Al-khwarizmi
Engineering
Journal

Al-Khwarizmi Engineering Journal, Vol.3, No.1, pp 81-90 (2007)

Low Cost Hardware Back Propagation Algorithm

Ammar A. Hassan
Computer Engineering Department / College of Engineering
University of Baghdad

(Received 12 September 2005; accepted 30 October 2006)

Abstract:-

The first successful implementation of Artificial Neural Networks (ANNs) was published a little over a decade ago. It is time to review the progress that has been made in this research area. This paper provides taxonomy for classifying Field Programmable Gate Arrays (FPGAs) implementation of ANNs. Different implementation techniques and design issues are discussed, such as obtaining a suitable activation function and numerical truncation technique trade-off, the improvement of the learning algorithm to reduce the cost of neuron and in result the total cost and the total speed of the complete ANN. Finally, the implementation of a complete very fast circuit for the pattern of English Digit Numbers NN has four layers of 70 nodes (neurons) on single chip using Xilinx FPGA technique is given.

The main goal of this paper is how to achieve the suitable activation function and weights for this network that gives minimum hardware cost when all stages of this ANN algorithm is implemented on FPGA.

1.Introduction

Artificial neural networks (ANNs) are a form of artificial intelligence, which have proven useful in different areas of application, such as pattern recognition [1] and function approximation/ prediction [2]. The most popular neural network is the multi-layer perceptron trained using the error back propagation algorithm [3]. However, an important obstacle in using this network in many applications is the slow training and the lack of clear methodology to determine the network topology before training starts. It is then desirable to speedup the training and allow fast experimentation with various topologies. One possible solution is an implementation on a reconfigurable computing platform (e.g Field Programmable Gate Arrays) FPGA.

Reconfigurable computing is a means of increasing the processing density (i.e. greater performance per unit of silicon area) above and beyond that provided by general-purpose computing platform [4]. Field Programmable Gate Arrays (FPGAs) are a medium that can be used for reconfigurable computing, since they allow for custom design of fine-grain logic compared to course-grain logic found in general-purpose computing platforms. FPGAs are a form of programmable logic, which offer flexibility in design like software, but with performance speeds closer to Application Specific Integrated Circuits (ASICs). With the ability to be reconfigured an endless amount of times after it has already been manufactured, FPGAs have traditionally been used as a prototyping tool for hardware designers. However, as growing die capacities of FPGAs have increased over the years, so has their use in reconfigurable computing applications too [2, 4].

2.Problem Formulation

The design problem in ANN using FPGA is a precision vs. area trade-off**.** One way to help achieve the density advantage of reconfigurable computing over general-purpose computing is to make the most efficient use of the hardware area available. In terms of an optimal precision vs area Trade-off, this can be achieved by determining the minimum allowable precision, whose criterion is to minimize hardware area usage without sacrificing quality of performance. Because a reduction in precision introduces more error into the system, minimum allowable precision is actually a question of determining the maximum amount of uncertainty (i.e. quantization error due to limited precision) that an application can withstand before performance begins to degrade. Hence, determining a minimum allowable precision and suitable numeric representation to use in hardware is often dependent upon the application at hand, and the algorithm used. Fortunately, suitable precision for backpropagation-based ANNs has already been empirically determined in the past.

Selecting weight precision is one of the important choices when implementing ANNs on FPGAs. Weight precision is used to trade-off the capabilities of the realized ANNs against the implementation cost. A higher weight precision means fewer quantization errors in the final implementations, while a lower precision leads to simpler designs, greater speed and reductions in area requirements and power consumption. One way of resolving the trade-off is to determine the “minimum precision”, required to solve a given problem. Traditionally, the minimum precision is found through “trial and error” by simulating the solution in software before implementation. Holt and Baker [5] studied the minimum precision required for a class of benchmark classification problems and found that 16-bit fixed-point is the minimum allowable precision without diminishing an ANN’s capability to learn these benchmark problems.

3.ANN Example

3.1.Structure of the Circuit

Following example [6] present how these two parameters effect on hardware resources. Remember that in neural network resources capacity differ from network to other according to number of neurons in each layer. The existence of hidden units allows the network to develop complex feature detectors, or internal representations. Fig.(1) shows the application of four layers network to the problem of recognizing english digit numbers. The two dimensional grid containing the numeral “7” forms the input layer (it can be arranged this two dimension array as one dimension, so that, number of input neurons is thirteen).

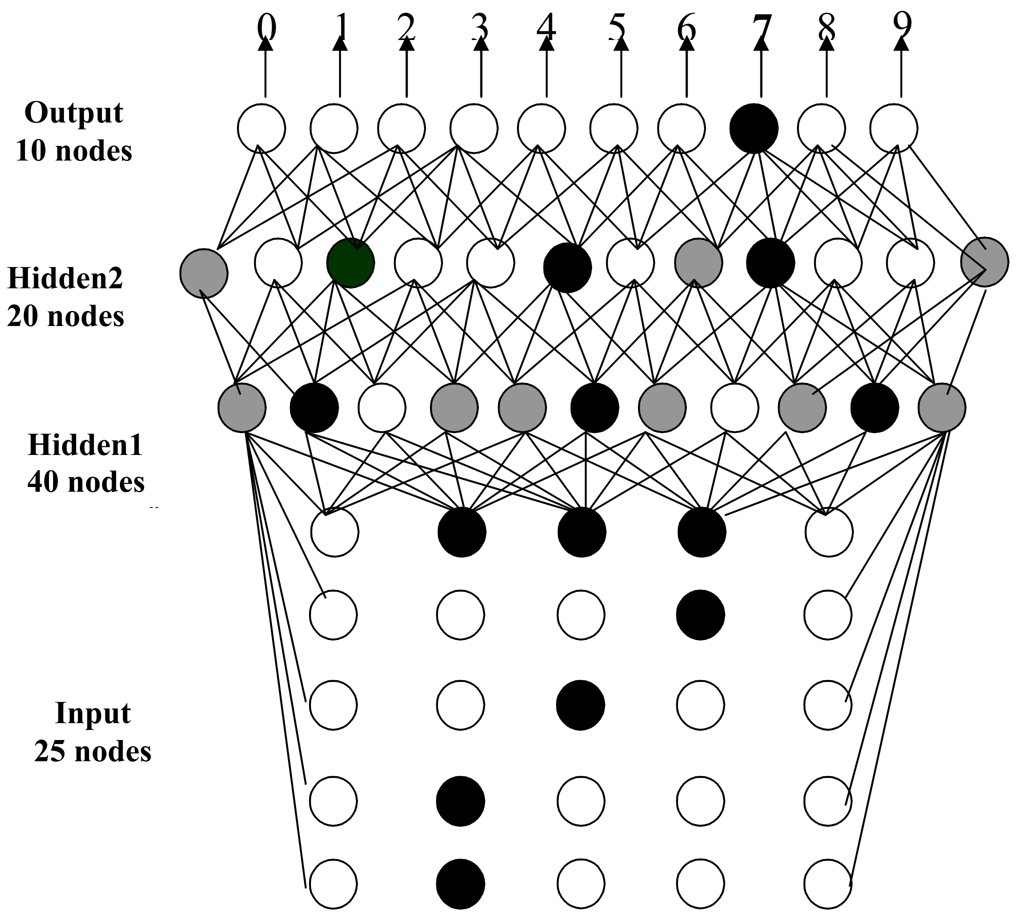

Fig.(1): Multilayer Network to Learn to Classify English Numbers

The first hidden layer is formed from 40 units each unit might be strongly activated by horizontal line in the input, while the second hidden layer molded from 20 hidden units, each unit fully connected with each unit in the first hidden layer. The output layer has ten units that represent the value of the digit number. Knowing that, the behavior of these hidden units is automatically learned not preprogrammed, then the computing weight of each layer in the network accuracy by using C++ programming language and implementing this result on FPGA as shown in the next section.

3.2.Implementation ANN Components on FPGA

Digital ANN architecture proposed in the previous section is an example of a reconfigurable computing application, where all stages of the algorithm reside together on the FPGA at once. These components based in idea of implementation on simple (or basic) arithmetic operations (such as addition and multiplication operations).

The floating-point precision is the problem of which an engineer must deal with when testing and validating circuits since it limits quantization errors according to number of bits in each node.

Therefore, truncation technique will be used in design to reduce the floating-point precision value and employ minimum hardware resources available on FPGA. Adding to this, choosing the optimal activation function suitable for implementation with truncation technique.

Previously as declared, the ANN algorithm components (such as number of hidden layers and number of units in each layer) differ from application to another. The English digit numbers NN is the application used in this work to implement ANN on FPGA, which is composed from four layers.

The quantization error after 5,000 iteration using two types of activation functions are given in table (1).

Activation Function	**Quantization Error without Truncation (30-bit)**	**Direct Truncation After Learning (10-bit)**
Sigmoid	2%	13-19%
Bipolar	4%	18-26%

Table (1): Quntization Error with and without Truncation

By using sigmoid function and without truncation (30-bit floating-point), quntization error is 0.02 (i.e. 98% of the corrected output). While, by the same method, but using bipolar function instead of sigmoid function, quntization error will be given 0.04.

4.Arithmetic Architecture for FPGA based ANNs

To calculate the cost required for English digit number NN in FPGA, it should be declared the way of selecting activation function against truncation approach as following: -

4.1.Sigmoid Function without Truncation (30-bit)

Before calculating the cost of FPGA based digit NN, it should be known that this function produced floating-point values. Then, it required a large number of logic storage as basic logic cells (LCs) called Look- UP- Tables (LUTs). These LCs are formed as RAM to store the output value of this function in each unit in the NN algorithm.

For 30-bit flotation-point, RAM cost implementation in FPGA 30-bit input and 30-bit output, for each unit in NN algorithm. Then, when sigmoid function applied in algorithm design is increased number of LCs, so that, it increased the over all hardware resources of NN algorithm.

The average cost of each unit (node) is calculated according to equation (1).

$$Cost\ /\ unit = n * \alpha + n * \beta + \delta \quad (1)$$

Where, n = Number of bits in data bus.

α = Cost for each Multiplier in FPGA.

β = Cost for each Adder / Subtractor in FPGA.

δ = Cost for each LUT in FPGA for sigmoid function.

i- For n=30 bit with sigmoid function α is a (30*30)-bit Multiplier and require about (2250) cell. β is a 30-bit Adder / Subtractor and require about (225) cell. Finally δ requires about (125000) cell using CORDIC approach.

So that, the cost of arithmetic components / unit (for 30-bit sigmoid) ≈ 200,000 cell. Therefore for English Digit NN algorithm from previously, there are seventy units then: -

Total Cost (for English Digit NN) = 200,000 * 70 = 14,000,000 cell.

This process can be achieved approximately by 17 of propagation gates delay.

ii- When applying truncation technique through the learning process with sigmoid function on ANN, the unit output precision reduces to be 10-bit and using equation (1).

Total Cost / unit ≈ 4000 cell.

Where n = 10-bit, α = 300 cell, β = 75 cell, and δ = 60 cell.

Total Cost (for English Digit NN) = 4000 * 70 = 280,000 cell with over all propagation delay reduced to 7 propagation gates delay.

From this result, when applying this technique in NN algorithm design it can reduce number of LUTs on FPGA and so reduce the over all hardware resources capacity.

4.2.Bipolar Function

As demonstrated from previous section, it required a large number of cells to implement NN based on sigmoid function. Then, bipolar function instead of sigmoid function in a NN algorithm can be used. Where, this function does not need storage cells for the output values in LUTs, because the output from this function is either "-1" or "1", this technique called *multiplierless technique*.

This approach has α =1 cell and δ=0 cell Therefore, the cost for each unit in a NN algorithm based bipolar function calculated using equation (2).

$$Cost\ / unit = n + n * \beta \qquad (2)$$

n = Number of bits in data bus.
β = Cost for each Adder / Subtractor in FPGA.

i- Without truncation technique (i.e. n=30-bit) β = 215 cell, therefor the cost / unit ≈6,500 cell. This process exceeds in 17 propagation gates delay. Then, to implement the English digit NN, that included 70 active neurons the total cost ≈ 455,000 cell.

ii- With truncation technique (i.e. n=10-bit) β = 64 cell, therefor the cost / unit = 650 cell. Also, this process exceeds in 7 propagation gates delay. So that, to implement the English Digit NN that included 70 neurons the total cost ≈ 45,500 cell.

Then, the total LUTs cost of NN can be reduced when applying bipolar function with truncation technique on FPGAs.

5.Proposed Backpropagation Algorithm [5]

An ANN using the classic backpropagation algorithm [6] as in fig.(2) has three basic phases named (1, 2 and 3) of execution the proposed backpropagation algorithm has same structure of the classic backpropagation algorithm with three additional steps for the proposed backpropagation algorithm named (1a, 2a and 3a) as following:

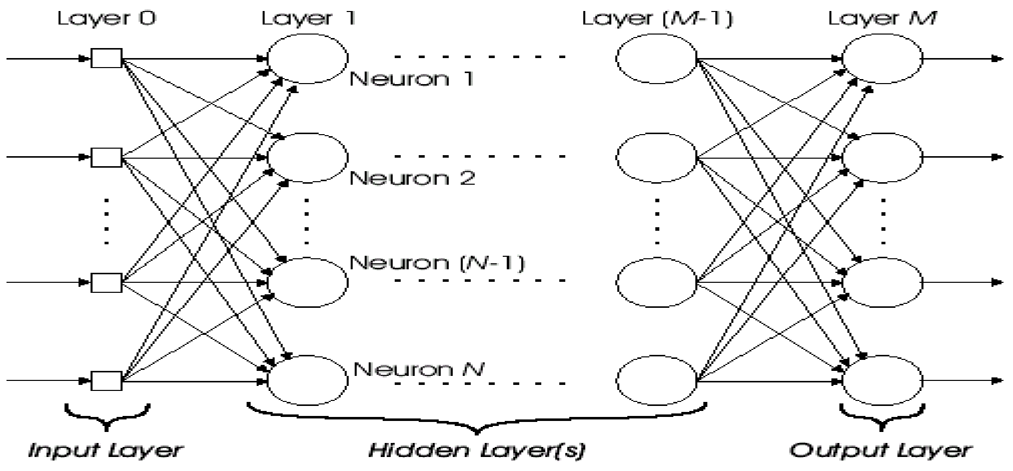

Fig. (2): Generic structure of an ANN.

5.1.Initialization

The following initial parameters have to be determined by the ANN trainer a priori: (i) $W_{Kj}^{(s)}(n)$ is defined as the synaptic weight that corresponds to the connection from neuron unit j in the (S-1) th layer, to K in the S th layer of the neural network. (ii) η is defined as the learning rate and is a constant scaling factor. (iii) $\theta_K^{(s)}$ is defined as the bias of a neuron, which is similar to synaptic weight in that it corresponds to a connection to neuron unit K in the (S-1)th layer of the ANN ,but is NOT connected to any neuron unit j in the(S-1) th layer.

(1a) Set counter =0

5.2.Forward Computation

During the forward computation, data from neurons of a lower layer (i.e (S-1) th layer), are propagated forward to neurons in the upper layer (i.e. (S) th layer) via a feedforward connection network. The structure of such a neural network is shown in Figure 2, where layers are numbered 0 to M, and neurons are numbered 1 to N. The computation performed by each neuron (in the hidden layer) is as follows:

$$H_k^{(s)} = \sum_{j=1}^{N(s-1)} w_{kj}^{(s)} o_j^{(s-1)} + \theta_k^{(s)} \quad (3)$$

Where j < k and s=1,….,M

$N(s-1)$ = Number of neurons in the (S-1)th layer of the ANN.

$H_K^{(s)}$ = Weight sum of the K th neuron in the Sth layer.

$W_{Kj}^{(s)}$ = Synaptic weight which corresponds to the connection from neuron unit j in the (S-1) th layer to neuron unit K in the S th layer of the neural network

$o_j^{(s-1)}$ Output of the jth neuron in the (S-1) th layer.

On the other hand, the output computation of neurons in any layer is as follows:

$$o_k^{(s)} = f(H_k^{(s)}) \quad (4)$$

Where k =1,…,N and s= 1,…,M

$o_k^{(s)}$ = neuron output of the Kth neuron in the sth layer.

$f(H_k^{(s)})$ = activation function computed on the weighted sum $H_k^{(s)}$.

Note that some sort of sigmoid function is often used as the nonlinear activation function, such as the following logsig function as in equation (5) or bipoler as in equation (6) as follows:

$$f(x)_{\log sig} = \frac{1}{1+\ \exp(-x)}$$

(5)

$$f(x)_{bipoler} = \begin{cases} 1 & for\ x \ge 0 \\ -1 & for\ x < 0 \end{cases}$$

(6)

(2a) Increment counter

For cycle counter = counter +1.

5.3.Backward Computation

In this step, the weights of the networks are updated. Criterion for the learning algorithm is to minimize the error between the expected (or teacher) value and the actual output value that was determined in the Forward Computation. The following steps are performed:

i- Starting with the output layer, and moving back towards the input layer, calculate the local gradients, as follows:

$$\varepsilon_k^{(s)} = \begin{cases} t_k - o_k^{(s)} & s = M \\ \sum_{j=1}^{N(s+1)} w_{kj}^{s+1}\ \delta_j^{(s+1)} & s = 1,...,M-1 \end{cases}$$

(7)

t_k = The target output for K^{TH} neuron in the M layer.

$N(s+1)$ = Number of neurons in the (S+1)th layer of the ANN.

Where $\varepsilon_k^{(s)}$ = error term for the K^{TH} neurons in the S^{TH} layer.

The difference between the teaching signal b _ and the neuron output $o_K^{(s)}$

$\delta_j^{(s+1)}$ = Local gradient for the j th neuron in the (S+1)th layer.

$$\delta_k^{(s)} = \varepsilon_k^{(s)}\ f^{\backslash}\ (H_k^{(s)}) \qquad s = 1,....,M$$

(8)

ii- Using the local gradients calculated in step 1, calculate the weight (and bias) changes for all the weights as follows:

$$\Delta w_{kj}^{(s)} = \eta\delta_k^{(s)}\ o_j^{(s-1)} \qquad k=1,....,Ns\ and\ j=1,...,N_{s-1}$$

(9)

Where $\Delta w_{kj}^{(s)}$ is the change in synaptic weight (or bias) ***CORRESPONDING TO THE GRADIENT OF ERROR FOR CONNECTION FROM NEURON UNIT J IN THE (S+1) th LAYER, TO NEURON K IN THE S th LAYER.***

iii- Once all weight (and bias) changes have been calculated in step 2, update all the weights (and biases) as follows:

$$w_{kj}^{s}(n+1) = \Delta w_{kj}^{s}(n) + w_{kj}^{s}(n)$$

(10)

Where K = 1,….., N neurons in the ***S*** th ***LAYER*** and j = 1,………, N neurons in the ***(S-1)*** th ***LAYER.***

$W_{Kj}^{(s)}(n+1)$ = Update synaptic weight (or bias) to be used in the (N+1) th iteration of the Forward Computation.

$\Delta W_{Kj}^{(s)}(n)$ = Change in synaptic weight (or bias) calculated in the N th iteration of the Backward Computation, where n =the current iteration.

$\Delta W_{Kj}^{(s)}(n)=$ Synaptic weight (or bias) to be used in the N th iteration of the Forward and Backward Computations, where n = the current iteration.

(3a) Truncation

For each counter > Q makes a truncation as in equation (11) to the values of the weights $W_{Kj}^{(s)}(n)$.

$$W_{Kj}^{(s)}(n) = Int\,(W_{Kj}^{(s)}(n)/\,2^n\,) * (\,2^n\,-1) \quad (11)$$

Where Q is a small integer value depends on the ANN, it defaults between 5- to- 50.

n:- Number of bits in data bus.

For truncation in the range of 10-bit this equation becomes as in equation (12).

$$W_{Kj}^{(s)}(n) = Int\,(W_{Kj}^{(s)}(n)/1024) * 1023 \quad (12)$$

6.Comparative study

The simple example is the four-layer network to the problem of recognizing English digit numbers that has 70 nodes. The implementation of this NN using ISE 4.1i on Xilinx platform [7, 8] shows the effect of the proposed algorithm on the total cost and the final error as in table (2).

ACTIVATION FUNCTION	NUMBER OF BITS	ALGORITHM	FINAL ERROR %	TOTAL NN COST / CELL	MAX SPEED OF NN / MHZ
SIGMOID	30	CLASSIC	2	14,000,000	35
BIPOLAR	30	CLASSIC	4	455,000	35
SIGMOID	10	CLASSIC WITH TRUNCATION	16	280,000	74
BIPOLAR	10	CLASSIC WITH TRUNCATION	22	45,500	74
SIGMOID	10	PROPOSED	6	455,000	74
BIPOLAR	10	PROPOSED	9	45,500	74

7.Conclusion

Generally, any complete circuit of ANN has a very high cost because the high resolution of the weight values required is about 30-bit or more. The truncation of this data to low number such as 10-bit will reduce the total cost to 5% from the total cost but increase the error to more than 10 times.

In addition to this, the effect of the transfer function type such as sigmoid or bipolar function declared the cost of the classic NN, where sigmoid function is 30 times over the bipolar transfer function but the bipolar has in average twice the error.

This work has shown capability choice of different transfer functions for NN algorithm that are suitable to reduce the hardware cost with an optimal choice of precision value, by applying proposed truncation algorithm to reduced precision value from 30-bit to 10-bit with minimum possible error.

The matching of the proposed algorithm with bipolar transfer function will reduce the cost to 0.3% from the total cost and increases the speed to twice the original speed but increase the error from 2% to 10%. While, the direct truncation with classic learning algorithm with bipolar transfer function will give a very high error reaching to 25% while the truncation using the proposed learning algorithm with bipolar transfer function will reduce the error to less than 10% with the same cost and same speed.

8.References

[1] M. Skrbek, "Fast Neural Network Implementation", Neural Network World, vol. Vol. 9, n. No. 5, pp. 375–391, 1999.

[2] J. G. Eldredge, "FPGA Density Enhancement of a Neural Network Through Run-Time Reconfiguration", Master's thesis, Department of Electrical and Computer Engineering, Brigham Young University, May 1994.

[3] D.E Rumelhart, J.L McClelland and PDP Research Group, Parallel Distrubuted Processing: Explorations in the Microstructure of Cognition, volume 1: Foundations, MIT Press, Cambridge, Massachusetts, 1986.

[4] A. Dehon, "The Density Advantage of Configurable Computing", IEEE Computer, vol. 33, n. 5, pp. 41–49, April 2000.

[5] Holt, J.L., T.E. Baker. Back propagation simulations using limited precision calculations, in Proceedings of International Joint Conference on Neural Networks. 1991. pp 121-126 vol. 2.

[6] E. K. Knight, "Artificial Intellingence" McGraw-Hill New York, Second Edition, 1991.

[7] Xilinx.Com, "VirtexTM2.5V FPGAs", Data Sheet, www.Xilinx.com, May 1999.

[8] Wolf, D.F., Romero, R. A. F., Marques, E. Using Embedded Processors in Hardware Models of Artificial Neural Networks. In proceedings of SBAI - Simpśio Brasileiro de Automao Inteligente. 2001. pp 78-83.

9.List of symbols

ANN: Artificial Neural Network.
ASIC: Specific Integrated Circuit.
CORDIC: COordinate Rotation DIgital Computer.
FPGA: Field Programmable Gate Array.
LC: Logic Cell.
LUT: Look- UP- Table.
RAM: Random Access Memory.

بناء مستوى واطئ الكلّفة لخوارزمية الانتشار العكسي

عمار عادل حسن
قسم هندسة الحاسبات/ كلية الهندسة
جامعة بغداد

الخلاصة:
من أولى التطبيقات الناجحة التي تم نشرُها للشبكات العصبية الاصطناعية (ANNs) كانت قليلة وقبل أكثر من عقد. لذلك حان الوقت لمراجعة التقدّم الذي تم في مجال هذا النوع من البحوث. يمتاز هذا الملخص بتوفير الفكرة الأساسية حول تطبيق أصناف الأنواع المتوفرة للبوابات المرتبة بصيغة صفوف قابلة للبرمجة (FPGAs) لبناء الشبكات العصبية الاصطناعية. تقنيات مختلفة التطبيق وأفكار للتصميم سيتم مناقشتها لاحقاً, مثلاً الحصول على الدالة الفاعلة والمناسبة وتقنية التقليم العددية. كذلك, العمل على تحسين خوارزمية التعلم للتقليل من كلفة بناء الخلية العصبية وبالتالي تقليل الكلفة الكلية وتحسين أداء الشبكة العصبية. وأخيراً, بناء دائرة متكاملة لها السرعة العالية لتميز أشكال الأرقام الإنكليزية من خلال شبكة عصبية اصطناعية لها أربعة طبقات من خلال (٧٠) عقدة (خلية عصبية) على رقاقة واحدة باستخدام تقنية Xilinx FPGA.