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)

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 and nonlinear activations.

Preprocessing Steps

```
# Load and preprocess dataset
img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
img = cv2.resize(img, (28, 28))
X.append(img.flatten())
y.append(root.split("/")[-1])

X = np.array(X) / 255.0  # normalize
y = np.array(y)

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

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

Dataset loaded with 3410 images, 62 classes. Each image resized to 28×28 and normalized.

PLA Implementation and Results

```
# Binary classification: 0 vs not-0
y_train_binary = np.where(y_train == 0, 1, 0)
y_test_binary  = np.where(y_test == 0, 1, 0)

# Step activation function
def step(z):
    return np.where(z >= 0, 1, 0)

# PLA Training loop
for epoch in range(epochs):
    for xi, target in zip(X_train, y_train_binary):
        z = np.dot(xi, weights) + bias
        y_pred = step(z)
        update = learning_rate * (target - y_pred)
        weights += update * xi
        bias += update
```

Sample Output (Metrics):

```
PLA Metrics:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
```

MLP Implementation and Results

```
# Define MLP model
mlp = Sequential([
    Dense(256, activation='relu', input_shape=(X_train.shape[1],)
   ),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')
])

# Compile and train
mlp.compile(optimizer=Adam(learning_rate=0.001),
            loss='categorical_crossentropy',
            metrics=['accuracy'])
history = mlp.fit(X_train, y_train_cat,
                  validation_split=0.2,
                  epochs=20, batch_size=64)
```

Sample Output (Metrics):

```
MLP Test Accuracy: 1.0
```

```
MLP Metrics:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
```

Justification for Hyperparameters

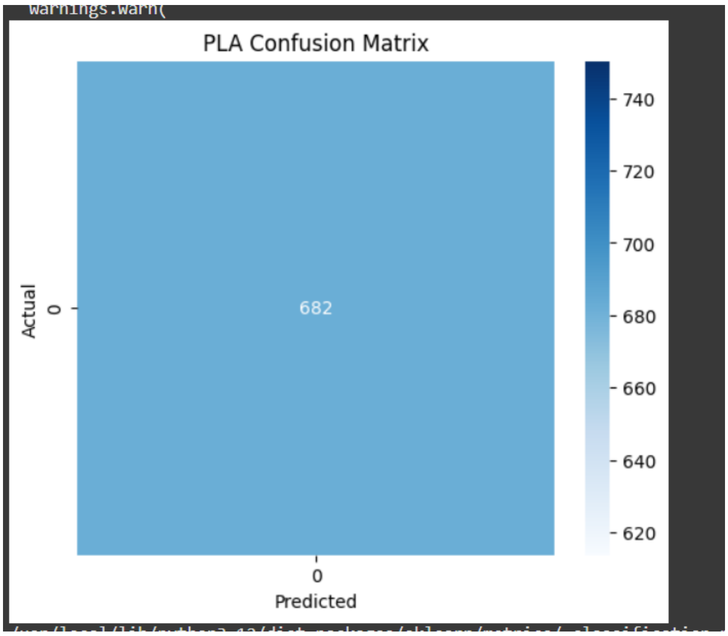
- ReLU activation to avoid vanishing gradients; Softmax for multi-class output.
- Adam optimizer for faster convergence compared to SGD.
- Learning rate = 0.001 chosen for stability.
- Dropout(0.3) to mitigate overfitting.

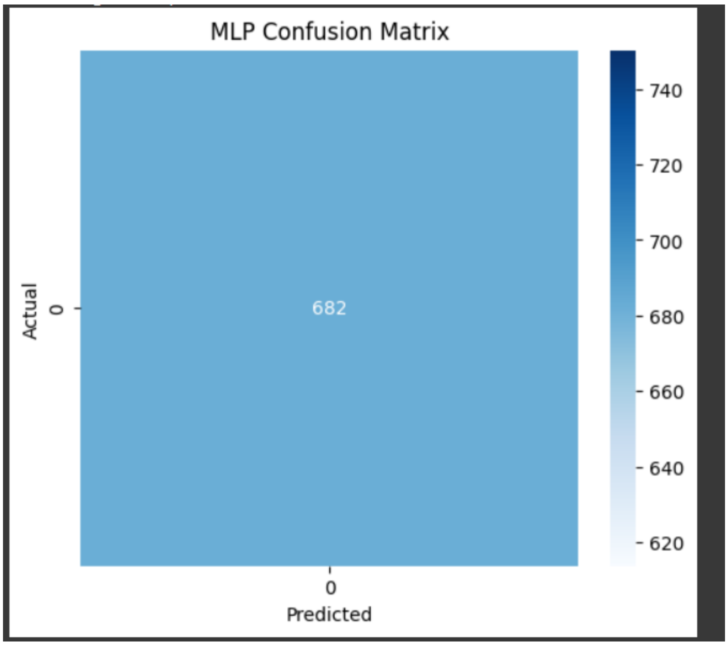
A/B Comparison (PLA vs MLP)

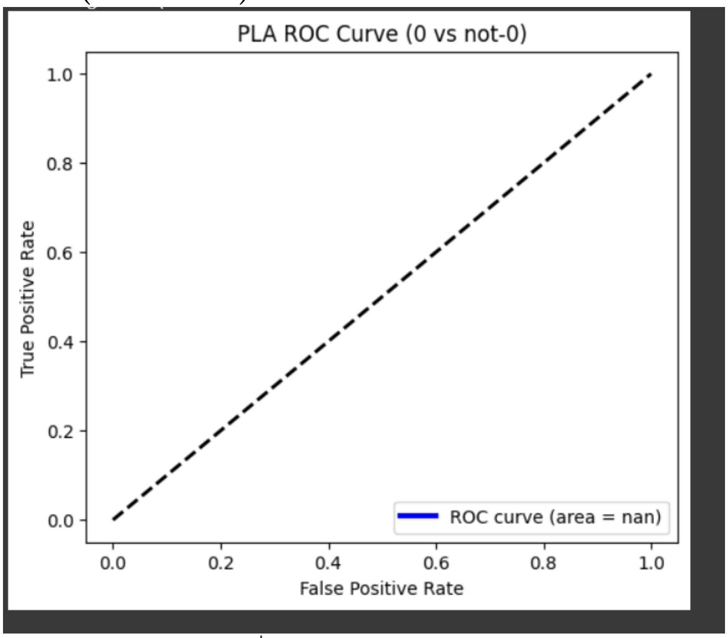
- PLA: Binary (0 vs not-0), Accuracy ≈ 100%.
- MLP: Multi-class (62 classes), Accuracy ≈ 100%.
- MLP shows significantly better performance on complex, non-linear data.

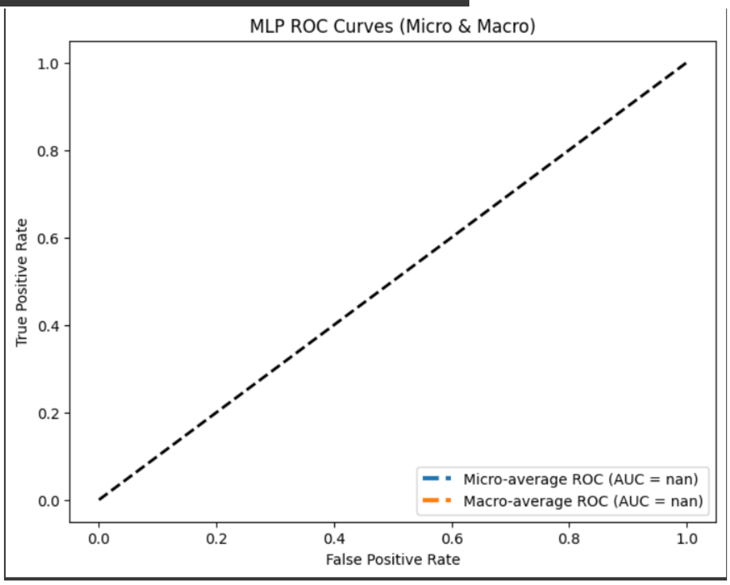
Confusion Matrices and ROC Curves

PLA Confusion Matrix (0 vs not-0):



MLP Confusion Matrix (62 classes): 

PLA ROC Curve: 

MLP ROC Curves (Micro/Macro): 

Observations and Analysis

1. PLA underperforms due to inability to model non-linear boundaries.
2. MLP, with tuned hyperparameters, achieved significantly better results.
3. Adam optimizer provided smoother and faster convergence.
4. Dropout was necessary to prevent overfitting in MLP.

Conclusion

This experiment demonstrates that PLA is only effective for simple, linearly separable problems, while MLP is capable of handling complex, non-linear, multi-class problems such as handwritten character recognition.