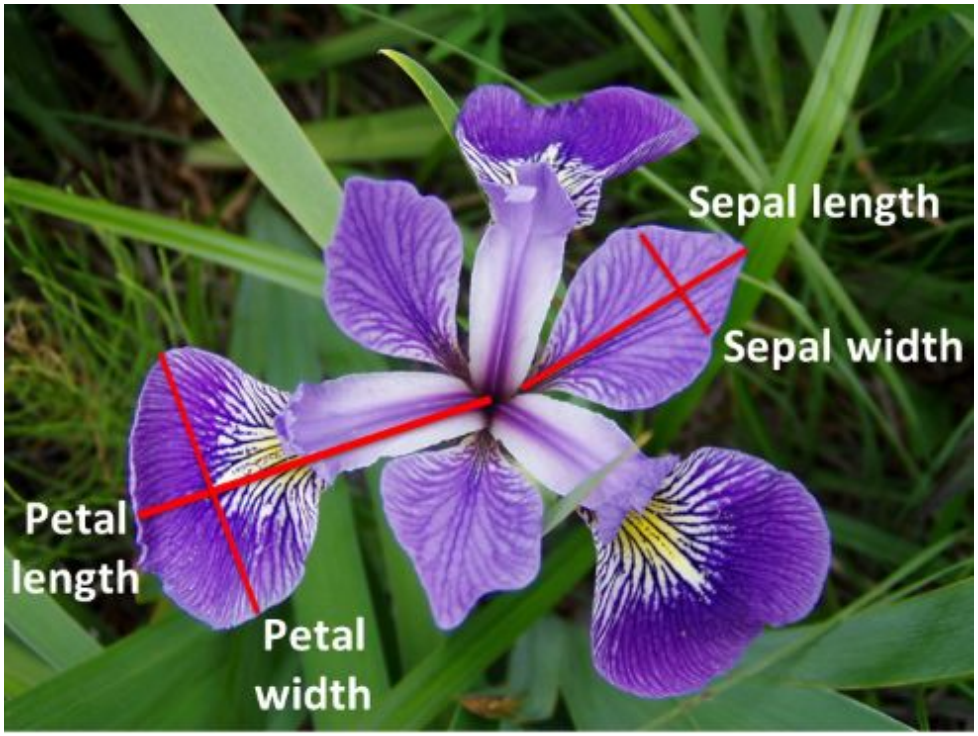
LAB 2 REPORT

Module Definition

Developing a neural network accelerator



Jannis Wolf - Julien Labarre

21.10.2020

PROCESSOR DESIGN

1. Description of the module

Neural networks (NN) exist in tons of different flavours. The development of our NN accelerator will be iterative. So the given specification is the initial step of the development cycle, but is subject to change in the next few weeks. Our first approach is going to be a fully connected network with ReLU activation function, due to the simplicity of the math involved. Section 1. summarizes the description of the module. The data set to test the functionality is going to be the Iris flower data set (see section 2. for details). The last section will outline some potential extensions.

A. Textual description of the module.

1. Feature Vector in floating point:
    - Starting point is loading the input data which is normally given in floating point.

2. Quantization step to get 8 Bit integer representation:
    - Implementing floating point arithmetic is complex and cumbersome. So sticking to integer arithmetic is the way to go. Research showed that 8 Bit can be precise enough so that the rounding error is smaller than the noise in the input data (Warden).

3. Fully Connected Layer (FC):
    - The math here is very easy, as it is only a matrix multiplication. The bias can either be included in the multiplication, by stacking an additional dimension with value 1 to the input or by simply adding it afterwards. The latter being probably more efficient, as it saves some multiplications.

Furthermore, saturation arithmetic has to be implemented to avoid overflows.

4. ReLU Layer:
    - Should be pretty straightforward to implement in VHDL by looking at the sign bit.

5. Reverse quantization after softmax
    - Maybe not necessary because of a softmax layer, if we want to keep it general (like possibility to do regression) this is a necessary step.

6. Predicted probabilities
    - In a first iteration we want to solve a simple classification task (Iris flower dataset, see section 2.). Therefore it is a good idea to use the softmax function in the last layer. This is probably easier to implement on the CPU. The softmax function squeezes the input into the ranch between 0 and 1 and makes sure that the outputs add up to 1, so it can be seen as probabilities. Like mentioned before, this could make the reverse quantization step redundant.

B. Diagram/s of the module and its internal structure

A surface level description of the internal structure was given in section 1.A. Figure 1 shows a diagram of this. It is a bit superficial, but the implementation stage starts after this report.

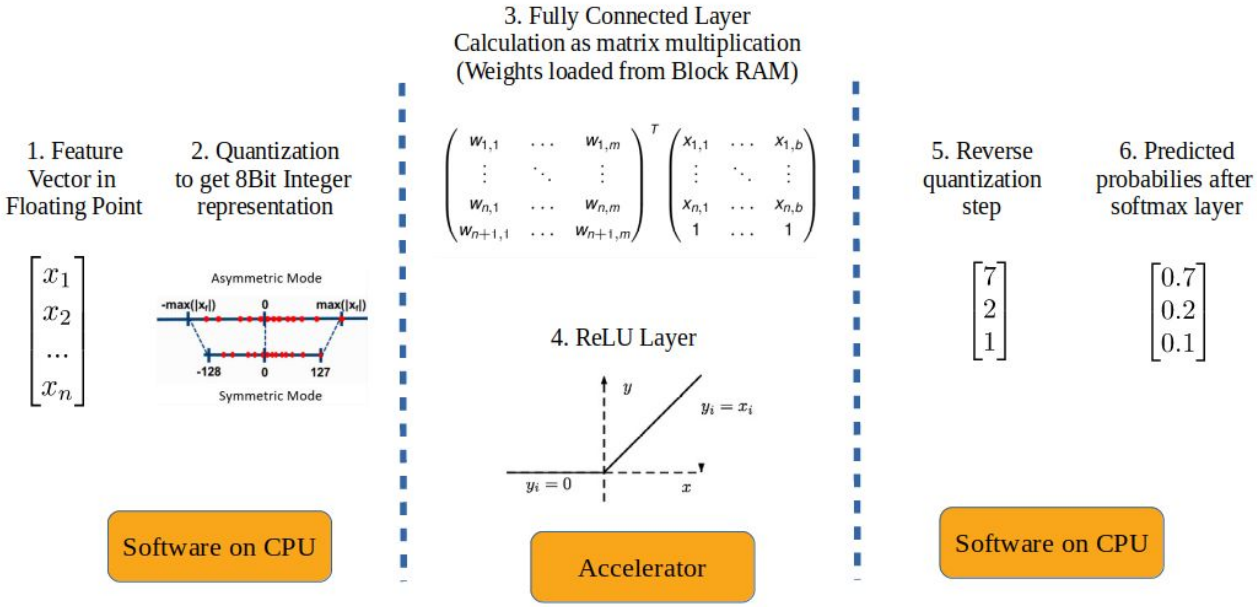


Figure 1: The six steps involved separated into CPU (left and right) and accelerator (central). For details see description in section 1.A.

C. Definition of inputs and outputs of the module

Input -> 8 Bit integer array

- maybe with batch processing, it is a 2D integer array, but we have to see with which input the pipelining is implemented most efficiently

Output -> integer array

- exact size still to be determined while implementing

D. Definition of inputs and outputs of the internal sub-modules

No plans for a structure in submodules yet.

2. Verification

As a first verification of our hardware design we will use the Iris flower data set. It consists of 150 4-dimensional feature vectors who group into three classes. See Table 1 for examples. The dataset has great characteristics for testing neural network performance with respect to prediction. Computationally, a 4D feature vector does not produce very extensive calculations. But it is a good way to verify our first implementation.

	I. setosa	I. versicolor	I. virginica
Sepal length	5.1	7.0	6.3
Sepal width	3.5	3.2	3.3
Petal length	1.4	4.7	6.0
Petal width	0.2	1.4	2.5

Table 1: Horizontally, the three classes of the Iris flower data set can be seen. In the columns, the four features are listed.

The first two classes are linearly separable, but the third class needs hidden layers with non linearities to produce good results, as we need a non linear decision boundary. See Figure 2 which plots two axes of the dataset (Wikipedia). There is a great paper about the hyperparameter configuration for minimal neural networks (Masarczyk). We will train the networks mentioned in the paper in python using tensorflow and keras and initialize the Block RAM with the trained weights. A first working model will then do the calculations of the fully connected layers with ReLU activation functions.

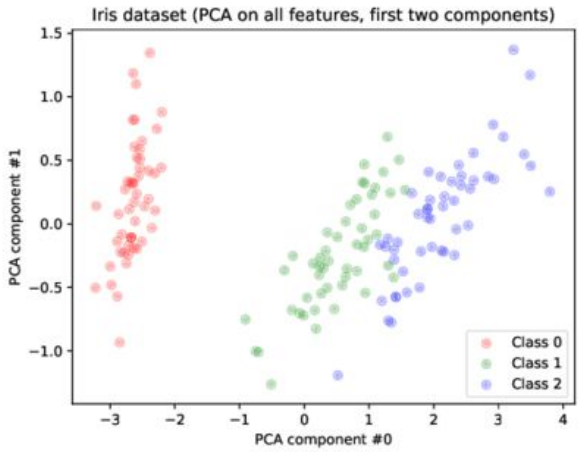


Figure 1: Plot of the two main axes of the Iris Flower data set obtained through a principal component analysis (PCA).

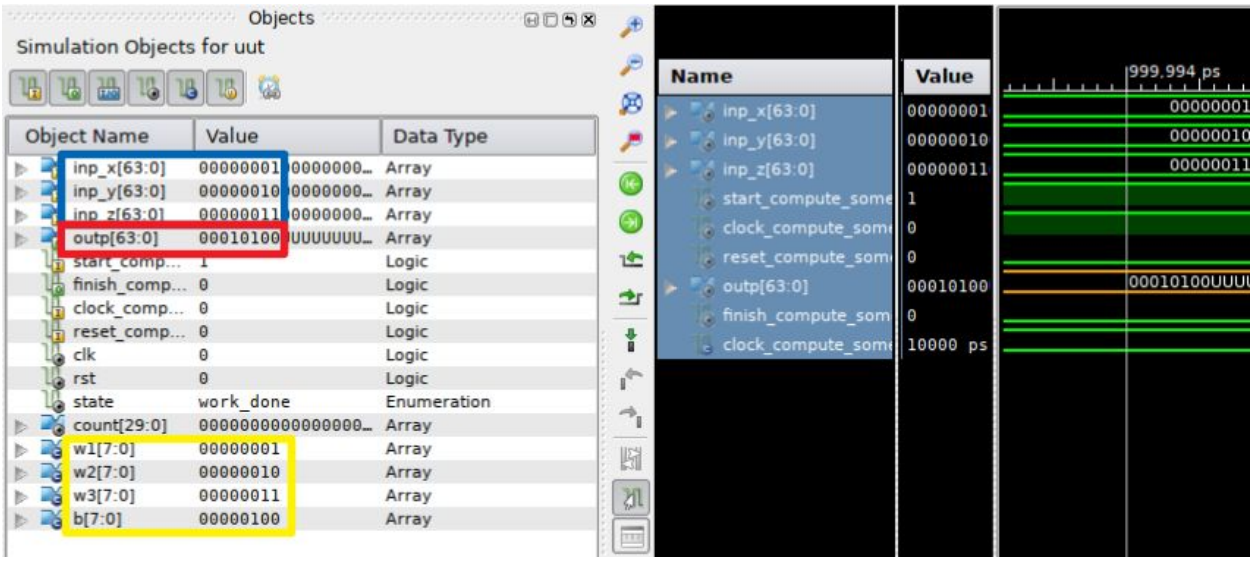


Figure 3: Simulation of a one neuron perceptron. The weights and bias is outlined in yellow. The input values are marked in blue and the output in red.

So in conclusion, the first verification of the design will be with one small fully connected layer. See Figure 3 for an example of the simulation of one neuron. The weights and bias is outlined in yellow. The input values are marked in blue and the output in red. The calculation is shown in the following formula:

$$y = x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + b \quad \Leftrightarrow \quad outp = inpx * w1 + inpy * w2 + inpz * w3 + b \ .$$

3. Potential extensions

Deep Learning is a great research topic, so naturally there are many possible extensions, which are listed below.

- More layers stacked:
  - New datasets with bigger feature vectors require deeper and wider layers
  - Limitations on the resources from the FPGA

- Different layer types:
  - Convolutional Layer (fast implementation as matrix multiplication with toeplitz matrix)
  - Recurrent or LSTM/GRU (very difficult implementation, but especially the LSTM and GRU with their electronic circuit structure are interesting models on specialized hardware
  - Softmax (maybe it is easier to do the softmax layer on the CPU)

- More activation functions then only ReLU inside the accelerator
  - Sigmoid
  - TanH

- Architectural
  - Inception layers (probably very hard to achieve, as they are very complex and need much space on the limited resources of the FPGA, furthermore

needs convolutional layers)
- Residual Neural Networks (the flow of data in residual neural networks could be very nicely transported on specialized hardware, but due to the sheer unlimited depth of these networks, this seems unrealistic too unfortunately)

4. References

Masarczyk, Wojciech. “Optimum size of feed forward neural network for Iris data set.” 2017, http://ceur-ws.org/Vol-1853/p03.pdf. Accessed 22 10 2020.

Warden, Peter. “Quantization Screencast.” 01 05 2020, https://www.youtube.com/watch?v=-jBmqY_aFwE&feature=emb_title. Accessed 22 01 2020.

Wikipedia. “Iris Flower Dataset.” 2020, https://en.wikipedia.org/wiki/Iris_flower_data_set. Accessed 22 10 2020.