Deep Learning Lab 1 Report

Shusrith S

PES1UG22AM155

January 18, 2025

1 Introduction

The MNIST (Modified National Institute of Standards and Technology) dataset is a widely used collection of 70,000 grayscale images of handwritten digits, each of size 28x28 pixels, representing the digits 0 through 9. It is divided into a training set of 60,000 images and a test set of 10,000 images, making it a standard benchmark for evaluating machine learning models, particularly for image classification tasks.

2 Architecture and Framework used

A feed-forward neural network is used to classify these images and identify the digit represented in the image. A sigmoid activation function is used in the model with a batch size of 20 and Adam optimizer. The PyTorch library is used for its efficient syntax and easy of use. The dataset is loaded from TensorFlow into PyTorch DataLoader objects. These objects facilitate faster loading of data along with parallelization, shuffling, batching and sampling.

TQDM is used to wrap the training process and show progress bars and loss for training and validation. It provides updates on the training process and integrates with the DataLoader. Since comparison plots are needed in this report, TensorBoard has been used extensively. It keeps track of training and validation loss during the training process and creates interactive plots and graphs. It also allows comparison between various runs, a key objective of this report.

The model is trained for 100 epochs for the first submission. All consequent comparison models are trained for 50 epochs. Tuning of the model revealed optimal training occured at a learning rate of 0.001 and led to a high accuracy of 96.5% on the test data. All training was done on an NVIDIA RTX 4050 laptop GPU.

3 One hidden layer

Figure 1: Single hidden layer with 50, 100 and 200 neurons

A single hidden layer was used. 50 neurons had higher train and val loss while 100 and 200 neurons had nearly the same loss. The time taken to train also reduced with an increase in the number of neurons, most likely because the GPU was better utilized with an increase in number of neurons, since there was only one hidden layer.

4 Two hidden layers

Figure 2: Two hidden layers with 50, 100 and 200 neurons

Model contained two hidden layers. Clear drop in loss with increase in number of neurons is observed. Time taken to train increases with increase in size of the model.

5 Three hidden layers

Figure 3: Three hidden layers with 50, 100 and 200 neurons

Three hidden layers were used. Marked increase in training times, with 200 neurons taking lesser time, showing better GPU utilization. Increase in number of neurons shows increase in performance across all graphs. No signs of overfitting is seen.

6 Varying optimizers

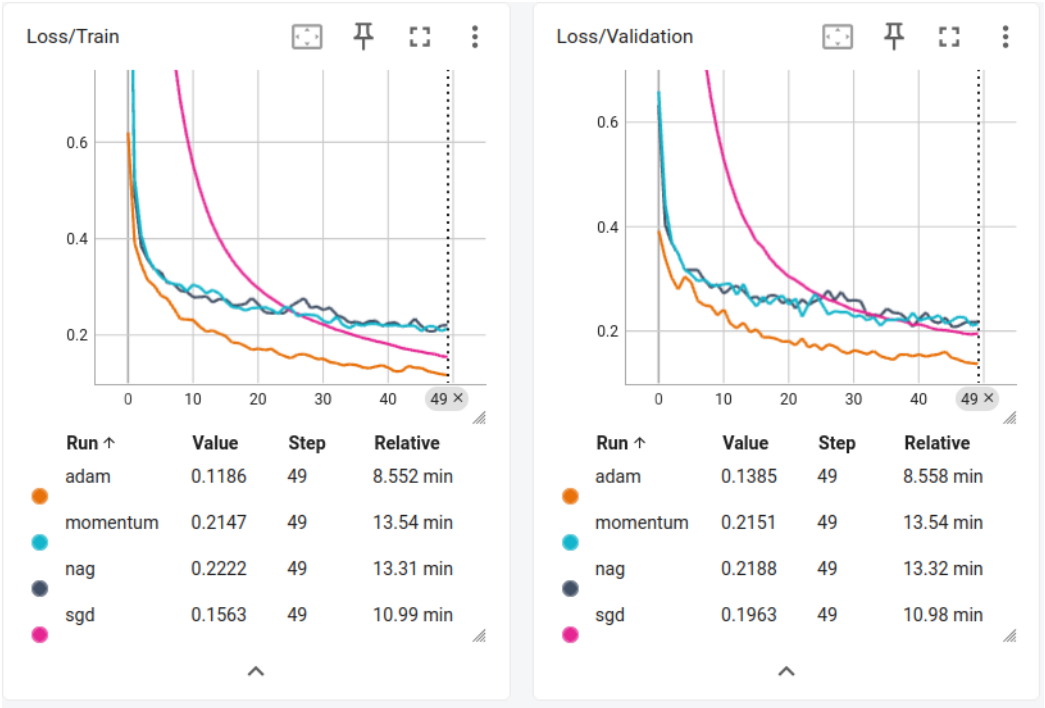

Figure 4: Comparing Adam, NAG, Momemntum, SGD optimizers

The four optimizers were used for 50 epochs each on the same architecture. Adam proved to be the most optimal and SGD was a close second with a very smooth curve. Since SGD takes individual samples, the loss starts off at a very high value and drops sharply. Despite the optimization in SGD, Adam was the fastest.

7 Varying Activation Functions

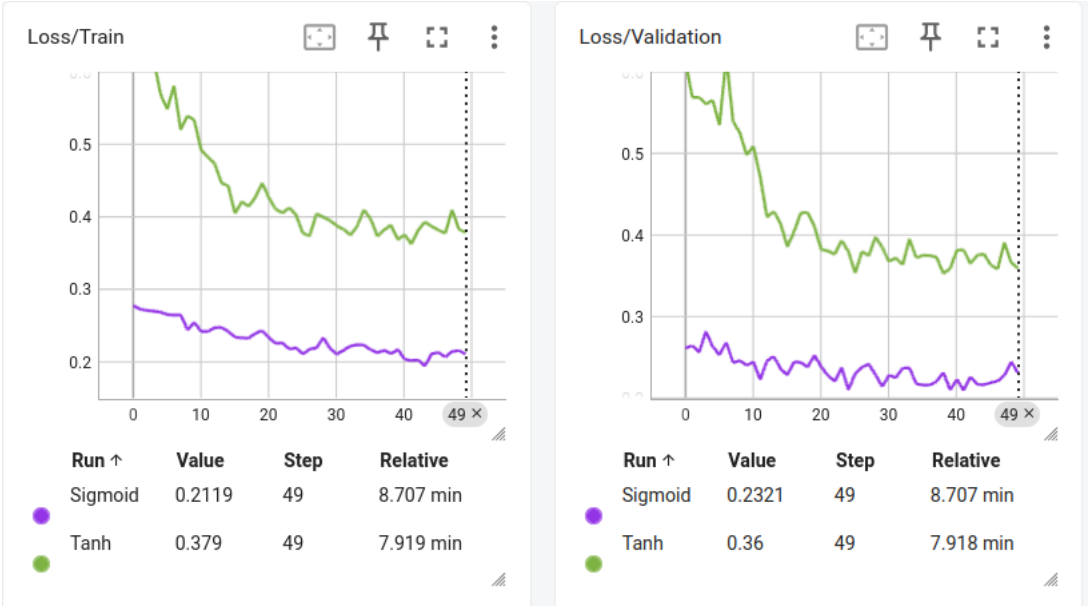

Figure 5: Comparing Sigmoid and Tanh activation functions

Sigmoid function has a very clear advantage over tanh in this scenario. Since it is a classification task, sigmoid works much better to model the data in the form of probabilities between 0 and 1. It also works well with cross entropy loss and as a result, performs much better than tanh

8 Varying loss functions

Figure 6: Comparing CrossEntropy and MSE loss

It is difficult to graphically compare the two losses because they operate on different scales. Cross entropy is preferred because it is a classification task and cross entropy drives the probability of prediction towards the positive class. To calculate MSE, an additional softmax has to be performed and labels need to be one-hot encoded for calculating loss. MSE is preferred for regression tasks.

9 Varying batch size

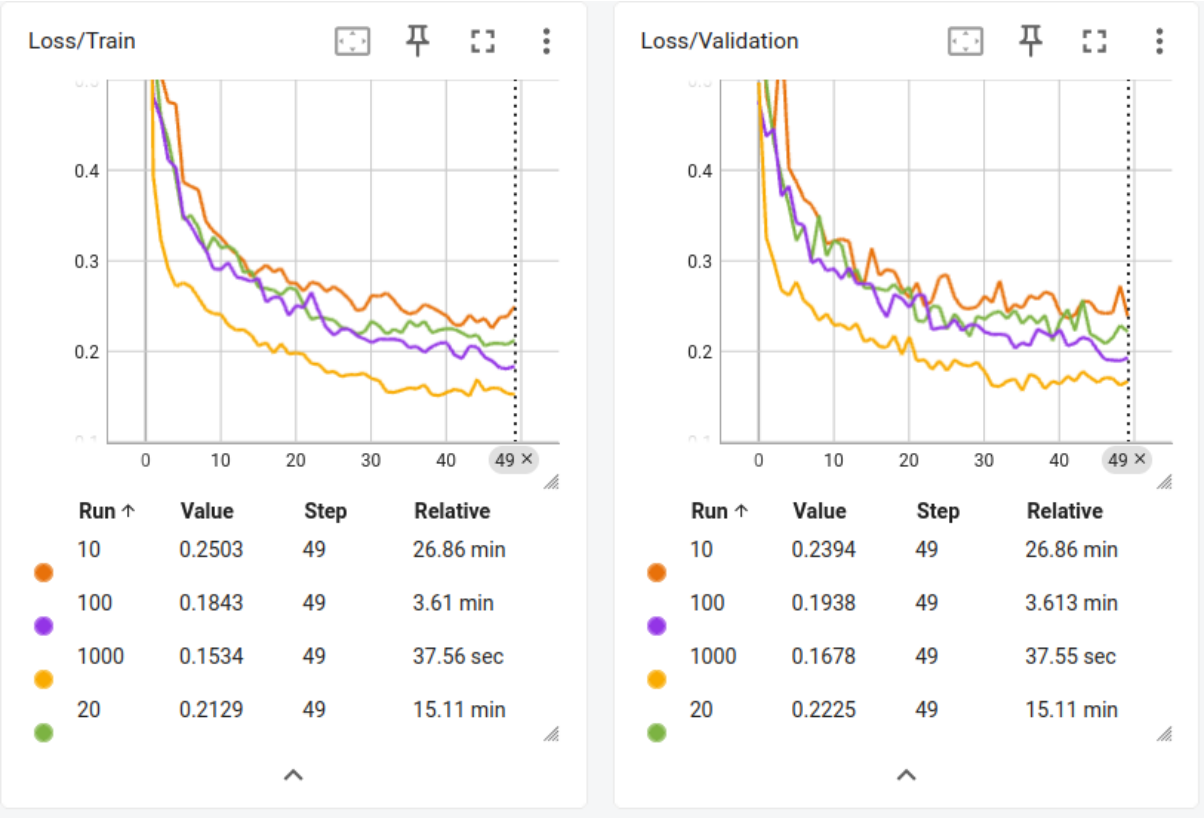

Figure 7: Comparing batch size of 10, 20, 100, 1000

Batch size has a significant impact on the training process. The speed of training exponentially increases with an increase in batch size, because the GPU can be better utilized. Training takes only 40 seconds with a batch size of 1000 compared to 27 minutes with a batch size of 10. The loss is also a lot lower because larger batch size allows better gradient calculation with less noise in the gradients, allowing better weight updates.