Documentation for ConvNet Training

Zahra Khodagholi

October 20, 2023

1 Introduction

This document provides a comprehensive description of the provided code for training different configurations of a convolutional neural network (ConvNet) using the MNIST dataset.

2 Libraries Imported

- **time**: Used for timing and benchmarking.
- **torch**: The main library for deep learning functionalities.
- **torch.nn**: Used for defining and manipulating neural network layers.
- **torch.nn.functional**: Provides functions like activation functions and convolution operations.
- **torch.optim**: Contains standard optimization routines like SGD, Adam, etc.
- **torchvision**: Provides access to popular datasets, model architectures, and image transformations for computer vision.
- **torch.utils.tensorboard**: TensorBoard visualization tool.
- **argparse**: Used for command-line option and argument parsing.
- **numpy**: Essential for numerical operations.
- **matplotlib.pyplot**: For plotting graphs and visual representations.

3 ConvNet Model

3.1 Initialization

The `ConvNet` class extends `nn.Module`. In its constructor, it defines several layers based on the mode. Five different configurations (or modes) of the network can be used.

3.2 Forward Pass

The forward pass routes the input through the respective model depending on the selected mode. The modes differ mainly in activation functions and the number of layers used.

4 Training and Testing Functions

4.1 Train

This function performs the training of the model for one epoch. It:

- Loads a batch of data and labels.
- Computes the model predictions.
- Calculates the loss between predictions and true labels.
- Backpropagates errors and updates model weights.
- Computes and returns average training loss and accuracy for the epoch.

4.2 Test

The testing function evaluates the model on test data:

- Predicts using the model.
- Computes the loss for predictions.
- Calculates accuracy on the test set.
- Returns the average test loss and accuracy.

5 Main Execution

5.1 Configurations and Hyperparameters

The code uses `argparse` to take command-line arguments, allowing users to specify:

- Mode for the ConvNet architecture.
- Learning rate.
- Number of training epochs.
- Batch size.
- Directory for logging.

5.2 Data Loading and Transformations

The code uses the MNIST dataset. It applies transformations like converting images to tensors and normalizing.

5.3 Training Loop

For each epoch:

- The model is trained using the `train` function.
- The model's performance is evaluated using the `test` function.
- Training and testing statistics are stored for visualization.

5.4 Visualization

Post-training, loss, and accuracy statistics are plotted using `matplotlib`.

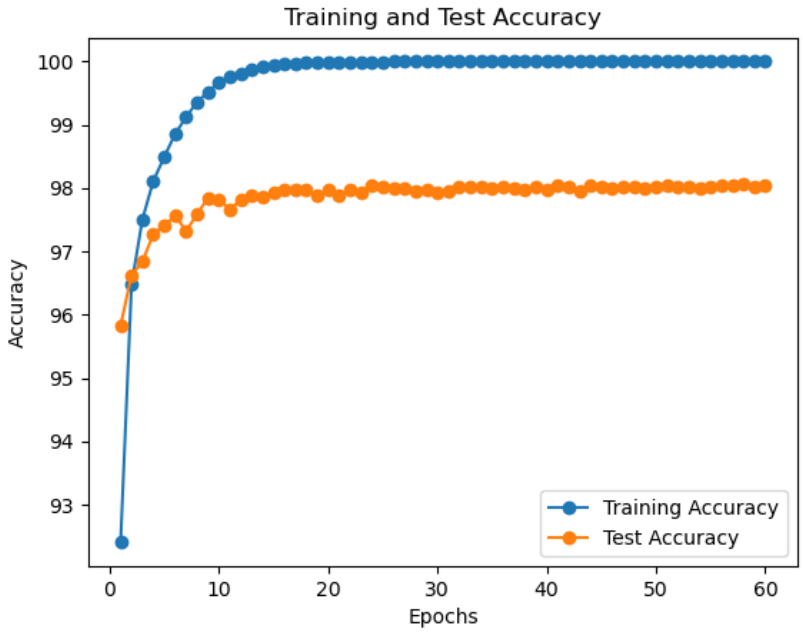

Figure 1: Training and Test Accuracy over Epochs

Observations from the Training and Test Accuracy plot:

- The training accuracy starts near 93% and rapidly approaches 100% within the first 10 epochs. This quick rise suggests that the model adapts well to the training data early on.

- Test accuracy begins near 94% and sees a swift increase in the first few epochs. It then experiences a plateau post the 10th epoch before it slightly dips and stabilizes around 98%.
- The notable gap between training and test accuracy up to the 40th epoch suggests potential overfitting during the initial training phase. However, post the 40th epoch, the gap narrows, indicating a better generalization on the test set.

Training and Test Loss

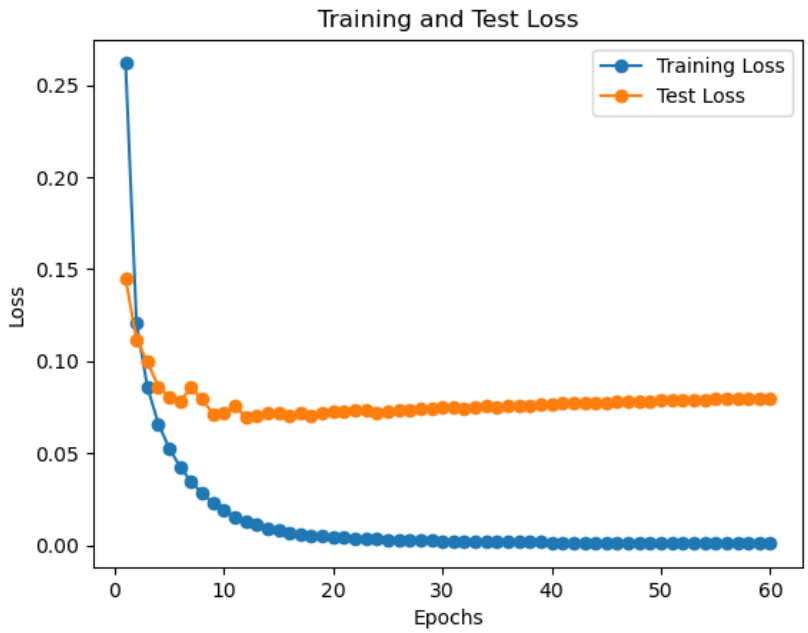

Figure 2: Training and Test Loss over Epochs

Insights from the Training and Test Loss plot:

- The training loss starts notably high but observes a drastic reduction within the first 10 epochs. After this rapid decrease, the loss tapers off and stabilizes, showing minor fluctuations.
- Test loss begins slightly elevated but reduces substantially in the first 10 epochs. After this decline, it maintains a more gradual descent until converging closely with the training loss by the 40th epoch.
- The convergence of the training and test loss after the 40th epoch suggests that the model is becoming better at generalizing its predictions for unseen data. The relatively parallel trajectories of the losses post this point indicate consistent performance.

6 Conclusion

This code provides a comprehensive setup for training a ConvNet on the MNIST dataset with various configurations. Users can easily modify hyperparameters and visualize training and testing performances.