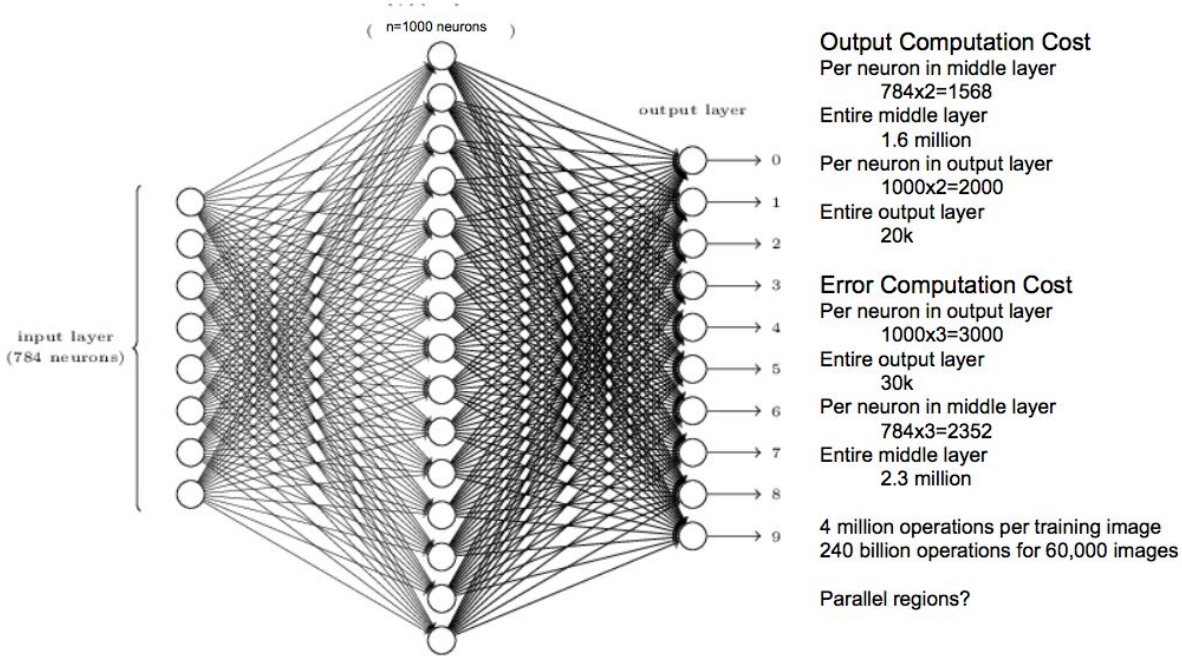
Recognizing Numbers using Neural Nets

Nathik Salam

Background

This project uses artificial neural nets which mimic the human brain to recognize numerical digits from 0-9. The motivation behind choosing this project is because I'm building a robotic arm that uses artificial neural nets to learn how to pick up an object all by itself and I took advantage of this project to give me a very small exposure into that subject. This is also one of the fastest growing technology field right now, in our very own University about half the thesis defenses in the EE department this quarter involved neural nets. My aim is to use these networks on embedded chips to do real time learning and computation.



Implementation

This neural net has 3 layers, the first layer all the way on the left of the image above is the input layer which is fed by the pixels of a 28x28 image. Every single one of these inputs feed to each and every neuron on the layer afterwards, called the hidden layer comprising of a 1000 neurons. Each of these neuron calculates the following sum

$$activation = \sigma(bias + \sum_{inputs} w_i * input)$$

where each neuron has a w or weight vector of the size of inputs and a bias. The σ is the sigmoid function. Since these neuron calculations are independent of each other, it can be parallelized. Similarly to the inputs feeding into each of the neurons in hidden layer, each of the activations feed into the 10 neurons in the output layer, each neuron corresponding to the number between 0-9. The neuron that holds the highest value in

this layer is chosen as the correct output. This area is again a parallel region. Not to mention during the learning phase, the computed value is subtracted from the correct value called the error. This error is then used to recalibrate the weight vectors of the neurons, this part also has two parallel regions, first in the output layer and then the hidden layer.
Another parallel region is the summation of each neuron which can be tiled.

Previous Work
The only part where I utilized work done by others was to convert image files into C pixel arrays, the rest I used as reference to learn from.

Results

	Time	Accuracy	Threads
CSL Lab machine	> 60 minutes	97.6%	1
Home Mac	45 minutes	97.7%	1
Intel Xeon Processor			
GNU Compiler	50 minutes	97.7%	1
Intel Compiler			
No parallel	632 seconds	97.6%	1
Parallelizing Layer	84 seconds	97.7%	16
Parallelizing Layer & Tiling	114 seconds	97.8%	16
Intel Xeon phi Coprocessor			
Parallelizing Layer	105 seconds	97.7%	228
Parallelizing Layer & Tiling	170 seconds	97.7%	228

Discussion
Specific Information

To optimize this code, I just added openmp for loop pragma to the loop in the layer class that iterates through the calculation for each neuron. Since I used objected oriented programming, I only had to add this pragma to one area of the code. Another optimization I experiment without any success was calculating the sum for each neuron using tiling, I tried tiles of different sizes to any avail. Two thing that I did not experiment with and I probably should have was vectorization and memory alignment.

Concise discussion
As shown in the results section at the best case there was 35% decrease in computation time without making any compromises to the accuracy of the program.

Sources

Professor John Seng's lecture notes on neural nets
http://neuralnetworksanddeeplearning.com
http://natureofcode.com
https://mmlind.github.io/Simple_1-Layer_Neural_Network_for_MNIST_Handwriting_Recognition/