Implementation of Neural Network on FPGA

Ranjith M S[1], Sampanna T[2], Niranjan Ganapati Hegde[3], Shruthi R[4]

[1,2,3]Student, Dept. of ECE, The National Institute of Engineering, Mysuru, India
[4]Assistant Professor, Dept. of ECE, The National Institute of Engineering, Mysuru, India

---***---

Abstract - *Deep neural networks and their application is rapidly changing many of the fields by bringing a different way of solution to the problems. In fact, it is being applied in every field right from Signal processing to the communication. Although it solves many problems easily, training and deploying it is limited due to various reasons. These deep neural networks require most powerful computational devices like GPU in order to train them and large RAM, memory is required for its operation limiting its usage in many of the applications. Even if these resources are provided through cloud the deployment of system faces problems like more latency as there is round trip to the server, connectivity, privacy as the data needs to leave the system. All these problems could be overcome using the FPGA's for the deployment of the neural network as they are low cost and reconfigurable. In fact, FPGA's are faster than GPU during inference in many of the cases making it suitable for the deployment in the real time application. FPGA's are faster than GPU's as the neural network is implemented as a hardware in case of FPGA's, where only hardware delays are introduced but in case of CPU or GPU large number of floating-point operations needs to be done using ALU. We will implement a trained neural network of TensorFlow on FPGA using Verilog HDL to perform a regression task.*

Key Words: Deep learning, FPGA, Regression, Verilog HDL, Artificial Intelligence, Neural Networks, TensorFlow

1. INTRODUCTION

Currently a variety of applications like image processing are deployed on FPGA's because of their advantages like cost-optimized system integration, differentiated designs for a wide range of embedded applications, a significant speed advantage compared to processor-based solutions. Even complicated calculations can be carried out in an extremely short time using FPGA's.

Deep neural networks are employed for the various tasks in various fields like object detection, speech recognition, natural language processing, segmentation most of these are deployed on GPU. We use FPGA to employ deep neural network for the regression task. Hardware is generated for the trained neural network which includes many floating-point operations to be performed before the prediction is done i.e., to obtain the output. Initially the network is implemented and trained in python environment using TensorFlow and the values of the weights are obtained as an array from the trained model and these floating-point numbers are converted into IEEE 754 double precision format. The whole network is then broken into floating point multiplication and addition which is implemented using Verilog HDL. Since Verilog HDL does not perform floating point arithmetic by default we implement the modules to perform floating point addition and multiplication on IEEE 754 double precision format based on their sign, exponent and mantissa part.

Since we are using only dense networks we use *He initialization* [1] as compared with *Xavier Initialization* [2] for initializing the weights while training the network.

""overfitting" is greatly reduced by randomly omitting half of the feature detectors on each training case. This prevents complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors. Instead, each neuron learns to detect a feature that is generally helpful for producing the correct answer given the combinatorially large variety of internal contexts in which it must operate. Random "dropout" gives big improvements on many benchmark tasks" [3].

The optimizers such as Adam [4], rms [5], LARS [6] leads for the faster convergence and better optimization without getting stuck at the saddle points or at the local minima's. "*One of the key elements of super-convergence is training with one learning rate cycle and a large maximum learning rate*" [7]. To speed up training we use *Batch Normalization* [8].

2. EXPERIMENTAL SETUP

2.1 Dataset

The dataset contains sold prices of many houses with their information like overall area in square feet. The maximum value of the area is 1.306800e+06 whereas the maximum value of the price column is 3600. If these raw values of the dataset are passed into the network for training, it leads to overshooting and also slows the rate of convergence as the loss surface is not smooth. The loss surface will have large number of saddle points making it difficult for training. These problems could be overcome by doing the mean normalization on the data before training and same has to be applied during the inference.

Mean normalization is given by,

$$x_i := \frac{x_i - \mu_i}{s_i}$$

Where μ(i) is the average of all the values for a feature and *s(i)* is the standard deviation. After mean normalization the mean of a feature is nearly 0 and standard deviation is nearly 1. Which counters the vanishing gradient problem.

Fig-1 represent how the price varies with area before the mean normailzation is applied.

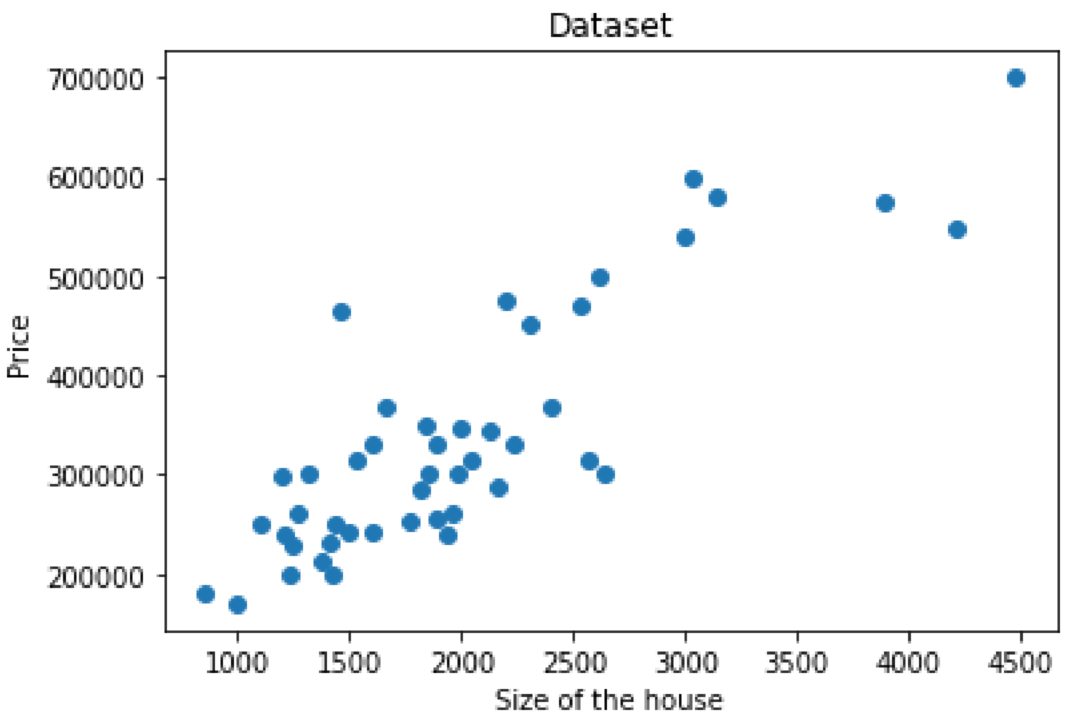


fig-1: Dataset

2.2 Training the Neural Network

We implemented two architectures for the same dataset, perceptron and the neural network (multi-output perceptron). The perceptron is implemented in python environment and the gradient descent algorithm is used to find the optimal values of the weight and bias. Mean squared error is used as a cost function based on this the parameters are optimized.

Neural network is implemented and trained using TensorFlow library in python environment. The network has 2 hidden layers and an output layer. First hidden layer has 8 hidden units to which ReLU activation is applied the purpose of activation function is to introduce the non-linearities in the network. The output of the activation is applied to the second dense layer with the 4 hidden units, output of which is then applied with ReLU activation and these values are used to predict the final output. The network architecture is as shown in fig-2. And the structure parameter and its dimension are as in table-1.

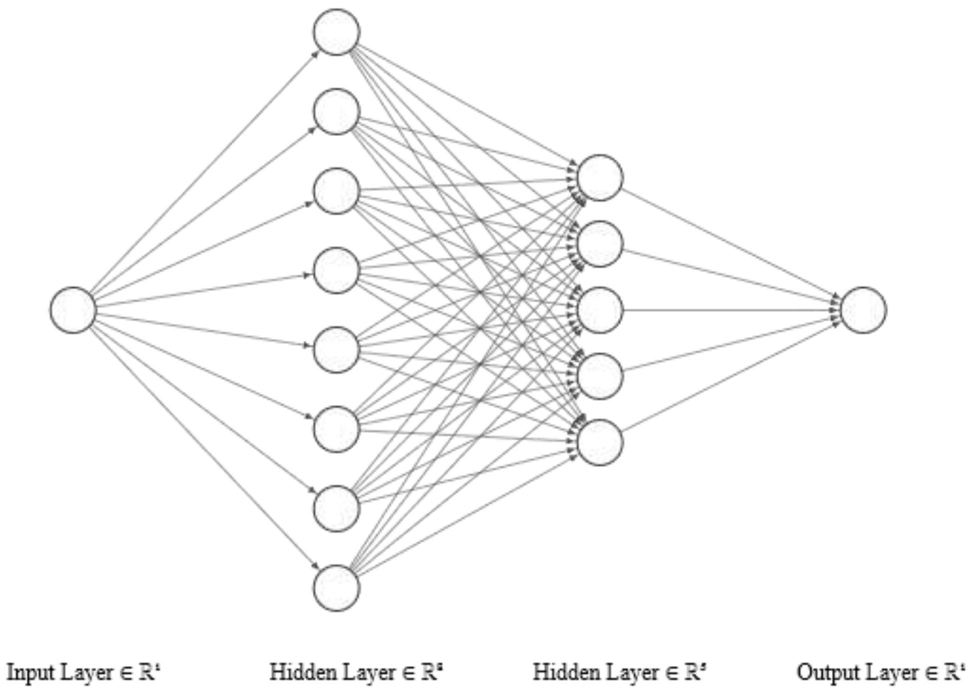

Input Layer $\in \mathbb{R}^1$
Hidden Layer $\in \mathbb{R}^8$
Hidden Layer $\in \mathbb{R}^5$
Output Layer $\in \mathbb{R}^1$

fig -2: Neural network architecture

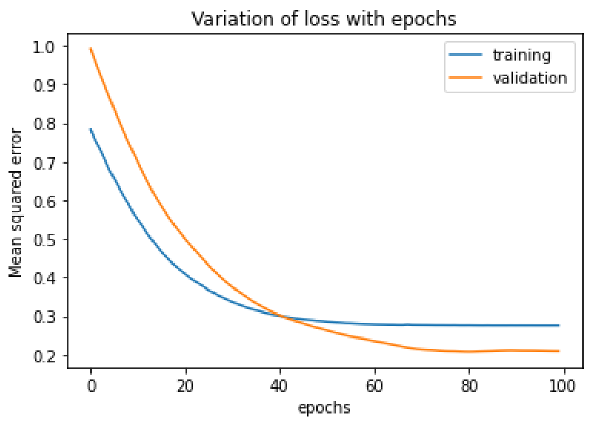
Table -1: Network structure

Model structure and parameters		
Layer	**Output shape**	**NO. of parameters**
Input	(None, 1)	0
Dense_1	(None, 8)	16
Dense_2	(None, 4)	36
Output	(None, 1)	5
Total params: 57 Trainable params: 57 Non-trainable params: 0		

The network is trained using Adam optimizer with a batch size of 32 and the data is split into 90% for training and 10% for validation. Mean squared error is used as a loss function.

Table -2: Network results after training

Final loss	
	Mean Squared Error
Training	0.2753
validation	0.2088



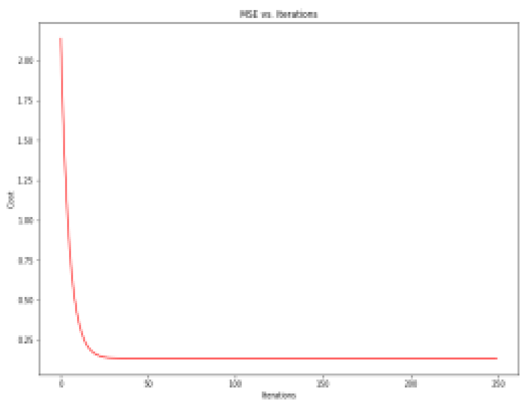


fig -3: loss during training for neural network **fig -4**: loss during training in perceptron

From table-2 we see the mean squared error values after training the neural network. Since the values of training error and validation error are near it indicates that the model is not overfitted to the training data.

Fig-3 indicates how the loss (mean squared error) varied during the training of the neural network and fig-4 indicates how the loss (mean squared error) varied during the training (only training data) of the perceptron. In both the cases loss reached very low value by the end of the training indicating that they have learnt to predict the proper values of the target.

The model was trained on Tesla K80 GPU which can do up to 8.73 Teraflops single-precision and up to 2.91 Teraflops double-precision performance with NVIDIA GPU Boost

3. IMPLEMENTATION OF TRAINED MODEL IN VERILOG HDL

The trained network is implemented in Verilog HDL to perform the inference. Since the network implementation includes the floating-point arithmetic and Verilog does not have any functions for the floating-point operations the operations have to be implemented based on the sign, exponent and mantissa. The optimized weights are converted into IEEE 754 double precision format having 64 bits.

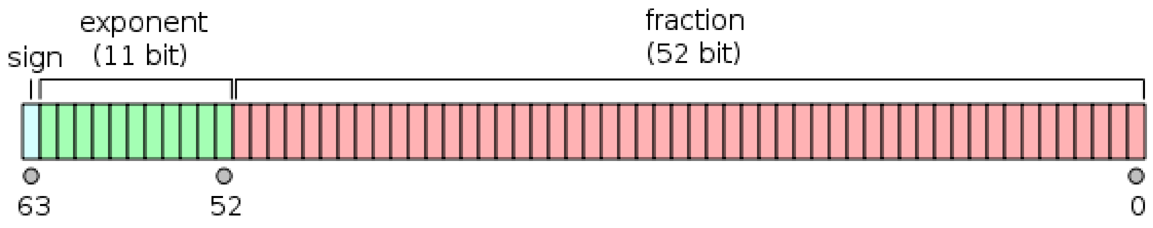


fig-5 IEEE 754 double precision format

Fig-5 shows the representation of the IEEE 754 double precision format. First bit indicates sign 1 indicate negative number and 0 indicate positive number. Bit 62 to 52 indicates the exponent part and other bits indicates fractional part.

In floating point multiplication, the result will be negative if any one of the numbers is negative, result will be positive if both are negative or positive which indicates the behaviour of XOR gate. Hence the resultant sign bit of the output is determined by XOR of two sign bits of the numbers.

Exponent part of the resultant is determined by using the formula,

$E_r = E_1 + E_2 - 1023$

Where E_1 and E_2 are the exponent part of two numbers. Hence the 11-bit adder is required for the exponent part of the resultant.

Fractional part is obtained by the normal multiplication between fractional parts of the number. And result is rounded to 52 bits. Hence the multiplier is implemented for the fractional part.

Hence all the trained network parameters are converted into IEEE 754 double precision format and the whole network is implemented using these floating-point operation modules. The module now takes the input and operations are done with the optimized weights to get the prediction.

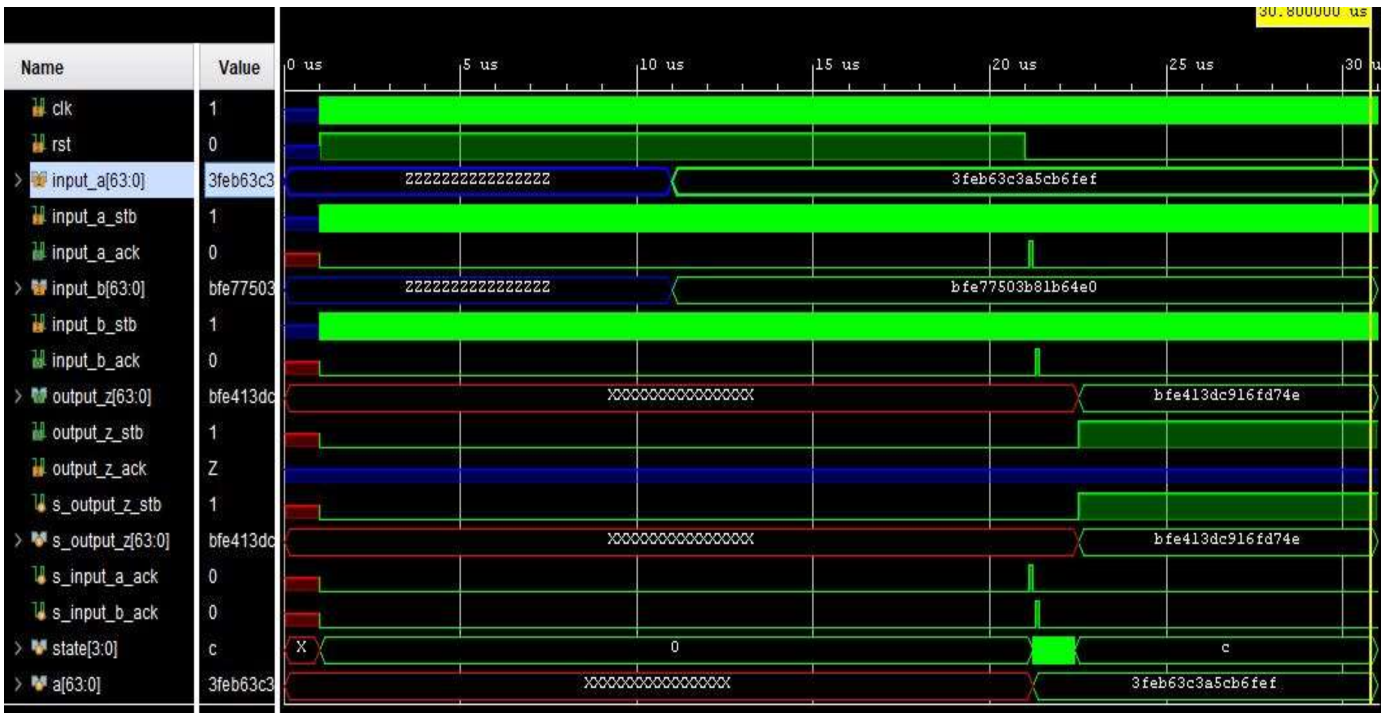


Fig-6: Simulation output

From fig-6 we observe the output prediction (output_z) of the implemented hardware for the given mean normalized input square feet.

4. RESULTS AND CONCLUSIONS

The implemented hardware using Verilog HDL produced the same result as that of the GPU or CPU which computes the result using ALU.

The hardware implementation reduces the latency when compared with GPU or CPU in many cases. But the implementation of the neural network on the hardware takes more time and the training of neural network is not possible in FPGA as there are more training loops involved and the gradient computation and their update.

REFERENCES

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

[2] Bengio, Yoshua and Glorot, Xavier. Understanding thedifficulty of training deep feedforward neural networks.InProceedings of AISTATS 2010, volume 9, pp. 249–256, May 2010.

[3] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, Ruslan R. Salakhutdinov Improving neural networks by preventing co-adaptation of feature detectors arXiv:1207.0580v1 [cs.NE]

[4] Diederik P. Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization. arXiv:1412.6980v9 [cs.LG]

[5] Geoffrey Hinton. 2012. Neural Networks for MachineLearning - Lecture 6a - Overview of mini-batch gradi-ent descent.

[6] Yang You, Igor Gitman, Boris Ginsburg. Large Batch Training of Convolutional Networks. arXiv:1708.03888v3 [cs.CV]

[7] Leslie N. Smith, Nicholay Topin. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates. arXiv:1708.07120v3 [cs.LG]

[8] Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:1502.03167v3 [cs.LG]

[9] Alexandre de Brébisson, Étienne Simon, Alex Auvolat, Pascal Vincent, Yoshua Bengio. Artificial Neural Networks Applied to Taxi Destination Prediction. arXiv:1508.00021v2 [cs.LG]

[10] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S.,Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G.,Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:1603.04467 [cs.DC]

[11] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

[12] Plotly Technologies Inc. (2015). Collaborative data science. Montreal, QC: Plotly Technolo-gies Inc.

[13] S. Sahin, Y. Becerik1i, S. Yazici, "Neural network implementation in hardware using FPGAs", *NIP Neural Information Processing*, vol. 4234, no. 3, pp. 1105-1112, 2006.

[14] Ranjith M S, S Parameshwara. Optimizing Neural Networks for Embedded Systems. IRJET Volume 7, Issue 4, April 2020 S.NO: 211

[15] Yufeng Hao. A General Neural Network Hardware Architecture on FPGA
arXiv:1711.05860 [cs.CV]

[16] S. Coric, I. Latinovic and A. Pavasovic, "A neural network FPGA implementation," *Proceedings of the 5th Seminar on Neural Network Applications in Electrical Engineering. NEUREL 2000 (IEEE Cat. No.00EX287)*, Belgrade, Yugoslavia, 2000, pp. 117-120.