MPCS 51087
Problem Set 3
Machine Learning for Image Classification

Sonia Sharapova

CNetID: sharapova

Introduction:

In this project, we compare performance of strategies used for matrix-matrix multiplication. Implementing a neural network to test the performances on, this report goes over three techniques for speeding up matrix multiplication: a naive approach (manual calculation), the cBLAS library for the CPU, and the CuBLAS library for a GPU implementation.

To standardize our benchmarks and mitigate discrepancies, the set of parameters remained constant at a learning rate of 0.1, a batch size of 200, and a total of 50 epochs. The activation function used was ReLu, Softmax for computations in the last layer, and weights initialization using the Kaiming initialization scheme. Moreover, the data was split into 50,000 images for training and 10,000 images for validation, using these metrics for loss evaluation, and then further evaluated on an unknown test set. This analysis found that the use of these libraries significantly improved the runtime of the program.

Parameters:

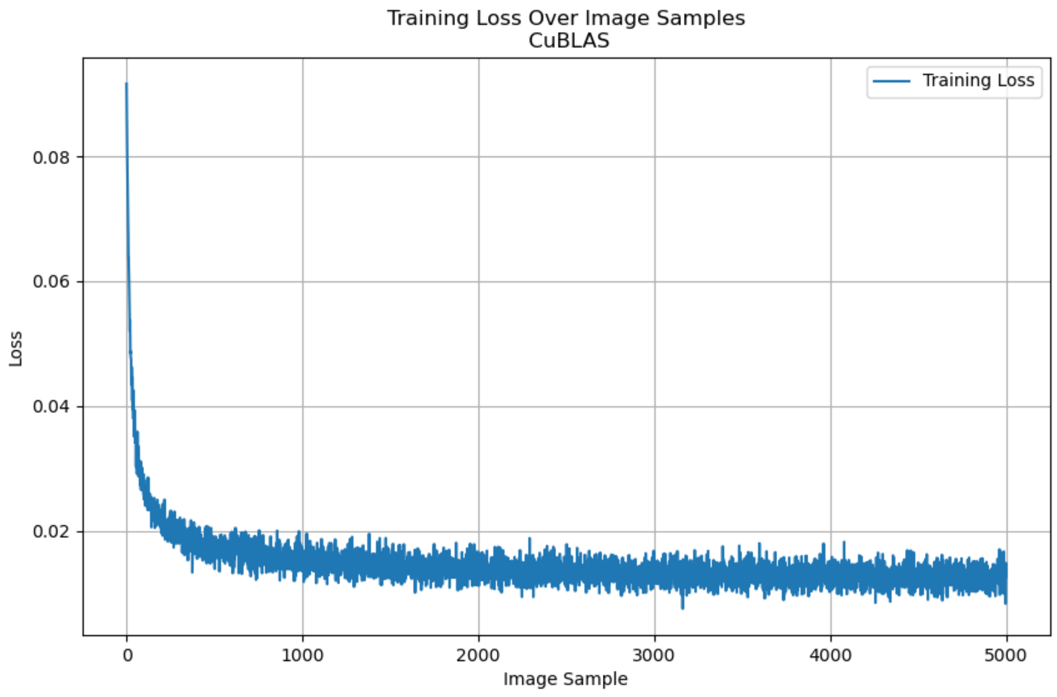
Number of hidden layers: 1
Dimension of hidden layers: 800
Number of epochs: 50
Batch Size: 200
Alpha: 0.1

Running the program:

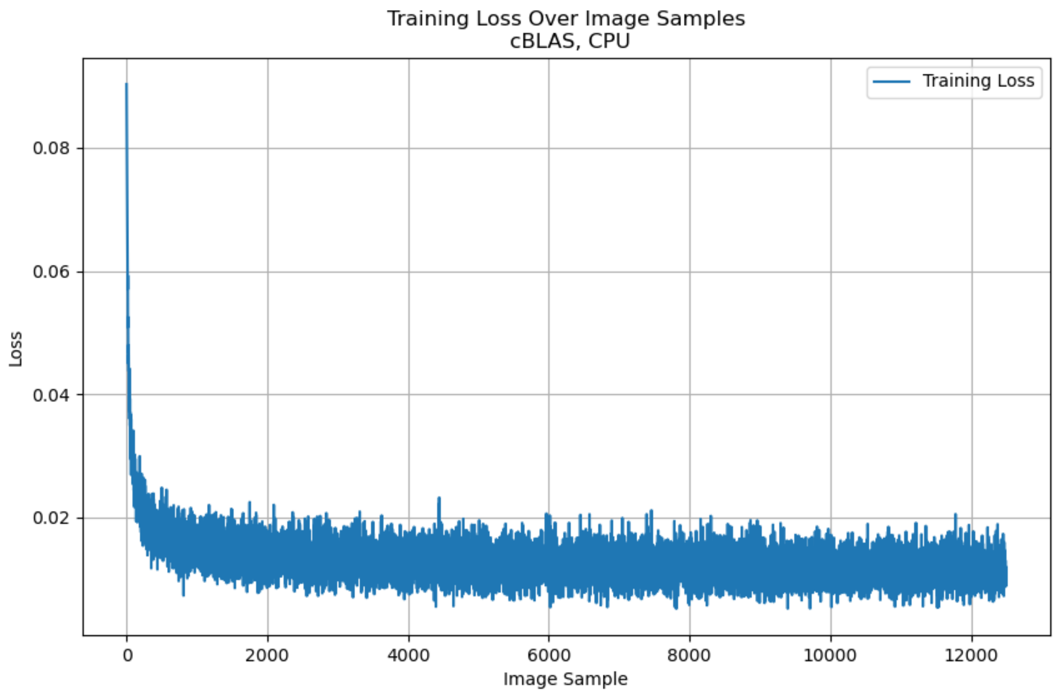
```
$ make cpu_blas
$ make run_cpu_blas

$ make cpu_manual
$ make run_cpu_manual

$ make cuda_blas
$ make run_cuda
```

Loss plots:

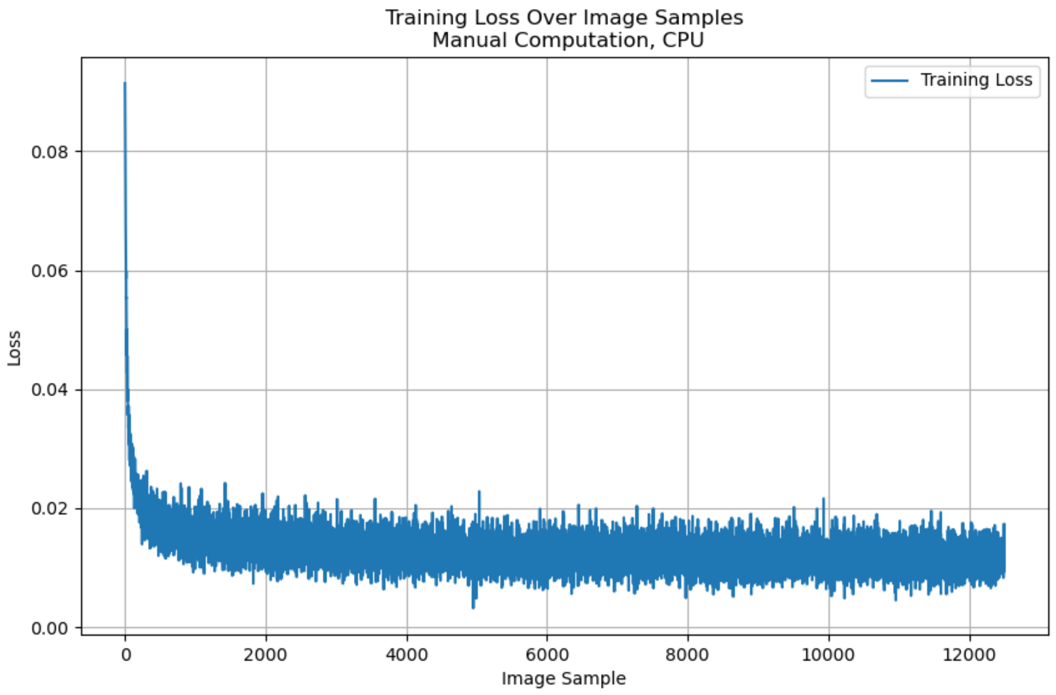
CuBLAS:



cBLAS:



Manual Computation:



With CuBLAS, the loss was not as noisy. This may, however, have been caused by a implementational oversight.

CUDA vs. CPU Performance:

To compare the BLAS implementation for matrix-matrix multiplication, CuBLAS was implemented to compare against a GPU version. The use of BLAS has already sped up the performances of the model, but implementing cuRand improved it further. Compared to BLAS and the manual implementation for the matrix multiplications, the speedups were as follows:

Version	Processor	Accuracy	Grind Rate	Training Time
GPU CuBLAS	Tesla V100	95.00%	118174.820	25.38s
CPU native		95.50%	11278.614	265.99s
CPU BLAS		95.50%	44589.355	67.28s

Observations and Analysis:

- SPEEDUP:
  The speedup between the CPU naïve version and the CuBLAS implementation was an increase of about a factor of 10. Even with just cBLAS, there was a speedup of about a factor of 4 compared to the manual computation. Some of the performances may have been affected by overhead in the program, such as printing to the console or linking the CUDA file to the C execution.

- GRINDRATE:
  The grind rate for CuBLAS was much higher than that of the naive implementation or the cBLAS implementation. Between the manual CPU version and the CuBLAS version, there was in increase by a factor of 10.

- ACCURACIES:
  Training converged quickly, with an accuracy of over 90% within the first 3 epochs. A sample list of accuracies over the epochs can be found as a test file in the repository.

- FUNCTIONS:
  Compared to the sigmoid activation function, the ReLU activation function produced between results with accuracies better by ~15%

Shortcomings:

- When randomizing my image selection, a better approach would have been to shuffle the dataset and then select the batch from the set in a consecutive manner. Instead, I randomly pick random images from the set (where batch size specifies how many random images I select), but this means the same image might come up multiple times in the same batch.
- GPU: I was unable to unable to implement a fully working CUDA version in which the network is parallelized, so I only applied CUDA to the matrix-matrix multiplications using CuBLAS. This did show a speedup in the times, but a full implementation might have shown even better results. The draft of the CUDA implementation is in the folder but is not executable.
- The network uses all of the necessary functions and initialization schemes. The biggest issue is the randomization.
- Speed: My speed seemed to be quick without the use of any BLAS libraries, but I could not pinpoint the reason for this.

EXTRA:

Some sample runs can be found in the “extra” folder in the repository. This includes samples of accuracies over the runs and losses during training. The CUDA version for the network is also there, but does not run...