VHDL trained multilayer perceptron neural network generator with Python3 + Keras

Author: Héctor Ochoa Ortiz Date: 2020-01-12
Subject: Hardware Description Languages & Circuits Design - Karadeniz Teknik Üniversitesi

Introduction

VHDL trained multilayer perceptron neural network generator with Python3 + Keras, vhdl-trained-nn for short, is a project for Hardware Description Languages & Circuits Design subject, Computer Engineering department, Karadeniz Teknik Üniversitesi. The project files can be found in https://github.com/Robot8A/vhdl-trained-nn.

Vhdl-trained-nn consists of two Python3 files: *main.py* and *vhdl.py*; and two VHDL packages: *activationFunct.vhd* and *FloatPT.vhd*. This last package is a slightly modified version of the one found at https://github.com/xesscorp/Floating_Point_Library-JHU.

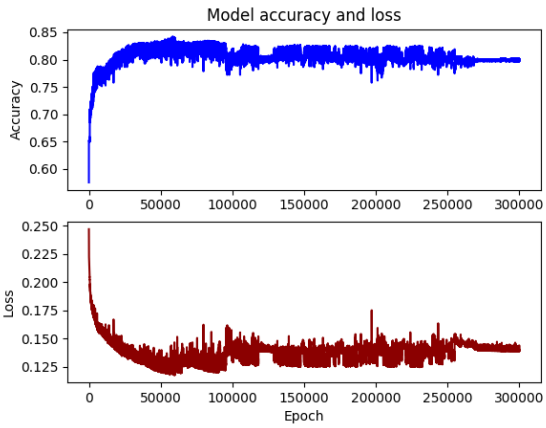
The learning and neural network modeling is made with the Keras library, this, alongside Python3, helps with the developing of the project by providing an already tested and widely used Neural Networks framework with an easy to use and powerful programming language.

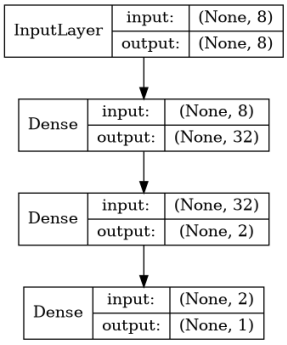
Main program

The main program, *main.py*, models and trains a multilayer perceptron neural network. The user inputs, passed as parameters to the program, define the model.

```
main.py <trainingFile> <inputNumber> <neuronNumberForEachLayer> <trainingEpochs> <batchSize>
```

After training, it exports two images: *acc_loss.png* and *model.png*. In this pictures we can see the accuracy and mean squared error loss of the model along the training epoch, and the layer model, respectively.

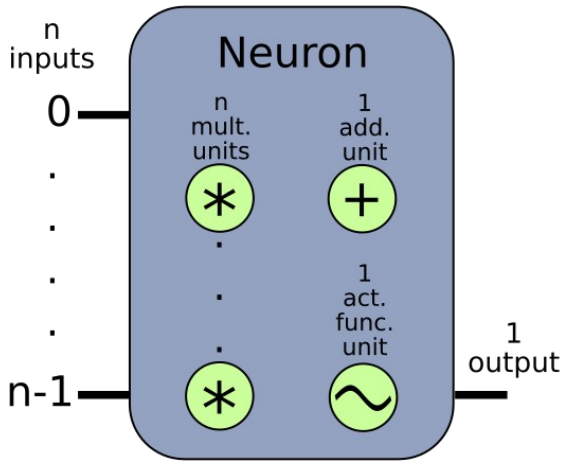




Finally, with the trained weights, it invokes the *create()* function in *vhdl.py* which generates the VHDL file *neuralNetwork.vhd*.

A training dataset can be downloaded from https://www.kaggle.com/kumargh/pimaindiansdiabetescsv, this was the training set used in the picture. There are several more example sets available on the web.

VHDL neural network

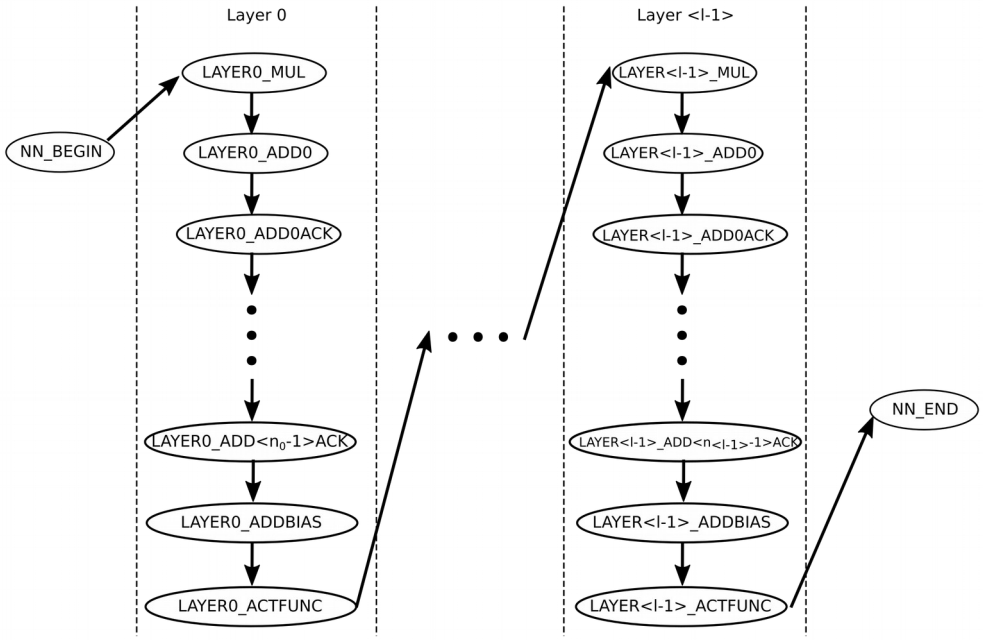


The generated file has the desired number of neurons per layer, with every neuron containing the calculated weights and biases as constants, aswell as n multiplication units (with n being the number of neurons of the previous layer, or the number of inputs to the neural network if it's the first layer) that can compute in parallel, one addition unit that is used multiple times to generate the subresults, and one activation function unit to create the final output of the neuron. The output follows this algorithm:

$$activationFunc\left(bias + \sum_{a=0}^{n} \left(input_a * weight_a\right)\right)$$

The arithmetical units (multiplication and addition with 32-bit floats) come in the package *FloatPT.vhd* and the activation function is located in the *activationFunct.vhd* file. Due to time restrictions in the developing of this project, I was only able to program one activation function: hard_sigmoid. This function returns 0 if x < -2.5, 1 if x > 2.5 or 0.2*x + 0.5 if -2.5 <= x <= 2.5. In future versions, more activation functions could be implemented.

The neural network state machine consists of the state NN_BEGIN, followed by, for each layer l: LAYER<l>_MUL; n (with n being the number of neurons of the previous layer, or the number of inputs to the neural network if it's the first layer) LAYER<l>_ADD<n> and LAYER<l>_ADD<n>ACK; LAYER<l>_ADDBIAS and LAYER<l>_ACTFUNC. And finally NN_END. The ACK states are necessary because the arithmetic units need one cycle to work again after they compute their output, this cycle is used to acknowledge the correct retrieval of the data by the main process, and is completed after the main process turns the go signal of the unit off.



The work was tested with the Modelsim developing and simulation IDE, and an example *wave.do* file is provided with the project. The generated VHDL file has fixed inputs of value 0, this can be changed editing the code.

Conclusions

This is still an open project and an open research topic. This work only lays the foundations of what can be a very huge project. It has been made in about 50 work hours of one single person, with the time constraints of the subject deadline and other work the author of this project had to do.

The following list states future improvements and/or research topics this project can have.

Future work

- Implement more activation functions
- Optimize the number of cycles per layer using n/2 addition units
- Generate automatic wave.do and testbench files alongside the main VHDL code
- Separate the layers in a package
- Pipeline the neural network
- Implement nn. with fixed point decimal numbers instead of floats to improve performance
- Translate into actual hardware
- Any other improvement that can be done

Documentation

https://en.wikipedia.org/wiki/Perceptron
https://en.wikipedia.org/wiki/Multilayer_perceptron
https://en.wikipedia.org/wiki/IEEE_754
https://keras.io/
https://keras.io/activations/
https://docs.python.org/3/