FPGA Realization of Activation Function for Neural Network

Shang-Ping Pan ZhaoFang Li and Yu-Jung Huang
Department of Electronic Engineering
I-Shou University
Kaohsiung 84008, Taiwan
e-mail:yjhuang@isu.edu.tw

Wei-Cheng Lin
Medical Engineering Department
E-DA Hospital
Kaohsiung City 824, Taiwan

***Abstract*—This paper presents the FPGA implementation of neuron block units based on a sigmoid activation function for artificial neural networks (ANNs) applications. The proposed system was simulated using Modelsim XE II and synthesized using Synopsys design compiler based on TSMC 90 nm technology file. The functionality of backward propagation was successfully verified as compared with the python simulation results.**

Keywords—Sigmoid activation function; ANN; FPGA

I. INTRODUCTION

An Artificial Neural network (ANN) [1] have broad applicability to real-world problems and fast-growing Artificial Intelligence (AI) techniques used both in educational institutions and industries during recent years. Artificial neuron represents real neuron mathematically. The concept of deep learning comes from the study of the artificial neural network, multilayer perceptron which contains more hidden layers in a deep learning structure. In recent years, deep learning based on the artificial neural network has achieved great success in pattern recognition fields, such as speech recognition, image classification, and face recognition. The software-based ANNs have a disadvantage of slower execution compared with hardware-based ANNs in real-time applications [2]. A large number of hardware architectures have been proposed for the implementation of ANNs. ANN may be realized by using analog systems or digital systems. In addition, some of the existing platforms available for hardware implementation of ANN are Digital Signal Processing (DSP) chips, Application Specific Integrated Circuits (ASICs), Graphical Processing Unit (GPU) [3] or Field programmable gate array (FPGAs) [4].

II. ANN MODEL

As shown in Fig. 1, the block diagram of the sigmoid unit shows the model for the multilayer feed-forward ANN structure with the neuron structure used in the hidden layer. A neuron forms the basis for designing the ANNs. A neuron forms the basis for designing the ANNs. A neuron forms the basis for designing the ANNs. The output activation f for the neuron is described by the following equations

$$o_i = f(\sum_{j=1}^{m} w_{ij} x_j + b_i) \tag{1}$$

The w_{ij} denotes the weights connecting the jth input unit to the ith hidden unit. The weighted summation adds up the products of previous neurons multiplied by the corresponding

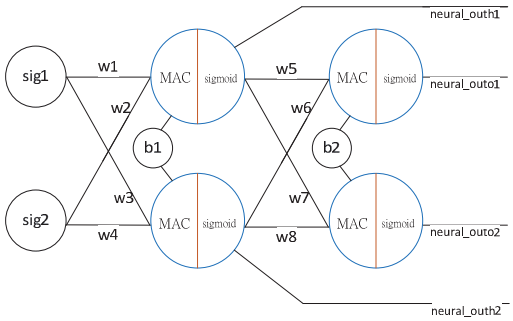

Fig. 1 A typical ANN structure

weights, and then, the sigmoid function $f(\vartheta)$ is operated to calculate the output. The nonlinear activation is typically chosen to be the sigmoidal function

$$f(\vartheta) = \frac{1}{1+\exp(-\vartheta)} \tag{2}$$

The training data consists of a set of X_j input patterns where j represents the pattern number and W_{ij} the corresponding trained weight in the ith hidden layer. The training procedure of the two-layer perceptron is implemented in Tensorflow. TensorFlow minimizes cross_entropy using the gradient descent algorithm with a learning rate of 0.01. Cross-entropy is defined as:

$$H_{y'}(y) = \sum_i y_i' \log(y_i) \tag{3}$$

Where y is our predicted probability distribution, and y′ is the true distribution (the one-hot vector with the digit labels). Gradient descent is a simple procedure, where TensorFlow simply shifts each variable a little bit in the direction that reduces the cost. Here, our loss function is the cross-entropy between the target and the activation function applied to the model's prediction. The role of activation functions is to make neural networks non-linear. The sigmoid non-linearity takes a real-valued number and "squashes" it into a range

978-1-5386-1445-7/18/$31.00 ©2018 IEEE

between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1.

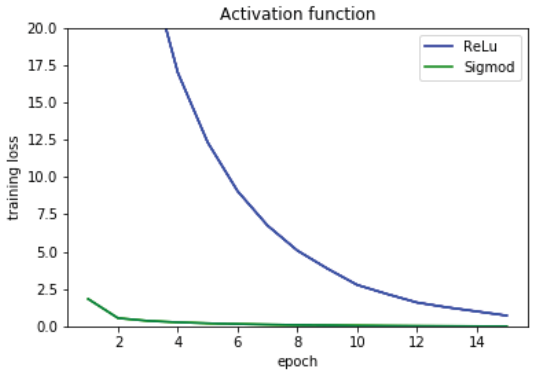

Fig. 2 Training Loss Over Iterations for the ANN Model

The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron. ReLU units can be fragile during training. a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any data point again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. Fig. 2 depicts the training loss for two types of the activation function. For a two-layer perceptron, the training loss of sigmoid function is less than that of ReLu function.

III. HARDWARE IMPLEMENTATION

The trained weight matrix obtained from Python simulations is used as the input vector of the ANN on the present FPGA implementation. To evaluate the hardware implementation of the neuron block units, the design has been simulated and synthesized using Synopsys design compiler logic synthesis tool. Table1 lists the logic element gate counts and the power consumption summary of the design complier report based on SMC 90 nm technology file.

Table 1 Design Complier Report

Power Group	Internal	Switching	Leakage	Total	%
Register	24.3903	9.9259E-3	1.154E9	25.5547	87.45
Sequential	2.7020E-3	7.6993E-7	4.0251E6	6.7279E-3	0.02
Combinational	9.7326E-2	0.2576	3.3044E9	3.6593	12.52
Total (mW)	24.4903	0.2675	4.4624	29.2207	99.99
Combination area		1164630.191792 um^2			
Buf/Inv area		55606.221034 um^2			
Noncombinational area		555566.869473 um^2			
Toral cell area		1720197.061265 um^2			

The simulation results of the output of the neuron block on ANN-based architecture with backward propagation are shown in Fig. 3. Neural_Output is the output of neurons implemented based on the present proposed architecture. The output results of neuron block units are consistent with the results from Python simulation as listed on Table 2, which confirm our successful implementation.

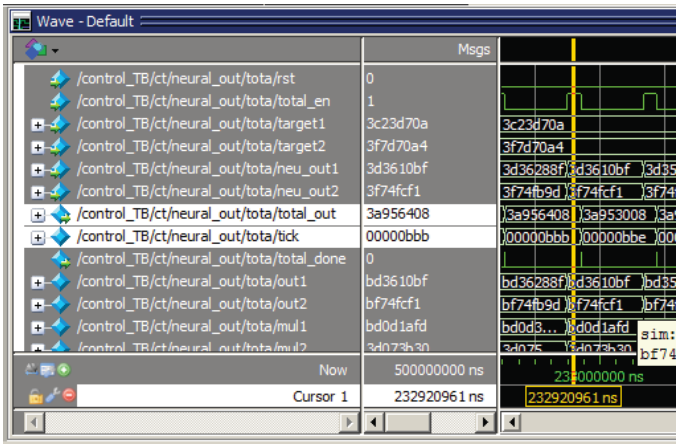

Fig. 3 Simulation result of neuron block unit in ANN

Table2 Software v.s. Hardware Computational Results

Iteration (n)	Python (value)	modelsim (i754)	Modelsim (value)
0	0.291027774	0x3e950195	0.2910277
1	0.283547133	0x3e912d15	0.28354707
99	2.0046E-2	0x3ca4f5c0	2.013671E-2
999	1.1143E-3	0x3a956408	1.139761E-3
1999	4.4486E-4	0x39f09020	4.588375E-4
9999	3.5102E-5	0x381d0100	3.743265E-5

IV. CONCLUSION

The proposed ANN consists of sigmoid activation function with floating-point arithmetic is realized using FPGA devices. This work has the conclusion of successful FPGA implementation of neuron block unit using CORDIC algorithm for the approximation of activation functions. The python trained weight vector of backward propagation for the neuron block is used to verify our output of the present proposed ANN architecture.

ACKNOWLEDGMENT

This work was supported in part by the I-Shou University under Grants ISU-106-ICU-03.

REFERENCES

[1] Simon Haykin, Neural Networks: A Comprehensive Foundation, 2ed. Addison Wesley Longman (Singapore) Private Limited, Delhi, 2001

[2] D.S. Jinde, S.S. Thorat, Neural network implementation using FPGAs, Int. J. Com-put. Sci. Inf. Technol. 5 (3) (2014) 3431–3433.

[3] F.O. Zamorano, J.M. Jerez, G. Juarez, J.O. Perez, L. Franco, High precision FPGA implementation of neural network activation functions, Intelligent EmbeddedSystems (IES), Orlando, FL. IEEE (2014) 55–60.

[4] B. Li et. al, Large-scale recurrent neural network on GPU, Neural Networks (IJCNN), 2014 International Joint Conference on, 2014: 4062-4069.