EE705 Course Project: Hardware implementation of Machine Learning classifiers

Mihir Kavishwar (17D070004)
Anubhav Agarwal (17D070026)
Shailee Suryawanshi (17D070049)
Prashant Kurrey (17D070057)

***Abstract*—This report summarizes our work in the area of machine learning hardware. We attempted to implement two popular machine learning models in hardware - Neural Network and Support Vector Machine (SVM). Both of these implementations were tested on an open source data set of medical records of patients with diabetes. Testing accuracy of 70% was achieved for SVM classifier and 80% for Neural Net classifier. We wrote the code in Verilog used Quartus tools for simulation.**

***Index Terms*—machine learning, verilog, fpga, neural network, support vector machine**

I. Introduction

Over the last few years there has been growing interest in exploring hardware based solutions to solve some of the fundamental problems associated with Machine Learning techniques. As the size of datasets and complexity of machine learning models grows, the need for faster computation and lower power consumption becomes extremely critical.

As shown in Figure 1 and 2, Machine Learning typically consists of 2 parts:

1) Training
2) Inference

The former is usually not as time critical as the later if we want to deploy a product in the field since training has to be done only once. Therefore, our focus was only to implement the trained models in hardware. This allows us to a high level language like Python for training and directly used the trained values of parameters. The hardware implementation was done in Verilog HDL. Since Verilog has a module based syntax, we started by writing modules for the simplest units (adder, multiplier, etc) and then combined them to build larger modules (neuron, svm kernel, etc). The design was kept as configurable as possible so that we can tune the hyperparameters such as neural net layer sizes, number of support vectors in SVM, etc. IEEE 754 Full Precision floating point representation was used to for data. The design can be uploaded on an FPGA which can be then used in the field for some application.

II. Support Vector Machine

The model was first trained in the Python and then the support vectors and dual coefficients were exported to a hex file. The hex file was imported in the hardware implementation as ROM memory. The training was done by changing the parameter

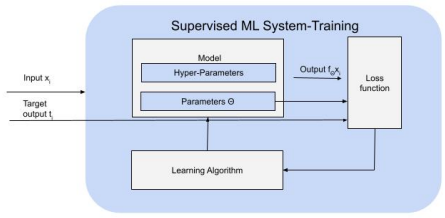

Fig. 1. Training

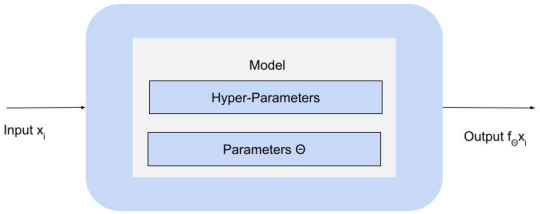

Fig. 2. Inference

to get the maximum accuracy. The following parameters are selected for SVM for this particular data set :

- Kernel: RBF
- γ: 0.01
- Number of Support Vectors: 385 with 8 features each

In SVM, the RBF kernel for binary classification is following:

$$\Sigma(y_i \alpha_i K(x, x_i)) + b > 0 : \implies class = 1$$

$$K(x, xi) = e^{-\gamma ||X - X_i||^2}$$

The above equation consists of addition, subtraction and multiplication. For all of this we have used the single 32 bit floating point adder and multiplier. The hardware SVM first takes the normalised input which consists of 8 features, 32 bit each. Then one support vector is taken and all the features are subtracted from the features of support vector multiplied with each other to get the $||X - X_i||^2$. Then this value is multiplied with $-\gamma$. This value acts as input to the exponential term. For the computation of exponential term, we implemented the Taylor series for exponential upto 15 terms since the computations again boil down to multiplication and addition.

Here the kernel $K(x, xi) = e^{-\gamma ||X - X_i||^2}$ is multiplied with the dual coefficient and similarly we will compute this for all

the support vectors and then add them together and finally with the bias value. Finally we check the sign bit. Negative number implies the class 0 , positive number implies class 1.

III. Neural Network

For simplicity, a 3-layer neural network model was trained in python. The number of neurons in each layer was kept configurable. ReLU activation function was used in the hidden layer. In the training model, at the output layer, sigmoid activation was used for binary classification while softmax was used for multi-class classification.

$$f_{relu}(y_i) = max(0, y_i)$$

$$f_{sigmoid}(y_i) = \frac{1}{1 + e^{-y_i}}$$

$$f_{softmax}(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{N} e^{y_j}}$$

Since we are only interested in the final decision of the classifier, sigmoid was replaced by step and softmax was replaced by winner-take-all in the hardware implementation of the trained model.
Fundamentally, feedforward neural networks are just matrix multiplications interposed with non-linearities. For example, the inference equation for a trained binary neural network can be written as:

$$output = sign\left(w'_{l_2} f_{relu}\left(w'_{l_1} x_{in} + b_{l_1}\right) + b_{l_2}\right)$$

The weights and biases we imported from the trained model directly. We implemented the adder, multiplier, multiply-and-accumulate (MAC), ReLU and winner-take-all in hardware. Figure 3 shows the RTL netlist of a NN model with 10 neurons in hidden layer. Figure 4 shows the modules instantiated in a single neuron.

IV. Simulation Results

A testbench was written in verilog to compute the accuracy of our hardware implementations. A hardwired verification based methodology was followed. The testbench reads inputs from an input-rom and output-rom, drives inputs in the DUT and compares the dut output with the expected output stored in output rom. A count is maintained of correctly predicted outputs and accuracy is computed at the end based on what fraction of dut outputs matched expected outputs.

TABLE I
Test accuracy on diabetes dataset [3]

	Neural Net	SVM
Test accuracy	67.2 %	80.2 %

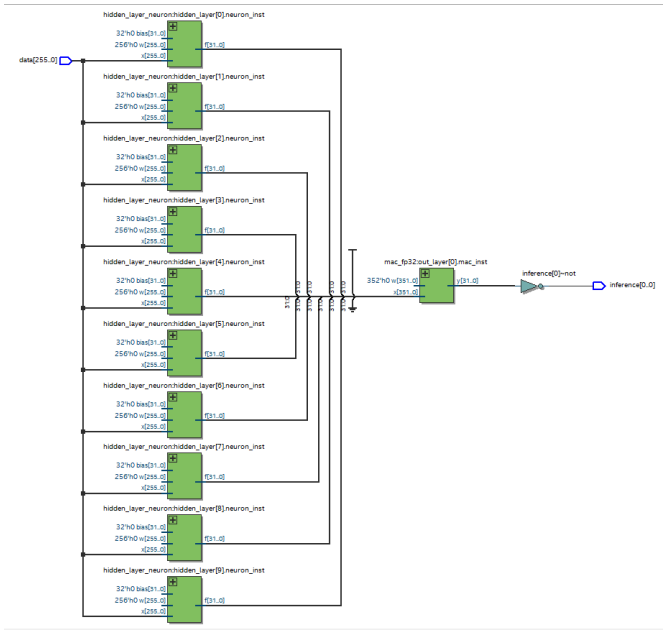

Fig. 3. Neural Network RTL Netlist

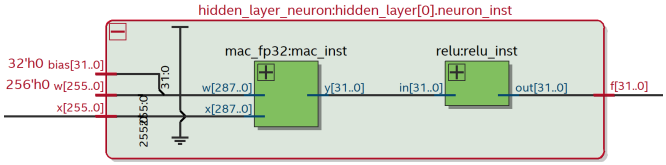

Fig. 4. Hidden Layer Neuron RTL Netlist

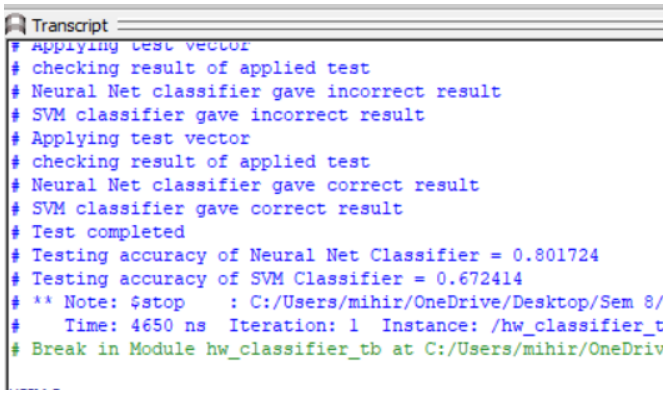

Fig. 5. Simulation result showing test accuracy of both models

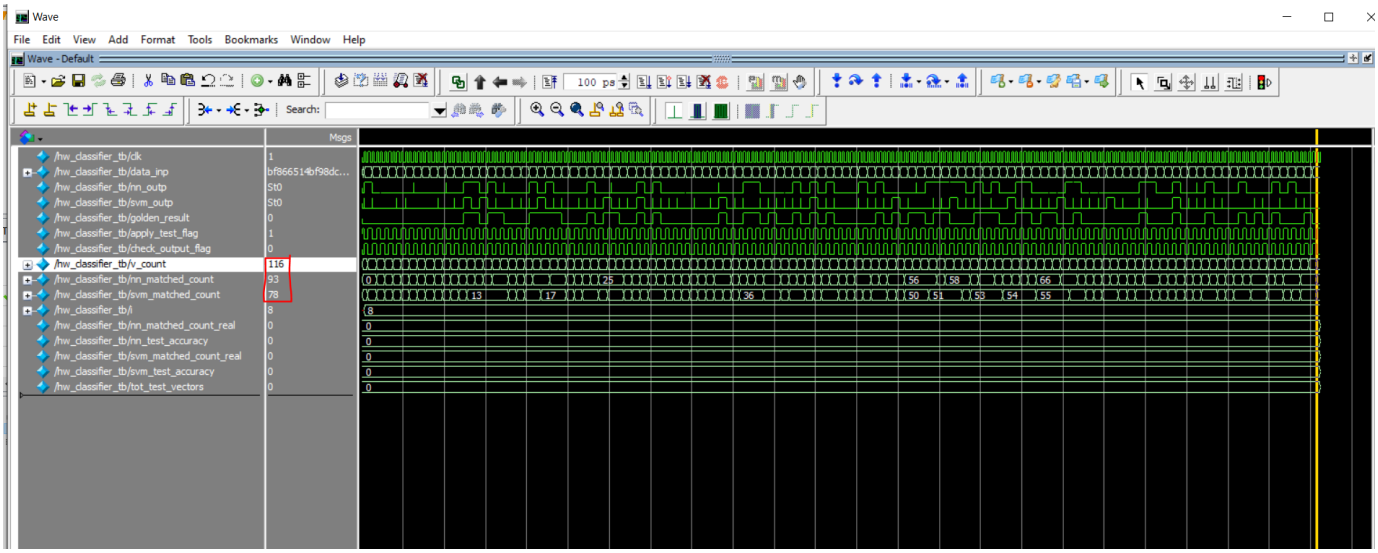

Fig. 6. Simulation result showing transient waveform of different signals

Accuracy for NN $= \frac{93}{116} = 80.2\%$, SVM $= \frac{78}{116} = 67.2\%$.

REFERENCES

[1] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[2] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier
[3] https://www.kaggle.com/uciml/pima-indians-diabetes-database
[4] https://github.com/sudhamshu091/32-Verilog-Mini-Projects/tree/main/Floating%20Point%20IEEE%20754%20Addition%20Subtraction