**NUGGET.h**

The nugget class is a simple parametrized neural network class that is built on the matrix class (which has its own documentation). It allows a user to create a neural network with hidden layers, control the size and number of these hidden layers, what activation functions are used between the layers, how the network is trained etc. This tool is not very optimized for performance (It only uses CPU threads) and as such, is best suited for smaller projects and relatively shallow neural nets. Its relative simplicity makes it ideal for learning neural nets and experimenting with them. This guide will walk you through an example that demonstrates the various functionalities of nugget.

**Compiling and setup**

* Have the header files for both the nugget and the matrix classes in your working directory.
* #include the nugget header file in your main .cpp file.
* To compile the code, you must use C++ 14 or more recent for the RNG used in the matrix class to work.
* Compiling with OpenMP will allow the matrix class to run on several CPU threads, significantly speeding up the program.

**Initializing a nugget**

The default constructor for nugget is as follows:

nugget(int inputs, int outputs, std::vector<int> hid\_layers, std::string init)

Where ‘inputs’ is the size of the input layer of the neural net, ‘outputs’ is the size of the output layer, ‘hid\_layers’ is a vector of integers which define the number and size of the hidden layers, and ‘init’, which determines how the weights are initialized (“normal” or “uniform” Xavier initialization).

Below is an example of a neural net initialized to take in 784 pixels from the MNIST database (hand drawn digits from 0 to 9), outputs a classification from 0 to 9 (output layer of size 10), have 3 hidden layers of 80 neurons each, and uses uniform Xavier initialization:

std::vector<int> hidden\_layers = { 80, 80, 80 };// Vector describing hidden layers

nugget test\_nug(784, 10, hidden\_layers, "uniform");// Initializing the NN

A nugget can also be initialized by reading a save file from a previously trained nugget NN like so:

nugget newnug("SaveFile.txt"); // Initializing a NN called ‘newnug’ using the save file // ‘SaveFile.txt’

**Training a nugget**

Once a nugget is initialized, the ‘train’ member function allows the user to train the NN on a data set.

void nugget::train(const mat& Data, const mat& labels, int it, std::string activ, std::string o\_activ, double alpha) // train() function definition

Where ‘Data’ is a matrix or vector containing the training data, ‘labels’ contains the data labels used to evaluate the NN, ‘it’ is the number of epochs the NN will be trained for, ‘activ’ is the type of activation function used for the hidden layers (“ReLu” or “sigmoid”,) ‘activ\_o’ is the output activation function (currently softmax is hardcoded as the only option,) and ‘alpha’ is the training rate (usually between 0.3 and 1.5.) A last argument can be used to create a save file with a string used to name the file, like so:

test\_nug.train(data, labels, 1200, "ReLu", "softmax", 0.1, "TestSave.txt");

// training the NN ‘test\_nug’ with 1200 epochs, ReLu activation function, a learning rate // of 0.1, and saving the NN matrices to ‘TestSave.txt’

The train() function will print out the current accuracy score of NN every 10 epochs. Once the number of epochs entered is reached, the NN saves the resulting weight and bias matrices to the save file name entered as the last argument to the function, or will save to a default save if no name is provided.

**Running a nugget**

Once a nugget NN has been initialized and trained or has been initialized using a save-file from a previous run, it can be used to process data directly using the run() function. The run() function takes 1 argument at a minimum in the form of a vector or matrix of data to process, formatted the same way as the training data. Labels can also be provided for data validation, also using the same format as used for the training, and as a second argument to the function like so:

test\_nug.run(Tdata, Tlabels);

If labels are provided, the accuracy will be displayed it the console window for the run.