Julio Grillo

Nguyen

CISC 190

17 May 2017

JNeural

For my final project I decided to dabble in neural networks and artificial intelligence. After many google searches and a fair amount of reading I created two Java packages, JMatrix and JNeural. JMatrix, as you may have guessed, is used for working with matrices. The core of the library is the FloatMatrix class which holds a matrix ( 2D array) as a 1D array and provides mathematical methods such as add, subtract and multiplication. There is also Math class which contains a multithreaded dot product method. After testing, however, the multithreading only proved faster with matrices in the thousands of rows and columns.

JNeural provides several classes, most importantly the NeuralNetwork class which holds the network’s dimensions and layers as FloatMatrixes. This class also contains the core methods for neural networks such as run() and train(). The run method simply computes and output based on an array of inputs passed as a parameter to run(). The train method is slightly more complicated to use as its purpose is to adjust the network weights so that the network produces a smaller error. This requires both inputs and known correct outputs, essentially answers. For example, with a basic AND gate, the inputs FALSE, TRUE would yield FALSE and thus FALSE would be the answer. If we wanted the network to train into an AND gate FALSE, TRUE would be an input in the training set and FALSE would be and output in the training set. For the purpose of uniformity and ease the TrainingSet class was created.

The TrainingSet class contains an addToSet() method which takes two single dimensional arrays and inserts them to their respective FloatMatrix.The FloatMatrix size is not dynamic and a new one must be created within every addToSet() call.

The last *important* part of JNeural is the Activation interface. Each neuron in the network must pass their input sum through an activation function before a passing the value to next neuron. This is to properly emulate the namesake brain cell. For backpropagation (training), the Activation function and its derivative are needed. This proved a perfect place to implement an interface; the Activation interface defines two methods, function and prime, for use in the run() and train() methods.

The element of this package is the Eliminator interface. This is simply a class which will be called every training iteration before the weights are adjusted. This is for user created equations which help networks optimize past local minimums. I have included such an equation called LocalMinProbability which uses the network’s location within the training cycle (simply the current iteration divided by the number of iterations to be executed), and the network’s error above a threshold to amplify the weight changes.

This framework has quite a few shortcomings, such as lack of multithread support and GUI. In the future this program would mostly be written as openCL kernels and and methods such as train and run would be their own kernels which a Java, Python, CPP , etc program would bind to. OpenCL would allow me to take advantage of high concurrency devices such as a GPU. A GUI or some sort of visuals would help in understanding the neural network and provide a good learning tool.

Psuedo code:

* Neural Network
  + Layers - matrices of layers
  + Train
    - Run the network forwards
    - Compare results with answers in training set and propagate the error to the previous layer
      * Find error of each layer by multiplying the error by the weight between the outer-most layer (already calculated error) and the inner layers
      * Find the node delta by multiplying the error and activation1(node sum)
      * Find the adjustment by multiplying activation(node sum) and the node delta
      * Adjust weights by LEARN\_RATE and the adjustment
  + Run
    - Mulitply the inputs by the layer weights until the last one
* Matrix
  + get(r, c) - get row and column
  + Add
  + Sub
  + Mult
  + Dot
* Demo
  + Trains XOR
  + Displays last result value

OUTPUT

[ 0.00000000]

[ 0.99999976]

[ 0.99999976]

[ 0.00000043]