Fall 2021 - 216A Project

Designed by Uneeb Rathore

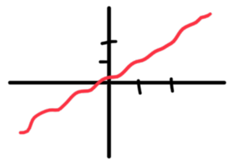
Hand-Written Number Classification by Hardware Neural Network

Background

A lot of development has gone into studying how the neurons in the brain function and trying to replicate them in software and hardware. In this project we will use the most famous and successful model of the neuron called the '*perceptron*' which takes a weighted sum of all its inputs and runs it through a non-linear function. This model was invented by the pioneer of the field of '*Deep Learning*', Frank Rosenblatt and like his first proposed Neural network which just consisted of one layer, in this project you are going to mimic his work, by implementing a single layer neural network (or one layer of neurons) in hardware. Since this is not a Machine Learning class, all the Machine Learning aspects of this project have already been done for you so you can focus on the implementation side of it. However it is important to give an overview so that you have a sense of direction as to what your hardware will do.

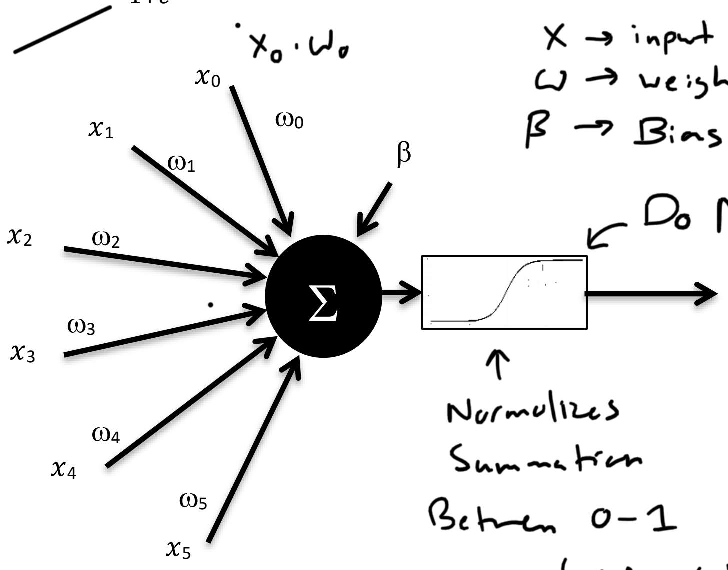
Below is an example of a Perceptron Neuron, which takes a weighted sum of all its inputs and applies the sigmoid function ($\frac{1}{1+e^{-x}}$). The β is the bias of the neuron:

Monotonic Functions
-only goes one direction



Therefore if $in_2 > in_1$, $Out_2 > out_2$

$X \rightarrow$ input
$\omega \rightarrow$ weight
$\beta \rightarrow$ Bias

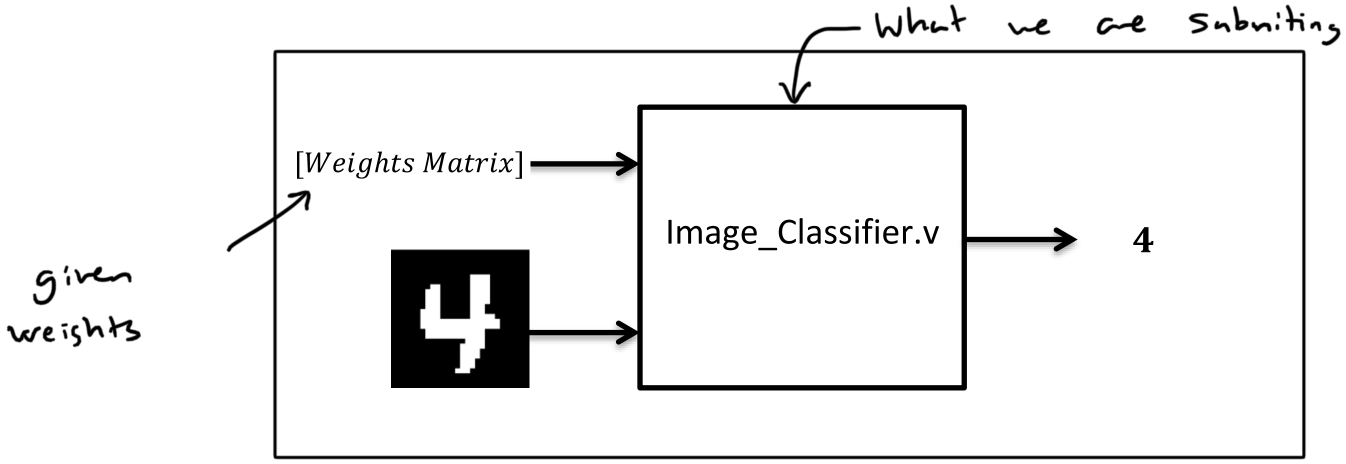


Do Not implement Sigmoid

Normalizes Summation Between 0-1 (or Any set Range)

Project Description

The network you will implement is an image classifier. It takes in a 28x28 pixel gray scale image of a ***hand written*** number, (white pen on a black background) recognizes it and outputs what number it is. You will synthesize your design and try to lower the {Energy x Area} Product



The architecture of the network is as shown below:

Example (Dont have to implement this)

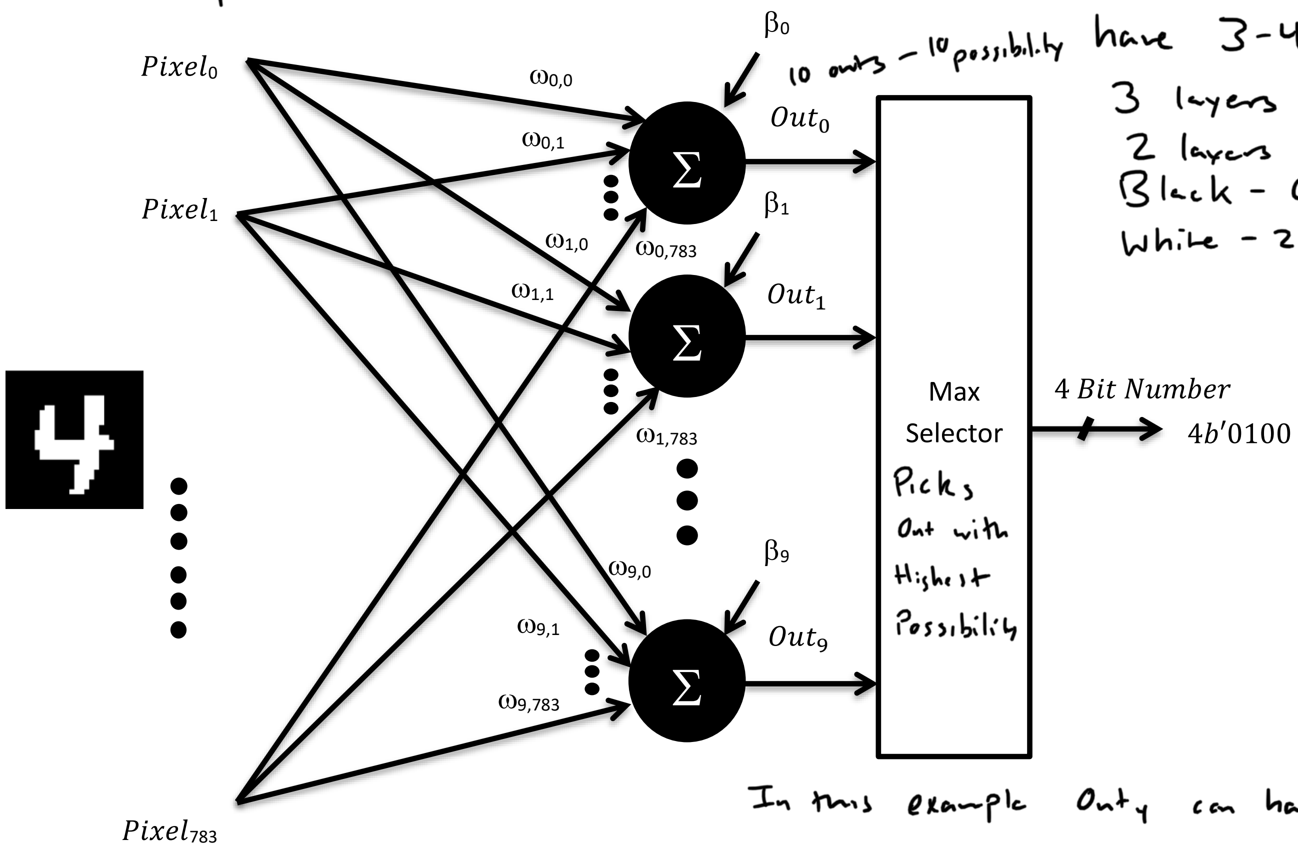
images in txt files have 3-4 layers

3 layers - RGB

2 layers here BLK & W

Black - 0

White - 255



In this example Out_4 can have 99%

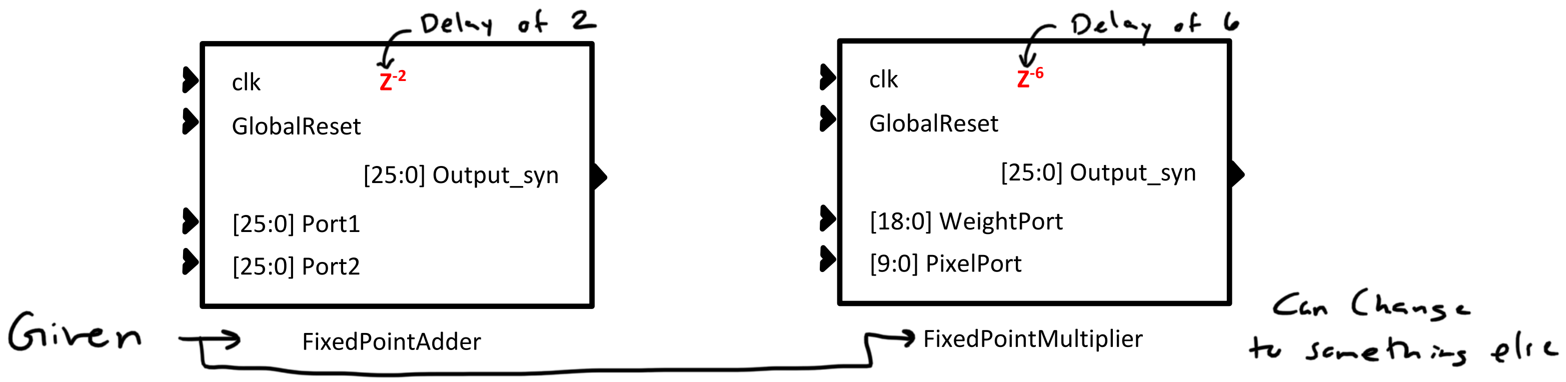
In this architecture the input pixels come from a 28x28 image. There are total of 784 pixels in the image. Each pixel corresponds to a grey scale representation of intensity ranging from 0 to 255. Each of these values is multiplied by a set of pre-trained weights (ω). Each Neuron takes inputs from all pixels and produces a single output. There are total of 10 Neurons in the layer. The output of the Neuron, in a way represents the probability, that the input is that particular number. Therefore the highest output represents the given input number. The Max Selector at the end recognizes this and outputs the corresponding number. For example, the outputs of the Neurons after weight multiplication, summation and sigmoid might look like this:

[Out_0 Out_1 Out_2 Out_3 Out_4 Out_5 Out_6 Out_7 Out_8 Out_9] = [0.1,0.02,0.004,0.098,**0.81**,0.074,0.115,0.007,0.0088,0.03]

This means that the network believes with the highest probability that the output is a '**4**'. The Max Selector block detects all the neural probabilities and outputs a four bit binary number corresponding to the highest probability → 4b'0100 = 4d'4.

What will be provided?

You will be provided with a Verilog implementation of a pipelined *fixed point adder* and *multiplier* to use in your design. You can treat them as 'black boxes' and instantiate them in your top module. Note that these blocks have synchronous active-high reset (GlobalReset) and the active clock (clk) edge is the positive edge. The **red**-lettered Z^{-6} and Z^{-2} indicate that the multiplier and the adder have latency of 6 and 2 clock cycles, respectively (they are pipelined).



include in Simulation

File Name	Description
define.h	Contains definitions used by the provided design blocks.
SynLib.v	Contains structures used by the provided design blocks.
FixedPointAdder.v	Fixed point adder
FixedPointMultiplier.v	Fixed point multiplier.
Image_Classifier.v	Example shell for your block

You will also be provided with an example shell of what your final module should look like. Remember to open the shell and look inside. The module has an input valid which will tell you that the inputs are now valid and you should start your processing, and an out valid bit which you should assert when your module is ready to display its output. The Network has already been pre-trained and the weights will be provided to you in the form of a matrix.

You may choose not to use the adder and multipliers given if you wish to implement a more optimized version – optimized for area, power or delay. The ones provided to you might not be optimal for this task.

You will also be provided with the Matlab Base code that was used to come up with the weights in the first place so that you may play around with it if you'd like.

The composition of the numbers provided to you are follows :

Parameter

Weights range from	-0.25 to 0.25
Pixel range from	0 to 255
Multiplier has inputs	{10,0} & {19,18} (where the format is {total bits, fractional bits}
Multiplier has output	{26,18}
Adder has inputs	{26,18} & {26,18}
Adder has output	{26,18}

Test Bench

Classifier given 83% Accurate

You will be provided with a test bench and a few images which should all pass your classifier.

All images should pass

Suggested Timeline

The project will span five weeks. You will need roughly three weeks to develop RTL, one week for functional verification, and one week for synthesis and optimization. We strongly encourage you to work in groups of 3 however exceptions will be considered with good reason.

Design Specs

Your verilog code should be synthesizable. Remember we are looking for **the best {Energy x Area} product.**

You may not use more than **250 Multipliers** in your design. You may do frequecy scaling, VDD scaling, pipelining, parallelism, scheduling, and a host of other techniques at your disposal. The accuracy requirement for the network is 83 % on the MNIST Dataset. Although we will only provide you with a few images in the test bench, we will upload the full Matlab based Image set for you to play with if you want to change the architecture of the net for a more optimized computation. Remember, if you change the net or change the wordlength, your network should pass the 83 % accuracy mark on the MNIST Dataset. We will test your code with a 'large' random assortment of images from the MNIST data set.

Submission requirements

Submit following files by email (ee216a@gmail.com) with "**Project submission: FirstName1_FirstUID1 FirstName2_UID2**" in the subject line:

- Submit your design as a verilog block named *Image_Classifier* which takes in 784 inputs as the image, and 7850 inputs as the weights, a 1-bit start signal, and outputs a 4-bit number with a 1-bit done signal. An example shell of this will be provided.

- Submit the AREA, POWER and TIMING report logs for your design. Make sure you are not synthesizing latches instead of flip-flops.
- Submit a 3 slide Summary of the composition of your team, the mesurments of the design, and a few bullet points on what particular improvments are in your design that allows it to achieve a better energy area product than others.
- Submit a Notepad (.txt) file titiled team.txt with the respective names and UID numbers of all team members in the group.
- Other required Files

• Image_Classifier.v	Your Verilog design
• Timing-SID.txt	Timing report
• Power-SID.txt	Power report
• Area-SID.txt	Area report
• Summary-SID.ppt	Summary report (PPT template provided)

- Place all of this in a zip file Project _UID1_UID2.zip and email it to ee216a@gmail.com

Grading

Your project will be graded based on following criteria:

Functional Single Neuron:	20%
Functional Full Classifier:	60%
Efficiency Metric:	25% $\leftarrow$ Accuracy

You may also apply these techniques for extra credit:

Word-length optimization:	10%
Different algorithm for image classification:	15%

Grading for Students without a group

Your project will be graded based on following criteria:

Functional Single Neuron:	20%
Functional Full Classifier:	75%
Efficiency Metric:	25%

You may also apply these techniques for extra credit:

Word-length optimization:	10%

Common FAQs

1) Do I need to register the input image and weights?
   - You must register (not latch) the input image (or at least the part you need) when the start signal comes. The image is only valid during that period. The weights do not need to be latched or registered.
2) Why don't you talk about the exponential/non-linear block? How are we expected to do that!?
   - You are not supposed to implement that block because the non-linear block is monotonic, the maximum after the block will be the maximum before the block
3) What do you mean by 'large' random assortment of images from the MNIST data set? What if your random assorment of images all fail my modified algorithem but I pass on another subset of random assorments from the full dataset?
   - Believe in the law of large numbers and don't target 83% accuracy, target a higher accuracy to be on the safe side.
4) Does my deisgn have a minimum throughput limit?
   Yes, From the moment that the input image is valid, you get 10,000 clock cycles to process the image and give an output, which is more than enough for any reasonable design. If you exceed this limit, the testbench will assume that your hardware stalled and will count that as a failed result!
5) How will the efficency metric be graded?
   - For all the viable designs, we will plot their relative efficencies. Then grade the efficencies on a curve. If your efficency is lower than other ppl you will get a lower slice of the 20% and if your efficency is at the top of the class then you will get the entire 20%.
6) The number of input pixels range from 0-783, why does the test bench have pixels from 0-**784** ? whats the extra 784th input?
   The 784th pixel is always 1, and its corresponding weight is the bias. You are free to use this information to your advantage.

Happy Coding and Best of Luck!