Introduction to Parallel Scientific Computing

Parallel Implementation of Artificial Neural Networks for Hand-written Recognition (OpenMP)

Submitted By:	*Roll No.*
Monu Tayal	2018201042
Danish Mukhtar	2018201016
Shubham Pokhriyal	2018201080

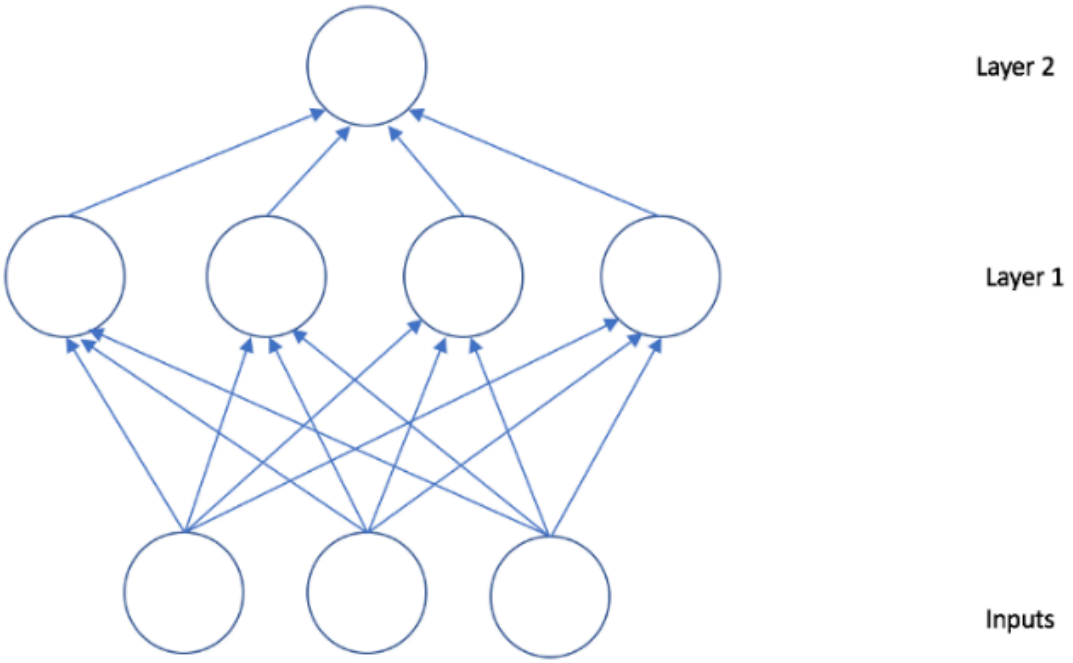
INTRODUCTION

In this project we have implemented hand-written digit recognition using artificial neural networks. Artificial neural network is implemented in both classical and parallel computing style to recognise handwritten digits. The idea is to take a large number of handwritten digits, known as training examples, and then develop a neural network which can learn from those training examples. By increasing the number of training examples, the network can learn more about handwriting, and so improve its accuracy.

Performance of both of our implementation is compared according to time taken by the models. Parallelism is exploited in all the possible matrix operations that need to done over weight, layers and bias matrices. We have used one hidden layer in our neural networks implementation. Size of input layer and hidden layers taken are 784 and 100 respectively.

OVERVIEW OF NEURAL NETWORKS

Neural Networks are a key piece of some of the most successful machine learning algorithms. The development of neural networks have been key to teaching computers to think and understand the world in the way that humans do. Essentially, a neural network emulates the human brain. Brains cells, or neurons, are connected via synapses. This is abstracted as a graph of nodes (neurons) connected by weighted edges (synapses).



This neural network has two layers, three inputs, and one output. Any neural network can have any number of layers, inputs, or outputs. The layers between the input neurons and the final layer of output neurons are hidden layers of a deep neural network.

1) **Neurons**. A neural network is a graph of neurons. A neuron has inputs and outputs. Similarly, a neural network has inputs and outputs. The inputs and outputs of a neural network are represented by input neurons and output neurons. Input neurons have no predecessor neurons, but do have an output. Similarly, an output neuron has no successor neuron, but does have inputs.

2) **Connections and Weights**. A neural network consists of connections, each connection transferring the output of a neuron to the input of another neuron. Each connection is assigned a weight.

3) **Propagation Function**. The propagation function computes the input of a neuron from the outputs of predecessor neurons. The propagation function is leveraged during the forward propagation stage of training.

4) **Learning Rule**. The learning rule is a function that modifies the weights of the connections. This serves to produce a favored output for a given input for the neural network. The learning rule is leveraged during the backward propagation stage of training.

Forward Propagation

The input *X* provides the initial information that then propagates to the hidden units at each layer and finally produce the output y^. The architecture of the network entails determining its depth, width, and activation functions used on each layer. Depth is the number of hidden layers. Width is the number of units (nodes) on each hidden layer since we don't control neither input layer nor output layer dimensions. There are quite a few set of activation functions such *Rectified Linear Unit, Sigmoid, Hyperbolic tangent, etc*. Research has proven that deeper networks outperform networks with more hidden units. Therefore, it's always better and won't hurt to train a deeper network (with diminishing returns).

Backward Propagation

Allows the information to go back from the cost backward through the network in order to compute the gradient. Therefore, loop over the nodes starting at the final node in reverse topological order to compute the derivative of the final node output with respect to each edge's node tail. Doing so will help us know who is responsible for the most error and change the parameters in that direction. The following derivatives' formulas will help us write the back-propagate functions: Since *b^l* is always a vector, the sum would be across rows (since each column is an example).

WORK DONE

1. Neural network is implemented from scratch without using any inbuilt library.
2. Handwritten digit dataset from MNIST is used for both training and testing of our neural network.
3. 10,000 examples are used where 80% is used to train the model and 20% to test the model.
4. Image is represented as 28*28 pixel intensity vector.
5. Neural network is implemented with one hidden layer of 100 neurons.
6. Accuracy and Time taken by model is analysed for both classical and parallel approach.

RESULTS AND ANALYSIS

1. Batch Size = 500

```
gunno@tayal-lenovo:~/Downloads$ export OMP_NUM_THREADS=8
gunno@tayal-lenovo:~/Downloads$ ./a.out
Datasize: 10000
Batchsize: 500
Epochs: 20

Accuracy: 0.946375
Time taken by program is : 155.928067 sec
gunno@tayal-lenovo:~/Downloads$ export OMP_NUM_THREADS=4
gunno@tayal-lenovo:~/Downloads$ ./a.out
Datasize: 10000
Batchsize: 500
Epochs: 20

Accuracy: 0.946375
Time taken by program is : 176.622885 sec
gunno@tayal-lenovo:~/Downloads$ export OMP_NUM_THREADS=2
gunno@tayal-lenovo:~/Downloads$ ./a.out
Datasize: 10000
Batchsize: 500
Epochs: 20

Accuracy: 0.946375
Time taken by program is : 284.733047 sec
gunno@tayal-lenovo:~/Downloads$ export OMP_NUM_THREADS=1
gunno@tayal-lenovo:~/Downloads$ ./a.out
Datasize: 10000
Batchsize: 500
Epochs: 20

Accuracy: 0.946375
Time taken by program is : 505.507775 sec
gunno@tayal-lenovo:~/Downloads$
```

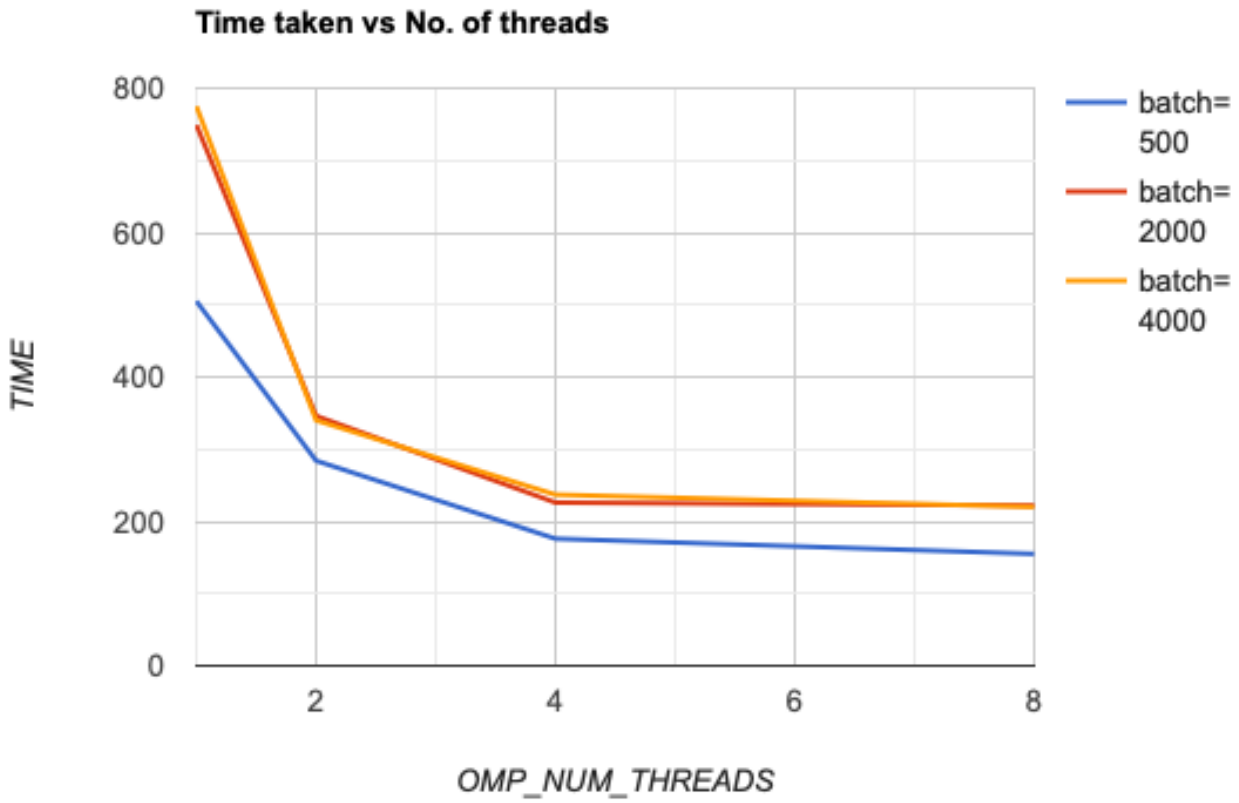
2. Batch Size = 2000

```
gunno@tayal-lenovo:~/Downloads$ export OMP_NUM_THREADS=8
gunno@tayal-lenovo:~/Downloads$ ./a.out
Datasize: 10000
Batchsize: 2000
Epochs: 20

Accuracy: 0.890625
Time taken by program is : 222.257675 sec
gunno@tayal-lenovo:~/Downloads$ export OMP_NUM_THREADS=4
gunno@tayal-lenovo:~/Downloads$ ./a.out
Datasize: 10000
Batchsize: 2000
Epochs: 20

Accuracy: 0.890625
Time taken by program is : 226.465865 sec
gunno@tayal-lenovo:~/Downloads$ export OMP_NUM_THREADS=2
gunno@tayal-lenovo:~/Downloads$ ./a.out
Datasize: 10000
Batchsize: 2000
Epochs: 20

Accuracy: 0.890625
Time taken by program is : 346.961462 sec
gunno@tayal-lenovo:~/Downloads$ export OMP_NUM_THREADS=1
gunno@tayal-lenovo:~/Downloads$ ./a.out
Datasize: 10000
Batchsize: 2000
Epochs: 20

Accuracy: 0.890625
Time taken by program is : 749.379091 sec
gunno@tayal-lenovo:~/Downloads$
```

3. Batch Size = 4000

```
gunno@tayal-lenovo:~/Downloads$ export OMP_NUM_THREADS=8
gunno@tayal-lenovo:~/Downloads$ ./a.out
Datasize: 10000
Batchsize: 4000
Epochs: 20

Accuracy: 0.870375
Time taken by program is : 220.805444 sec
gunno@tayal-lenovo:~/Downloads$ export OMP_NUM_THREADS=4
gunno@tayal-lenovo:~/Downloads$ ./a.out
Datasize: 10000
Batchsize: 4000
Epochs: 20

Accuracy: 0.870375
Time taken by program is : 237.229119 sec
gunno@tayal-lenovo:~/Downloads$ export OMP_NUM_THREADS=2
gunno@tayal-lenovo:~/Downloads$ ./a.out
Datasize: 10000
Batchsize: 4000
Epochs: 20

Accuracy: 0.870375
Time taken by program is : 340.176716 sec
gunno@tayal-lenovo:~/Downloads$ export OMP_NUM_THREADS=1
gunno@tayal-lenovo:~/Downloads$ ./a.out
Datasize: 10000
Batchsize: 4000
Epochs: 20

Accuracy: 0.870375
Time taken by program is : 775.440279 sec
gunno@tayal-lenovo:~/Downloads$
```

Time taken vs No. Of Threads



Conclusion

1. Accuracy of our model is **94.6%** with batch size of 500.
2. Accuracy is consistent for different number of threads. Threads has no effect on the accuracy of our model as it should be.
3. Time taken decreases with increase in number of threads as matrix computation takes place in parallel.
4. After a threshold increasing number of threads will not improve time taken due overheads eg. context switch.

REFRENCES

1. https://medium.com/analytics-vidhya/neural-networks-for-digits-recognition-e11d9dff00d5
2. http://yann.lecun.com/exdb/mnist/ (MNIST dataset)
3. https://www.openmp.org
4. https://www.coursera.org/learn/neural-networks-deep-learning