CSL7590 - Deep Learning
Assignment-1

By Sumeet S Patil (B22CS052)

Project Overview

This practical exercise focuses on developing a custom neural network implementation using fundamental Python libraries. The primary objectives include constructing a multilayer perceptron architecture and designing an efficient backpropagation algorithm for parameter optimization. The network will be evaluated on handwritten digit recognition tasks using standard benchmarking protocols. Google Colab

Dataset Preparation

The experiment utilizes the MNIST database of handwritten digits, accessed through PyTorch's data utilities. Notably, PyTorch serves solely for data retrieval purposes, with all neural network operations implemented using NumPy for fundamental matrix computations.

- Training samples: 60,000 grayscale images
- Evaluation samples: 10,000 grayscale images

Figure 1: Visual representation of MNIST digit samples with corresponding labels

Three distinct data partitioning strategies were implemented:

- Primary training-test splits: 70%-30%, 80%-20%, and 90%-10%

Network Architecture Design

The implemented neural network features the following structural components:

Layer Configuration

- Input dimension: 784 (28×28 pixel flattening)
- Hidden layer specifications: Input from user, as of now 128,64
- Output dimension: 10 (corresponding to digit classes 0-9)

Training Parameters

- Hidden layer dimensionality: 256 units per layer
- Training iterations: 25 epochs
- Mini-batch size: 22 samples
- Initial learning rate: 1e-2
- Nonlinear activations: Sigmoid, Tanh, ReLU variants (Used only Tanh and ReLU as of now)

Parameter Initialization

- Weight matrices: Randomized initialization with fixed seed (52)
- Bias vectors: Constant initialization (unit values)

Activation Functions

- Hidden layers: Selectable activation (Sigmoid/Tanh/ReLU/Leaky ReLU)
- Output layer: Softmax normalization to ensure probabilistic check

Implementation Details

Forward Propagation

Computational graph operations include:

- Linear transformations (weighted sums)
- Activation mappings
- Cross-entropy loss calculation

Gradient Computation

Backpropagation mechanics implement:

- Partial derivative calculations through computational graph
- Parameter updates via stochastic gradient descent
- Loss/accuracy tracking per training iteration

Training Protocol

- Mini-batch stochastic optimization
- Epoch-wise loss/accuracy aggregation

Performance Evaluation

Experimental results across different configurations are summarized below:

Metric	Hyperbolic Tangent			Rectified Linear Unit		
	90-10	80-20	70-30	90-10	80-20	70-30
Testing Loss	0.56	0.58	0.63	0.53	0.44	0.49
Testing Accuracy	83.13%	83.5%	81.50%	86.88%	87.89%	87.38%

Table 1: Performance comparison across activation functions and data partitions

Key Findings

- **Total trainable parameters**:
  - Weights: $784\times128 + 128\times64 + 64\times10$
  - Bias terms: $128 + 64 + 10 = 522$
- **Total non-trainable parameters: 8**
  - Learning rate: 1
  - Batch size: 1
  - Epochs: 1
  - Activation Function: 1

  - Layers size (taken the same size for both hidden layers): 1+2+1 = 4
- Performance observations:
  - ReLU activations outperformed Tanh variants
  - Training data proportionality positively correlated with accuracy
  - Common misclassifications: 4→9, 3→5, 8→5

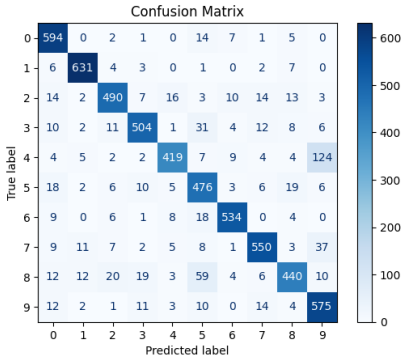


Figure 2: Classification error matrix highlighting frequent confusion patterns

Design Considerations

- Fixed learning rate (non-adaptive)
- Iteration count constraint (25 epochs)
- Cross-entropy loss formulation
- Fixed mini-batch size (22 samples)

References

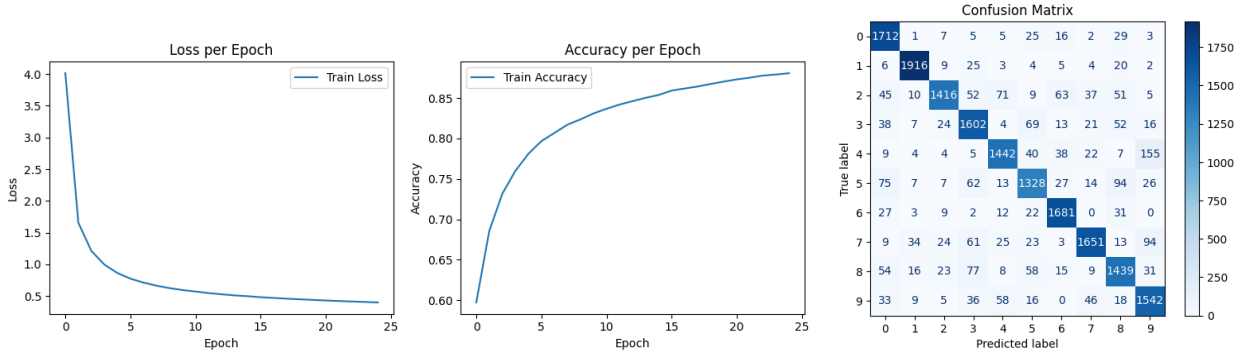
- Class notes
- Matplotlib documentation for plotting
- Seaborn documentation for heat map

Appendix

These are all the plots and the confusion matrices:

Split: 70:30:

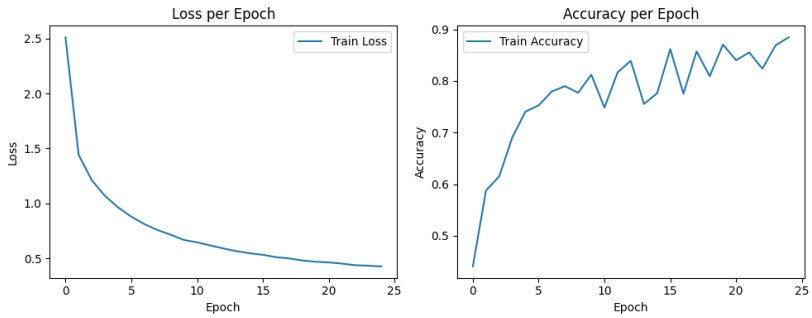
- Tanh Activation function



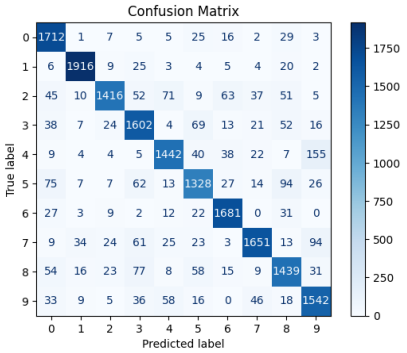
(a) Training loss vs epochs and Training accuracy vs epochs

(b) Testing confusion matrix

- ReLU Activation function



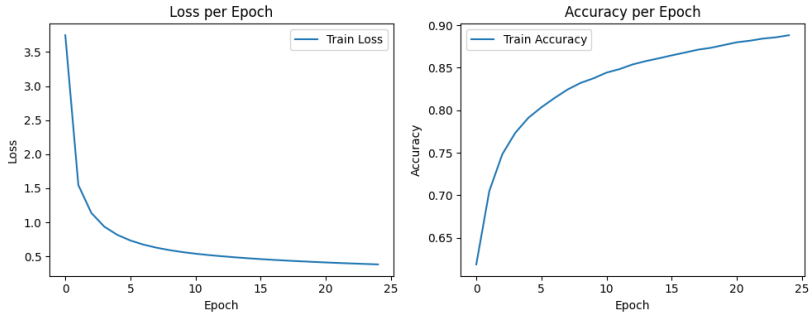
(a) Training loss vs epochs and Training accuracy vs epochs



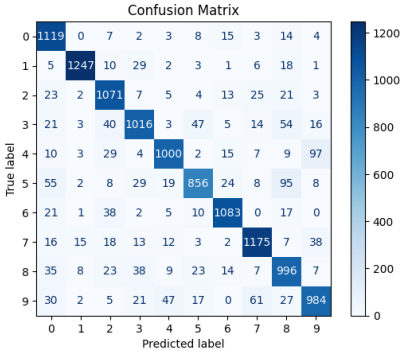
(b) Testing confusion matrix

Split: 80:20:

- Tanh Activation function

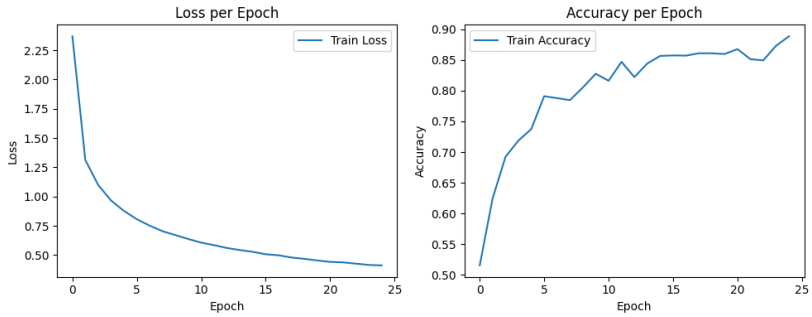


(a) Training loss vs epochs and Training accuracy vs epochs

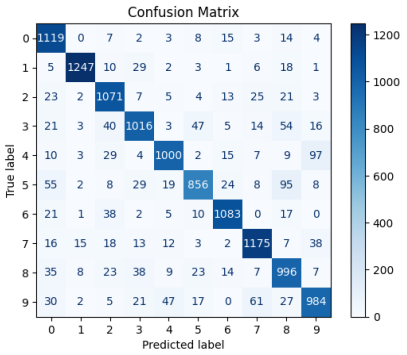


(b) Testing confusion matrix

- ReLU Activation function



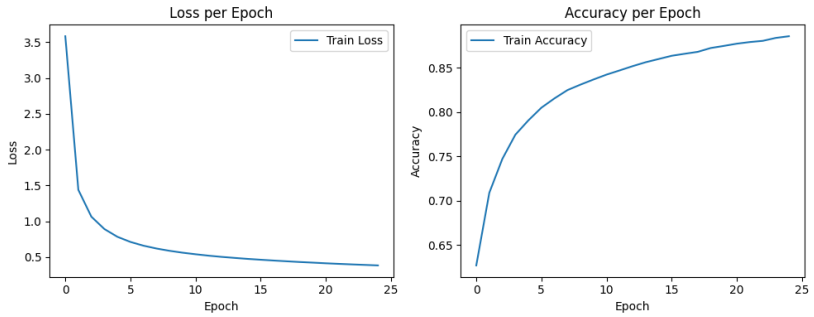
(a) Training loss vs epochs



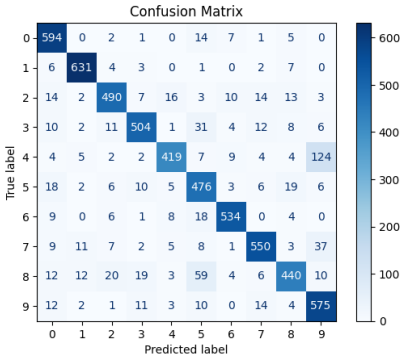
(b) Testing confusion matrix

Split: 90:10:

- Tanh Activation function

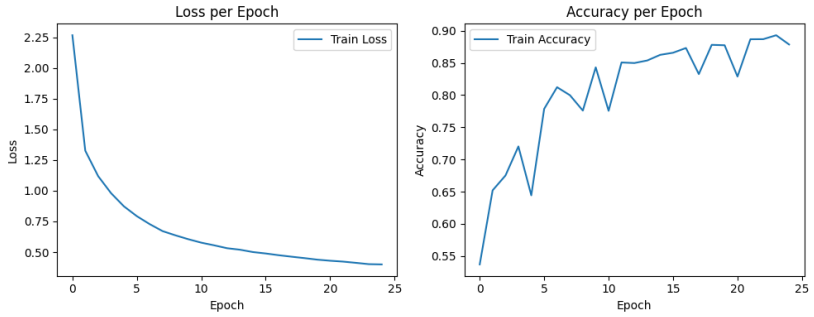


(a) Training loss vs epochs and Training accuracy vs epochs

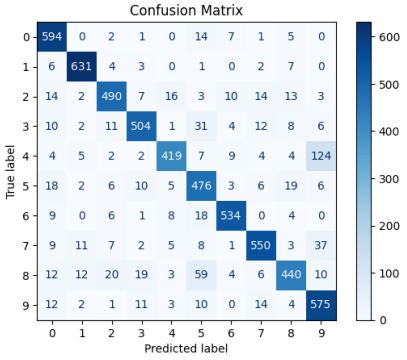


(b) Testing confusion matrix

- ReLU Activation function



(a) Training loss vs epochs and Training accuracy vs epochs



(b) Testing confusion matrix