University of Chicago

Winter 2023

GPU Machine Learning

Assignment 4 : High Performance Computing

Divya Pattisapu

03-13-2023

1 Python implementation results

- For 20 epochs, batch size 128, 3 hidden layers, and 20 hidden units per layer, the grind rate is 2884 images per second. It runs in 20.8 seconds.
- Best accuracy achieved is 90.32 percent when number of units in each hidden layers and training epochs is 25

2 Serial implementation results

- For 20 epochs, batch size 128, 3 hidden layers, and 20 hidden units per layer, the grind rate is 1775 images per second. It runs in 33 seconds.
- This is not better than the python version since numpy parallelizes the matrix operations.

3 OpenMP implementation results

3.1 Time taken with increasing number of threads

- The time taken reduces with increasing number of threads.
- On a 12th Gen Intel(R) Core(TM) i7-12700H machine with 20 logical cores, the plot **below** represents the time taken to run 20 epochs, batch size 128, 3 hidden layers, and 50 hidden units per layer.

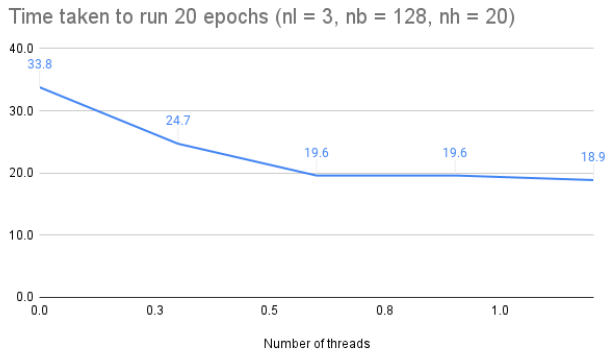

Figure 1: Time taken to run 20 epochs (nl = 3, nb = 128, nh = 20)

3.2 Increasing number of units per layer

- I observed that the test accuracy is sometimes greater than the train accuracy and both the values are pretty close. This suggests that we make the model more complex since there is underfitting.
- While the accuracy improves when I change the number of units per layer from 20 to 50 (Improves from 87 percent to 90.6 percent), the very little gap between train and test accuracy suggests that the model can go deeper. Perhaps deeper than 3 hidden layers will improve the accuracy further.
- The test and train accuracy are very close as shown in the image **below**.

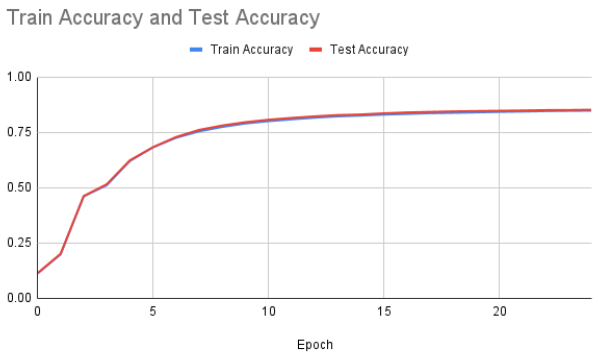

Figure 2: Train Accuracy and Test Accuracy (nl = 3, nb = 128, nh = 50, ne= 20)

Epoch number	Train accuracy	Test accuracy
0	0.112367	0.1135
5	0.745683	0.7494
10	0.8517	0.8572
15	0.877567	0.8818
20	0.8879	0.8908
25	0.894	0.8956
30	0.899517	0.8986
35	0.9037	0.9045
40	0.9079	0.908
45	0.911383	0.9101 (best and grind rate=626 images per sec)

3.3 Accuracy for different number of threads

- The accuracy improves when I increase the number of threads, which is unexpected since the seed is the same and the random number generated should be the same. I do not know how to explain it. It improves from 86 to 89 percent from 1 to 16 threads.

3.4 Changing learning rate over epochs

- With decreasing learning rate over epochs, (by a factor of 0.9 per 5 epochs), the accuracy improves to 91 percent (test - 91.0 percent and train - 91.1 percent)
- Decreasing the learning rate over epochs and increasing the number of epochs with number of units per layer at 50 gives the best accuracy - 91 percent (test accuracy). This can be seen in the table below.

4 Implementation details

4.0.1 Code walk through

- Initialization code : The weights are initialized using Gaussian function, and the biases are initialized equal to 1.
- One hot encoding : After loading the dataset, the labels need to be one hot encoded.
- Shuffle : For a good split, the data needs to be shuffled. This is done by randomly shuffling the indices of the image dataset.

- The Batches are made by splicing the dataset
- Forward propagation, the affine transformation (weights times inputs plus biases) which is activated using sigmoid for all but the final layer which is activated using the softmax activation function.
- Then the backprogation is done where we know the final loss, we use the chain rule and the activated transformations to find the remaining gradients backwards.
- The gradients are updated using the gradients calculated.
- Accuracy is calculated using the weights and biases.