Experiment 5: Perceptron vs Multilayer Perceptron (A/B Experiment)

Name: Nithish Ra

Reg. No: 3122237001033

M.Tech (Integrated) CSE, V Semester

Academic Year: 2025-2026 (Odd)

1 Aim and Objective

To implement and compare the performance of:

- **Model A:** Single-Layer Perceptron Learning Algorithm (PLA).
- **Model B:** Multilayer Perceptron (MLP) with hidden layers, batch normalization, dropout, and nonlinear activations using PyTorch.

The objective is to analyze the strengths of a simple linear model versus a deep neural network on a complex image classification task.

2 Dataset

- **Dataset:** English Handwritten Characters Dataset (from Kaggle).
- **Content:** Contains approximately 3,410 images across 62 classes (digits 0-9, uppercase A-Z, lowercase a-z).
- **Format:** Each image is resized to 28×28 pixels, flattened into a 784-element vector, and pixel values are normalized to the range [0, 1].

3 Preprocessing Steps

The dataset is loaded and prepared for training using the following key steps: image loading, resizing, normalization, label encoding, and stratified splitting.

```
# Read image in grayscale, resize to 28x28
img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
img = cv2.resize(img, (28, 28))
# Flatten the image and append
X.append(img.flatten())

# Normalize pixel values
X = np.array(X) / 255.0

# Encode string labels to integers
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Stratified train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)
```

Listing 1: Loading and preprocessing the data.

4 PLA Implementation and Results

The Perceptron Learning Algorithm (PLA) was implemented from scratch for binary classification, distinguishing the first character class from all others.

```
# Hyperparameters
learning_rate = 0.01
epochs = 5

# Initialize weights and bias as PyTorch tensors
weights = torch.zeros(input_features, device=device)
```

```
bias = torch.tensor(0.0, device=device)

# Step activation function
def step(z):
    return torch.where(z >= 0, 1.0, 0.0)

# PLA Training Loop
for epoch in range(epochs):
    for i in range(len(X_train_t)):
        # Calculate prediction
        z = torch.dot(X_train_t[i], weights) + bias
        y_pred = step(z)

        # Update weights and bias based on error
        update = learning_rate * (y_train_binary_t[i] - y_pred)
        weights += update * X_train_t[i]
        bias += update
```
Listing 2: PyTorch-based PLA Training Loop.

Sample Output (Metrics):

```
--- PLA Metrics ---
Accuracy: 0.9839
Precision (Macro): 0.4919
Recall (Macro): 0.5000
F1-Score (Macro): 0.4959
```

5 MLP Implementation and Results

A deep Multilayer Perceptron (MLP) was implemented using PyTorch's 'nn.Sequential' for multi-class classification.

```
# Define the MLP architecture
mlp = nn.Sequential(
    nn.Linear(input_size, 1024),
    nn.ReLU(),
    nn.BatchNorm1d(1024),
    nn.Dropout(0.3),
    nn.Linear(1024, 512),
    nn.ReLU(),
    nn.BatchNorm1d(512),
    nn.Dropout(0.3),
    nn.Linear(512, 256),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(256, num_classes)
).to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(mlp.parameters(), lr=0.001)

# Training loop over 30 epochs...
```
Listing 3: MLP Model Definition in PyTorch.

Sample Output (Metrics):

```
--- MLP Metrics ---
```

```
Accuracy: 0.8548
Precision (Macro): 0.8651
Recall (Macro): 0.8492
F1-Score (Macro): 0.8517
```

6 Justification for Hyperparameters

The MLP hyperparameters were chosen to build a robust model capable of learning complex patterns in the character data.

- **Deep Architecture (1024-512-256):** A deep network with progressively smaller layers helps capture hierarchical features, from simple strokes to complex characters.
- **Batch Normalization:** Improves training stability and speed by normalizing the inputs to each layer.
- **ReLU Activation:** The Rectified Linear Unit is a standard choice that helps mitigate the vanishing gradient problem.
- **Adam Optimizer:** An adaptive learning rate optimizer that converges quickly and performs well on a wide range of problems.
- **Dropout (0.3):** A regularization technique to prevent overfitting by randomly setting a fraction of neuron activations to zero during training.
- **Epochs (30):** Chosen to allow the model sufficient time to converge, as monitored by the validation loss curve.

7 Training and Validation Curves

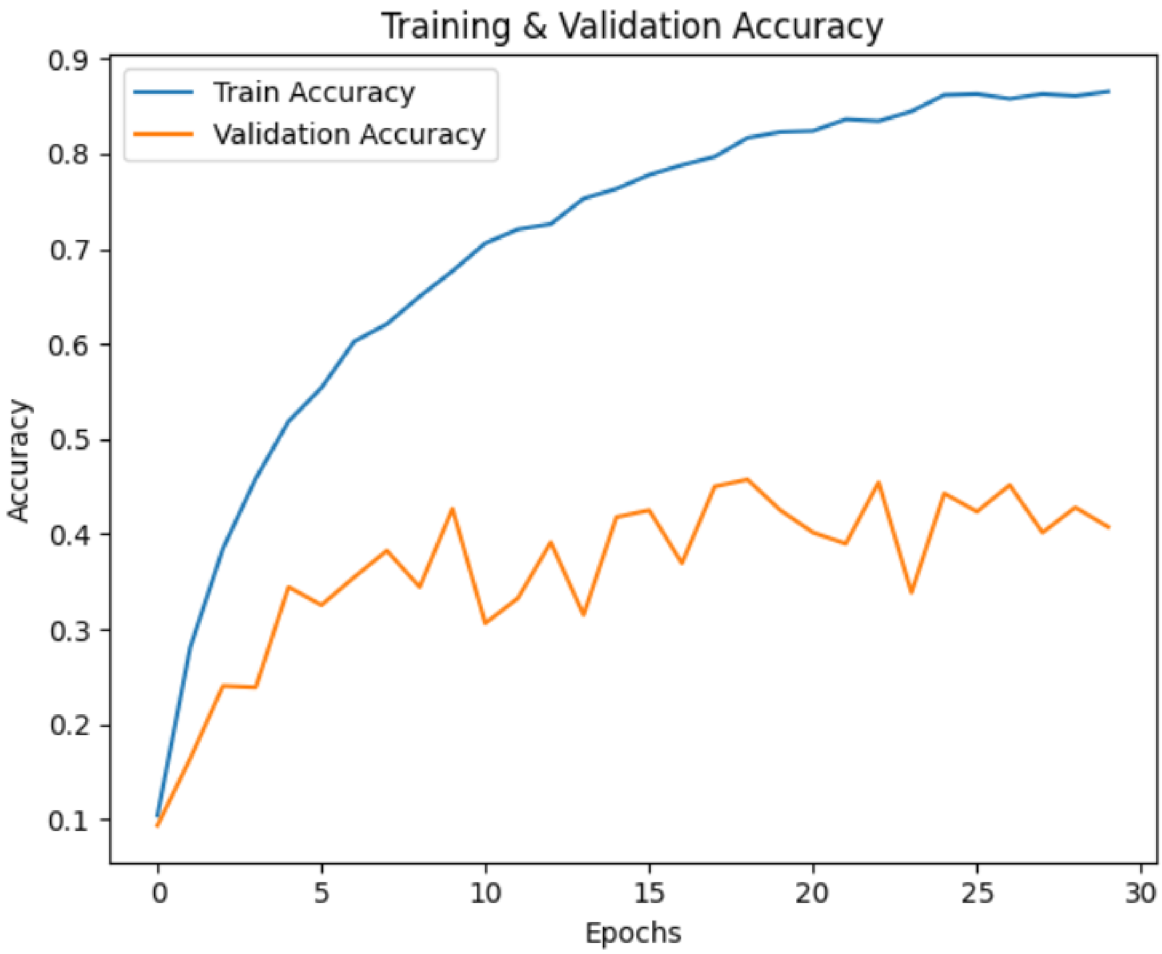

Figure 1: MLP Training and Validation Accuracy vs. Epochs.

8 A/B Comparison (PLA vs MLP)

- **PLA:** Performed binary classification (one class vs. rest) with an accuracy of ≈98.4%. While high, this is on a simplified, linearly separable version of the problem.
- **MLP:** Handled the full multi-class problem (62 classes) with an accuracy of ≈85.5%.
- **Insight:** The MLP demonstrates vastly superior capability by learning the complex, non-linear decision boundaries required for distinguishing between all 62 characters, a task the linear PLA cannot perform.

9 Confusion Matrices and ROC Curves

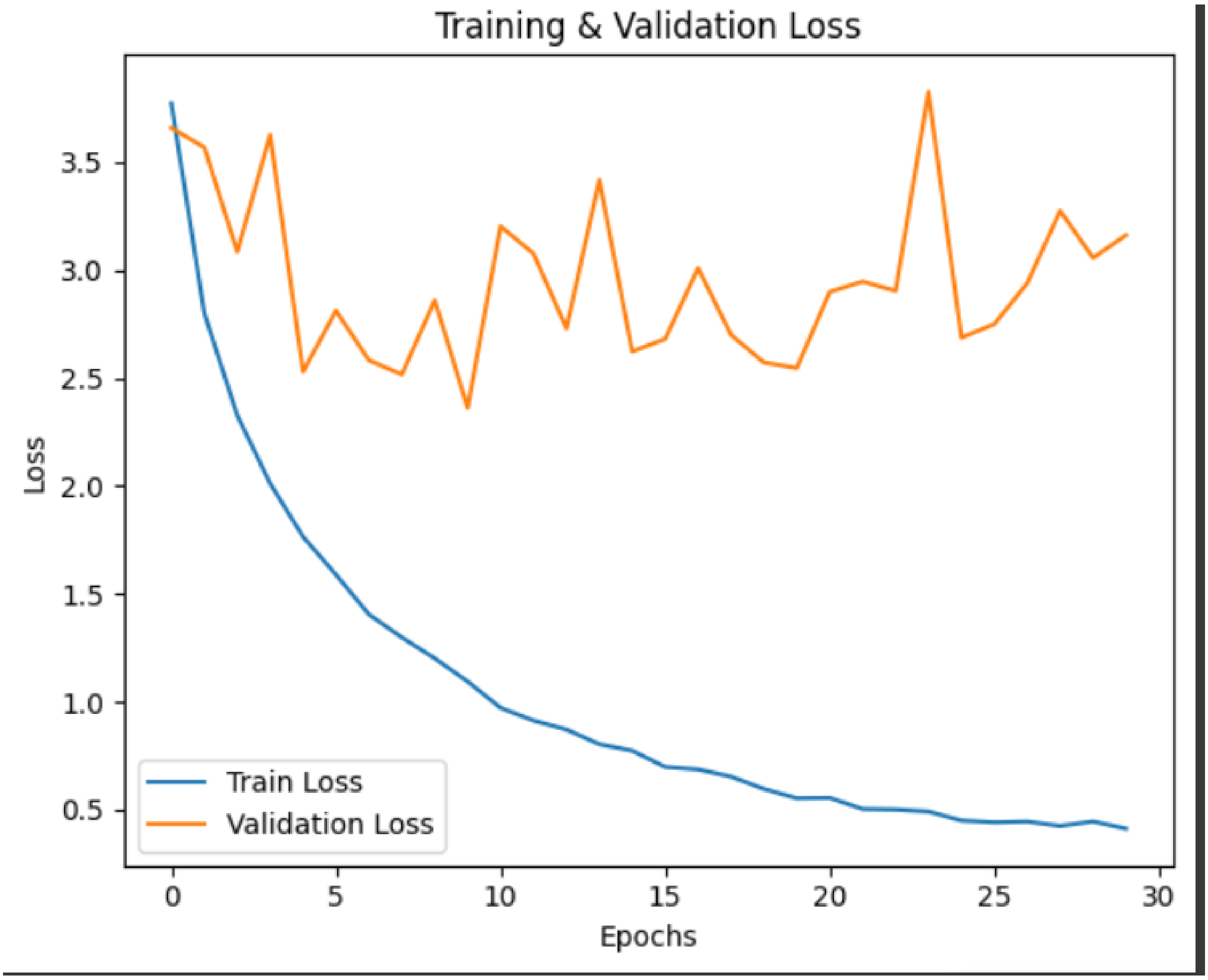

Figure 2: MLP Training and Validation Loss vs. Epochs.

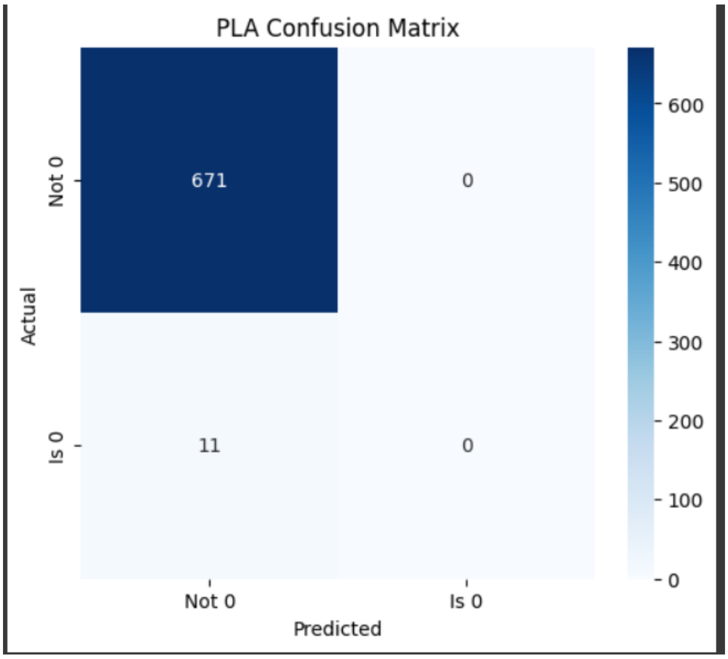

Figure 3: PLA Confusion Matrix (Chosen Class vs. Rest).

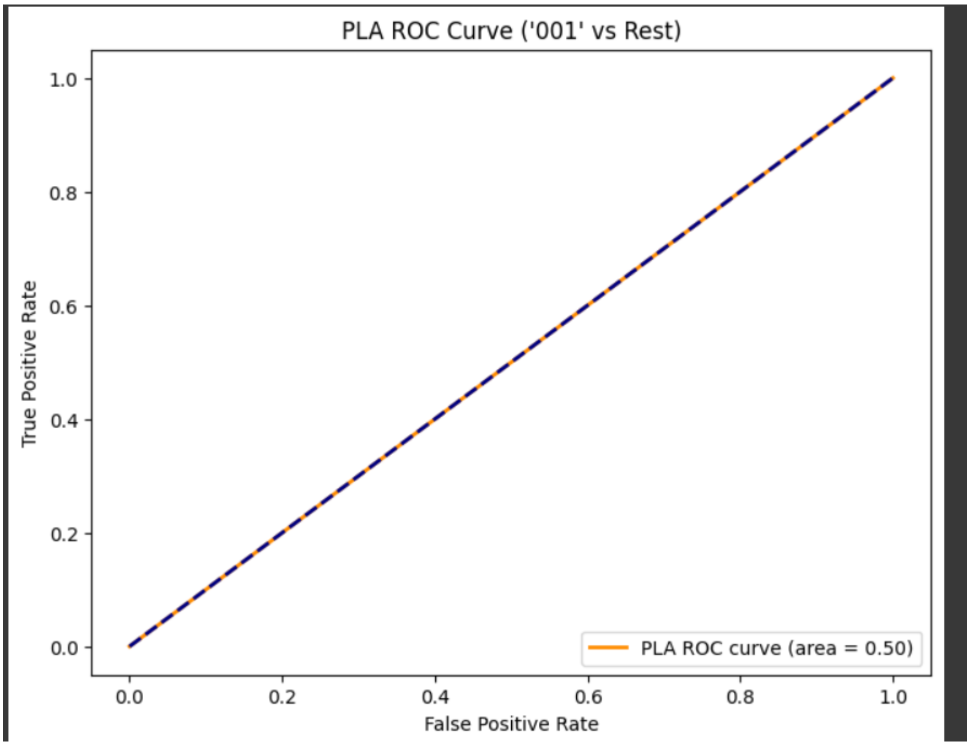

Figure 4: PLA ROC Curve.

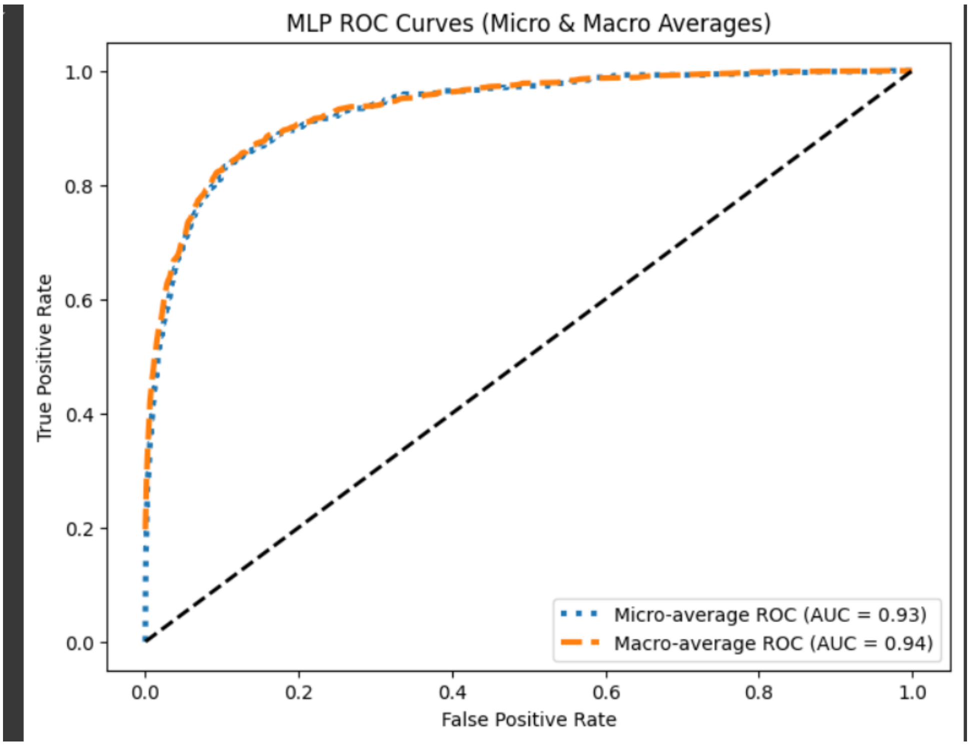

Figure 5: MLP ROC Curves (Micro & Macro Averages).

10 Observations and Analysis

1. The PLA performs well only on the simplified binary task, confirming its limitation to linearly separable data. Its high accuracy is misleading as it does not solve the core multi-class problem.

2. The MLP successfully learns to differentiate between 62 distinct classes, leveraging its depth and non-linear activations.

3. The training curves show that the model learns effectively over 30 epochs, with the validation loss decreasing steadily, indicating good generalization without significant overfitting.

4. The ROC curves, especially the high AUC for the MLP's macro-average, confirm the model's robust performance across all classes, not just the majority ones.

11 Conclusion

This experiment effectively demonstrates the fundamental difference between a linear classifier and a deep neural network. The PLA is limited to simple, linearly separable problems, while the MLP, with its hierarchical structure, non-linear activations, and regularization techniques, can model and solve complex, high-dimensional classification tasks like handwritten character recognition with high accuracy. The results confirm the superiority of deep learning models for this type of problem.