EEL 4783: Spring 2022

Verilog Implementation of a Feed Forward Neural Network

1. Problem Description:

In this project, you will implement a feed forward artificial neural network using verilog that performs handwritten digit classification on a subset of the MNIST dataset. The neural network implementation should be able to predict the digit when given an input image and network parameters. Finally, using the network implementation, you will calculate the network accuracy on the MNIST dataset.

2. Neural Network Architecture:

The Neural Network that you will implement consists of an input layer, one hidden layer with 30 neurons and one output layer. Each Neuron in the hidden layer and the output layer adds bias to the calculated activations. Additionally, the hidden layer uses a ReLU activation function. The network that you implement will only perform inference and not training i.e., given an input, the network should be capable of performing digit prediction. Visually, the network looks like the image below

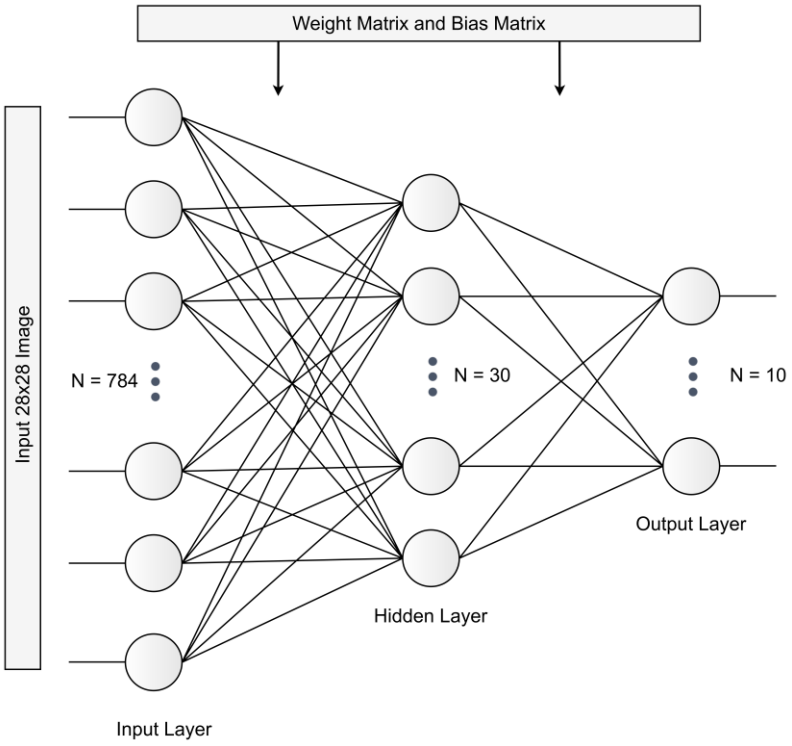

Figure 1: Neural network architecture

A neural network consists of layers of neurons. Each neuron of a layer receives an input from the previous layer (the input layer will receive a flattened version of the 28 x 28 image) and performs a weighted sum of the inputs. Additionally, each neuron adds a bias value to the calculated weighted sum. This resultant value is passed through a non-linear function called an activation function to produce an output. The output of each neuron in a layer is collectively called the activations of the layer. Neural networks can vary in implementation, as such, we will be using the following terminology

Input Layer:

For our implementation, neurons of the input layer do not perform the weighted sum, bias operation and do not perform an activation function. Instead, it is used to feed the inputs to the next hidden layer.

Hidden Layer:

The neurons of the hidden layer perform the weighted sum of inputs and add a bias to the resultant. Additionally, it performs the activation function on the resultant sum and produces an output as shown in Fig.2

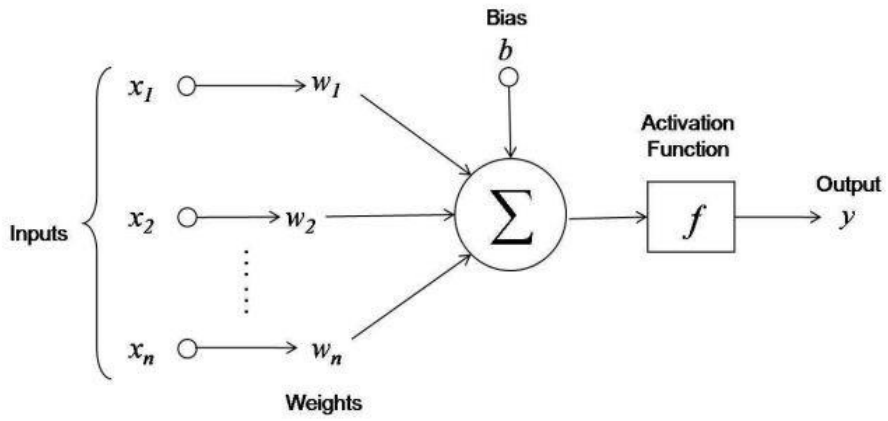

Figure 2: Internal working of a neuron of the hidden layer[1]

Output Layer:

The neuron of the output layer performs the weighted sum of inputs and adds a bias to the resultant. However, it does not perform an activation function. The neuron of the output layer with the max activation value will be the prediction of the entire network for that input image.

2.1 Network Specifications

[1] Image from Medium article

The implemented neural network should have the following specifications

	Input Layer	Hidden Layer	Output Layer
Neurons	28 x 28 = 784	30	10
Activation	None	ReLU	None
Bias	No	Yes	Yes
Weighted Input	No	Yes	Yes

Table 1: Specifications of the neural network layers

For more information about Neural networks and their functioning, refer to the following resources:

- But what is a neural network? | Chapter 1, Deep learning (Youtube)
- Activation Functions in Neural Networks (Medium)

3. Challenges of a hardware implementation

While implementing a neural network in a high-level language that can run on a general-purpose processor is relatively straightforward, implementing a neural network on hardware can pose new challenges.

3.1 Usage of floating-point numbers

Due to the precise nature of the weights and biases of a neural network, floating-point numbers are used to represent the fractional weights and biases. However, implementing floating-point arithmetic such as multiplication and addition used in a neuron can significantly increase the hardware complexity and resource utilization.

For our implementation, we will be using a fixed-point binary representation of the numbers which greatly simplifies the design complexity at the expense of network accuracy. Additionally, we will use the fixed-point binary representation to perform all operations and store in the registers throughout this project.

3.2 Negative weights and biases

A trained network can have negative numbers for weights and biases. Generally, to represent negative numbers in binary, we either use an additional "sign-bit" to represent the sign of the number or represent a negative binary number in its two's complement format.

However, for simplicity of design, we will not use negative weights, inputs and biases in our implementation. As such, all the multiplication and addition operations performed in the neural network will be done using positive numbers and result in non-negative numbers.

Resources for fixed point binary arithmetic in verilog:

- https://projectf.io/posts/fixed-point-numbers-in-verilog

4. Step-By-Step Implementation Guide

Below are the recommended steps to be followed to implement the neural network.

4.1 Input to the network

The MNIST dataset is a collection of 70,000 28x28 pixel grayscale image representation of a handwritten number as shown in Fig.3. Each pixel of the image can have a value between 0 and 255 with the lower end representing light pixels and the higher end representing darker pixels.

Figure 3: Images of the MNIST dataset [2]

Each image in the dataset is converted to a 2D pixel value representation. This is further flattened into a 1D array. An example of this process is shown in Fig.4

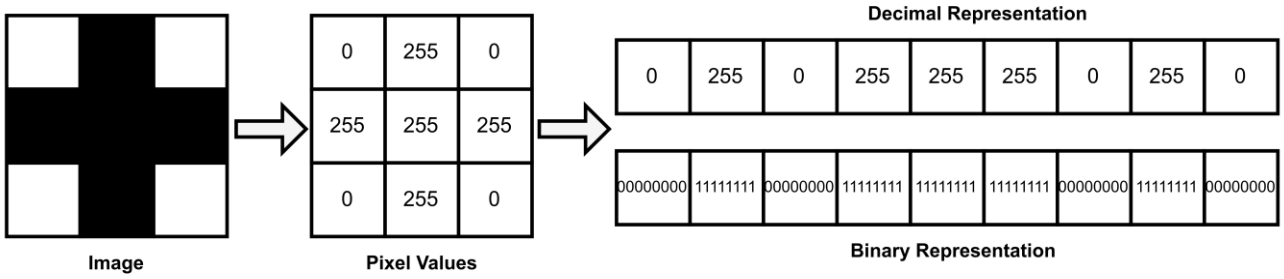

Figure 4: Grayscale Image to 1D array of 8-bit binary numbers

The input files stored in project/images will be a .txt file that stores the flattened 16-bit fixed point binary representation of the 28x28 pixel values. The underscore, "_", in the input file is used to separate the integer and fractional values. A binary value of 00000000_11000010 represents a

[2] Image from Wikipedia

decimal value of 0.7578125. Each input file represents a single image and contains 784 (28x28) 16-bit fixed point binary numbers.

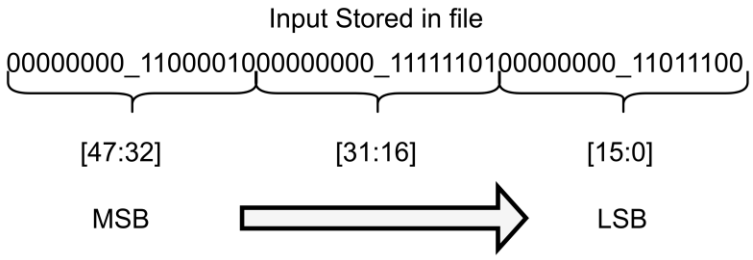

Figure 5: Input text file format

You must first read the file and store its contents into a register and use the register as input to your neural network. Reading the file and storing in a register is shown in Section 4.7

4.2 Input Layer

The first step to implementing the network is to create a verilog module, input layer. This input layer acts as an interface between the image input and the next hidden layer.

Input:

The image input will have 784 values, each 16-bit long (as we are using a 16-bit fixed point representation) which will be the input to the input layer. In the provided stub code in "project/code/input_layer.v", the precision and image size are set as configurable parameters. You may choose to use them in your code wherever required.

Output:

The output should also be a wire/register that holds exactly 784, 16-bit values and should act as a pass through between the input and the output. For ease of use, all the inputs and outputs of all the modules will be flattened and stored in a single register/ wire of the required dimension.

4.3 Neuron

The next module to be created is a neuron. A neuron should be capable of taking N 16-bit inputs from the previous layer and N 16-bit weights and 1 16-bit bias value and should perform the following operation.

$$Output = \Sigma_{i=0}^{N}(Input[i] \cdot Weight[i]) + Bias$$

Here N is the number of neurons of the previous layer and the output is a single 16-bit number. For more information about multiplication and useful verilog statements, refer to the following:

- https://projectf.io/posts/fixed-point-numbers-in-verilog

- https://fpgatutorial.com/verilog-generate/

The testbench file for a single neuron is provided in “project/testbench/neuron_tb.v” and the stub code can be found in “project/code/neuron.v”

4.4 Hidden Layer

Design a module hidden_layer that instantiates multiple neurons. The number of neurons can be set as a configurable parameter (M).

Input:

The input to the hidden layer will be the activations of the previous layer. If the previous layer has N neurons, the input to the current layer will be N 16-bit values. The module hidden_layer must instantiate M neurons and must provide the proper interfacing between the input to the hidden layer and input to the constituent neurons.

Weights and Bias:

The weights and bias of the hidden layer will be read from the provided .txt files located in “project/weights/hidden_layer_weight.txt” and “project/bias/hidden_layer_bias.txt”. Each file in the folder is the weights and bias for each layer of network. For a hidden layer with M neurons, the weights file will have M*N 16-bit values where N is the number of neurons of the previous layer.

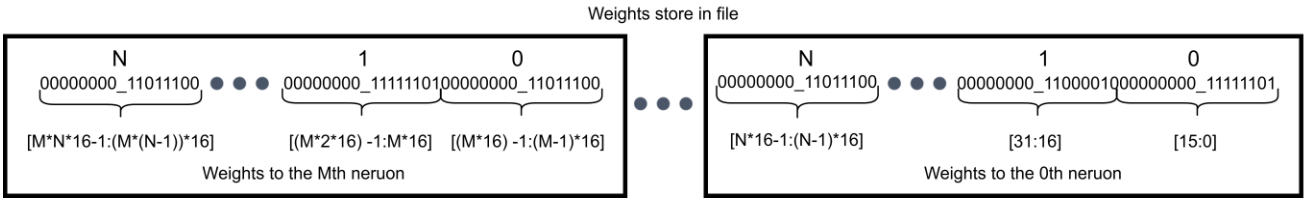

Figure 6: Weight text file format

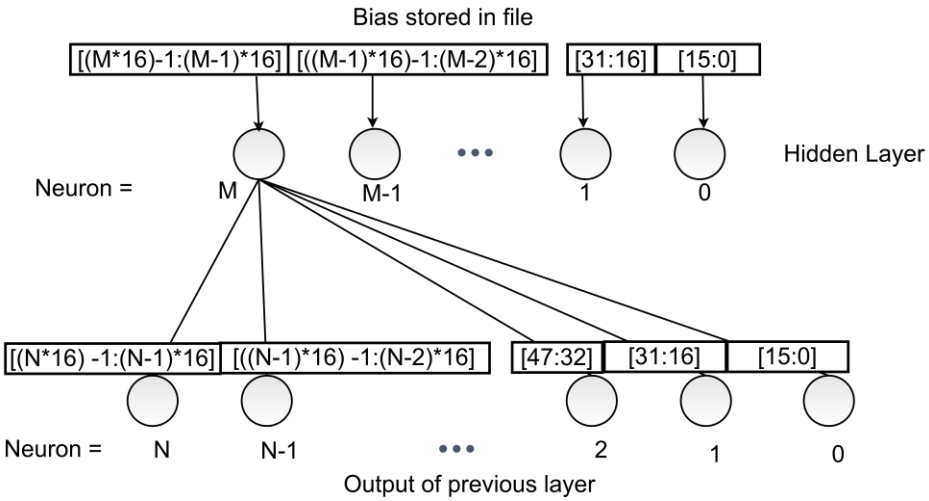

Figure 7: Bias and input to the hidden layer

The bias file will have M 16-bit values where M is the number of neurons of the hidden layer.

Output:

Each output of the constituent neuron must then be concatenated to form a register/wire of length (M*16) which will be the output of the hidden layer.

The testbench file for a hidden_laeyr is provided in “project/testbench/hidden_layer_tb.v” and the stub code can be found in “project/code/hidden_layer.v”

4.5 Activation function

For the activation function you can write your function code in the project/code/hidden_layer.v file. The pseudo code for ReLU activation function can be written as

```
if (input > 0):
        return input
else:
        return 0
```

4.5 Output layer

The output layer is the same as hidden layer without the activation function.

4.6 Network Prediction

The activation/output of each neuron of the output layer is the final output of the neural network. The index of the neuron (X) with the maximum activation value will be the predicted output of our network. Due to our naming convention, the final prediction will be (9-X)

4.7 Providing Input to Vivado Simulation

All the input image, weights and bias are stored in 16-bit fixed point binary representation. Use the verilog functions `$readmemb("file.txt", memory_array)` to read a binary “file.txt” and store its contents in the reg memory_array.

For more information about $readmemb and other functions, refer to the following:

https://projectf.io/posts/initialize-memory-in-verilog/

4.8 Debugging

The testbench to each module can be found in “project/testbench” folder. Make sure each of the testbench for each module runs and provides the required output before trying to run the whole network.

5. Experiments and Report

After finishing the implementation of the neural network, perform the predictions on a subset of the MNIST dataset. The folder "project/images" contains 100 binary image files, and the folder "project/labels" contains the file with the actual label corresponding to each image. Compare the predicted output and the label to calculate the accuracy of the network on the 100 images. The testbench for accuracy stub code is provided in "project/testbench/accuracy_tb.sv". Write a short 2 to 4-page report showing the output of the simulation for the input image files image_0, image_1 and image_2. Along with the simulation waveforms, write down about the challenges you faced during the implementation and any additional modifications you have made and fill in the tables self-evaluation checklist.

6. What to Submit on Webcourses

- All the source code and testbench files
- Report as described in Section 5
- Filled in self-evaluation checklist

Place the source code and testbench files in "project/code" and "project/testbench". Place the report and filled-in self-evaluation checklist in the project folder and compress it. Submit the compressed folder to webcourses

7. Grading

The grading will be done using the following rubric

Points (100)	Task	Verification methodology
10	input_layer.v	Using testbench, input_layer_tb.v
10	neuron.v	Using testbench, neuron_tb.v
10	hidden_layer.v	Using testbench, hidden_layer_tb.v
10	Output_layer.v	Using testbench, output_layer_tb.v
10	Neural_network.v	Using testbench, neural_network_tb.v
40	Report	Correct accuracy and few challenges/ changes
10	Checklist	Self-evaluation Checklist completed

Table 2: Grading Rubric

8. Additional Help

Reach out to the TA (Sanjay) for any questions

Email: sanjaygandham@knights.ucf.edu

TA office hours: Tu 4-5pm, Th 4-5pm