Implementación Paralela del Algoritmo Backpropagation en un Cluster de Computadoras

Marcelo Alfonso, Carlos Kavka, Marcela Printista
Departamento de Informática, Universidad Nacional de San Luis
San Luis, 5700, Argentina
email: malfo,ckavka,mprinti@unsl.edu.ar

Abstract

En la actualidad existe una gran diversidad de arquitecturas paralelas, en donde se han definido muchos modelos de implementación del algoritmo de entrenamiento de redes neuronales backpropagation o propagación hacia atrás. La obtención de una buena performance depende básicamente del modelo de implementación y la arquitectura paralela disponible.

El presente trabajo consiste en la definición de un modelo de implementación del algoritmo de backpropagation en un ambiente de programación paralela y su implementación utilizando un cluster de computadoras conectadas entre sí.

Esta implementación permite ser utilizada, no necesariamente en una arquitectura paralela especifica en donde la comunicación no introduce un overhead, como ocurre en la utilización de un clustrer de estaciuones de trabajo.

De esta forma la implementación realizada permite no solo, mejoras en la performance del algoritmo de entrenamiento sino que ademas mediante la utilización de un parámetro ϵ que se utiliza para determinar que un cambio en un peso de entrenamiento superior a ϵ debe ser considerado, en otro caso no.

Keywords: red neuronal, backpropagation, paralelismo.

1 INTRODUCCIÓN

El algoritmo backpropagation es uno de los algoritmos mas usados para el entrenamiento de redes neuronales. El proceso de aprendizaje para ciertos problemas puede llegar a ser muy costoso en cuanto al tiempo de computación, que puede estar en el orden de días o semanas. En algunos casos, esperar este tiempo para que la red neuronal sea capaz de aprender, es inaceptable. Por ello el algoritmo de entrenamiento podria ser particionado en procesos de aprendizaje mas fáciles de resolver e implementarlos en arquitecturas de computación paralela.

Los modelos de implementación paralela de backpropagation mas utilizados se dividen en dos categorias: basado en la red y modelo basado en patrones de entrenamiento . El modelo basado en la red, particiona la red neuronal en diferentes procesadores. Cada procesador posee un subconjunto de unidades y pesos y es responsable de procesar esos componentes. El modelo basado en patrones de entrenamiento, los patrones de entrenamiento son particionados entre los diferentes procesadores. Estas implementaciones paralelas son llevadas a cabo en arquitecturas paralelas de proposito general. Muy pocas implementaciones consideran la utilización de redes de estaciones de trabajo. En este trabajo realizamos una implementación paralela del algoritmo backpropagation utilizando un cluster de estaciones de trabajo interconectadas entre si.

2 EL ALGORITMO BACKPROPAGATION

El algoritmo backpropagation es el mecanismo de entrenamiento que se utiliza para que una red neuronal del tipo percéptron de múltiples capas aprenda un problema determinado [3]. La Figura 1 muestra una representación gráfica de un Perceptrón de múltiples capas.

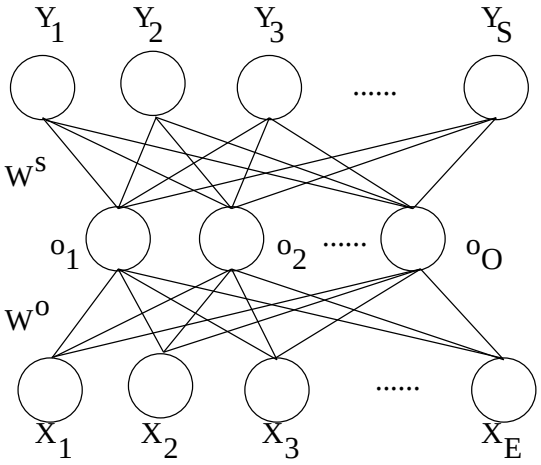


Figura 1: El Perceptrón de múltiples capas

En una red de este tipo existe una capa de entrada con **E** neuronas, una capa de salida con **S** neuronas y al menos una capa oculta con **O** neuronas internas.

El algoritmo de entrenamiento backpropagation, tiene dos fases, una hacia delante(fordward) y otra hacia atrás(backward). Durante la primera fase el patrón de entrada es presentado a la red y propagado a través de las capas hasta llegar a la capa de salida. Obtenidos los valores de salida de la red, se inicia la segunda fase, comparándose éstos valores con la salida esperada para obtener el error. Se ajustan los pesos de la última capa proporcionalmente al error. Se pasa a la capa anterior con una retropropagación del error, ajustando los pesos y continuando con este proceso hasta llegar a la primer capa. De esta manera se van modificado los pesos de las conexiones de la red para cada patrón de aprendizaje del problema, del que conocíamos su valor de entrada y la salida deseada que debería generar la red ante dicho patrón.

La técnica backpropagation requiere el uso de neuronas cuya función de activación sea continua. En general, la función utilizada es del tipo sigmoidal.

El algoritmo de aprendizaje se ejecuta un número determinado de veces (comúnmente denominado épocas), en donde en cada una se toman todos los patrones de entrenamiento.

3 IMPLEMENTACIONES PARALELAS DEL ALGORITMO DE BACKPROPAGATION

Las mayoría de las implementaciones paralelas del algoritmo BackPropagation pueden agruparse en dos grandes categorías[6]:

1. Modelo basado en patrones de entrenamiento.
2. Modelo basado en la red neuronal.

3.1 Modelo basado en patrones de entrenamiento

Este modelo es comúnmente llamado paralelismo de datos porque se particiona el conjunto de patrones de entrenamiento y no la red. Cada procesador posee una copia local de la red formada por un conjunto de pesos, la cual es entrenada con un subconjunto de patrones, actualizando los correspondientes pesos. Una vez que cada procesador actualizó los pesos para el subconjunto de patrones asignado, estos valores son globalmente coleccionados y se debe realizar una actualización global de los pesos, para mantener su consistencia; generalmente se realiza al finalizar la ejecución de una época. En la Figura 2 vemos como pueden está particionado los patrones de entrenamientos.

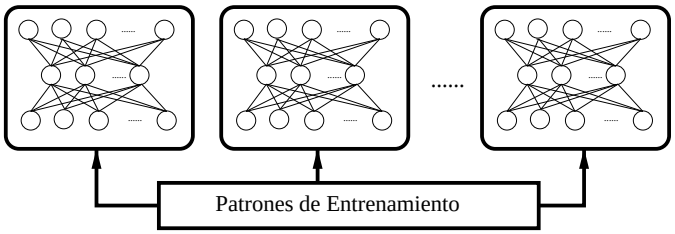


Figura 2: Paralelismo en los patrones entrenamiento

3.2 Modelo basado en la red neuronal

Consiste en distribuir la red (subconjunto de pesos) entre los procesadores. De acuerdo a la distribución de los pesos tenemos distintos modelos.

3.2.1 Pipeline

La distribución de pesos es tal que cada procesador evalúa una capa. Cada procesador(PE_I) recibe un mensaje del procesador(PE_{I-1}) que procesa la capa anterior. Lo interesante de este modelo es que el procesador (PE_{I-1}) procesa el patrón(A) y le envía la información al procesador(PE_I) el cual continua procesando el patrón(A), concurrentemente PE_{I-1} puede procesar el patrón siguiente. Es decir que este método realiza un pipeline a nivel del paso Forward y el paso Backward. (ver Figura 3).

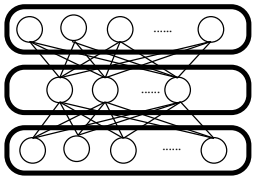


Figura 3: Paralelismo pipeline

3.2.2 Slicing Vertical

Los pesos que llegan a una unidad oculta y una de salida son mapeados en un procesador (ver Figura 4). Cada procesador almacena solamente los pesos de las neuronas que le son asignadas, comúnmente una neurona oculta y una de salida. El proceso en una neurona oculta consiste en computar el valor que corresponda y lo envía a los demás procesadores y luego continua calculando el valor de la neurona de salida asignada. Se calcula el error de la capa oculta basado en el error en la salida. Para

la actualización de los pesos se deben utilizar métodos para mantener la consistencia de los datos. Si el número de neuronas ocultas o de salidas supera la cantidad de procesadores, cada uno procesará mas de una neurona oculta y/o de salida.

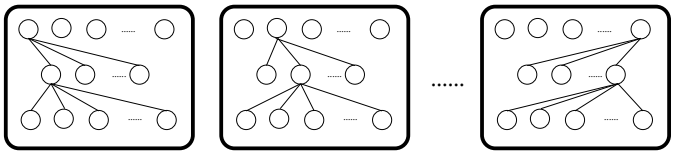


Figura 4: Paralelismo Slicing vertical

3.2.3 Paralelismo Synapse

La distribución de pesos es tal que cada procesador calcula una suma parcial de una neurona de salida (ver Figura 5). La computación es de gránulo mas fino que en el Slicing vertical. Los resultados parciales de cada procesador deben ser enviados a todos los demás procesadores mediante un brodcast antes que la próxima capa pueda ser procesada. La ventaja es que el error de la capa oculta puede ser evaluada sin necesitar ninguna comunicación entre los procesadores.

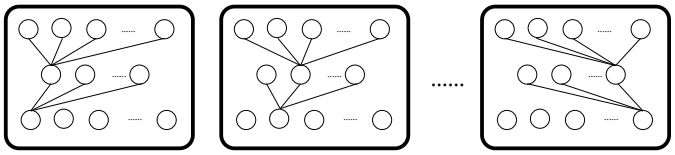


Figura 5: Paralelismo Synapse

4 IMPLEMENTACIÓN REALIZADA

La implementación algoritmo realizada es una combinación del paralelismo basado en la red y paralelismo basado en el conjunto de entrenamiento.

Por un lado tenemos los procesadores que computan el paso forward del algoritmo particionando el conjuntos de patrones de entrenamiento; donde la cantidad de particiones depende de la cantidad de procesadores con esta funcionalidad.

Por otro lado tenemos los procesadores que computan el paso backward, los cuales son responsables de un subconjunto de unidades ocultas, con sus pesos correspondientes a las unidades de entrada y de salida con que están conectadas (ver Figura 6).

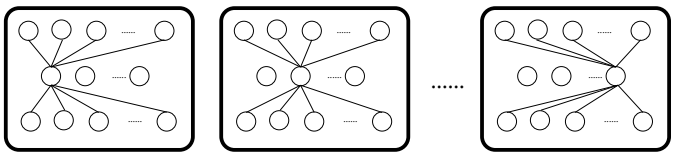


Figura 6: Modelo

En la Figura 7, podemos observar la partición de tareas y la red de interconexión.

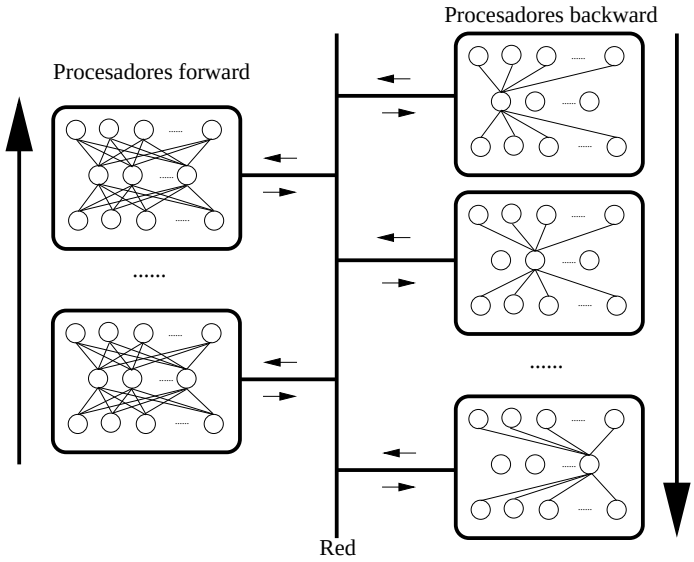


Figura 7: Partición de tareas

Para poder llevar a cabo el control y administración del proceso de entrenamiento se utiliza un procesador que cumple la función de árbitro, es el encargado, entre otras cosas, de dar el punto de partida del entrenamiento y lo detiene cuando la red aprende o ejecuta el número de épocas previstas.

En las siguientes secciones se describe con mayor claridad la estructura del modelo.

4.1 Arbitro

El árbitro es el encargado de todas las tareas de inicialización y de preparación de la red para la ejecución del proceso de entrenamiento, como así también es el encargado de decidir la finalización. Su principal actividad es recibir las actualizaciones de pesos y calcular el error una vez por época.

Al comenzar la ejecución, este procesador lee la red inicial. Una vez realizado esto, forma un mensaje con los pesos de las unidades ocultas y de salidas, lo envía utilizando un broadcast a todos los demás procesadores. Una vez que todos los procesadores recibieron el mensaje, se calcula el error inicial y a partir de este momento están todos los procesadores listos para iniciar el entrenamiento de la red. A partir de este momento el árbitro va manteniendo una copia actualizada de los pesos de las unidades ocultas y los pesos de la unidades de salida, mediante la recepción y el procesamiento de los mensaje de actualización provenientes de los procesadores backward. También en los mensajes recibidos viene como dato el número de época que se está ejecutando en ese momento, lo cual permite saber cuando una época finaliza y poder en ese momento realizar el cálculo del error, determinar el tiempo que ha transcurrido hasta ese momento y la cantidad de pesos que se modificaron en dicha época. Además, si el número de épocas ejecutadas es mayor al número de épocas previstas, se debe finalizar el entrenamiento de la red. Lo mismo ocurre cuando el error calculado es menor al error aceptable, lo cual nos está indicando que la red aprendió. Para finalizar el entrenamiento, todos los procesadores deben conocer esta decisión. El árbitro logra este efecto mediante el envío de un mensaje a todos los procesadores con la orden de finalización.

4.2 Forward

Cada procesador toma un subconjunto de patrones de entrada (la cantidad de patrones para cada uno es igual al número total de patrones dividido por la cantidad de procesadores de este tipo).

Realizan el paso forward del algoritmo de backpropagation propiamente dicho, calculando el valor de activación de las unidades ocultas, de las unidades de salida y el delta de las unidades de salida; luego envía información a los procesadores backward por cada patrón que procesan. Los procesadores deben tener actualizados los pesos, lo cual se logra a través de la recepción de mensajes que son enviadas por los procesadores backward.

Su ejecución se detiene al recibir un mensaje proveniente del árbitro con la orden de terminar.

La cantidad de procesadores de este tipo se pasa como parámetro y está determinada por el tipo de problema a resolver, la cantidad de patrones de entrenamiento y así también, por la cantidad de procesadores que se disponga para la ejecución.

4.3 Backward

Cada procesador tiene un subconjunto de la red, formado por todos los pesos asociados con las unidades ocultas que le corresponde a cada uno.

El proceso para este tipo de procesadores consiste en recibir información de los procesadores forward, para poder realizar el paso backward del algoritmo de backpropagation propiamente dicho, el cual consiste en calcular el valor de los deltas de las unidades ocultas y obtener el nuevo valor de todos los pesos asociados con las unidades ocultas que corresponden. Luego envía las nuevas actualizaciones a los procesadores forward y al árbitro.

Su ejecución se detiene al recibir un mensaje proveniente del árbitro con la orden de terminar.

4.4 Comunicación entre Procesos

Para la implementación se utiliza MPI en donde la comunicación entre procesos es mediante la utilización de mensajes[4]. Los mensajes involucrados en el proceso paralelo descripto (ver Figura 8) se detallan a continuación.

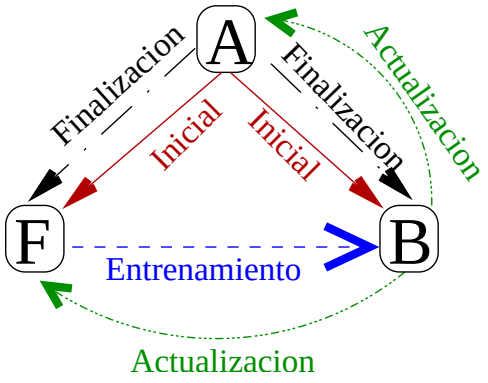


Figura 8: Mensajes intervinientes

4.4.1 Mensaje inicial

Este mensaje es enviado por el procesador que actúa como árbitro a todos los procesadores y contiene la información necesaria para poder entrenar la red. Este mensaje permite sincronizar todos los procesadores, porque hasta que todos los procesadores reciban el mensaje no pueden seguir ejecutando la siguiente instrucción. El mensaje es enviado una única vez al comenzar la ejecución, es de tamaño fijo y contiene la siguiente información: los pesos de las unidades ocultas y los pesos de las unidades de salidas.

El tamaño del mensaje esta dado por:

$$|M| = ((O * E) + (O * S)) * |DATO|$$

donde, E es la cantidad de unidades de entrada, O es la cantidad de unidades ocultas, S es la cantidad de unidades de salida y $|DATO|$ es el tamaño que ocupa cada dato del mensaje

4.4.2 Mensaje de entrenamiento

Este mensaje es enviado por el procesador que realiza el **paso forward** a los procesadores que realizan el **paso backward** con los resultados de haber ejecutado este paso. Se envía por cada patrón que se procesa y tiene tamaño fijo para cada red. Contiene la siguiente información: el número de épocas ejecutadas, los valores de salida de las unidades ocultas, los errores de las unidades de salida y el número que identifica el patrón.

El tamaño del mensaje esta dado por:

$$|M| = (O + S + 2) * |DATO|$$

donde, O es la cantidad de unidades ocultas, S es la cantidad de unidades de salida y $|DATO|$ es el tamaño que ocupa cada dato del mensaje

Este mensaje tanto en el envío como en la recepción, el tipo es no bloqueante, debido a que los procesos deben además estar continuamente chequeando que el árbitro no haya pedido finalizar. Es decir, que puede ocurrir que el árbitro pida que se detenga el entrenamiento. Pero por otro lado el proceso forward debe, además, chequear si existen mensaje de actualización, en cuyo caso debería actualizar los pesos. En cambio el proceso backward no pueden hacer otra cosa hasta que el mensaje realmente fue recibido.

4.4.3 Mensaje de actualización

Este mensaje es enviado por el procesador que realiza el **paso backward** a los procesadores que realizan el **paso forward** y al **árbitro** con los pesos que se modificaron en un valor superior al parámetro ϵ. Contiene la siguiente información: el número de épocas ejecutadas, los pesos que deberán ser actualizados con sus correspondientes valores **(peso**$_x$ **nuevo_valor_peso**$_x$ **. . . peso**$_y$ **nuevo_valor_peso**$_y$**)**.

La particularidad de este mensaje es que es de tamaño variable, que estará dado por:

$$|M| = (cantidad\_pesos\_a\_modificar * 2) + 2 \quad [1]$$

En el caso que ningún peso se modificó en un valor superior al ϵ no se debe enviar el mensaje, ya que no tiene sentido enviar un mensaje solamente con la época.

El envío del mensaje es de tipo no bloqueante, por la situación del posible mensaje de finalización que puede arribar en el momento que se está enviando el mensaje.

Mientras que en la recepción, el árbitro usa una comunicación de tipo bloqueante, ya que de no recibir una actualización no debe hacer otra actividad.

Pero el proceso forward, debe utilizar una recepción no bloqueante ya que, de no recibir el mensaje de actualización esperado, debe poder continuar con su tarea. Esto, por otro lado, le permite al proceso forward poder atender el mensaje de finalización proveniente del árbitro.

[1] Se agrega un -1 al final del mensaje para indicar el fin del mismo

4.4.4 Mensaje de finalización

Este mensaje es enviado por el procesador que es árbitro a todos los procesadores con el mensaje de finalización. Se envía una única vez antes de terminar la ejecución, es de tamaño fijo (1). El envío del mismo se hace mediante una iteración enviando un mensaje a cada procesador con un mensaje de finalización del entrenamiento. La recepción en cada uno de los procesadores se realiza mediante una recepción no bloqueante, que fue iniciada al comienzo del entrenamiento y que ahora podrá ser finalizada y de ese modo detener la ejecución de todos los procesos.

5 EL PARÁMETRO ϵ

La implementación al ser realizada sobre un cluster formado por estaciones de trabajo, en donde el tiempo de comunicación influye considerablemente en la performance del algoritmo [7] [5], se decidió introducir un parámetro denominado ϵ [1] [2], el cual permite controlar el tamaño y/o el número de los mensajes, que son enviados desde los procesadores backward a los procesadores forward y al árbitro. En estos mensajes sólo se transmite la actualización de los pesos cuyo cambio fue superior al valor del ϵ. Al poder disminuir el tamaño de los mensajes se produce una disminución en el overhead producido al enviar dicho mensaje.

En la sección Resultados obtenidos, se muestra claramente el impacto que produce el ϵ, en cuanto al tiempo de ejecución del entrenamiento al variar su valor, sin perjudicar la calidad del aprendizaje de la red.

6 RESULTADOS OBTENIDOS

Con el objetivo de mostrar los resultados, se decidió utilizar el reconocimiento de rostros[2] que es un problema de reconocimientos de patrones que tiene un cierto grado de dificultad y complejidad; el cual se puede adaptar muy bien para ser tratado con la implementación propuesta.

El problema del reconocimiento de rostros, consiste en poder, a partir de un conjunto de imágenes de los rostros de personas, identificar: una o varias personas, el estado de ánimo en que se encuentra, si tiene anteojos o nó, si tiene bigote o nó.

En nuestro caso se usó un conjunto de 40 imágenes de tamaño 94 por 100 pixels, en escalas de grises, correspondiente a los rostros de 6 personas distintas en donde cada una está: con anteojos o sin ellos, triste, feliz, enojado o en expresión neutra. En la Figura 6 se muestra las imágenes de una de las personas.

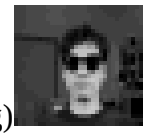
(a)  (b)  (c)  (d)  (e)  (f)  (g) 

Figura 9: Imágenes de una persona con algunas de los poses o características que posee: (a) con anteojos y enojado, (b) sin anteojos y feliz, (c) con anteojos y feliz, (d) sin anteojos y neutral, (e) con anteojos y neutral, (f) sin anteojos y triste, y (g) con anteojos y triste.

El objetivo consiste en que la red sea capaz de identificar cada una de las personas. El conjunto de entrenamiento consiste de imágenes y la salida consiste de un indicador de a que persona corresponde.

[2] Esta idea fue presentada en el CACIC 2000

Se espera que una vez finalizado el entrenamiento la red retornada sea capas de identificar a las 6 personas, sin importar el estado de animo y si usa o nó anteojos.

La red toma como entradas cada uno de los 9400 pixels de la imagen, cuyo valor esta dado por la intensidad en la escala de grises. Estos valores están entre 0(negro) y 255(blanco), los cuales son llevado al rango [0,1] dividiendo el valor por 255.

El número de unidades ocultas fue definido en 20 y como salida se utilizan 6 unidades, las cuales toman el valor 1 en la posición que corresponde a la persona identificada y 0 en las demás.

La cantidad de pesos con la que cuenta la red esta dada por la siguiente expresión:

$$\#pesos = (\#entradas * \#ocultas) + (\#salidas * \#ocultas)$$

$$\#pesos = (9\,400 * 20) + (6 * 20)$$

$$\#pesos = 188\,120$$

Este es la cantidad de pesos que el algoritmo debe actualizar, hasta lograr que la red minimice el error y obtener así el aprendizaje de la red para los patrones de entradas dados.

En la totalidad de las ejecuciones la red aprende en una época comprendida entre la 140 y 160, con lo cual se decidió para mostrar los resultados detener todas las ejecuciones cuando alcanzamos la época 150; para poder comparar más claramente las gráficas obtenidas.

En las secciones siguientes se realiza el análisis de la ejecuciones con distintos parámetros ejecutando en forma secuencial y paralela con 7 procesadores.

El objetivo es, por un lado mostrar los beneficios de utilizar el parámetro ϵ teniendo en cuenta la performance de la implementación paralela y, por otro lado, poder decir que la implementación paralela mejora la implementación tradicional del algoritmo de aprendizaje backpropagation en términos de tiempo de ejecución. En todas las ejecuciones se tomaron 5 semillas seleccionadas aleatoriamente para que se puedan comparar entre sí las distintas ejecuciones tomando los mismos pesos iniciales.

6.1 Ejecución Secuencial

En esta sección se analiza la implementación secuencial realizada del problema utilizando el factor de actualización épsilon (ϵ). Esto se utiliza para probar que si sólo actualizamos los pesos cuando el cambio a introducir supera el valor del épsilon, no se introduce overhead al entrenamiento y que la red aprende en todos los casos. A continuación se muestran las gráficas del error para cada una de las tres tipos de ejecuciones, mostrando el mínimo, máximo y promedio del error producido en cada época.

La Figura 10 muestra el error que se produjo detallado por épocas para el tipo de ejecución se-

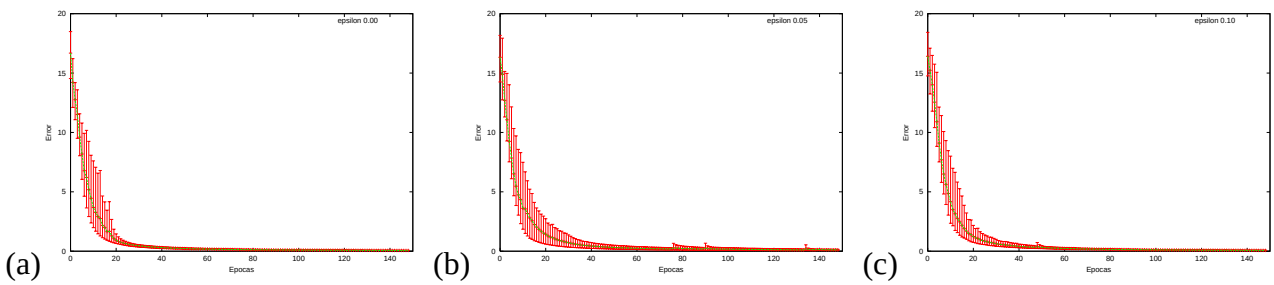


Figura 10: Secuencial. Error por épocas para ϵ de: (a)0.00, (b)0.05 y (c)0.10

cuencial con los distintos valores del ϵ. El valor del error en la época 150 fue de 0.061139 para ϵ 0.00, 0.111800 para ϵ 0.05 y 0.067799 para ϵ 0.10.

Como se observa, las tres ejecuciones tienen un comportamiento similar, logrando bajar el error. La red aprende o reconoce los rostros de las personas utilizadas para tal fin; aunque en el caso que utilizemos un épsilon (ϵ) de 0.05 el error disminuye un poco mas lentamente. Además se observa que los valores máximos, mínimos y promedios, tienen poca amplitud entre sí.

Estas gráficas nos permiten afirmar que la variación del parámetro épsilon (ϵ), no afecta el entrenamiento de la red, sino que todo lo contrario disminuye la amplitud entre el valor máximo y mínimo en el cálculo del error, es decir que en todas las ejecuciones la disminución del error se produce en forma similar.

Las próximas gráficas muestran el tiempo de entrenamiento necesario por cada época, utilizadas para analizar la performance de las ejecuciones secuenciales con diferentes valores del épsilon (ϵ). Se muestran para cada una de los tres tipos de ejecuciones, el tiempo mínimo, el máximo y el promedio, transcurrido durante la ejecución, por cada una de las épocas.

La Figura 11 muestra el tiempo transcurrido desde el comienzo del entrenamiento. El tiempo

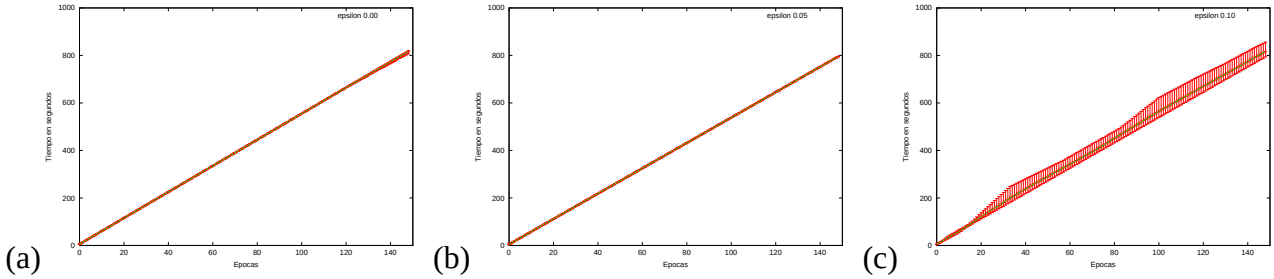


Figura 11: Secuencial. Tiempo de entrenamiento por épocas para ϵ de: (a)0.00, (b)0.05 y (c)0.10

transcurrido durante el entrenamiento fue de 816.569641 segundos(aproximadamente 14 minutos) para ϵ 0.00, 794.773560 segundos(aproximadamente 13 minutos) para ϵ 0.05 y 816.529785 segundos(aproximadamente 14 minutos) para ϵ 0.10.

Como era de esperar las tres gráficas tienen un comportamiento similar. Ya que al usar o no el épsilon (ϵ) no varia en si el código a ejecutar. Con un valor de ϵ igual a 0 por cada patrón se deben asignar a todos los pesos el nuevo valor del mismo. Mientras que para un ϵ distinto de 0 sólo se deben asignar a los cambios a los pesos que fueron modificados en un cambio mayor al del ϵ.

Podemos afirmar que la variación del parámetro épsilon (ϵ), no afecta ni la performance, ni la calidad del entrenamiento de la red, lo cual demuestra que una red neuronal del tipo backpropagation, puede ser entrenada utilizando un algoritmo tradicional de entrenamiento o introduciendo la utilización del parámetro ϵ.

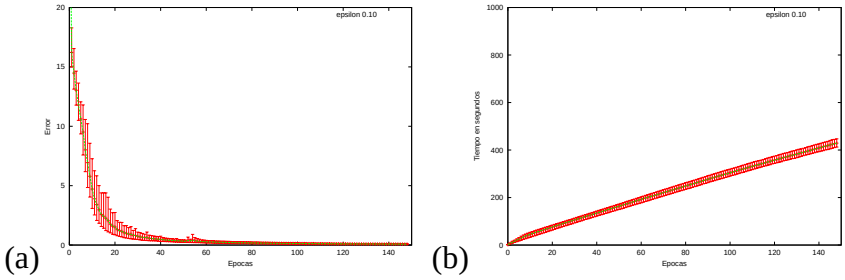
6.2 Ejecución paralela

En esta sección se analiza la implementación paralela realizada del problema utilizando el factor de actualización épsilon (ϵ). En las ejecuciones secuenciales se vió que era posible utilizar dicho factor, sin ningun impacto en la performance. Su objetivo primordial, es reducir en número y tamaño los mensajes de actualización que son enviados por los procesadores backward a los demás; influyendo directamente en la disminución en el tiempo de comunicación.

Se realizaron diversas ejecuciones con distintos parametros, obteniendo bueno resultados. A continuación mostramos los resultados obtenidos para la siguiente configuración en particular: se seleccionó el número de procesadores a utilizar en 7, donde el número de procesadores forward 2, 1 es el procesador que cumple la función de árbitro y los 4 restantes backward.

En este caso, se observa claramente que se está combinando la partición de los patrones, de tamaño 20 cada subconjunto y el particionado de la red.

La Figura 12 muestra el error por épocas para el tipo de ejecución paralela con 7 procesadores.



(a) (b)

Figura 12: 7 proc. con un ϵ de 0.10. (a)Error por épocas, (b)Tiempo de entrenamiento por épocas.

El valor del error en la época 150 es de 0.076350. Ademas se muestra el tiempo de entrenamiento necesario por cada época, utilizadas para analizar la performance, con el valores del épsilon (ϵ) igual a 0.10. Se muestra el tiempo mínimo, el máximo y el promedio, transcurrido durante la ejecución, por cada una de las épocas. El tiempo transcurrido durante el entrenamiento fue de 428.976746 segundos, aproximadamente 7 minutos. Esto nos esta indicando que el tiempo de entrenamiento disminuyo considerablemente con respecto a la implementación secuencial

La Figura 13 muestra la cantidad de pesos modificados por cada épocas, para el tipo de ejecución

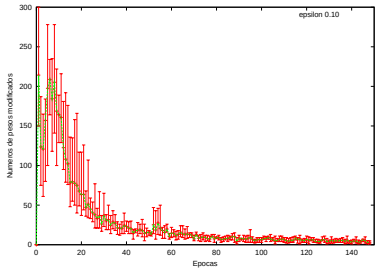


Figura 13: 7 proc. con un ϵ de 0.10. Pesos modificados por épocas.

paralela con 7 procesadores.

7 CONCLUSIONES

En este trabajo se describieron y compararon los modelos de implementación paralela del algoritmo backpropagation existentes. El éxito de los diferentes modelos depende de la existencia de máquinas paralelas que traten eficientemente las comunicaciones, reduciendo los tiempos de espera que se producen cuando el algoritmo está esperando mensajes de otros procesadores. En muchos casos, los equipos paralelos necesarios para aplicar estos algoritmos son demasiado caros, y su performance en otro tipo de equipos es inapropiada.

Teniendo en cuenta estas limitaciones, y además, al disponer de un cluster de estaciones de trabajo, se ha propuesto e implementado en este trabajo un modelo paralelo mixto, particionando los patrones de entrenamientos y la red entre los procesadores intervinientes. Una de las principales limitaciones del uso de un cluster de estaciones de trabajo, es que la comunicación entre los procesadores juega un papel muy importante en la performance de implementaciones paralelas en donde se utiliza

mucha comunicación, la cual está directamente relacionado con el tiempo y tamaño utilizado por los mensajes.

Para minimizar el efecto de la disminución de performance producida por la comunicación, se aplicó un cambio muy importante al algoritmo de entrenamiento backpropagation, el cual consiste en actualizar únicamente los pesos que superan un cierto valor (ϵ), permitiendo disminuir la cantidad y el tamaño de los mensajes. En los algoritmos tradicionales, aún los cambios más pequeños en los pesos implican su inmediata actualización.

Se ha logrado reducir el overhead de comunicación de forma tal que no se necesita una máquina puramente paralela, sino que el algoritmo puede producir una mejora significativa de tiempos de entrenamiento aún usando un cluster de estaciones de trabajo conectadas a través de una red de área local. Es importante destacar que en los experimentos realizados, no se tenía control total del cluster, sino que la ejecución de nuestros algoritmos debía competir con algoritmos de otros usuarios. Aún así, fue posible observar la mejora en los tiempos de ejecución de los algoritmos de entrenamiento.

La implementación propuesta permite mostrar que en el entrenamiento, no es necesario enviar todos los cambios que se producen en los pesos de la red; sino, que solo enviando los que superan un cierto valor (ϵ) se logra que la red aprenda.

La disminución en el overhead de comunicación permite que este algoritmo pueda ser implementado en diversas arquitectura paralela, con diferentes modelos de conexión.

REFERENCIAS

[1] M. Alfonso, C. Kavka, and M. Printista. A low communication overhead parallel implementation of the back-propagation algorithm. *Anales del VI CACIC*, Octubre 2000.

[2] T. Michell. *Machine Learning*. Carnegie Mellon University, 1996.

[3] M. Minsky and S. Papert. *Perceptrons*. MIT Press, 1969.

[4] P. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann Publishers, Inc., 1997.

[5] D. Culler R. Martin, A.M. Vahdat and T. Anderson. Effects of communication latency, overhead and bandwidth in a cluster architecture. Technical Report 94720, 1996.

[6] N. Sundarajan and P. Saratchandran. *Parallel Architectures for Artificial Neural Network: Paradigmas and Impementation*. Instituto of Electrical and Electronics Engineers, 1998.

[7] B. Wilkinson and M. Allen. *Parallel Programming - Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice Hall, 1999.