# CUDA Implementation of a Fully Connected ANN

A graphical representation of a fully-connected feed-forward ANN is depicted in Figure 1. It is called fully-connected because each neurons in a given layer is connected to all neurons in its adjacent layers, both before and after except for input and output layers. Input of this neural network is directly assigned to each of its input neurons. The value of each neurons on hidden layers as well as output layer are computed based on the value of its previous neurons.

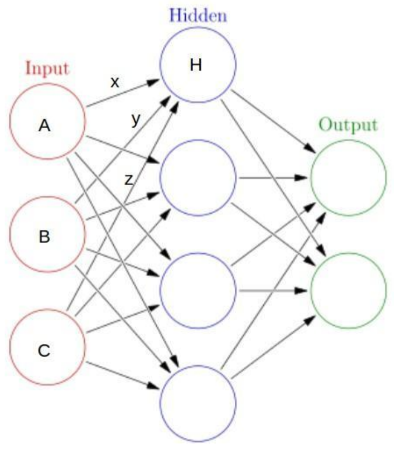

Fig. 1. Illustration of fully-connected feed-forward ANN.

Mathematically, with an example of illustration in Figure 1, to calculate the value of neuron H, we can use the equation H = f(Ax+By+Cz), where A, B, and C are the value of the neurons in the previous layer of H. There is an associated weight of each connections between neuron H and neurons in its previous layer and they are x, y, and z for weight of connection between H and A, B, and C, respectively. The f(n) in that equation is an activation function with input n. This computation can continue until all the output neurons values are computed and those values are the output of the neural network model.

There are two modes of feed-forward neural network execution: feed-forward run mode and train mode. For feed-forward run, the values of input neurons is fed into the next layer until all the output neurons values are computed. For train run, it is the same as feed-forward run but after all the output values are computed, the delta of each output values with the desired values are computed and used to compute mean squared error (MSE). The delta can be used again as an input of another run, called back-propagation run, to distribute the delta into each of neurons in hidden layers. The back-propagation run is similar with the feed-forward run but it propagates backward and activation derived function is used instead of activation function. Finally, once all the delta are distributed to each of neurons, all the weight of connections connected to it is updated based on certain algorithm, such as gradient descent.

(a) Check http://leenissen.dk/fann/wp/ for a serial/OpenMP parallel implementation of the above procedure on the CPU, this will be your baseline and the datasets given in its git repo https://github.com/libfann/fann/tree/master/datasets the benchmarks. Compile and run fann on at least two of these datasets and measure train and test times.

(b) Write CUDA implementation of the important functions like feed-forward run, delta and MSE computations, back-propagation run, weight update. Compare with the serial/OpenMP implementation to check the functionality is correct

(c) Add CUDA optimizations as discussed in class. Plot runtime graphs of runtimes with serial implementation, your naive CUDA implementation and each successive optimizations, to show the effect of each optimization on performance.

(d) Submit source code, Makefile and readme for compiling and running your code on HPC. Create a short report describing your optimizations and the performance graphs. ZIP everything and submit on Moodle.