architecture*. Technical Report CMU-CS-91-100, School of Computer Science. Carnegie Mellon University.

[9] Freeman, James A. & Skapura David M. *Neural Networks*. Addsion Wesley, 1993.

[10] Hebb, D.O., *The Organization of Behavior, a Neuropsychological Theory*, John Wiley, New York, 1949.

[11] Hecht-Nielsen, R., *Neurocomputing*, Addsion-Wesley Publishing Co., Reading, MA, 1990.

[12] Him, Y.K. and J.B. Ra. *Weight value initialization for improving training speed in the backpropagation network*. Department of Electrical Engineering, Korea Advanced of science and Technology.

[13] Hetz, J., A. Krogh and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Co., Redwood City, CA, 1991.

[14] Hyunsoo, Yoon., Jong H. Nang., S.R. Maeng. *A distributed backpropagation algorithm of neural networks on distribuited-memory multiprocessors*. University of Maryland. College Park.

[15] Hoefeld, M and Falhman, S.E. (1991) *Learning with limited numerical precision using cascade correlation algorithm.* Technical Report CMU-CS-91-130, School of Computer Science. Carnegie Mellon University.

[16] Hopffield, J. John. *Neural Networks and physical system with collective computational abilities.* Proc. Natl. Acad. Sci. USA. 79, pp. 2254-2558, Abril de 1982. Biofisica.

[17] Hwan Kai & Fayé A. Briggs. *Computer Architecture and Parallel Processing.* McGrawHill, Inc., USA, 1987.

[18] Jacobs, R.A. *Increased rate of convergence through learning rate adaptation.* Neural Networks, 1988.

[19] Kashi, Kimmo and Jukka Vanhala. *Neural networks and MIMD-multiprocessors.* Tampere University of Technology. Microelectronics Laboratory.

[20] LI, Gang. Hussein Alnuweiri and Yuejian Wu. Despartment of Electrical Engineering and Department of Computer Science. University of Columbia. 1993. *Acceleration of back-propagation through initial weight pre-training with delta rule.*

[21] Majumder, A., and R. Dandapani. Motorola Inc and Department of ECE University of Colorado Springs. *Neural networks as massively parallel automatic test pattern generators.*

[22] McCulloch, W.S., and W. Pitts, *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Math. Bio., Vol. 5, 1943, pp. 115-133

[23] Minsky, M., and Papert, S., *Perceptrons,* MIT Press, Cambridge, MA, 1969.

[24] Prados, Donald L. *A fast supervised-learning algorithms for large, multilayered neural networks.* Electrical Engineering Department. University of New Orleans.

[25] Okawa, Yoshikuni and Takayuki Suyama. *Paralell computation of neural networks in a processor pipeline with partially shared memory.* NTT Communication Science Laboratories.

[26] Omidvar, O.M. *An Implementation of backpropagation algorithm on a massively parallel processor.* University of the District of Columbia. Washington, DC. National Institute of Standard and Technology. Gaithersburg, Maryland.

[27]Park, D.J., B.E. Jun and J.H. Kim.. *Novel fast training algorithm for multilayer feedforward neural network.*

[28] Perez, M. Yolanda and Dilip Sarkar. Department of Mathematics and Computer Science. University of Miami. 1993. *Back-propagation algorithm with controlled oscillation of weights.*

[29] Petrowski, Alain., Gerard Dreyfus, Senior Member, IEEE, and Claude Girault. *Performance analysis of a pipelined backpropagation parallel algorithm.*

[30] Qiu, G., M.R. Varley and T.J. Terrell. *Accelerated training of backpropagation networks by using adaptive momentum step.*

[31] Riedmiller, Martin and Heinrich Braun. Institute fur Logik, *Komplexitat und Deduktionssyteme. A direct adaptive methods for faster backpropagation learning: The RPROP algorithm.*

[32] Rumelhart, D.E., Hinton, G.E., and William R.J., *Parallel Distributed Processing.*

[33] Rumelhart, D.E and J.L. McClelland, *Parallel Distributed Processiing.* MIT Cambridge.

[34] Rosenblatt, F., *The Perceptrons : a probabilistic model for information storage and organization in the brain*, Physhol. Rev., Vol. 65, 1958, pp. 386-408.

[35] Rumelhart, D.E., G.E. Hinton, and R.J. Williams. *Learning internal representation by error propagation*, in Rumelhart, D.E. and McCleland, J.L. [Eds], Parallel Distributed Processing, MIT Press, Cambridge, MA, 1986, Vol.1 pp.318-362.

[36] Sánchez-Guzmán R.A, Mayol-Cuevas W.W & J. Figueroa. Laboratorio de Sistemas Complejos. *Optimización de muestreo en la capa de entrada de redes neuronales utilizando optimización por algoritmo genetico.* Universidad del Valle de México-CEM.

[37] Sandoval, carlos A. *A modified learning algorithm for back-propagation networks.* Computer Science. ITESM-CEM.

[38] Szu, Harold, Charles Yeh, George Rogers, Michael Jenkins and Ali Farsaie. Naval Surface Warfare Center. Naval Postgraduate School. *Speed up performances on MIMD Machines.*

[39] Weir, M.K. *A method for self-determination of adaptative learning rates in backpropagation.* Neural Networks, 1991.

[40] Wohl, Peter and Thomas W. Cristopher. Department of Computer Science. Illinois Institute of Technology. IIT Center Chicago. *MIMD implementation of neural networks through pipelined, parallel communication trees.*

[41] Werbos, P.J., *Beyond Regression : New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. Thesis, Harvard University, 1974.

[42] Widrow, B., *Generalizations and Information Storage in Network of Adaline 'Neurons'.*, Spartan Books, Washington, DC, 1962, pp. 435-461.

[43] Widros, B. and M.E. Hoff Jr., *Adaptative Switching Circuits*, 1960 IRE Western Electronic Show and Convention Record, New York, 1960, pp. 96-104.

[44] Xu, H.Y., G.Z. Wang, and C.B.Baird. *A fuzzy neural networks technique with fast backpropagation learning*. International Joint Conference on Neural Networks, 1992.

[45] Yoon, Hyunsoo., Jong H. Nang and S.R. Maeng. *A distribuited backpropagation algorithm of neural networks on distribuited-memory multiprocessors*. Center for Artificial Intelligence Research & Department of Computer Science. Korea Advanced Institute of Science and Technology.

[46] Zell, Andres., Niels Mache, Michael Vogt, Markus Huttel. *Problem of massive parallelism in neural networks simulation*. University of Sttugart. Institute for parallel and distribuited high performance systems (IPVR). Germany.

Anexo A

Programas Fuentes

Estructuras de datos de BPN

Para la representación de la red neuronal se necesitan básicamente una representación de su estado y una del proceso. Para su representación utilizamos diferentes estructuras que a continuación se definen.

Variables del sistema

float **net;

Estructura de datos que contiene las salidas netas de cada neurona para cada una de las capas de la red, es decir es una estructura matricial de la forma $net[i,j]$, donde i representa la capa y j es la j-ésima neurona de la capa i. En forma matemática se puede ver como: $neta_{pk}{}^{o} = \sum_{j=1}^{I} W_{kj}{}^{o} I_{pj} + \theta_{k}{}^{o}$

float **output;

Estructura de datos que contiene las salidas de cada nodo, es decir, es el resultado de aplicar $f(net[i,j])$. Dicha estructura existe por capa y por nodo respectivamente. En forma matemática se puede ver como: $O_{pk} = f_k^{\,o}(neta_{pk}^{\,o})$.

float **umbral;

Estructura de datos que contiene los valores de los umbrales para cada neurona de la red. Existe un valor de umbral por cada nodo de la red.

float **dumbral;

Estructura de datos que contiene los valores de las variaciones de los umbrales para cada neurona de la red.

float ***weight;

Estructura de datos que representa los valores de pesos de la red neuronal. Existe para cada capa de la red y representa el valor de pesos entre pares de neuronas de capas sucesivas, de tal forma que $weight[c][n\_i][n\_j]$, donde c es la capa asociada y la conexión existe entre la neurona i y la neurona j. En forma matemática se puede ver como: $W_{ji}^{\,indice}$ que representa el peso de la conexión procedente de la i-ésima unidad de entrada.

float ***dweight;

Existe para cada capa de la red y representa las modificaciones de pesos entre pares de neuronas de capas sucesivas, de tal forma que $dweight[c][n\_i][n\_j]$ es el valor en que se deberá modificar el vector de pesos, tal que c es la capa asociada y la conexión existe entre la neurona i y la neurona j. En forma matemática se puede ver como: $\Delta_p W_{kj}^{o}(t) = \eta(y_{pk} - o_{pk}) f_k^{o'}(neta_{pk}^{o}) I_{pj}$.

float **pattern;

Estructura de datos que representa el conjunto de patrones de entrenamiento. Representan los valores de X_{pi}, donde i representa el nodo del *p-ésimo* patrón.

float **teacher;

Estructura de datos que representa las salidas deseadas para cada patrón de entrenamiento. En forma matemática se puede ver como y_{pk}, que es la salida del *k-ésimo* nodo del patrón p.

int patterindex;

Determina un patrón de entrenamiento a utilizar del conjunto de patrones (pattern) de manera aleatoria.

float error;

Estructura de datos que contiene el valor total de la red dado un patrón de entrada, es equivalente a $E_p = \sum_{k=1}^{M} \delta_{pk}^{2}$

float eta;

Factor o coeficiente de aprendizaje de la red (η).

float beta;

Valor de momento de la red (α).

float nlayer;

Numero de capas de la red.

float nneuron[n_neuronas];

Numero de neuronas de la red especificadas por capa. *Existen asignaciones dinámicas dentro del programa, razón que permite realizar la asignación indispensable de memoria para la arquitectura de la red seleccionada.*

float npattern;

Numero de patrones de entrenamiento.

float epsilon;

Valor de error máximo deseado de la red (ε).

Funciones del sistema

float difsigmoid(float x);

Función encargada de calcular el valor de $f(net[i,j])$ utilizando para ello la función sigmoidal. Podemos escribir este proceso en código C en la siguiente forma:

```
float difsigm(float x)
  {
   float auxiliar;
   auxiliar=exp(x);
   return (4.0/sq(auxiliar+1.0/auxiliar));
  }
```

void StartModel(void);

Función encargada de hacer la asignación dinámica de memoria para las estructuras dinámicas del sistema. Podemos escribir este proceso en código C en la siguiente forma:

```
void StartModel(void)
 {
  int i,j,k;
  FILE *input;
  net=(float **)calloc(nlayer,sizeof(float *));
  output=(float **)calloc(nlayer,sizeof(float *));
  umbral=(float **)calloc(nlayer,sizeof(float *));
  dumbral=(float **)calloc(nlayer,sizeof(float *));
  delta=(float **)calloc(nlayer,sizeof(float *));
  weight=(float ***)calloc(nlayer,sizeof(float **));
  dweight=(float ***)calloc(nlayer,sizeof(float **));

  for(i=0;i<nlayer;++i)
  {
       net[i]=(float *)calloc(nneuron[i],sizeof(float));
```

```
        output[i]=(float *)calloc(nneuron[i],sizeof(float));
        umbral[i]=(float *)calloc(nneuron[i],sizeof(float));
        dumbral[i]=(float *)calloc(nneuron[i],sizeof(float));
        delta[i]=(float *)calloc(nneuron[i],sizeof(float));
        for(j=0;j<nneuron[i];++j)
        {
          umbral[i][j]=randomfloat;
          dumbral[i][j]=0.0;
        }
    }

    for(i=0;i<nlayer-1;++i)
    {
     weight[i]=(float **)calloc(nneuron[i],sizeof(float *));
     dweight[i]=(float **)calloc(nneuron[i],sizeof(float *));
     for(j=0;j<nneuron[i];++j)
     {
      weight[i][j]=(float *)calloc(nneuron[i+1],sizeof(float));
      dweight[i][j]=(float *)calloc(nneuron[i+1],sizeof(float));
      for(k=0;k<nneuron[i+1];++k)
      {
       weight[i][j][k]=randomfloat;
       dweight[i][j][k]=0.0;
      }
     }
    }

    pattern=(float **)calloc(npattern,sizeof(float));
    teacher=(float **)calloc(npattern,sizeof(float));

    for(i=0;i<npattern;++i)
    {
     pattern[i]=(float *)calloc(nneuron[0],sizeof(float));
     teacher[i]=(float *)calloc(nneuron[nlayer-1],sizeof(float));
    }
    input=fopen(inputfilename,"r");

    for(i=0;i<npattern;++i)
    {
     for(j=0;j<nneuron[0];++j)
     fscanf(input,"%f",&pattern[i][j]);
     for(j=0;j<nneuron[nlayer-1];++j)
     fscanf(input,"%f",&teacher[i][j]);
    }
    fclose(input);
}
```

void FreeModel(void);

Función encargada de liberar la memoria de la computadora.

```
void FreeModel(void)
 {
  int i,j;
  for(i=0;i<nlayer;++i)
  {
   free(net[i]);
   free(output[i]);
   free(umbral[i]);
   free(dumbral[i]);
   free(delta[i]);
  }
  for(i=0;i<nlayer-1;++i)
  {
   for(j=0;j<nneuron[i];++j)
   {
    free(weight[i][j]);
    free(dweight[i][j]);
   }
   free(weight[i]);
   free(dweight[i]);
  }

  for(i=0;i<npattern;++i)
  {
   free(pattern[i]);
   free(teacher[i]);
  }

  free(net);
  free(output);
  free(umbral);
  free(dumbral);
  free(delta);
  free(weight);
  free(dweight);
  free(pattern);
  free(teacher);
  }
```

void ForwardTrans(void);

Rutina encarga de efectuar la propagación de la red hacía adelante. Podemos escribir este proceso en código C en la siguiente forma:

```
void ForwardTrans(void)
        {
        int i,j,k;

        /* En este capa no existen conexiones anteriores,          */
        /* la salida es el resultado de sumar cada uno de          */
         * los patrones con el valor del umbral por neurona         */

        for(i=0;i<nneuron[0];++i)
                {
                net[0][i]=pattern[patternindex][i]+umbral[0][i];
                 *
                output[0][i]=sigmoid(net[0][i]);
                }

        /* La salida es el producto de la salida de cada           */
        /* neurona por el peso asociado a las dos neuronas         */
        /* adicionándole el valor del umbral y seguidamente        */
        /* aplicamos la función sigmoidal a este valor             */
        /* este proceso se hace para cada capa y cada nodo         */
        /* neta_pj^h = Σ_{i=1}^{n} W_ji^h X_pi + θ_j^h ; I_pj = f_j^h(neta_pj^h)  */

        for(i=0;i<nlayer-1;++i)
                {
                for(k=0;k<nneuron[i+1];++k)
        {
                        net[i+1][k]=0.0;
                        for(j=0;j<nneuron[i];++j)
                          net[i+1][k]+=weight[i][j][k]*output[i][j];
                        net[i+1][k]+=umbral[i+1][k];
                        output[i+1][k]=sigmoid(net[i+1][k]);
        }
        }

        /* Calculo del error de la red                             */
        /* E_p = Σ_{k=1}^{M} δ_pk^2                                */

        error=0.0;
```

```
for(i=0;i<nneuron[nlayer-1];++i)
error+=sq(teacher[patternindex][i]-output[nlayer1][i]);
}
```

void BackTrans(void);

Rutina encargada de efectuar la retropropagación del error de la red a cada conexión de la red, es decir, asigna de manera proporcional el error total de la red a cada conexión asociada dependiendo de los pesos asociados.

```
void BackTrans(void)
    {
    int i,j,k;
    float auxiliar;
     * Para cada neurona de la capa de salida determinar        *
     * los ∂_{p.k}^o que son los valores delta de salida,        *
     * Determinación del error para las unidades salida          *
    /* δ_{pk}^o = (y_{pk} - o_{pk}) f_k^{o'}(neta_{pk}^o) = δ_{pk} f_k^{o'}(neta_{pk}^o)   */
    for(i=0;i<nneuron[nlayer-1];++i)
        delta[nlayer-1][i]=
            (teacher[patternindex][i]-output[nlayer-1][i])*

        difsigm(net[nlayer-1][i]);
    /* Determinación de los términos de error para las          */
    /* unidades ocultas, δ_{pj}^h = f_j^{h'}(neta_{pj}^h) Σ_k δ_{pj}^o W_{kj}^o   */
    for(i=nlayer-1;i>0;--i)
        for(j=0;j<nneuron[i-1];++j)
            {
            delta[i-1][j]=0.0;
            for(k=0;k<nneuron[i];++k)
                delta[i-1][j]+=weight[i-1][j][k]*delta[i][k];
            delta[i-1][j]*=difsigm(net[i-1][j]);
            }
    /* Actualización de los pesos de las capas de la red                    */
    /* W_{kj}^o(t+1) = W_{kj}^o(t) + ηδ_{pk}^o I_{pj} ; W_{ji}^h(t+1) = W_{ji}^h(t) + ηδ_{pj}^h X_{pi}   */
    for(i=0;i<nlayer-1;++i)
    {
        for(j=0;j<nneuron[i];++j)
            for(k=0;k<nneuron[i+1];++k)
                {
                dweight[i][j][k]*=beta;
                dweight[i][j][k]+=eta*output[i][j]*delta[i+1][k];
                weight[i][j][k]+=dweight[i][j][k];
                }
```

```
        }
/* Modificación de los umbrales */
for(i=0;i<nlayer;++i)
        for(j=0;j<nneuron[i];++j)
                {
                dumbral[i][j]*=beta;
                dumbral[i][j]+=eta*delta[i][j];
                umbral[i][j]-=dumbral[i][j];
                }
}
```

Programa Principal

void main(void);

Programa principal que efectúa Back-Propagation para un conjunto de muestras de la población, tomando como variables de control: el coeficiente de aprendizaje (η), el conjunto de patrones seleccionados aleatoriamente de la población de patrones $(SIZE)$ y el número máximo de iteraciones $(MAXITERATION)$.

El proceso termina cuando se cumple para $v = n_p N$ patrones la condición de terminación $\varepsilon < error\_permitido$.

```
void main(void)
        {
        int i,j,k,p;
        puts("problema a analizar: ************ ");
        randomize();
        eta=0.0;

        /* inicialización de variables */

        for(i=0;i<SIZE;++i)result[i]=0;
        outstream=fopen(outputfilename,"w");

        /* efectúa los cálculos con diferentes razones de          */
        /* aprendizaje η_inicial = 0.1, step = 0.025, η = 0.3.     */
        /* p determina el cálculo para diferentes η                */
```

```
for(p=0;p<12;++p)
        {
        eta+=0.025;
        for(j=0;j<MAXITERATION;--j)result[j]=0;

         * MAXITERATION es el máximo de iteraciones     *
         * a efectuar en el proceso de entrenamiento     *
         * Sea k = 0 y v = n_p N , donde n_p es el número *
         * de patrones que participan en el proceso      *
         * Repetir el siguiente proceso para un número   *
         * veces igual al tamaño deseado de la muestra   */

        for(i=0;i<SIZE;--i)
                {
                StartModel();
                k=0;
                for(j=0;j<MAXITERATION;--j)
                        {
                        patternindex=random(npattern);
                        ForwardTrans();
                        BackTrans();

                        /* verificación de convergencia*/

                        if(error<epsilon)
                                ++k;
                        else
                                k=0;

/* condición necesaria de tal forma que la probabilidad de selección de todos los
elementos de la muestra se maximize o dicho de otra forma la probabilidad 1/2^k pueda
considerarse razonablemente pequeña */


                        if(k==N*npattern)
                                {

                                /* frecuencia de convergencia      */
                                /* en la iteración  j              */

                                ++result[j];
                                break;
                                }
                        }
                FreeModel();
```

```
            }
        Report();
        }
fclose(outstream);
puts("end of program");
```