1.Logistic Regression

This notebook demonstrates Logistic Regression applied on the MNIST dataset (digits 0 and 1 only).

Import Required Libraries

```
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import numpy as np
from torch.utils.data import DataLoader
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from torch.utils.data import TensorDataset, DataLoader
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Load and Visualize MNIST Data

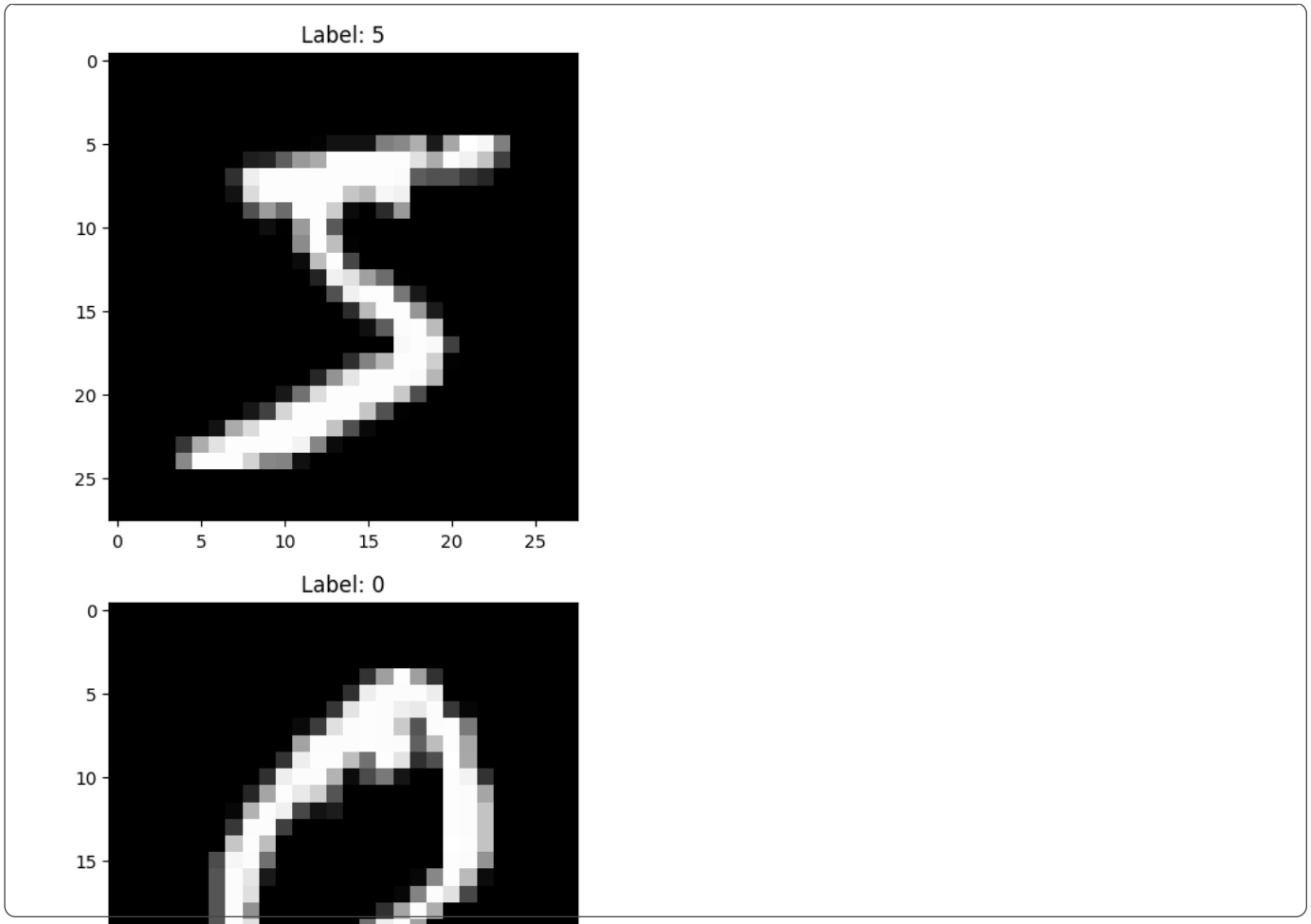
```
transform = transforms.Compose([
    transforms.ToTensor(),  # convert image to PyTorch tensor
    transforms.Normalize((0.0,), (1.0,))  # normalize to [0,1]
])

mnist_train = datasets.MNIST(root='data', train=True, transform=transform, download=True)
mnist_test = datasets.MNIST(root='data', train=False, transform=transform, download=True)
```

```
print(f"Number of training samples: {len(mnist_train)}")
```

```
Number of training samples: 60000
```

Sample Images

```
for i in range(5):
    image, label = mnist_train[i]
    plt.imshow(image.squeeze(), cmap='gray')
    plt.title(f"Label: {label}")
    plt.show()
```



Data Preprocessing

```
X = []
y = []

for img, label in mnist_train:
    if label in [0, 1]:
        X.append(img.view(-1))  # flatten 28x28 -> 784
        y.append(label)

X = torch.stack(X)
y = torch.tensor(y)
```

Split into Train, Validation, Test

```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, stratify=y_temp)
print(f"Train size: {X_train.shape[0]}, Val size: {X_val.shape[0]}, Test size: {X_test.shape[0]}")
```

```
Train size: 7599, Val size: 2533, Test size: 2533
```

Create DataLoaders

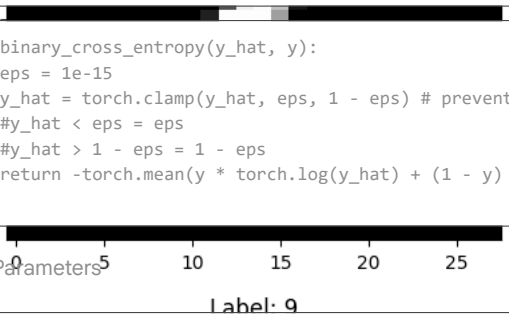
```
train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=64, shuffle=True)
val_loader = DataLoader(TensorDataset(X_val, y_val), batch_size=64)
test_loader = DataLoader(TensorDataset(X_test, y_test), batch_size=64)
```

Manual Logistic Regression Implementation (from scratch)

Helper Functions

```
def sigmoid(z):
    return 1 / (1 + torch.exp(-z))
```

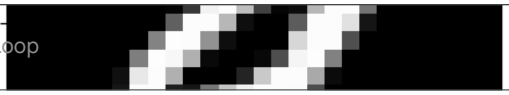
```
def binary_cross_entropy(y_hat, y):
    eps = 1e-15
    y_hat = torch.clamp(y_hat, eps, 1 - eps) # prevent log(0)
    #y_hat < eps = eps
    #y_hat > 1 - eps = 1 - eps
    return -torch.mean(y * torch.log(y_hat) + (1 - y) * torch.log(1 - y_hat))
```



Initialize Parameters

Label: 9

```
# Number of features = 784 pixels per image
n_features = 784

# Initialize weights and bias
W = torch.zeros((n_features, 1), requires_grad=True)
b = torch.zeros(1, requires_grad=True)
```



Training Loop

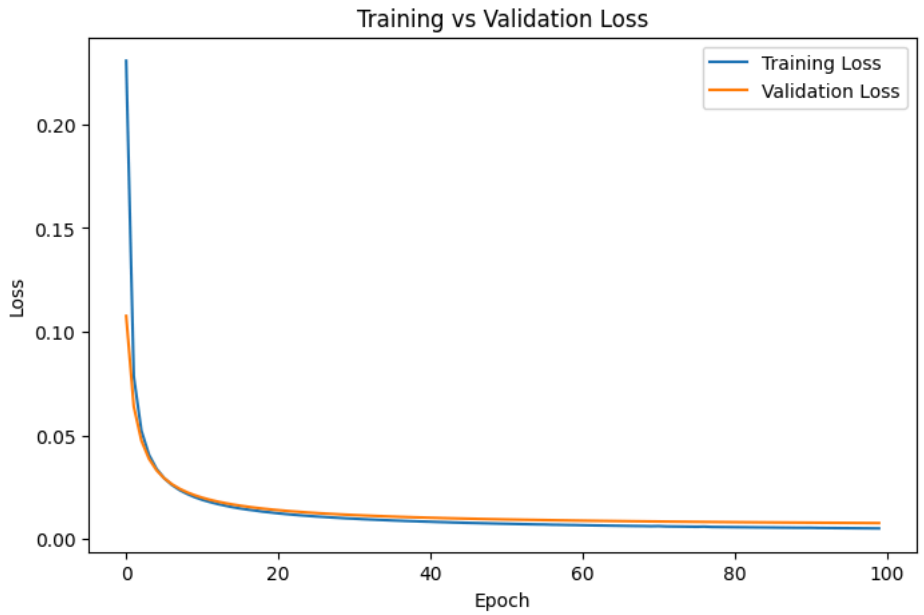
```
# --- Hyperparameters and trackers ---
learning_rate = 0.01
epochs = 100
train_losses = []
val_losses = []
val_accuracies = []
train_accuracies = []

# --- Training loop ---
for epoch in range(epochs):
    epoch_loss = 0.0
    correct_train = 0
    total_train = 0

    # ----- TRAINING PHASE -----
    for X_batch, y_batch in train_loader:
        # Forward pass
        y_pred = sigmoid(X_batch @ W + b)
        loss = binary_cross_entropy(y_pred, y_batch.unsqueeze(1).float())

        # Backward pass
        loss.backward()

        # Gradient update (manual SGD)
        with torch.no_grad():
            W -= learning_rate * W.grad
            b -= learning_rate * b.grad

        # Reset gradients
        W.grad.zero_()
        b.grad.zero_()

        # Accumulate batch loss
        epoch_loss += loss.item()
        # Compute training accuracy per batch
        preds = (y_pred > 0.5).int().squeeze()
        correct_train += (preds == y_batch).sum().item()
        total_train += y_batch.size(0)

    # Compute average training loss for this epoch
    avg_train_loss = epoch_loss / len(train_loader)
    train_losses.append(avg_train_loss)
    train_accuracies.append(correct_train / total_train)

    # ----- VALIDATION PHASE -----
    with torch.no_grad():
        val_loss_total = 0.0
        correct = 0
        total = 0
```

```
        for X_val_batch, y_val_batch in val_loader:
            y_val_pred = sigmoid(X_val_batch @ W + b)
            val_loss_total += binary_cross_entropy(y_val_pred, y_val_batch.unsqueeze(1).float()).item()

            # Compute accuracy
            y_val_pred_label = (y_val_pred > 0.5).int().squeeze()
            correct += (y_val_pred_label == y_val_batch).sum().item()
            total += y_val_batch.size(0)

        avg_val_loss = val_loss_total / len(val_loader)
        val_acc = correct / total

        val_losses.append(avg_val_loss)
        val_accuracies.append(val_acc)

    # Print progress for this epoch
    print(f"Epoch {epoch+1}/{epochs} | "
          f"Train Loss: {avg_train_loss:.4f} | "
          f"Val Loss: {avg_val_loss:.4f} | "
          f"Train Acc: {train_accuracies[-1]:.4f} | "
          f"Val Acc: {val_accuracies[-1]:.4f}")
```

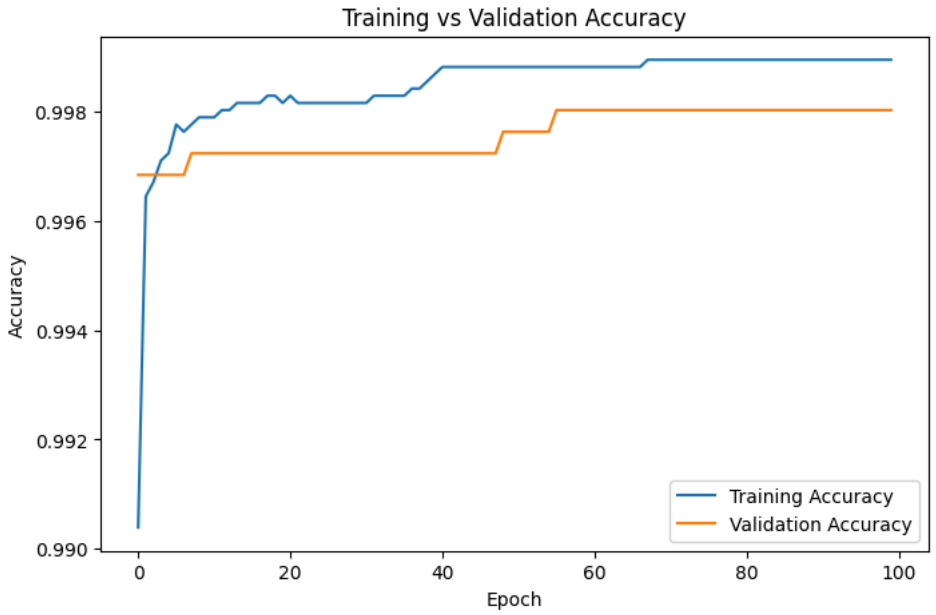
```
Epoch 1/100 | Train Loss: 0.2307 | Val Loss: 0.1076 | Train Acc: 0.9904 | Val Acc: 0.9968
Epoch 2/100 | Train Loss: 0.0784 | Val Loss: 0.0639 | Train Acc: 0.9964 | Val Acc: 0.9968
Epoch 3/100 | Train Loss: 0.0523 | Val Loss: 0.0475 | Train Acc: 0.9967 | Val Acc: 0.9968
Epoch 4/100 | Train Loss: 0.0406 | Val Loss: 0.0387 | Train Acc: 0.9971 | Val Acc: 0.9968
Epoch 5/100 | Train Loss: 0.0339 | Val Loss: 0.0332 | Train Acc: 0.9972 | Val Acc: 0.9968
Epoch 6/100 | Train Loss: 0.0294 | Val Loss: 0.0293 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 7/100 | Train Loss: 0.0262 | Val Loss: 0.0265 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 8/100 | Train Loss: 0.0237 | Val Loss: 0.0243 | Train Acc: 0.9978 | Val Acc: 0.9972
Epoch 9/100 | Train Loss: 0.0218 | Val Loss: 0.0226 | Train Acc: 0.9979 | Val Acc: 0.9972
Epoch 10/100 | Train Loss: 0.0202 | Val Loss: 0.0212 | Train Acc: 0.9979 | Val Acc: 0.9972
Epoch 11/100 | Train Loss: 0.0190 | Val Loss: 0.0200 | Train Acc: 0.9979 | Val Acc: 0.9972
Epoch 12/100 | Train Loss: 0.0179 | Val Loss: 0.0190 | Train Acc: 0.9980 | Val Acc: 0.9972
Epoch 13/100 | Train Loss: 0.0170 | Val Loss: 0.0181 | Train Acc: 0.9980 | Val Acc: 0.9972
Epoch 14/100 | Train Loss: 0.0162 | Val Loss: 0.0174 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 15/100 | Train Loss: 0.0154 | Val Loss: 0.0167 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 16/100 | Train Loss: 0.0148 | Val Loss: 0.0161 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 17/100 | Train Loss: 0.0142 | Val Loss: 0.0156 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 18/100 | Train Loss: 0.0138 | Val Loss: 0.0151 | Train Acc: 0.9983 | Val Acc: 0.9972
Epoch 19/100 | Train Loss: 0.0132 | Val Loss: 0.0147 | Train Acc: 0.9983 | Val Acc: 0.9972
Epoch 20/100 | Train Loss: 0.0129 | Val Loss: 0.0143 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 21/100 | Train Loss: 0.0124 | Val Loss: 0.0140 | Train Acc: 0.9983 | Val Acc: 0.9972
Epoch 22/100 | Train Loss: 0.0121 | Val Loss: 0.0136 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 23/100 | Train Loss: 0.0118 | Val Loss: 0.0133 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 24/100 | Train Loss: 0.0115 | Val Loss: 0.0131 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 25/100 | Train Loss: 0.0112 | Val Loss: 0.0128 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 26/100 | Train Loss: 0.0109 | Val Loss: 0.0126 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 27/100 | Train Loss: 0.0107 | Val Loss: 0.0124 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 28/100 | Train Loss: 0.0104 | Val Loss: 0.0122 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 29/100 | Train Loss: 0.0102 | Val Loss: 0.0120 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 30/100 | Train Loss: 0.0100 | Val Loss: 0.0118 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 31/100 | Train Loss: 0.0099 | Val Loss: 0.0116 | Train Acc: 0.9982 | Val Acc: 0.9972
Epoch 32/100 | Train Loss: 0.0097 | Val Loss: 0.0114 | Train Acc: 0.9983 | Val Acc: 0.9972
Epoch 33/100 | Train Loss: 0.0095 | Val Loss: 0.0113 | Train Acc: 0.9983 | Val Acc: 0.9972
Epoch 34/100 | Train Loss: 0.0093 | Val Loss: 0.0111 | Train Acc: 0.9983 | Val Acc: 0.9972
Epoch 35/100 | Train Loss: 0.0092 | Val Loss: 0.0110 | Train Acc: 0.9983 | Val Acc: 0.9972
Epoch 36/100 | Train Loss: 0.0090 | Val Loss: 0.0109 | Train Acc: 0.9983 | Val Acc: 0.9972
Epoch 37/100 | Train Loss: 0.0089 | Val Loss: 0.0107 | Train Acc: 0.9984 | Val Acc: 0.9972
Epoch 38/100 | Train Loss: 0.0087 | Val Loss: 0.0106 | Train Acc: 0.9984 | Val Acc: 0.9972
Epoch 39/100 | Train Loss: 0.0086 | Val Loss: 0.0105 | Train Acc: 0.9986 | Val Acc: 0.9972
Epoch 40/100 | Train Loss: 0.0085 | Val Loss: 0.0104 | Train Acc: 0.9987 | Val Acc: 0.9972
Epoch 41/100 | Train Loss: 0.0084 | Val Loss: 0.0103 | Train Acc: 0.9988 | Val Acc: 0.9972
Epoch 42/100 | Train Loss: 0.0083 | Val Loss: 0.0102 | Train Acc: 0.9988 | Val Acc: 0.9972
Epoch 43/100 | Train Loss: 0.0082 | Val Loss: 0.0101 | Train Acc: 0.9988 | Val Acc: 0.9972
Epoch 44/100 | Train Loss: 0.0080 | Val Loss: 0.0100 | Train Acc: 0.9988 | Val Acc: 0.9972
Epoch 45/100 | Train Loss: 0.0079 | Val Loss: 0.0099 | Train Acc: 0.9988 | Val Acc: 0.9972
Epoch 46/100 | Train Loss: 0.0078 | Val Loss: 0.0099 | Train Acc: 0.9988 | Val Acc: 0.9972
Epoch 47/100 | Train Loss: 0.0078 | Val Loss: 0.0098 | Train Acc: 0.9988 | Val Acc: 0.9972
Epoch 48/100 | Train Loss: 0.0077 | Val Loss: 0.0097 | Train Acc: 0.9988 | Val Acc: 0.9972
Epoch 49/100 | Train Loss: 0.0076 | Val Loss: 0.0096 | Train Acc: 0.9988 | Val Acc: 0.9976
Epoch 50/100 | Train Loss: 0.0075 | Val Loss: 0.0096 | Train Acc: 0.9988 | Val Acc: 0.9976
Epoch 51/100 | Train Loss: 0.0074 | Val Loss: 0.0095 | Train Acc: 0.9988 | Val Acc: 0.9976
Epoch 52/100 | Train Loss: 0.0073 | Val Loss: 0.0094 | Train Acc: 0.9988 | Val Acc: 0.9976
Epoch 53/100 | Train Loss: 0.0072 | Val Loss: 0.0094 | Train Acc: 0.9988 | Val Acc: 0.9976
Epoch 54/100 | Train Loss: 0.0072 | Val Loss: 0.0093 | Train Acc: 0.9988 | Val Acc: 0.9976
Epoch 55/100 | Train Loss: 0.0071 | Val Loss: 0.0092 | Train Acc: 0.9988 | Val Acc: 0.9976
Epoch 56/100 | Train Loss: 0.0071 | Val Loss: 0.0092 | Train Acc: 0.9988 | Val Acc: 0.9980
Epoch 57/100 | Train Loss: 0.0070 | Val Loss: 0.0091 | Train Acc: 0.9988 | Val Acc: 0.9980
Epoch 58/100 | Train Loss: 0.0069 | Val Loss: 0.0091 | Train Acc: 0.9988 | Val Acc: 0.9980
```

Loss and Accuracy Curves

```
plt.figure(figsize=(8,5))
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training vs Validation Loss')
plt.legend()
plt.show()
```



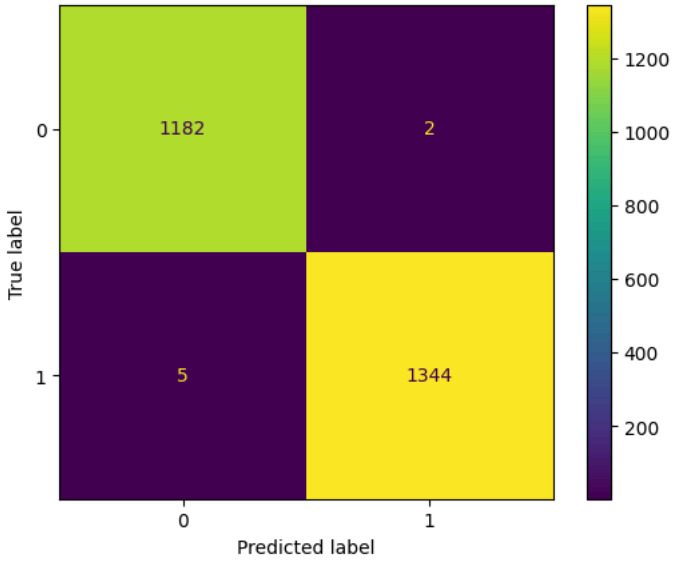

```
plt.figure(figsize=(8,5))
plt.plot(train_accuracies, label='Training Accuracy')
plt.plot(val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training vs Validation Accuracy')
plt.legend()
plt.show()
```



Evaluate on Test Set

```
with torch.no_grad():
    y_test_pred = sigmoid(X_test @ W + b)
    y_test_pred_label = (y_test_pred > 0.5).int().squeeze()

test_acc = (y_test_pred_label == y_test).float().mean()
print(f"Test Accuracy: {test_acc:.4f}")

cm = confusion_matrix(y_test, y_test_pred_label)
ConfusionMatrixDisplay(cm).plot()
plt.show()
```

```
Test Accuracy: 0.9972
```



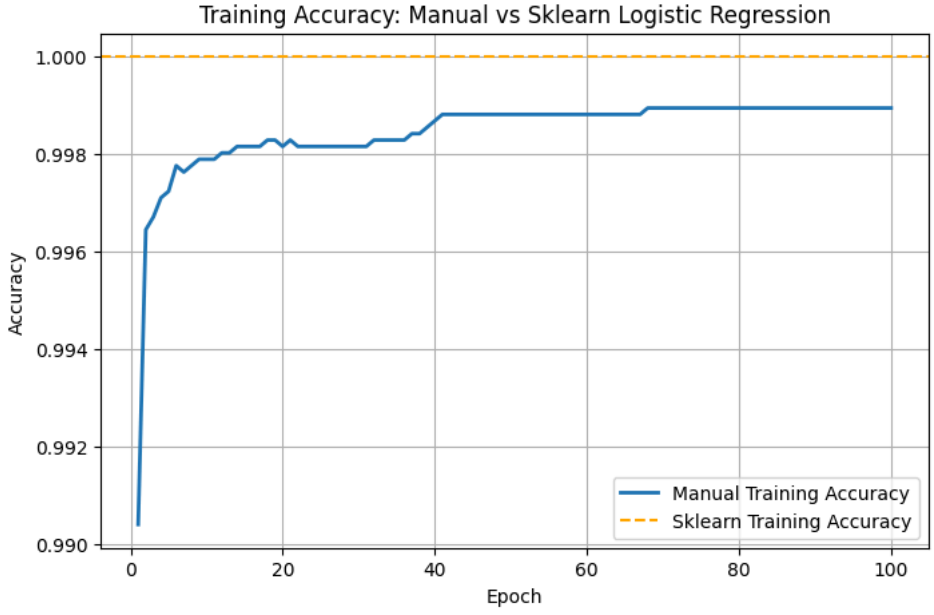
Comparison with Sklearn Logistic Regression

```
model_sklearn = LogisticRegression(max_iter=100, solver='liblinear')
model_sklearn.fit(X_train, y_train)

train_acc_sklearn = model_sklearn.score(X_train, y_train)
val_acc_sklearn = model_sklearn.score(X_val, y_val)
test_acc_sklearn = model_sklearn.score(X_test, y_test)

print(f"Sklearn Logistic Regression - Training Accuracy: {train_acc_sklearn:.4f}")
print(f"Sklearn Logistic Regression - Validation Accuracy: {val_acc_sklearn:.4f}")
print(f"Sklearn Logistic Regression - Test Accuracy: {test_acc_sklearn:.4f}")
```

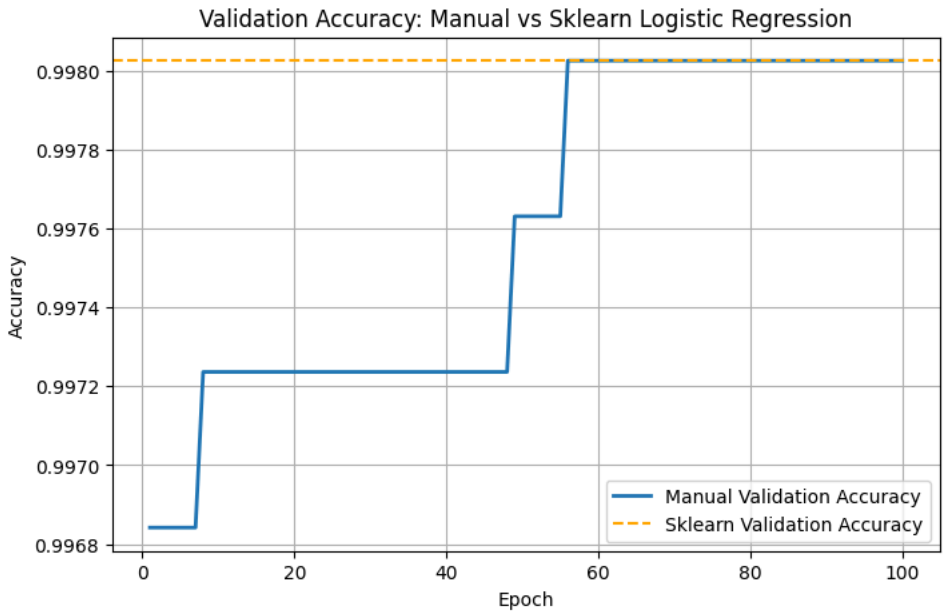
```
Sklearn Logistic Regression - Training Accuracy: 1.0000
Sklearn Logistic Regression - Validation Accuracy: 0.9980
Sklearn Logistic Regression - Test Accuracy: 0.9976
```

```
manual_final_train_acc = train_accuracies[-1]
manual_final_val_acc = val_accuracies[-1]
```

```
plt.figure(figsize=(8,5))
plt.plot(range(1, len(train_accuracies)+1), train_accuracies, label='Manual Training Accuracy', linewidth=2)
plt.axhline(y=train_acc_sklearn, color='orange', linestyle='--', label='Sklearn Training Accuracy')

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training Accuracy: Manual vs Sklearn Logistic Regression')
plt.legend()
plt.grid(True)
plt.show()
```



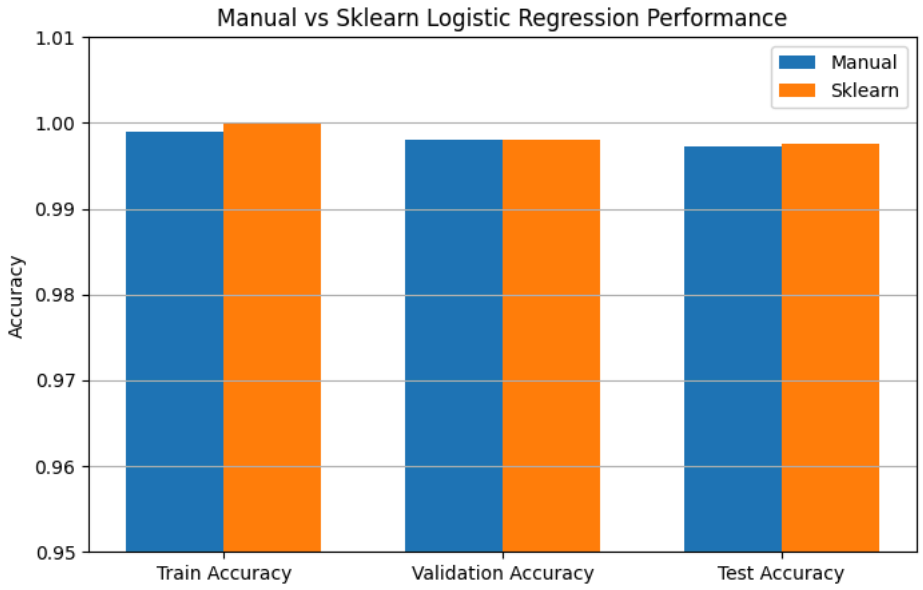

```
plt.figure(figsize=(8,5))
plt.plot(range(1, len(val_accuracies)+1), val_accuracies, label='Manual Validation Accuracy', linewidth=2)
plt.axhline(y=val_acc_sklearn, color='orange', linestyle='--', label='Sklearn Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Validation Accuracy: Manual vs Sklearn Logistic Regression')
plt.legend()
plt.grid(True)
plt.show()
```



Manual vs Sklearn Accuracy Comparison

```
labels = ['Train Accuracy', 'Validation Accuracy', 'Test Accuracy']
manual_values = [
    manual_final_train_acc,
    manual_final_val_acc,
    (y_test_pred_label == y_test).float().mean().item()
]
sklearn_values = [
    train_acc_sklearn,
    val_acc_sklearn,
    model_sklearn.score(X_test, y_test)
]
```

```
x = np.arange(len(labels))
width = 0.35

plt.figure(figsize=(8,5))
plt.bar(x - width/2, manual_values, width, label='Manual')
plt.bar(x + width/2, sklearn_values, width, label='Sklearn')
plt.ylabel('Accuracy')
plt.title('Manual vs Sklearn Logistic Regression Performance')
plt.xticks(x, labels)
plt.ylim(0.95, 1.01)
plt.legend()
plt.grid(axis='y')
plt.show()
```



Logistic Regression Using PyTorch Layers

```
transform = transforms.Compose([transforms.ToTensor()])
mnist_train = datasets.MNIST(root='data', train=True, transform=transform, download=True)
```

```
# Filter digits 0 and 1
X = []
y = []
for img, label in mnist_train:
    if label in [0, 1]:
        X.append(img.view(-1))
        y.append(label)

X = torch.stack(X)
y = torch.tensor(y, dtype=torch.float32)
```

```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, stratify=y_temp)
print(f"Train set size: {X_train.shape[0]}")
print(f"Validation set size: {X_val.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")
```

```
Train set size: 7599
Validation set size: 2533
Test set size: 2533
```

```
train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=64, shuffle=True)
val_loader   = DataLoader(TensorDataset(X_val, y_val), batch_size=64)
test_loader  = DataLoader(TensorDataset(X_test, y_test), batch_size=64)
```

```
model = nn.Linear(784, 1)
```

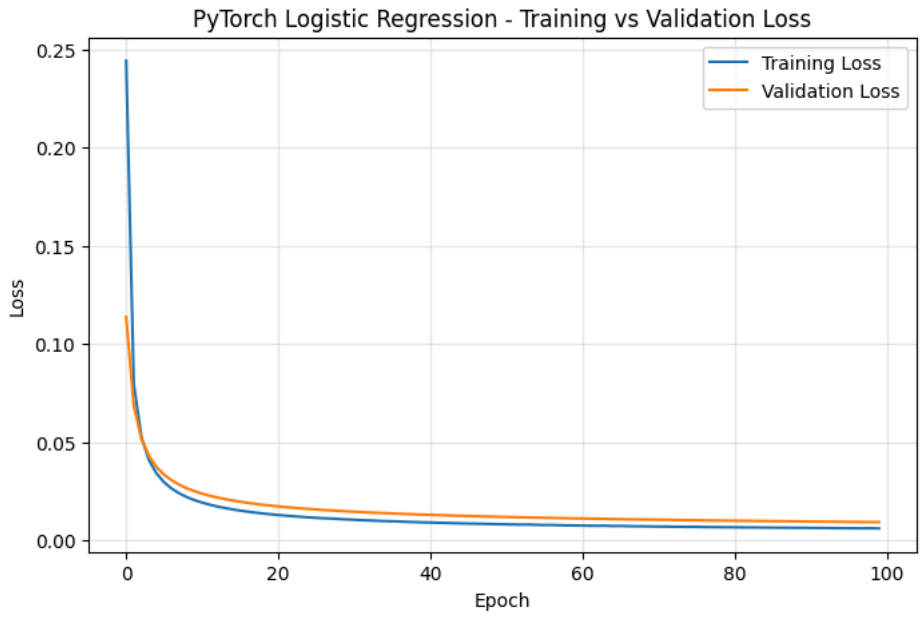
```
criterion = nn.BCEWithLogitsLoss()       # combines sigmoid + BCE for stability
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

```
epochs = 100
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []

for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    correct_train = 0
    total_train = 0

    # TRAIN
    for X_batch, y_batch in train_loader:
        optimizer.zero_grad()
        outputs = model(X_batch).squeeze()
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        preds = (torch.sigmoid(outputs) > 0.5).int()
        correct_train += (preds == y_batch.int()).sum().item()
        total_train += y_batch.size(0)

    avg_train_loss = running_loss / len(train_loader)
    train_acc = correct_train / total_train
    train_losses.append(avg_train_loss)
    train_accuracies.append(train_acc)

    # VALIDATION
    model.eval()
    val_loss = 0.0
    correct_val = 0
    total_val = 0
    with torch.no_grad():
        for X_val_batch, y_val_batch in val_loader:
            outputs_val = model(X_val_batch).squeeze()
            loss = criterion(outputs_val, y_val_batch)
            val_loss += loss.item()

            preds_val = (torch.sigmoid(outputs_val) > 0.5).int()
            correct_val += (preds_val == y_val_batch.int()).sum().item()
            total_val += y_val_batch.size(0)

    avg_val_loss = val_loss / len(val_loader)
    val_acc = correct_val / total_val
    val_losses.append(avg_val_loss)
    val_accuracies.append(val_acc)

    print(f"Epoch {epoch+1}/{epochs} | "
          f"Train Loss: {avg_train_loss:.4f} | Val Loss: {avg_val_loss:.4f} | "
          f"Train Acc: {train_acc:.4f} | Val Acc: {val_acc:.4f}")
```

```
Epoch 1/100 | Train Loss: 0.2443 | Val Loss: 0.1138 | Train Acc: 0.9697 | Val Acc: 0.9941
Epoch 2/100 | Train Loss: 0.0798 | Val Loss: 0.0686 | Train Acc: 0.9966 | Val Acc: 0.9949
Epoch 3/100 | Train Loss: 0.0528 | Val Loss: 0.0519 | Train Acc: 0.9967 | Val Acc: 0.9949
Epoch 4/100 | Train Loss: 0.0409 | Val Loss: 0.0430 | Train Acc: 0.9968 | Val Acc: 0.9949
Epoch 5/100 | Train Loss: 0.0341 | Val Loss: 0.0374 | Train Acc: 0.9971 | Val Acc: 0.9949
Epoch 6/100 | Train Loss: 0.0296 | Val Loss: 0.0335 | Train Acc: 0.9971 | Val Acc: 0.9953
Epoch 7/100 | Train Loss: 0.0264 | Val Loss: 0.0306 | Train Acc: 0.9971 | Val Acc: 0.9957
Epoch 8/100 | Train Loss: 0.0240 | Val Loss: 0.0284 | Train Acc: 0.9971 | Val Acc: 0.9961
Epoch 9/100 | Train Loss: 0.0221 | Val Loss: 0.0266 | Train Acc: 0.9972 | Val Acc: 0.9961
Epoch 10/100 | Train Loss: 0.0205 | Val Loss: 0.0251 | Train Acc: 0.9972 | Val Acc: 0.9961
Epoch 11/100 | Train Loss: 0.0193 | Val Loss: 0.0239 | Train Acc: 0.9972 | Val Acc: 0.9961
Epoch 12/100 | Train Loss: 0.0182 | Val Loss: 0.0228 | Train Acc: 0.9974 | Val Acc: 0.9961
Epoch 13/100 | Train Loss: 0.0173 | Val Loss: 0.0219 | Train Acc: 0.9974 | Val Acc: 0.9964
Epoch 14/100 | Train Loss: 0.0166 | Val Loss: 0.0211 | Train Acc: 0.9975 | Val Acc: 0.9964
Epoch 15/100 | Train Loss: 0.0158 | Val Loss: 0.0204 | Train Acc: 0.9975 | Val Acc: 0.9964
Epoch 16/100 | Train Loss: 0.0152 | Val Loss: 0.0198 | Train Acc: 0.9975 | Val Acc: 0.9968
Epoch 17/100 | Train Loss: 0.0147 | Val Loss: 0.0192 | Train Acc: 0.9975 | Val Acc: 0.9968
Epoch 18/100 | Train Loss: 0.0142 | Val Loss: 0.0187 | Train Acc: 0.9975 | Val Acc: 0.9968
Epoch 19/100 | Train Loss: 0.0138 | Val Loss: 0.0182 | Train Acc: 0.9975 | Val Acc: 0.9968
Epoch 20/100 | Train Loss: 0.0134 | Val Loss: 0.0178 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 21/100 | Train Loss: 0.0130 | Val Loss: 0.0174 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 22/100 | Train Loss: 0.0127 | Val Loss: 0.0170 | Train Acc: 0.9978 | Val Acc: 0.9968
```

```
Epoch 23/100 | Train Loss: 0.0123 | Val Loss: 0.0167 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 24/100 | Train Loss: 0.0121 | Val Loss: 0.0164 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 25/100 | Train Loss: 0.0118 | Val Loss: 0.0161 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 26/100 | Train Loss: 0.0115 | Val Loss: 0.0158 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 27/100 | Train Loss: 0.0113 | Val Loss: 0.0156 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 28/100 | Train Loss: 0.0112 | Val Loss: 0.0153 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 29/100 | Train Loss: 0.0109 | Val Loss: 0.0151 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 30/100 | Train Loss: 0.0107 | Val Loss: 0.0149 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 31/100 | Train Loss: 0.0106 | Val Loss: 0.0147 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 32/100 | Train Loss: 0.0104 | Val Loss: 0.0145 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 33/100 | Train Loss: 0.0103 | Val Loss: 0.0143 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 34/100 | Train Loss: 0.0100 | Val Loss: 0.0141 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 35/100 | Train Loss: 0.0099 | Val Loss: 0.0139 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 36/100 | Train Loss: 0.0098 | Val Loss: 0.0138 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 37/100 | Train Loss: 0.0097 | Val Loss: 0.0136 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 38/100 | Train Loss: 0.0095 | Val Loss: 0.0135 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 39/100 | Train Loss: 0.0094 | Val Loss: 0.0133 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 40/100 | Train Loss: 0.0093 | Val Loss: 0.0132 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 41/100 | Train Loss: 0.0092 | Val Loss: 0.0131 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 42/100 | Train Loss: 0.0090 | Val Loss: 0.0129 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 43/100 | Train Loss: 0.0090 | Val Loss: 0.0128 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 44/100 | Train Loss: 0.0088 | Val Loss: 0.0127 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 45/100 | Train Loss: 0.0087 | Val Loss: 0.0126 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 46/100 | Train Loss: 0.0087 | Val Loss: 0.0125 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 47/100 | Train Loss: 0.0086 | Val Loss: 0.0124 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 48/100 | Train Loss: 0.0085 | Val Loss: 0.0123 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 49/100 | Train Loss: 0.0084 | Val Loss: 0.0122 | Train Acc: 0.9976 | Val Acc: 0.9968
Epoch 50/100 | Train Loss: 0.0083 | Val Loss: 0.0121 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 51/100 | Train Loss: 0.0082 | Val Loss: 0.0120 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 52/100 | Train Loss: 0.0082 | Val Loss: 0.0119 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 53/100 | Train Loss: 0.0081 | Val Loss: 0.0118 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 54/100 | Train Loss: 0.0082 | Val Loss: 0.0117 | Train Acc: 0.9978 | Val Acc: 0.9968
Epoch 55/100 | Train Loss: 0.0080 | Val Loss: 0.0117 | Train Acc: 0.9979 | Val Acc: 0.9968
Epoch 56/100 | Train Loss: 0.0079 | Val Loss: 0.0116 | Train Acc: 0.9979 | Val Acc: 0.9968
Epoch 57/100 | Train Loss: 0.0079 | Val Loss: 0.0115 | Train Acc: 0.9980 | Val Acc: 0.9968
```

```
plt.figure(figsize=(8,5))
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('PyTorch Logistic Regression - Training vs Validation Loss')
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```




```
plt.figure(figsize=(8,5))
plt.plot(train_accuracies, label='Training Accuracy')
plt.plot(val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('PyTorch Logistic Regression - Training vs Validation Accuracy')
plt.legend()
```