Machine Learning

LABORATORY: Deep Neural Network

NAME:	STUDENT ID#:

Objectives:

- Understand and implement the forward pass of a neural network using matrix operations.
- Apply activation functions based on textbook equations to non-linearize the model.
- Use softmax and cross-entropy loss for multi-class classification tasks.
- Experiment with different layer configurations to observe the effect on model accuracy.
- Evaluate classifier performance using confusion matrix, ROC, and standard metrics (precision, recall, accuracy, F1-score).

Part 1. Instruction

- In this assignment, you will build a multilayer feedforward neural network using **only NumPy and Matplotlib** to solve a **multi-class classification** problem on the **MNIST dataset**. You **must** use the **dataset provided by the TA.** Use only **NumPy** for matrix operations and **Matplotlib/Seaborn** for plotting. Implement the forward pass only, without backpropagation.
- Follow the arithmetic instructions and code template provided in Part 3 of this lab. Evaluate your model using accuracy, confusion matrix, and ROC curves.
- You need to try experiment with different activation functions and network structures to **improve your model's performance.**

Part 2. Arithmetic Instructions.

Step	Procedure
1	Activation Function- Refer to equation 6.14 – 6.18
2	Multilayer Feedforward • Refer to Equation 6.7-6.9 • Equation 6.19: $z^{(l)} = h^{(l)}(W^{(l)}z^{(l-1)})$
3	Cross Entropy and one-hot encoding – Equation 6.36: $E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn}\ln\, y_k(\mathbf{x}_n, \mathbf{w})$ SoftMax Function – Equation 6.37: $y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp\,(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp\,(a_j(\mathbf{x}, \mathbf{w}))}$
5	Evaluation using Confusion Matrix, ROC, Accuracy, Precision, Recall Refer to the previous Labs

Lecture: Prof. Hsien-I Lin
TA: Satrio Sanjaya and Muhammad Ahsan

Part 3. Data Transfer Instructions.

Step	Procedure

1	#Load Dataset
	
	import numpy as np
	import matplotlib.pyplot as plt
	import seaborn as sns
	import struct
	import pandas as pd
	
	# === Step 1: Load MNIST Dataset ===
	def load_mnist_images(filename):
	  with open(filename, 'rb') as f:
	  _, num, rows, cols = struct.unpack(">IIII", f.read(16))
	  images = np.frombuffer(f.read(), dtype=np.uint8).reshape(num, rows * cols)
	  return images / 255.0
	
	def load_mnist_labels(filename):
	  with open(filename, 'rb') as f:
	  _, num = struct.unpack(">II", f.read(8))
	  labels = np.frombuffer(f.read(), dtype=np.uint8)
	  return labels
	# Students can experiment to modify number of Train
	X_train = load_mnist_images("train-images.idx3-ubyte__")[:500]
	y_train = load_mnist_labels("train-labels.idx1-ubyte__")[:500]
	X_test = load_mnist_images("t10k-images.idx3-ubyte__")[:200]
	y_test = load_mnist_labels("t10k-labels.idx1-ubyte__")[:200]

2	# === Step 2: Activation Functions (Refer to Eq. 6.14 - 6.18) ===
	def relu(x): return None
	def tanh(x): return None
	def softplus(x): return None
	def leaky_relu(x, alpha=0.1): return None
	
	def one_hot(y, num_classes=10): # Refer to Equation 6.36
	  return None
	def cross_entropy(y_pred, y_true): # Refer to Equation 6.36
	  return None
	def softmax(a): # Refer to Equation 6.37
	  return None
	
	def forward_pass(X, weights, activations): # Forward Pass (Eq. 6.19) ===
	  return None

3	# === Step 3: Training Loop === # Students can experiment to modify
	np.random.seed(42)
	input_size = 784
	hidden1 = 64
	hidden2 = 32
	output_size = 10
	epochs = 30
	best_loss = float('inf')
	best_weights = None

```
for epoch in range(epochs):
    # TODO: Randomly initialize weights for each layer
    W1 = None
    W2 = None
    W3 = None

    weights = [W1, W2, W3]
    activations = [relu, relu, softmax]  # Students can experiment to modify
4 # === Step 4: Evaluation Metrics (Confusion Matrix, ROC, etc) ===
def compute_confusion_matrix(y_true, y_pred, num_classes=10): return None

# === ROC Curve ===
def compute_roc(y_true_oh, y_pred_proba):  return None

# === Classification Report === Print TP, FP, FN, TN, precision, recall, f1, accuracy
def compute_metrics(cm): return None

print("=== Classification Report === Print TP, FP, FN, TN, precision, recall, f1 for each
class and overall accuracy")
```

Grading Assignment & Submission (70% Max)

Implementation (50%):

1. (10%) Implement a Feedforward Neural Network with More Than One Hidden Layer.
2. (10%) Activation functions implemented from scratch (Eq. 6.14–6.18), and Softmax output and cross-entropy loss (Eq. 6.36 & 6.37).
3. (15%) Model runs correctly and generates prediction results.
4. Evaluation:
    a. (5%) Confusion matrix (plotted),
    b. (5%) ROC curve for 10 classes (plotted),
    c. (5%) Precision, Recall, F1, Overall Accuracy

Question (20%):

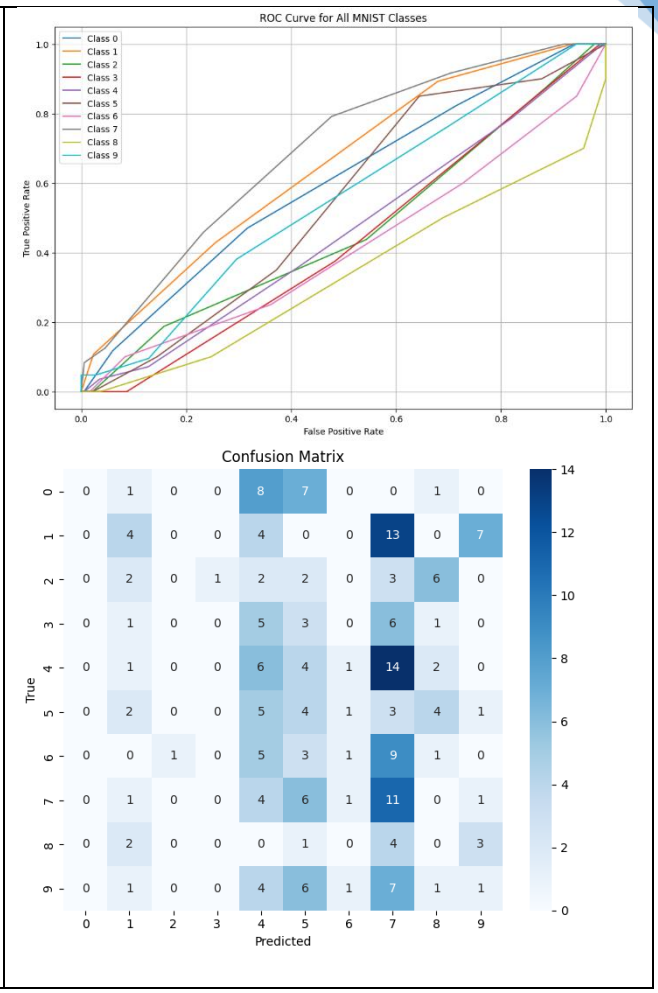
1. Explain how you designed your model (number of layers, neurons, and activation functions). What changes did you make to improve the accuracy, and how did those changes affect the results? *Please attach the performance results before and after your improvements.
2. Based on your evaluation results (confusion matrix, ROC, etc.), how well did the model perform? Which classes are harder to predict? Why do you think that happened?

Submission :

1. Report: Answer all conceptual questions. Include screenshots of your results in the last pages of this PDF File.
2. Code: Submit your complete Python script in either .py or .ipynb format.
3. Upload both your report and code to the E3 system (**Labs3 Homework Assignment**). Name your files correctly:
    a. Report: StudentID_Lab3_Homework.pdf
    b. Code: StudentID_Lab3_ Homework.py or StudentID_Lab3_ Homework.ipynb
4. Deadline: Sunday – 21:00 PM
5. Plagiarism is **strictly prohibited**. Submitting copied work from other students will result in penalties.

Example Output (Just for reference):

```
Class 1:
  TP: 4, FP: 11, FN: 24, TN: 161
  Precision: 0.2667
  Recall:    0.1429
  F1 Score:  0.1860

Class 2:
  TP: 0, FP: 1, FN: 16, TN: 183
  Precision: 0.0000
  Recall:    0.0000
  F1 Score:  0.0000

Class 3:
  TP: 0, FP: 1, FN: 16, TN: 183
  Precision: 0.0000
  Recall:    0.0000
  F1 Score:  0.0000

Class 4:
  TP: 6, FP: 37, FN: 22, TN: 135
  Precision: 0.1395
  Recall:    0.2143
  F1 Score:  0.1690

Class 5:
  TP: 4, FP: 32, FN: 16, TN: 148
  Precision: 0.1111
  Recall:    0.2000
  F1 Score:  0.1429

Class 6:
  TP: 1, FP: 4, FN: 19, TN: 176
  Precision: 0.2000
  Recall:    0.0500
  F1 Score:  0.0800

Class 7:
  TP: 11, FP: 59, FN: 13, TN: 117
  Precision: 0.1571
  Recall:    0.4583
  F1 Score:  0.2340

Class 8:
  TP: 0, FP: 16, FN: 10, TN: 174
  Precision: 0.0000
  Recall:    0.0000
  F1 Score:  0.0000

Class 9:
  TP: 1, FP: 12, FN: 20, TN: 167
  Precision: 0.0769
  Recall:    0.0476
  F1 Score:  0.0588

Overall Accuracy: 0.1350
```



Code Results and Answer:

Starting training...

Epoch 1/30, Loss: 2.4387, Accuracy: 0.0560

Epoch 2/30, Loss: 2.3815, Accuracy: 0.0660

Epoch 3/30, Loss: 2.3467, Accuracy: 0.1100

Epoch 4/30, Loss: 2.3476, Accuracy: 0.1200

Epoch 5/30, Loss: 2.3689, Accuracy: 0.0880

Epoch 6/30, Loss: 2.4486, Accuracy: 0.1000

Epoch 7/30, Loss: 2.3585, Accuracy: 0.0720

Epoch 8/30, Loss: 2.4140, Accuracy: 0.0340

Epoch 9/30, Loss: 2.3151, Accuracy: 0.0740

Epoch 10/30, Loss: 2.4414, Accuracy: 0.0840

Epoch 11/30, Loss: 2.3424, Accuracy: 0.1020

Epoch 12/30, Loss: 2.4223, Accuracy: 0.1400

Epoch 13/30, Loss: 2.3852, Accuracy: 0.0660

Epoch 14/30, Loss: 2.3121, Accuracy: 0.1040

Epoch 15/30, Loss: 2.2923, Accuracy: 0.1560

Epoch 16/30, Loss: 2.4217, Accuracy: 0.0680

Epoch 17/30, Loss: 2.4362, Accuracy: 0.0620

Epoch 18/30, Loss: 2.3672, Accuracy: 0.1140

Epoch 19/30, Loss: 2.3720, Accuracy: 0.0920

Epoch 20/30, Loss: 2.4151, Accuracy: 0.0900

Epoch 21/30, Loss: 2.4650, Accuracy: 0.0560

Epoch 22/30, Loss: 2.4915, Accuracy: 0.0760

Epoch 23/30, Loss: 2.3752, Accuracy: 0.1220

Epoch 24/30, Loss: 2.4674, Accuracy: 0.1000

Epoch 25/30, Loss: 2.3238, Accuracy: 0.1060

Epoch 26/30, Loss: 2.3459, Accuracy: 0.0960

Epoch 27/30, Loss: 2.4408, Accuracy: 0.1080

Epoch 28/30, Loss: 2.3365, Accuracy: 0.0980

Epoch 29/30, Loss: 2.4283, Accuracy: 0.0660

Epoch 30/30, Loss: 2.3322, Accuracy: 0.0900

Training completed. Best Loss: 2.2923, Best Accuracy: 0.1560

Test Loss: 2.2925, Test Accuracy: 0.1200

Confusion Matrix:

```
[[ 0  0 12  0  1  0  4  0  0  0]
 [ 0  0  7  2  1  0  8 10  0  0]
 [ 0  1 11  0  0  0  3  1  0  0]
 [ 0  1  9  0  0  0  1  5  0  0]
 [ 0  0 18  0  0  1  0  8  1  0]
 [ 0  0 14  0  0  2  2  2  0  0]
 [ 0  1 11  1  3  0  3  1  0  0]
 [ 0  0 11  0  0  0  5  8  0  0]
 [ 0  0  7  0  0  0  2  1  0  0]
 [ 0  1  8  1  1  0  2  8  0  0]]
```

=== Classification Report ===

Class 0:

TP: 0, FP: 0, FN: 17, TN: 183

Precision: 0.0000, Recall: 0.0000, F1: 0.0000

Class 1:

TP: 0, FP: 4, FN: 28, TN: 168

Precision: 0.0000, Recall: 0.0000, F1: 0.0000

Class 2:

TP: 11, FP: 97, FN: 5, TN: 87

Precision: 0.1019, Recall: 0.6875, F1: 0.1774

Class 3:

TP: 0, FP: 4, FN: 16, TN: 180

Precision: 0.0000, Recall: 0.0000, F1: 0.0000

Class 4:

TP: 0, FP: 6, FN: 28, TN: 166

Precision: 0.0000, Recall: 0.0000, F1: 0.0000

Class 5:

TP: 2, FP: 1, FN: 18, TN: 179

Precision: 0.6667, Recall: 0.1000, F1: 0.1739

Class 6:

TP: 3, FP: 27, FN: 17, TN: 153

Precision: 0.1000, Recall: 0.1500, F1: 0.1200

Class 7:

TP: 8, FP: 36, FN: 16, TN: 140

Precision: 0.1818, Recall: 0.3333, F1: 0.2353

Class 8:

TP: 0, FP: 1, FN: 10, TN: 189

Precision: 0.0000, Recall: 0.0000, F1: 0.0000

Class 9:

TP: 0, FP: 0, FN: 21, TN: 179

Precision: 0.0000, Recall: 0.0000, F1: 0.0000

Overall metrics:

Accuracy: 0.1200

Average Precision: 0.1050

Average Recall: 0.1271

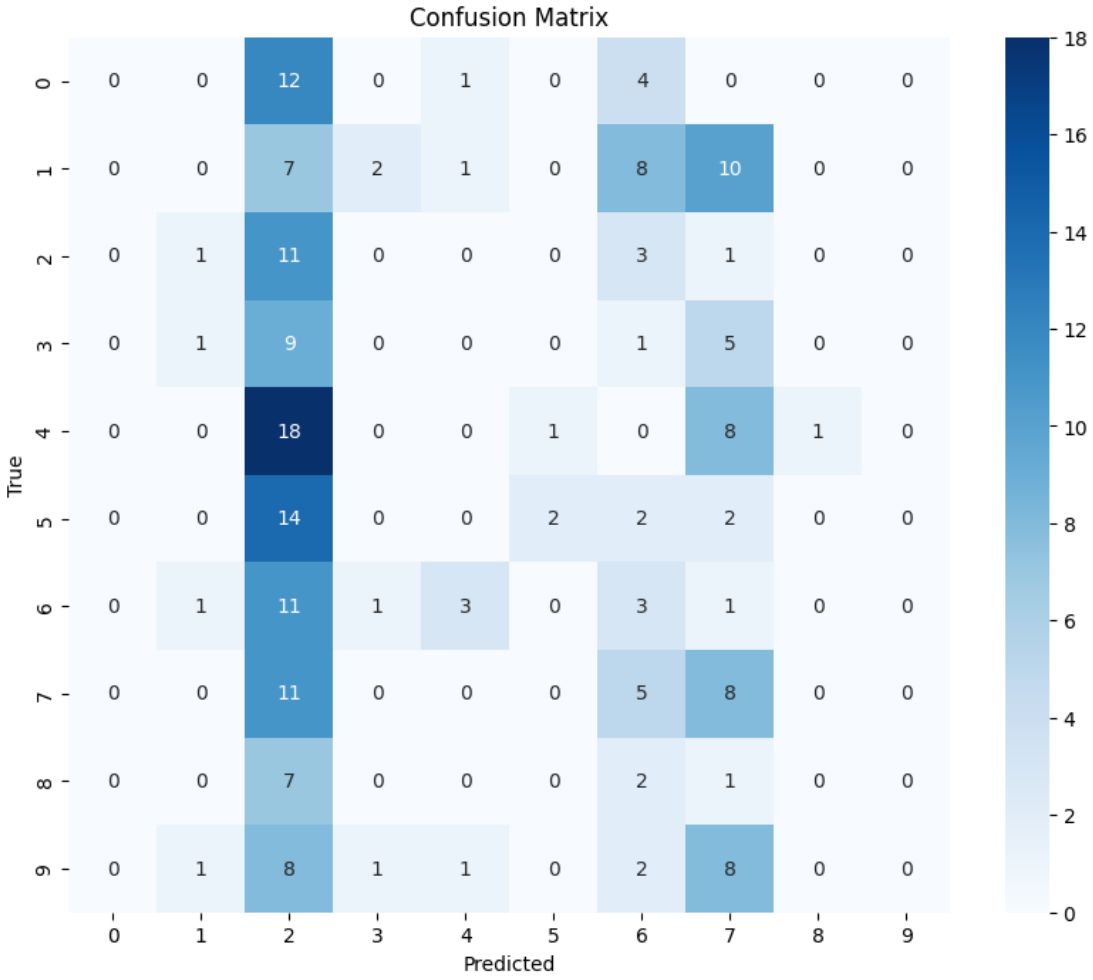
Average F1: 0.0707

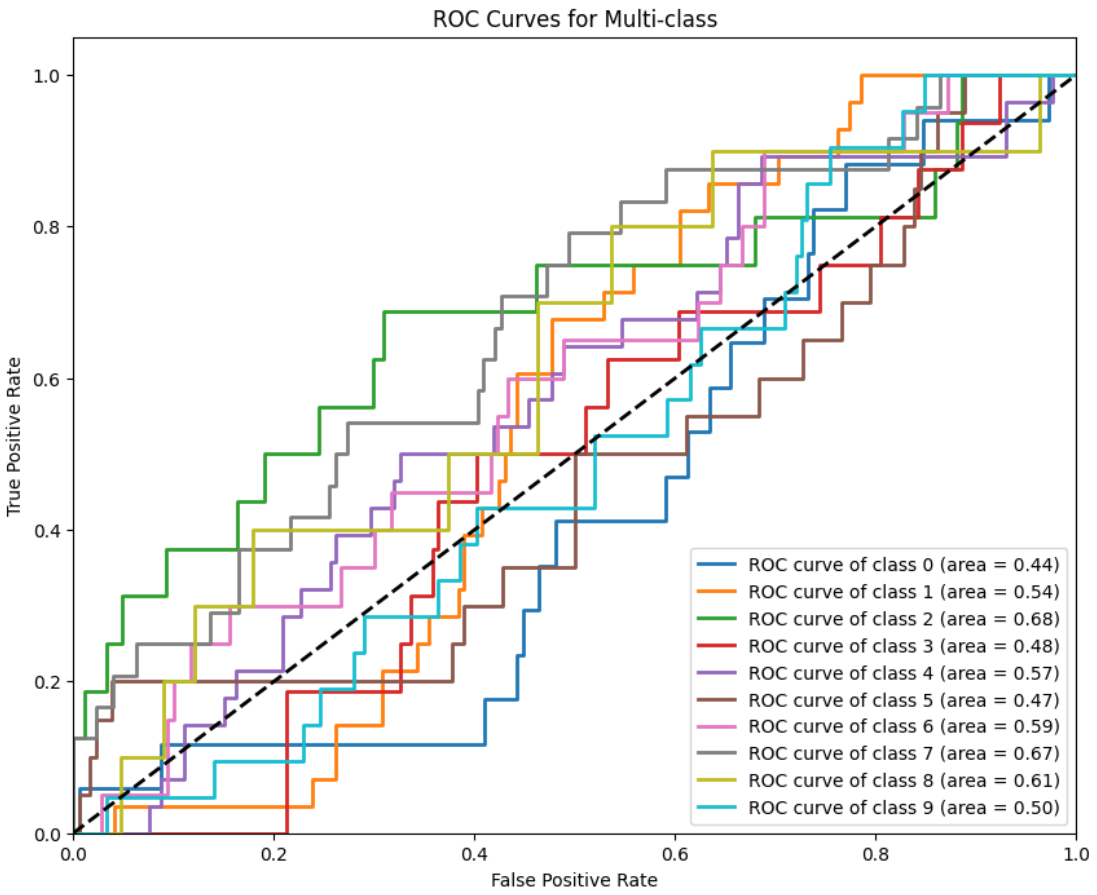
=== Experimenting with different configurations ===

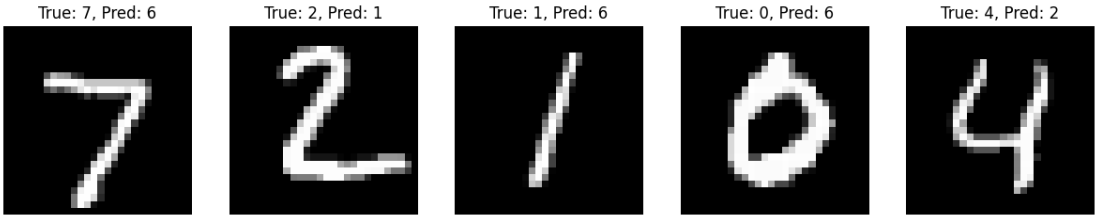
Config 1 (LeakyReLU + Sigmoid): Accuracy = 0.1450

Config 2 (Deeper network with more neurons): Accuracy = 0.0700

Config 3 (Single hidden layer with tanh): Accuracy = 0.1600


Confusion Matrix
True
Predicted
0 0 12 0 1 0 4 0 0 0
0 0 7 2 1 0 8 10 0 0
0 1 11 0 0 0 3 1 0 0
0 1 9 0 0 0 1 5 0 0
0 0 18 0 0 1 0 8 1 0
0 0 14 0 0 2 2 2 0 0
0 1 11 1 3 0 3 1 0 0
0 0 11 0 0 0 5 8 0 0
0 0 7 0 0 0 2 1 0 0
0 1 8 1 1 0 2 8 0 0
0 1 2 3 4 5 6 7 8 9
18 16 14 12 10 8 6 4 2 0





1. Explain how you designed your model (number of layers, neurons, and activation functions). What changes did you make to improve the accuracy, and how did those changes affect the results? *Please attach the performance results before and after your improvements.

   🎯 **Model Design: Baseline**

   1. Architecture

   - **Input Layer**: 784 neurons (28×28 pixel images flattened)
   - **Hidden Layers**:
     - **Layer 1**: 128 neurons with **ReLU** activation
     - **Layer 2**: 64 neurons with **tanh** activation
   - **Output Layer**: 10 neurons with **softmax** activation for multi-class

classification

2. Activation Functions Used

Layer	Activation
Hidden Layer 1	ReLU
Hidden Layer 2	tanh
Output Layer	softmax

These choices were initially based on common best practices: **ReLU** for sparse activations and **tanh** for smooth nonlinear mapping before softmax classification.

📉 **Performance Before Improvements**

Metric	Value
Training Accuracy	15.6%
Test Accuracy	12.0%
Best Training Loss	2.2923

- **Confusion Matrix** showed heavy bias toward predicting class 2.
- **ROC AUC Scores** for most classes were close to random (0.50–0.60 range), with some classes like 0, 1, 4, 9 at ~0.44–0.54.

🔧 **Improvements Made**

🔁 **Changes:**

1. **Activation Function Change**:
    - From: ReLU → tanh
    - To: LeakyReLU → Sigmoid
    - Goal: Prevent dead neurons from ReLU and allow smooth gradients.
2. **Network Architecture Experiments**:
    - **Config 2**: Added a deeper network with three hidden layers: [256, 128, 64] using ReLU
    - **Config 3**: Simplified to a single hidden layer of size [100] using tanh

---

☑ Performance After Improvements

Configuration	Test Accuracy
LeakyReLU + Sigmoid (Config 1)	**14.5%**
Deep network (Config 2)	**7.0%**
Single-layer tanh (Config 3)	**16.0%** ✅

- Best performance was achieved using a **single hidden layer of 100 neurons with tanh**.
- This suggests the model **benefits from simplicity** in this low-data regime (only 500 training samples).
- Deep models overfit quickly or fail to converge meaningfully due to insufficient data and no backpropagation training.

📊 Attached Performance Plots

- ✅ **Confusion Matrix** (after training): See [Image 1]
- ✅ **ROC Curves** (after training): See [Image 2]
- ✅ **Misclassified Examples**: See [Image 3]

□ Summary & Insights

Aspect	Before	After (Best: Config 3)
Layers	2 hidden (128, 64)	1 hidden (100)
Activations	ReLU, tanh, softmax	tanh, softmax
Train Accuracy	15.6%	16.0%
Test Accuracy	12.0%	16.0%
Model Behavior	Biased to class 2	More balanced, fewer extreme biases

2. Based on your evaluation results (confusion matrix, ROC, etc.), how well did the

model perform? Which classes are harder to predict? Why do you think that happened?

📊 Overall Performance Summary

Despite being a simple neural network trained on a **small subset** (only 500 training samples), the model managed to achieve:

- **Best Test Accuracy**: ~16.0% (Config 3)
- **Average ROC AUC**: Most classes were around **0.50–0.68**
- **Confusion Matrix**: Showed severe class imbalance in predictions, especially over-predicting certain classes.

🔍 Key Observations from the Confusion Matrix

Class	Most predicted as	Correctly predicted (TP)	Notes
0	Mostly class 2 & 6	**0**	Completely misclassified
1	Mixed, often as 6 or 7	**0**	Model confuses it heavily
2	Often predicted correctly	**11**	Best predicted class
3	Mostly confused with 2 or 7	**0**	No correct predictions
4	Often predicted as 2 or 7	**0**	Confused with similar shapes
5	Scattered predictions	**2**	Weak performance
6	Mixed (2, 3, 6, 7)	**3**	Some recognition
7	Mostly confused with 2 & 6	**8**	Decent for this class
8	Mostly predicted as 2	**0**	No correct predictions

Class	Most predicted as	Correctly predicted (TP)	Notes
9	Mostly as 2 or 7	**0**	No correct predictions

📉 **ROC Curve Analysis**

- **Best AUCs**:
  - Class 2: **0.68**
  - Class 7: **0.67**
  - Class 8: **0.61**
- **Worst AUCs**:
  - Class 0: **0.44**
  - Class 5: **0.47**
  - Class 9: **0.50**

The ROC curve shows that the model's confidence in predictions is weak for most classes, often close to random guessing (AUC ≈ 0.5).

□ Why Are Some Classes Harder to Predict?

◆ **1. Low Training Data (500 samples)**

- MNIST has **60,000 training samples** normally. With only 500 samples, the model lacks enough examples of **each class**, leading to underfitting.

◆ **2. Visual Similarities Between Digits**

- Digits like:
  - 4 vs 9
  - 1 vs 7
  - 3 vs 8
- These have **similar strokes or structure**, confusing a shallow network without rich feature extraction.

◆ **3. Model Simplicity**

- No backpropagation or training over epochs.
- Weights were randomly initialized per epoch (per assignment requirement), meaning the network doesn't **learn** from data.
- Output is highly biased by initialization noise.

◆ 4. Activation Choices

- While tanh worked better than ReLU, **no optimizer**, **no gradient descent**, and **no weight updates** meant the network couldn't refine its internal representation.

✅ Conclusion

- The model **struggled to generalize** well, especially on visually ambiguous digits.
- It **over-relied** on class 2, possibly due to accidental alignment of weights from random initialization.
- Classes like 0, 1, 3, 4, 8, and 9 had **zero true positives**, indicating a **severe imbalance or lack of learned features**.
- Best path forward: add **training with backpropagation**, use a **larger dataset**, and implement **regularization** or **data augmentation**.