VALIDACIÓN DEL USO DE ARITMÉTICA DISTRIBUIDA EN LA IMPLEMENTACIÓN DE REDES NEURONALES

ANDRÉS E. GAONA BARRERA
Ingeniero Electrónico, M.Sc
Profesor Auxiliar
Universidad Distrital "Francisco José de Caldas"
aegaona@udistrital.edu.co

RESUMEN
En este documento se presentan algunos resultados del uso de aritmética distribuida enfocados hacia implementaciones hardware de redes neuronales, dichos resultados confrontan consideraciones de área vs. precisión, a tener en cuenta por parte del diseñador antes de llevar la arquitectura de su red neuronal a algún dispositivo (microcontrolador, dispositivos de lógica programable o DSPs). El algoritmo desarrollado brinda un estimativo de cuánto se penaliza la precisión de la red a nivel hardware a medida que aumenta el número de bits empleados para representar las entradas. Además posee la ventaja que ejecuta las operaciones de manera digital emulando operaciones reales como se efectuarían en FPGA o CPLD, lo cual posteriormente puede ser utilizado para extraer el código para programar estos dispositivos.

PALABRAS CLAVES: red neuronal, aritmética distribuida, implementación hardware.

ABSTRACT
In this paper results of the Distributed Arithmetic application focused towards hardware implementations of neuronal networks are presented. These results confront important parameters like area and out precision, before implementing in some device (microcontroller, logic programmable devices or DSP's) the architecture of a neuronal network. The developed algorithm offers an approach on whatever is the network precision when increasing the size of the input words. In addition, it has the advantage that executes the operations of digital form to emulate real operations as they would take place in FPGA or CPLD, which later can be used to extract the code to program these devices.

KEYWORDS: *distributed arithmetic, neuronal networks, hardware implementations.*

1. INTRODUCCIÓN

Debido a la gran cantidad de operaciones que hay que realizar para encontrar la salida en una red neuronal ante un estímulo [1], muchos de los trabajos sobre este tema se han centrado en aplicaciones software más fáciles de implementar y en las cuales características de desempeño del sistema como velocidad de procesamiento y manejo masivo de datos pueden sacrificarse a causa de que los algoritmos realizados para este fin operan de manera serial, aumentando el tiempo de procesamiento considerablemente [2], [3].

Lo anterior abre un vacío en el estudio de redes neuronales en cuanto a opciones de implementación hardware que aunque son más difíciles de desarrollar muchas veces poseen un mejor desempeño que las software y pueden adaptarse a aplicaciones reales [2]-[4]. El inconveniente de éstas aplicaciones es la necesidad de realizar operaciones aritméticas sucesivas (multiplicación y suma) necesarias para evaluar el estado interno de las neuronas que componen una red y que consumen demasiados recursos hardware.

Surge como una alternativa para efectuar ciertas operaciones en una red neuronal el concepto de aritmética distribuida [5] - [8], el cual se basa en un esquema de sumas de productos usando tablas y/o memorias, donde son almacenados los valores constantes en una operación de este tipo. Es de utilidad la aritmética distribuida para implementar redes neuronales [6], [7], debido a que permite la selección ya sea de un modelo paralelo o serial según las necesidades de velocidad de procesamiento contra recursos hardware disponibles [5].

El presente trabajo intenta usar la aproximación de aritmética distribuida hacia implementaciones hardware, para ser introducida en el marco de redes neuronales y evaluar que tan viable es en este contexto. Para lo anterior, se compara la salida de una neurona contra el número de bits utilizados para representar las entradas y así tener un parámetro útil para seleccionar el número de bits de la implementación a nivel hardware de la red.

Este documento presenta en la sección 2 una revisión del estado del arte de aritmética distribuida. En la sección 3 se explica el funcionamiento del algoritmo desarrollado con sus características más relevantes. En la parte 4 se muestran los resultados al emplear una neurona y calcular

Fecha de Recepción: 31 Agosto de 2006
Fecha de Aceptación: 20 Marzo de 2007

su estado interno empleando aritmética distribuida. Y finalmente se concluye a cerca del trabajo realizado y se plantean algunas opciones de mejorarlo, en el trabajo futuro.

2. ARITMÉTICA DISTRIBUIDA (DA)

La aritmética distribuida (DA) es un método alternativo de realizar operaciones aritméticas que involucren la suma y el producto, en implementaciones software o hardware. S. White argumenta que la "DA es básicamente; pero no necesariamente, una operación computacional de bit serial que ejecuta el producto punto de una par de vectores en un paso directo y simple" [5].

Dentro de las ventajas de la aritmética distribuida se encuentran su fácil y eficiente proceso de implementación, además de la reducción del número de compuertas usadas. La desventaja que se le puede adjudicar a la DA es su baja velocidad de respuesta, debido a su naturaleza serial; pero ésta se puede convertir en ventaja cuando se emplean diversas técnicas para mejorarla como: fraccionamiento de los datos de entrada y/o operación en forma totalmente paralela [5], [8].

La DA se ha usado principalmente en el procesamiento digital de señales cuando es requerida una suma de productos como la mostrada en (1). Específicamente puede emplearse en el diseño e implementación de filtros digitales o en estructuras de diversas transformadas en el dominio de la frecuencia [5]-[9]. También es útil en sistemas de comunicaciones y en sistemas de control.

En (1) se muestra una suma de productos, que define la respuesta de un sistema lineal e invariante en el tiempo. Donde y es la respuesta del sistema, x_k es la variable de entrada y A_k es un arreglo de coeficientes constantes.

$$y(n) = \sum_{k=1}^{K} A_k \cdot x_k(n) \tag{1}$$

Si x_k es un número binario en complemento dos escalizado, entonces x_k se puede expresar así:

$$x_k(n) = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} \cdot 2^{-n} \tag{2}$$

donde b_{kn} son bits (0 ó 1), b_{k0} es el bit de signo y $b_{k\,N-1}$ es el bit menos significativo.

Al reemplazar (2) en (1) y reordenando las sumatorias se llega a:

$$y(n) = \sum_{n=1}^{N-1}\left[\sum_{k=1}^{K} A_k \cdot b_{kn}\right] \cdot 2^{-n} + \sum_{k=1}^{K} A_k \cdot -b_{k0} \tag{3}$$

Explícitamente (3) puede darse en términos de los todos los productos y sumas parciales de la siguiente forma:

$$\begin{aligned} y = & -\left[x_{10}\cdot A_1 + x_{20}\cdot A_2 + \ldots + x_{K0}\cdot A_K\right] \\ & + \left[x_{11}\cdot A_1 + x_{21}\cdot A_2 + \ldots + x_{K1}\cdot A_K\right].2^{-1} \\ & \vdots \\ & + \left[x_{1(B-2)}\cdot A_1 + x_{2(B-2)}\cdot A_2 + \ldots + x_{K(B-2)}\cdot A_K\right].2^{-(B-2)} \\ & + \left[x_{1(B-1)}\cdot A_1 + x_{2(B-1)}\cdot A_2 + \ldots + x_{K(B-1)}\cdot A_K\right].2^{-(B-1)} \end{aligned} \tag{4}$$

Si es observado el término en los corchetes de (3), esta expresión posee 2^K posibles valores como lo demuestran los términos en los corchetes de (4). Luego se pueden calcular estos valores y almacenarlos en una memoria o DALUT (Distributed Arithmetic Look Up Table); con lo cual se evita el cálculo de los productos parciales on-line. Los datos de entrada son usados para direccionar la DALUT, y la salida es guardada en uno o varios registros para ser sumados usando desplazamientos denotados por los factores exponenciales en (4) y obtener de esta manera el resultado final. Como ejemplo del contenido de una DALUT es mostrada la Figura 1.

0
A_1
A_2
$A_1 + A_2$
A_3
$A_1 + A_3$
$A_2 + A_3$
$A_1 + A_2 + A_3$

Figura 1. Contenido de una DALUT con K=3

3. EXPLICACIÓN DEL ALGORITMO DESARROLLADO

El enfoque para el desarrollo del algoritmo es que funcione de manera general, para ello los datos de entrada de la neurona fueron normalizados y cuantificados de la siguiente forma:

- Para los vectores de entrada es hallada la magnitud máxima de los datos.
- Son normalizados los vectores de entrada; es decir, son divididos los datos por el valor máximo de ellos, con esto teóricamente se tienen valores entre -1 y 1.
- A través del número de bits definidos para representar las entradas, son cuantificados los datos en complemento A2. El formato de representación en complemento A2 se utiliza para efectuar operaciones de suma y resta sin inconvenientes, además este formato es necesario para tener en cuenta el signo de los números, ya que los pesos

sinápticos de la neurona pueden ser tanto positivos como negativos.

- El vector de pesos también es normalizado y cuantificado con el mismo procedimiento explicado anteriormente.

A pesar de que las entradas son cuantificadas al número de bits definido previamente, es necesario concatenar "ceros" en la parte más significativa para los números positivos y de "unos" para los negativos, de tal forma que cumplan el tamaño de palabra máximo de salida dado por (5) y que resulta de efectuar las operaciones de multiplicación y sumas.

$$L = (2 \cdot b) + d \qquad (5)$$

donde L es la longitud de palabra de salida, b el número bits usados para cuantificar las entradas y d el número de entradas.

A partir de este momento es evaluado el estado interno de la neurona que tiene la forma dada por la ecuación (1) y que corresponde a la suma de productos en la que puede emplearse DA. Debido a que el algoritmo es desarrollado en MatLab y este ejecuta las instrucciones de forma serial, no es posible ejecutar el esquema de DA paralelo ideal por el alto grado de paralelismo de las redes neuronales, luego todas operaciones simulan el esquema serial como se enuncia a continuación:

- Es tomado el vector de pesos y se genera la DALUT correspondiente
- Con los datos de entrada se genera un vector encargado de direccionar la DALUT empezando por la parte más significativa del número.
- El vector creado anteriormente inicia el proceso de DA mostrado en la ecuación (5). Inicialmente el dato seleccionado de la DALUT es guardado y luego se hace un desplazamiento a izquierda (multiplicación por dos). El segundo dato tomado de la DALUT se suma con el número desplazado y se almacena. Este proceso continua hasta el momento en el cual el apuntador de la DALUT ha recorrido el vector creado en el paso anterior.
- Finalmente, el valor de salida se vuelve a transformar para que el resultado sea mostrado en el rango inicial de los datos de entrada de la neurona.

4. RESULTADOS Y VALIDACIÓN

Para mostrar como utilizar DA en aplicaciones neuronales, se ha tomado el esquema general de una neurona como el mostrado en la Figura 2 y se ha solucionado un problema de clasificación.

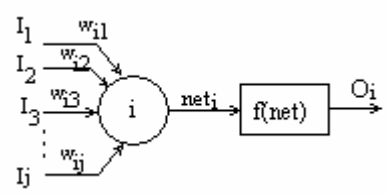

Figura 2 .Arquitectura de una neurona

El estado interno dado por net_i se define por (6) que es análoga a (1). Luego la preocupación principal al utilizar DA es que tanta resolución se debe tener para no perder precisión en la salida de la red.

$$net_i = \sum_{j=0}^{n} w_{ij} \cdot I_j \qquad (6)$$

El problema planteado es clasificar de la mejor manera las dos clases mostradas en la Figura 3.

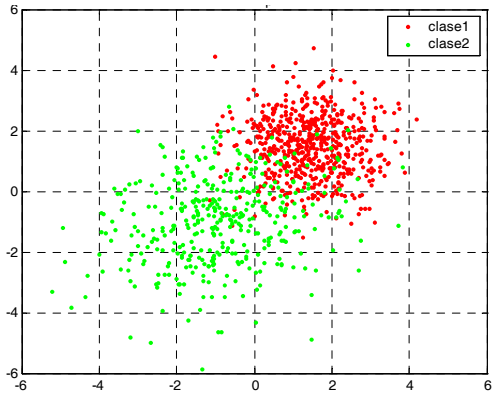

Figura 3. Datos correspondientes a dos clases

Entonces para encontrar el mejor clasificador es entrenada una neurona de manera tradicional utilizando el algoritmo del perceptrón, obteniendo que el estado interno de la neurona; es decir, sin aplicar la función de activación (discriminador duro) es:

$$net \doteq w_1.x + w_2.y + w_0 \qquad (7)$$

$$= 2.47.x + 2.54.y - 1.768 \qquad (8)$$

La solución obtenida y dada por (8) va a ser la implementa con la ayuda de DA usando el algoritmo explicado en la sección III. A continuación se muestran diferentes resultados variando el número de bits con los cuales se representan las entradas para una neurona "orientada a hardware" que soluciona el anterior problema:

Tamaño de palabra de entrada(bits)	*Tamaño de palabra de salida(bits)*	*Error (%)*
3	8	67.00
4	10	31.6226
5	12	20.1067
7	16	8.1783

8	18	2.5146
10	22	0.6181
12	26	0.2063
14	30	0.0871
16	34	0.0099

Tabla 1. Resultados de la salida de la neurona variando cantidad de bits

La tabla 1 muestra como se comporta la salida de la neurona ante 100 estímulos aplicados en las entradas y el error en la salida debido a las diferencias en el número de bits de cuantificación.

Los resultados muestran que la aproximación es valida cuando se cuantifican las entradas a un número superior de 10 bits, con los cuales se obtiene un error inferior al 1%. Pero no se puede ser muy estricto en esta afirmación ya que por ejemplo al cuantificar con 16 bits, el tamaño de la palabra de salida y por ende de recursos lógicos hardware crece excesivamente, 34 bits.

La Figura 4 muestra como se comporta el error con respecto al tamaño de la salida y tal vez es posible perder algo de precisión si se selecciona la cantidad de bits desde el momento en el cual la función inicia a ser plana, todo esto depende del grado de precisión del problema y de la experiencia del diseñador. Además, si el error introducido por la cuantificación es bajo, este sirve para regularizar la red neuronal.

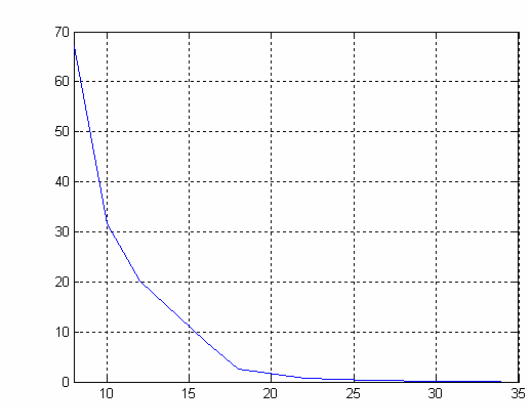

Figura 4. % Error vs. Tamaño de palabra de salida

5. CONCLUSIONES

Con el uso de aritmética distribuida es posible escoger entre tiempo de procesamiento vs. recursos hardware según condiciones del problema y dispositivos disponibles. El esquema de implementación es sencillo, ya que basta con programar en una memoria los coeficientes adecuados y con direccionamiento cuidadoso junto con desplazamientos a nivel de bits, proporcionan el estado interno de la neurona.

Es importante destacar que el proceso de obtención del estado interno de la neurona ha sido reducido a sumas o restas, según el caso, y a una escalización de los datos; con lo cual es simplificado enormemente la ejecución de las operaciones para tal fin.

Cuando se desarrollan sistemas como una red neuronal a nivel hardware es complicado tener estimativos de cómo se comportará con los datos de entrada que se poseen. Con el algoritmo desarrollado es posible verificar precisión y seleccionar con alto grado de confiabilidad el tamaño de palabra de entrada, salida e interno que tiene que manejar el sistema.

6. TRABAJO FUTURO

Desarrollar tanto el esquema paralelo como serial de aritmética distribuida en VHDL y evaluar la cantidad de recursos lógicos usados.

Para mejorar la precisión de salida de la red puede seleccionarse una normalización de la cuantificación que tenga en cuenta la varianza de los datos. Otra alternativa puede ser desarrollar formatos de punto flotante para solucionar de otra manera el problema.

Debido a que se está manejando toda la información a nivel de bits, puede utilizarse el algoritmo para desarrollar una herramienta que genere código para implementar la red a nivel hardware, por ejemplo en un lenguaje de descripción como VHDL.

7. BIBLIOGRAFÍA

[1] C. Bishop, “Neural Networks for Pattern Recognition”, Oxford University, 1995.
[2] N. Acosta and T. Marcelo, “Custom Architectures for Fuzzy and Neural Netwoks Controllers”, JCS&T, vol. 2, N° 7, October 2002.
[3] J.L Holt and T.E Baker, “Backpropagation simulations using limited precision calculations”, International Joint Conference on Neural Networks (IJCNN-91), vol. 2,July 1991, pp. 121–126.
[4] R. Molz, P. Engel, F. Moraes, L. Torres and M. Robert, “Codesign of Fully Parallel Neural Network for a Classification Problem”, 4th World Multiconference on Systemics, Cybernetics and Informatics SCI 2000
[5] S. A. White “Applications of Distributed Arithmetic to Digital Signal Processing: A tutorial Review”, IEEE ASSP Magazine Vol 6 No 3, 1989.
[6] V. Bochev, “Distributed arithmetic implementation of artificial neural networks”, IEEE Transactions on Signal Processing, Volume 41, Issue 5, May 1993.
[7] S. Park, J. Lim and S. Chae, “Digital implementation of discrete-time cellular neural networkswith distributed arithmetic”, International Conference on Neural Networks, 1997, vol 2.

[8] Xilinx® DSP Application Notes, “The Role of Distributed Arithmetic in FPGA-based Signal Processing”,1996.
[9] A. Sinha and A. Chandrakasan, “Energy Efficient Filtering Using Adaptive Precision and Variable Voltage”, Proceedings of IEEE Application Specific Integrated Circuits Conference, Washington, Sept. 1999.