Tarun Vallabhaneni
tvallabh

Project 3 – Final Specification

Introduction

This program implements a neural network in C for classifying handwritten digits from the MNIST dataset (60,000 training images and 10,000 test images). The architecture has been refined to include a single hidden layer with 800 neurons. The activation function for this hidden layer has been updated to ReLu, with the Kaiming initialization scheme employed for weight assignment, enhancing the network's capability for effective learning from the outset. Significantly, the output layer now employs the softmax activation function, paired with the categorical cross-entropy loss function, to better address the classification challenges presented by the MNIST dataset. The workflow was also altered to divide the 60,000 training images into 50,000 training inputs and 10,000 validation inputs. After each epoch using the 50,000 training images, I calculated the loss based on the validation set but only used the training images to update the weights, excluding the validation set from weight calculation. This adjustment aims to uncover the relationship between parameter choices, overfitting, and model performance.

The network was updated to process multiple input samples concurrently, emphasizing the use of matrix products to increase computational efficiency. This shift is supported by the introduction of BLAS implementations and custom-coded matrix-matrix multiplication routines, offering a comparative analysis of performance enhancements. In my CUDA optimization efforts, I primarily focused on enhancing matrix multiplication, the most computationally demanding operation, to improve overall model performance. Initially, I developed a custom implementation for these operations. Subsequently, for comparison, I integrated the cuBLAS V2 API, replacing our native solutions. While this approach yielded performance improvements, further gains are anticipated by executing the entire forward pass and backpropagation directly on the GPU. This strategy reduces the overhead associated with frequent memory transfers between the host and the GPU for each matrix multiplication.

Results

The parameters used were as follows:
- Learning rate: 0.1
- Number of epochs: 50
- Batch size: 200
- cuBLAS V2

Version	Processor	Accuracy	Grind Rate (imgs/sec)	Training Time (s)	TPB/Cores
GPU native	V100	98.08%	37265.52	80.50	128
GPU CuBLAS	V100	98.04%	36161.16	82.96	N/A
CPU native	Caslake3	98.05%	720.21	864.12	1
CPU BLAS	Caslake3	98.10%	25281.36	118.66	1

Table 1: Comparative Performance Analysis

Tarun Vallabhaneni
tvallabh

Loss Curves

Each loss curve was plotted by calculate the loss based on the validation set of 10,000 images and taking an average of that loss over the batch size. The loss was then further averaged over the number of batches.

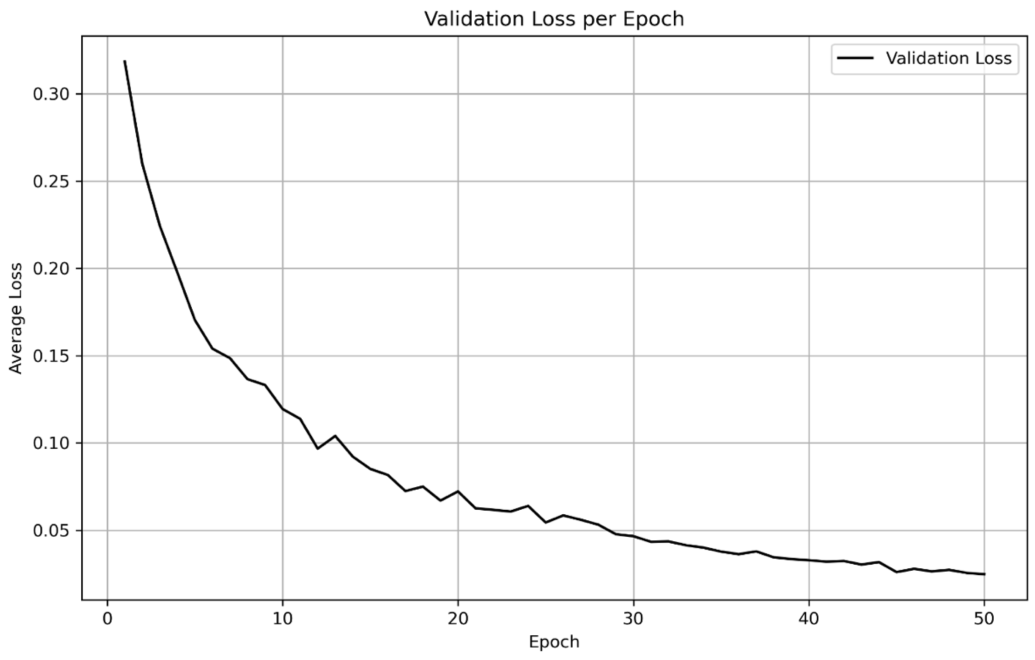

Figure 1: CPU BLAS Loss Curve

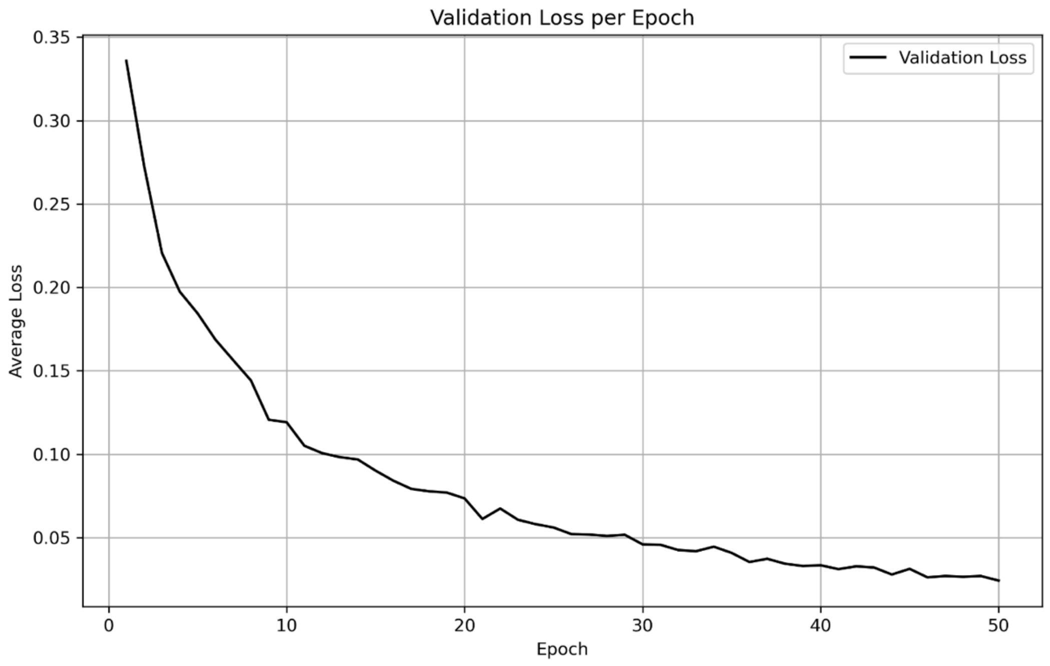

Figure 2: CPU Native Loss Curve

Tarun Vallabhaneni
tvallabh

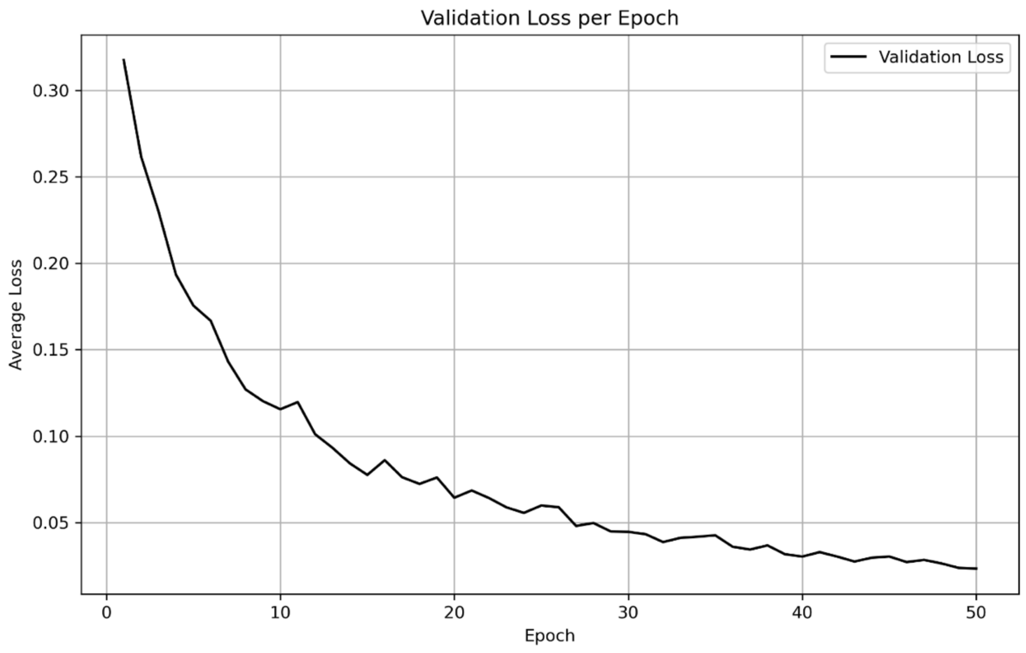

Figure 3: GPU Native Loss Curve

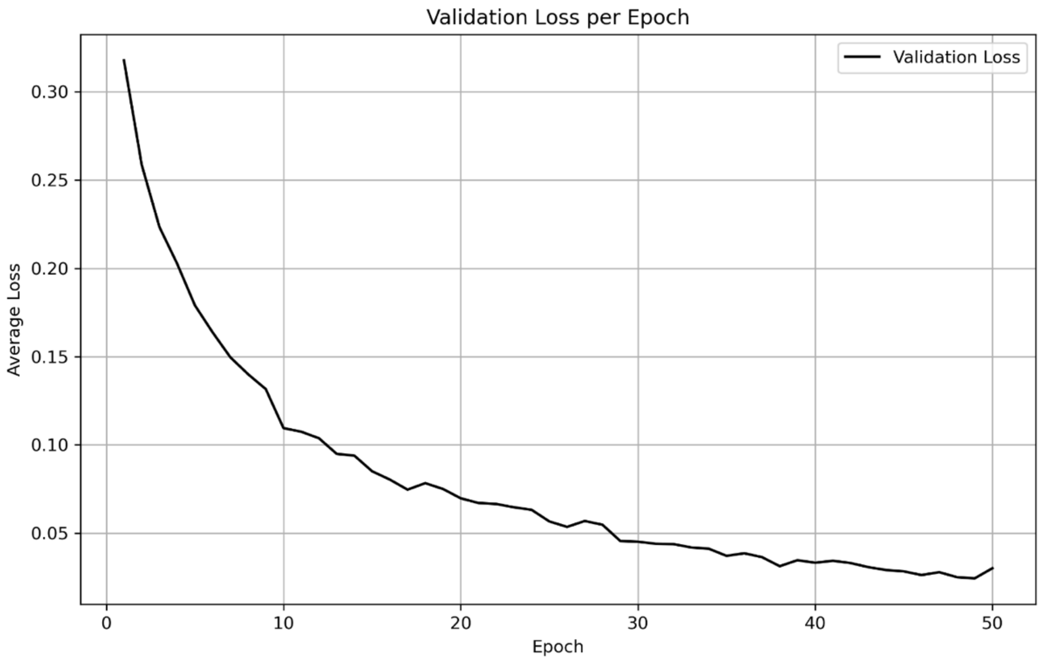

Figure 4: GPU cuBLAS Loss Curve

Note

In my implementation, the pseudo-random number generator was seeded with the current time, ensuring unique seeds for each run. Despite this, accuracy changes were minor, fluctuating by only about ±0.3%. This highlights the model's robust performance, showing its stability against seed variations.