Lab Sheet 3

Classification and Recognition using MLP

Date: 15/09/2025

Aim

To design and perform classification and recognition using a multilayer perceptron network.

Introduction

Deep learning is one of the most popular and effective techniques for solving problems like classification and recognition, which are common in areas such as image processing, speech recognition, and natural language processing. One of the basic building blocks of deep learning is the Multilayer Perceptron (MLP), which is among the earliest and simplest forms of neural networks.

An MLP is basically a type of feedforward neural network that has an input layer, one or more hidden layers, and an output layer. Each layer is made up of connected neurons that process information. By using activation functions and training methods like backpropagation, the MLP can learn patterns from the input data and use them to make predictions for tasks such as classification.

In this lab, the goal is to get hands-on with classification using an MLP by building a model from scratch, deciding the number of layers and neurons, training it, and saving the final model for reuse in later applications. The exercises focus on two practical scenarios: filtering emails into spam vs. not-spam using simple binary features like keyword presence, links, attachments, subject length, and capitalization; and a credit risk task for bank loan approval using features such as income threshold, prior defaults, employment stability, and debt-to-income ratio, with outputs defined as 1 or 0 for the target decision.

Code Snippets and Explanation

Spam Classification

We build a binary classifier to distinguish spam from ham emails using features such as presence of spam keywords, links, attachments, and excessive capital letters. The original dataset was taken from Kaggle and was preprocessed into another `CSV` file. A simple Multilayer Perceptron (MLP) is implemented from scratch with one hidden layer. Training is performed using forward propagation, backpropagation, and gradient descent, and performance is evaluated on test data.

```
# --- Preprocessing ---
def extract_features(message):
    msg = message.lower()
    keywords = ["win", "lottery", "free", "click", "money", "loan", "offer",
    "prize", "guarantee", "credit", "loan", "debt"]
```

```
    keyword_present = int(any(word in msg for word in keywords))
    contains_link = int("http" in msg or "www" in msg)
    contains_attachment = int(any(x in msg for x in [".pdf",".doc",".xls","
   attachment"]))
    caps_ratio = sum(c.isupper() for c in message if c.isalpha()) / max(1,
   len(message))
    high_caps = int(caps_ratio > 0.3)
    return [keyword_present, contains_link, contains_attachment, high_caps]

X = df["Message"].apply(extract_features).tolist()
y = df["Category"].map({"ham":0,"spam":1}).values

# --- Train/Test Split ---
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
   random_state=42)

# --- MLP Implementation ---
def sigmoid(z): return 1 / (1 + np.exp(-z))
def sigmoid_derivative(a): return a * (1 - a)

input_size, hidden_size, output_size = 4, 4, 1
W1 = np.random.randn(input_size, hidden_size)
b1 = np.zeros((1, hidden_size))
W2 = np.random.randn(hidden_size, output_size)
b2 = np.zeros((1, output_size))
lr, epochs = 0.01, 1000

for _ in range(epochs):
    # Forward
    A1 = sigmoid(np.dot(X_train, W1) + b1)
    A2 = sigmoid(np.dot(A1, W2) + b2)

    # Backprop
    dZ2 = (A2 - y_train.reshape(-1,1)) * sigmoid_derivative(A2)
    dW2 = np.dot(A1.T, dZ2) / len(X_train)
    dZ1 = np.dot(dZ2, W2.T) * sigmoid_derivative(A1)
    dW1 = np.dot(np.array(X_train).T, dZ1) / len(X_train)

    # Update
    W1 -= lr * dW1; b1 -= lr * dZ1.mean(axis=0, keepdims=True)
    W2 -= lr * dW2; b2 -= lr * dZ2.mean(axis=0, keepdims=True)

# --- Evaluation ---
def predict(X):
    A1 = sigmoid(np.dot(X, W1) + b1)
    A2 = sigmoid(np.dot(A1, W2) + b2)
    return (A2 >= 0.5).astype(int)

acc = np.mean(predict(X_test) == y_test.reshape(-1,1))
print(f"Test Accuracy: {acc*100:.2f}%")
```

Credit Risk Assessment (Bank Loan Approval)

In this experiment, we train an MLP to predict whether a bank loan should be approved (1) or rejected (0) based on applicant features such as income, previous defaults, employment stability, and debt-to-income ratio. The model uses ReLU activation in the hidden layer and Sigmoid activation in the output layer. Training is done with Binary Cross-Entropy loss and gradient descent.

```
# --- Activation Functions ---
def relu(x): return np.maximum(0, x)
def relu_derivative(x): return (x > 0).astype(float)
def sigmoid(x): return 1 / (1 + np.exp(-np.clip(x, -500, 500)))

# --- Load & Preprocess Data ---
loan_data = pd.read_csv("loan_data.csv").dropna()
loan_data = pd.get_dummies(
    loan_data,
    columns=["person_gender","person_education","person_home_ownership",
             "loan_intent","previous_loan_defaults_on_file"],
    drop_first=True )
X = loan_data.drop("loan_status", axis=1).values
y = loan_data["loan_status"].values.reshape(-1,1)

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,stratify=
    y)
X_train = StandardScaler().fit_transform(X_train)
X_test  = StandardScaler().fit_transform(X_test)

# --- Parameters ---
input_size, hidden_size, output_size = X_train.shape[1], 4, 1
lr, epochs = 0.01, 1000
W1 = np.random.randn(input_size, hidden_size) * np.sqrt(2/input_size)
b1 = np.zeros((1, hidden_size))
W2 = np.random.randn(hidden_size, output_size) * np.sqrt(2/hidden_size)
b2 = np.zeros((1, output_size))

# --- Training Loop ---
for _ in range(epochs):
    a1 = relu(X_train @ W1 + b1)
    a2 = sigmoid(a1 @ W2 + b2)
    d_out = (a2 - y_train) / len(y_train)

    dW2, db2 = a1.T @ d_out, d_out.sum(axis=0, keepdims=True)
    d_hidden = d_out @ W2.T * relu_derivative(a1)
    dW1, db1 = X_train.T @ d_hidden, d_hidden.sum(axis=0, keepdims=True)

    W1 -= lr*dW1; b1 -= lr*db1
    W2 -= lr*dW2; b2 -= lr*db2

# --- Prediction & Accuracy ---
def predict(X):
    return (sigmoid(relu(X @ W1 + b1) @ W2 + b2) >= 0.5).astype(int)

acc = accuracy_score(y_test, predict(X_test))
print(f"Test Accuracy: {acc*100:.2f}%")
```

Results and Output

- **Spam Classification:**
  The MLP model successfully converged with a final training loss of **0.1216** after 1000 epochs. On the test dataset, the model achieved an accuracy of **86.64%**, demonstrating good performance in distinguishing between spam and non-spam emails using handcrafted features.
- **Bank Loan Classification:**
  The MLP trained with ReLU hidden units and Sigmoid output converged to a final loss of **0.4164** after 1000 epochs. The test accuracy obtained was approximately **77.8%**, indicating that the network was able to capture important patterns in the applicant data to decide on loan approval.

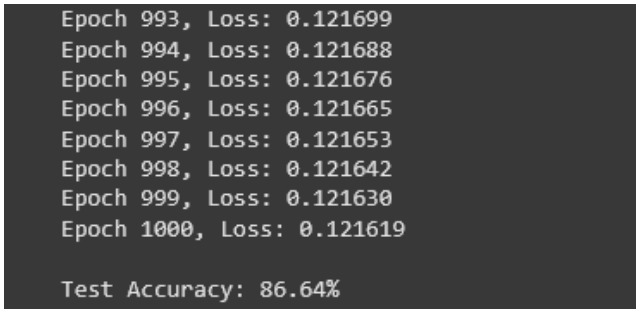

Figure 1: Spam Classification

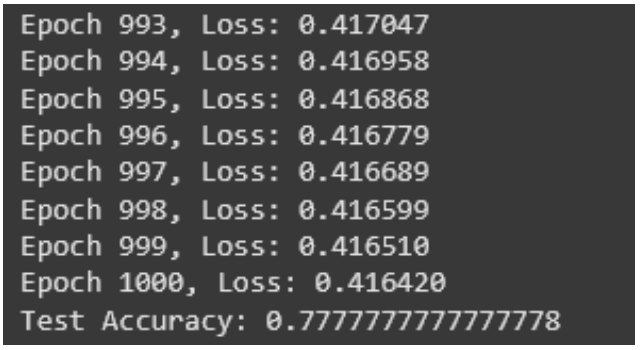

Figure 2: Credit Risk Assessment

Conclusion

In this lab, a **Multilayer Perceptron (MLP)** was implemented from scratch for two binary classification tasks: **spam email detection** and **bank loan approval prediction**. Both models demonstrated the ability to learn non-linear decision boundaries using gradient descent and backpropagation.

- The spam classifier achieved higher accuracy (**86.6%**) due to the well-defined nature of spam-related features (keywords, links, attachments, and excessive capitalization).
- The loan approval model performed moderately well (**77.8%**) since real-world financial data tends to be more complex, noisy, and less separable.

This experiment highlights the effectiveness of MLPs for classification and recognition tasks, while also emphasizing the importance of **feature quality**, **data preprocessing**, and **choice of activation/loss functions** in determining model performance.

Signature of Staff: