```
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.init as init
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
import pandas as pd
import time
from torch.utils.data import random_split, DataLoader
import torchvision.transforms as transforms
import torchvision.datasets as datasets
```

Logistic Regression

```
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)
```

```
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)
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
```

```
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)


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))


# 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)
```

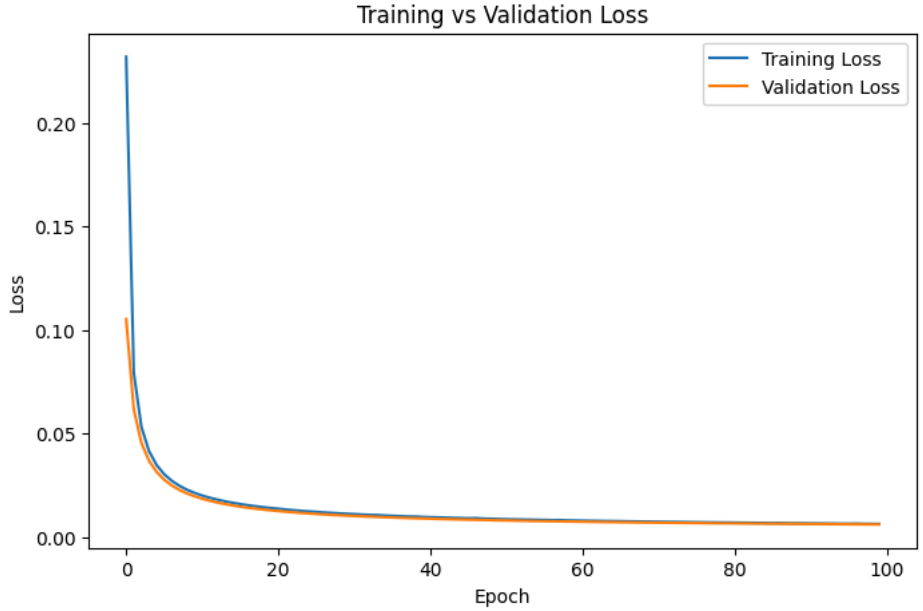
```
# --- Hyperparameters and trackers ---
learning_rate = 0.01
epochs = 100
train_losses = []
val_losses = []
val_accuracies = []
train_accuracies = []

# --- Training loop ---
start_time = time.time()
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}")
```

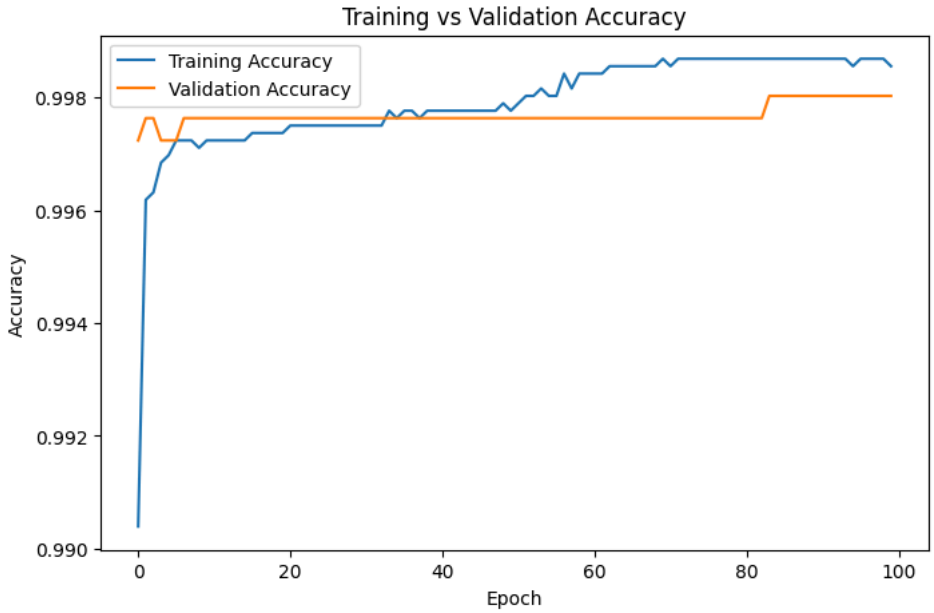
```
end_time = time.time()
lr_run_time = end_time - start_time
```

```
Epoch 1/100 | Train Loss: 0.2320 | Val Loss: 0.1053 | Train Acc: 0.9904 | Val Acc: 0.9972
Epoch 2/100 | Train Loss: 0.0795 | Val Loss: 0.0618 | Train Acc: 0.9962 | Val Acc: 0.9976
Epoch 3/100 | Train Loss: 0.0532 | Val Loss: 0.0455 | Train Acc: 0.9963 | Val Acc: 0.9976
Epoch 4/100 | Train Loss: 0.0415 | Val Loss: 0.0369 | Train Acc: 0.9968 | Val Acc: 0.9972
Epoch 5/100 | Train Loss: 0.0348 | Val Loss: 0.0315 | Train Acc: 0.9970 | Val Acc: 0.9972
Epoch 6/100 | Train Loss: 0.0303 | Val Loss: 0.0277 | Train Acc: 0.9972 | Val Acc: 0.9972
Epoch 7/100 | Train Loss: 0.0272 | Val Loss: 0.0249 | Train Acc: 0.9972 | Val Acc: 0.9976
Epoch 8/100 | Train Loss: 0.0248 | Val Loss: 0.0228 | Train Acc: 0.9972 | Val Acc: 0.9976
Epoch 9/100 | Train Loss: 0.0228 | Val Loss: 0.0211 | Train Acc: 0.9971 | Val Acc: 0.9976
Epoch 10/100 | Train Loss: 0.0213 | Val Loss: 0.0197 | Train Acc: 0.9972 | Val Acc: 0.9976
Epoch 11/100 | Train Loss: 0.0200 | Val Loss: 0.0185 | Train Acc: 0.9972 | Val Acc: 0.9976
Epoch 12/100 | Train Loss: 0.0190 | Val Loss: 0.0175 | Train Acc: 0.9972 | Val Acc: 0.9976
Epoch 13/100 | Train Loss: 0.0181 | Val Loss: 0.0167 | Train Acc: 0.9972 | Val Acc: 0.9976
Epoch 14/100 | Train Loss: 0.0172 | Val Loss: 0.0159 | Train Acc: 0.9972 | Val Acc: 0.9976
Epoch 15/100 | Train Loss: 0.0165 | Val Loss: 0.0153 | Train Acc: 0.9972 | Val Acc: 0.9976
Epoch 16/100 | Train Loss: 0.0159 | Val Loss: 0.0147 | Train Acc: 0.9974 | Val Acc: 0.9976
Epoch 17/100 | Train Loss: 0.0154 | Val Loss: 0.0142 | Train Acc: 0.9974 | Val Acc: 0.9976
Epoch 18/100 | Train Loss: 0.0149 | Val Loss: 0.0137 | Train Acc: 0.9974 | Val Acc: 0.9976
Epoch 19/100 | Train Loss: 0.0144 | Val Loss: 0.0133 | Train Acc: 0.9974 | Val Acc: 0.9976
Epoch 20/100 | Train Loss: 0.0140 | Val Loss: 0.0129 | Train Acc: 0.9974 | Val Acc: 0.9976
Epoch 21/100 | Train Loss: 0.0137 | Val Loss: 0.0126 | Train Acc: 0.9975 | Val Acc: 0.9976
Epoch 22/100 | Train Loss: 0.0133 | Val Loss: 0.0122 | Train Acc: 0.9975 | Val Acc: 0.9976
Epoch 23/100 | Train Loss: 0.0129 | Val Loss: 0.0119 | Train Acc: 0.9975 | Val Acc: 0.9976
Epoch 24/100 | Train Loss: 0.0127 | Val Loss: 0.0117 | Train Acc: 0.9975 | Val Acc: 0.9976
Epoch 25/100 | Train Loss: 0.0124 | Val Loss: 0.0114 | Train Acc: 0.9975 | Val Acc: 0.9976
Epoch 26/100 | Train Loss: 0.0122 | Val Loss: 0.0112 | Train Acc: 0.9975 | Val Acc: 0.9976
Epoch 27/100 | Train Loss: 0.0119 | Val Loss: 0.0109 | Train Acc: 0.9975 | Val Acc: 0.9976
Epoch 28/100 | Train Loss: 0.0116 | Val Loss: 0.0107 | Train Acc: 0.9975 | Val Acc: 0.9976
Epoch 29/100 | Train Loss: 0.0114 | Val Loss: 0.0105 | Train Acc: 0.9975 | Val Acc: 0.9976
Epoch 30/100 | Train Loss: 0.0113 | Val Loss: 0.0103 | Train Acc: 0.9975 | Val Acc: 0.9976
Epoch 31/100 | Train Loss: 0.0111 | Val Loss: 0.0102 | Train Acc: 0.9975 | Val Acc: 0.9976
Epoch 32/100 | Train Loss: 0.0109 | Val Loss: 0.0100 | Train Acc: 0.9975 | Val Acc: 0.9976
Epoch 33/100 | Train Loss: 0.0107 | Val Loss: 0.0098 | Train Acc: 0.9975 | Val Acc: 0.9976
Epoch 34/100 | Train Loss: 0.0106 | Val Loss: 0.0097 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 35/100 | Train Loss: 0.0104 | Val Loss: 0.0096 | Train Acc: 0.9976 | Val Acc: 0.9976
Epoch 36/100 | Train Loss: 0.0102 | Val Loss: 0.0094 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 37/100 | Train Loss: 0.0101 | Val Loss: 0.0093 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 38/100 | Train Loss: 0.0100 | Val Loss: 0.0092 | Train Acc: 0.9976 | Val Acc: 0.9976
Epoch 39/100 | Train Loss: 0.0099 | Val Loss: 0.0091 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 40/100 | Train Loss: 0.0097 | Val Loss: 0.0090 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 41/100 | Train Loss: 0.0096 | Val Loss: 0.0089 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 42/100 | Train Loss: 0.0095 | Val Loss: 0.0088 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 43/100 | Train Loss: 0.0094 | Val Loss: 0.0087 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 44/100 | Train Loss: 0.0093 | Val Loss: 0.0086 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 45/100 | Train Loss: 0.0092 | Val Loss: 0.0085 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 46/100 | Train Loss: 0.0091 | Val Loss: 0.0084 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 47/100 | Train Loss: 0.0091 | Val Loss: 0.0083 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 48/100 | Train Loss: 0.0089 | Val Loss: 0.0082 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 49/100 | Train Loss: 0.0088 | Val Loss: 0.0082 | Train Acc: 0.9979 | Val Acc: 0.9976
Epoch 50/100 | Train Loss: 0.0087 | Val Loss: 0.0081 | Train Acc: 0.9978 | Val Acc: 0.9976
Epoch 51/100 | Train Loss: 0.0086 | Val Loss: 0.0080 | Train Acc: 0.9979 | Val Acc: 0.9976
Epoch 52/100 | Train Loss: 0.0085 | Val Loss: 0.0079 | Train Acc: 0.9980 | Val Acc: 0.9976
Epoch 53/100 | Train Loss: 0.0085 | Val Loss: 0.0079 | Train Acc: 0.9980 | Val Acc: 0.9976
Epoch 54/100 | Train Loss: 0.0084 | Val Loss: 0.0078 | Train Acc: 0.9982 | Val Acc: 0.9976
Epoch 55/100 | Train Loss: 0.0083 | Val Loss: 0.0077 | Train Acc: 0.9980 | Val Acc: 0.9976
Epoch 56/100 | Train Loss: 0.0083 | Val Loss: 0.0077 | Train Acc: 0.9980 | Val Acc: 0.9976
Epoch 57/100 | Train Loss: 0.0082 | Val Loss: 0.0076 | Train Acc: 0.9984 | Val Acc: 0.9976
Epoch 58/100 | Train Loss: 0.0082 | Val Loss: 0.0076 | Train Acc: 0.9982 | Val Acc: 0.9976
```

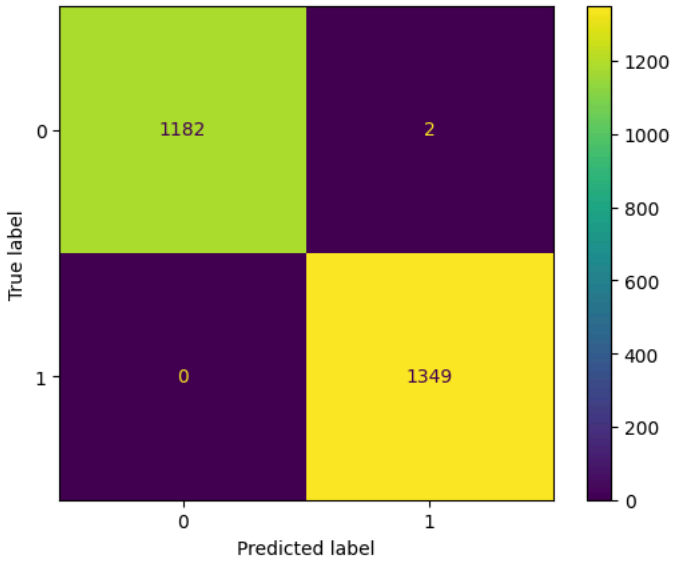
```
plt.figure(figsize=(8, 5))
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training vs Validation Loss')
plt.legend()
plt.show()

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()

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: 0.9992
```

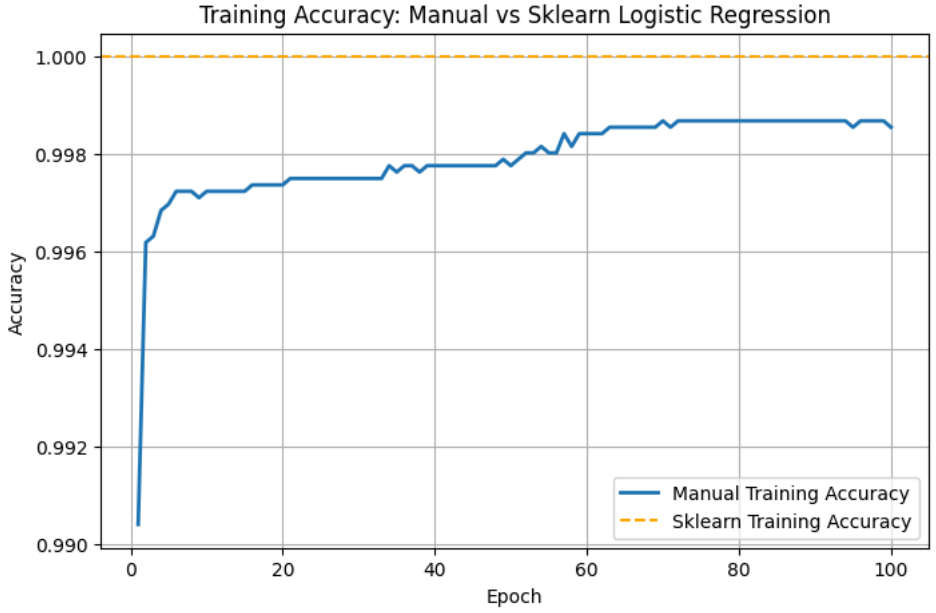


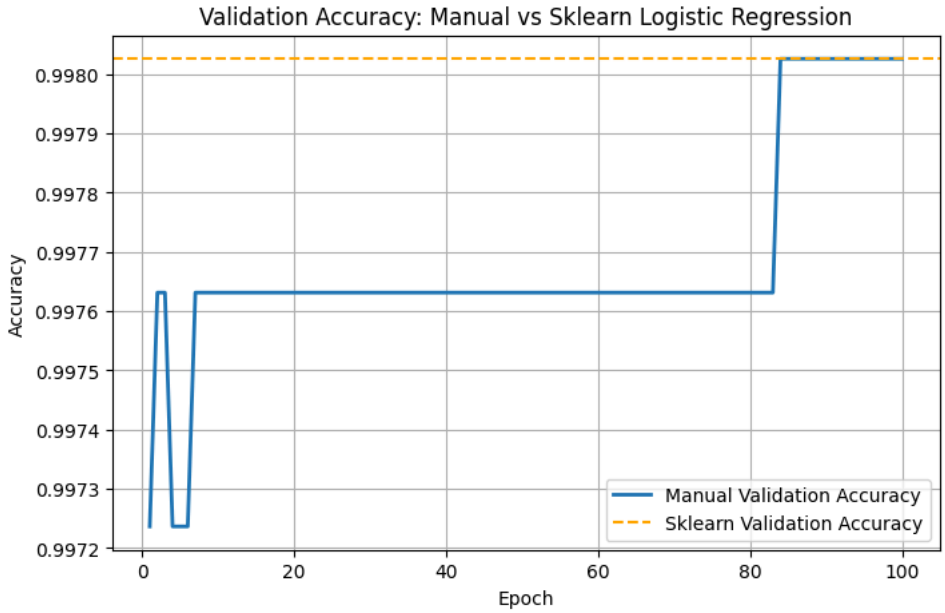

```
model_sklearn = LogisticRegression(max_iter=100, solver='liblinear')
model_sklearn.fit(X_train, y_train)
```

```
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}")
```

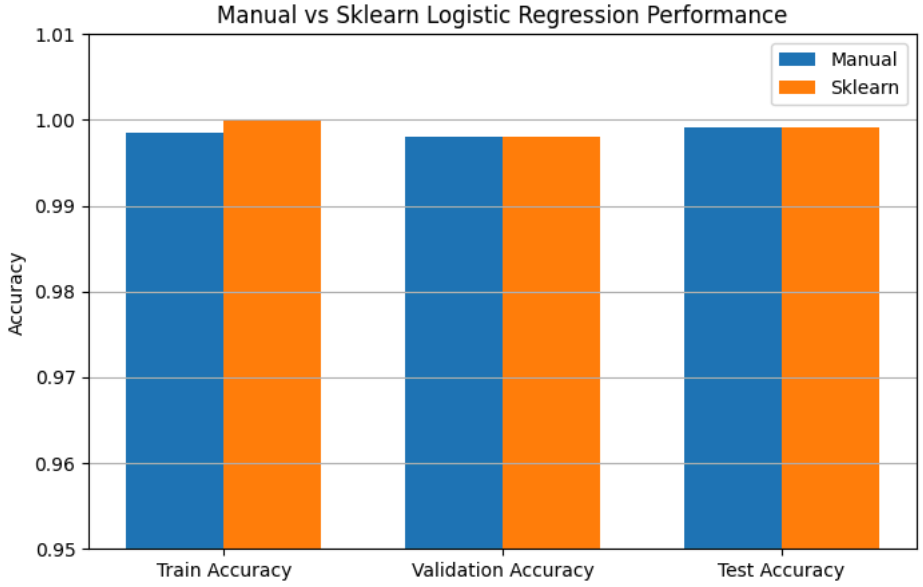
```
Sklearn Logistic Regression - Training Accuracy: 1.0000
Sklearn Logistic Regression - Validation Accuracy: 0.9980
Sklearn Logistic Regression - Test Accuracy: 0.9992
```

```
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()
```






```
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()
```




```
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_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)
```

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

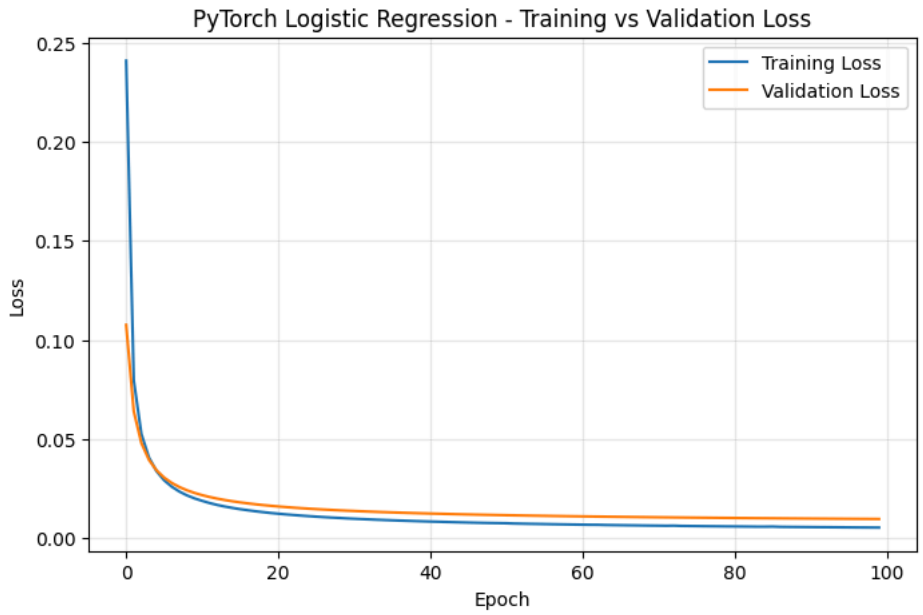
```
epochs = 100
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []

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: 0.2408 | Val Loss: 0.1076 | Train Acc: 0.9745 | Val Acc: 0.9945
Epoch 2/100 | Train Loss: 0.0796 | Val Loss: 0.0640 | Train Acc: 0.9968 | Val Acc: 0.9945
Epoch 3/100 | Train Loss: 0.0526 | Val Loss: 0.0480 | Train Acc: 0.9968 | Val Acc: 0.9953
Epoch 4/100 | Train Loss: 0.0406 | Val Loss: 0.0395 | Train Acc: 0.9972 | Val Acc: 0.9953
Epoch 5/100 | Train Loss: 0.0338 | Val Loss: 0.0342 | Train Acc: 0.9974 | Val Acc: 0.9957
Epoch 6/100 | Train Loss: 0.0292 | Val Loss: 0.0305 | Train Acc: 0.9975 | Val Acc: 0.9957
Epoch 7/100 | Train Loss: 0.0260 | Val Loss: 0.0279 | Train Acc: 0.9976 | Val Acc: 0.9957
Epoch 8/100 | Train Loss: 0.0235 | Val Loss: 0.0258 | Train Acc: 0.9978 | Val Acc: 0.9957
Epoch 9/100 | Train Loss: 0.0216 | Val Loss: 0.0242 | Train Acc: 0.9979 | Val Acc: 0.9957
Epoch 10/100 | Train Loss: 0.0201 | Val Loss: 0.0228 | Train Acc: 0.9979 | Val Acc: 0.9957
Epoch 11/100 | Train Loss: 0.0188 | Val Loss: 0.0217 | Train Acc: 0.9980 | Val Acc: 0.9961
Epoch 12/100 | Train Loss: 0.0177 | Val Loss: 0.0208 | Train Acc: 0.9980 | Val Acc: 0.9961
Epoch 13/100 | Train Loss: 0.0168 | Val Loss: 0.0199 | Train Acc: 0.9980 | Val Acc: 0.9961
Epoch 14/100 | Train Loss: 0.0160 | Val Loss: 0.0192 | Train Acc: 0.9980 | Val Acc: 0.9961
Epoch 15/100 | Train Loss: 0.0153 | Val Loss: 0.0186 | Train Acc: 0.9980 | Val Acc: 0.9961
Epoch 16/100 | Train Loss: 0.0146 | Val Loss: 0.0181 | Train Acc: 0.9980 | Val Acc: 0.9961
Epoch 17/100 | Train Loss: 0.0141 | Val Loss: 0.0176 | Train Acc: 0.9982 | Val Acc: 0.9961
Epoch 18/100 | Train Loss: 0.0136 | Val Loss: 0.0171 | Train Acc: 0.9980 | Val Acc: 0.9961
Epoch 19/100 | Train Loss: 0.0132 | Val Loss: 0.0167 | Train Acc: 0.9980 | Val Acc: 0.9961
Epoch 20/100 | Train Loss: 0.0127 | Val Loss: 0.0163 | Train Acc: 0.9980 | Val Acc: 0.9961
Epoch 21/100 | Train Loss: 0.0124 | Val Loss: 0.0160 | Train Acc: 0.9980 | Val Acc: 0.9961
Epoch 22/100 | Train Loss: 0.0120 | Val Loss: 0.0157 | Train Acc: 0.9982 | Val Acc: 0.9961
Epoch 23/100 | Train Loss: 0.0117 | Val Loss: 0.0154 | Train Acc: 0.9982 | Val Acc: 0.9961
Epoch 24/100 | Train Loss: 0.0114 | Val Loss: 0.0152 | Train Acc: 0.9982 | Val Acc: 0.9961
Epoch 25/100 | Train Loss: 0.0111 | Val Loss: 0.0149 | Train Acc: 0.9982 | Val Acc: 0.9961
Epoch 26/100 | Train Loss: 0.0109 | Val Loss: 0.0147 | Train Acc: 0.9982 | Val Acc: 0.9961
Epoch 27/100 | Train Loss: 0.0106 | Val Loss: 0.0145 | Train Acc: 0.9982 | Val Acc: 0.9961
Epoch 28/100 | Train Loss: 0.0104 | Val Loss: 0.0143 | Train Acc: 0.9982 | Val Acc: 0.9961
Epoch 29/100 | Train Loss: 0.0102 | Val Loss: 0.0141 | Train Acc: 0.9982 | Val Acc: 0.9961
Epoch 30/100 | Train Loss: 0.0100 | Val Loss: 0.0139 | Train Acc: 0.9982 | Val Acc: 0.9961
Epoch 31/100 | Train Loss: 0.0098 | Val Loss: 0.0137 | Train Acc: 0.9982 | Val Acc: 0.9961
Epoch 32/100 | Train Loss: 0.0097 | Val Loss: 0.0136 | Train Acc: 0.9982 | Val Acc: 0.9961
Epoch 33/100 | Train Loss: 0.0095 | Val Loss: 0.0134 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 34/100 | Train Loss: 0.0093 | Val Loss: 0.0133 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 35/100 | Train Loss: 0.0092 | Val Loss: 0.0131 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 36/100 | Train Loss: 0.0090 | Val Loss: 0.0130 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 37/100 | Train Loss: 0.0089 | Val Loss: 0.0129 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 38/100 | Train Loss: 0.0088 | Val Loss: 0.0128 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 39/100 | Train Loss: 0.0086 | Val Loss: 0.0127 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 40/100 | Train Loss: 0.0085 | Val Loss: 0.0126 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 41/100 | Train Loss: 0.0084 | Val Loss: 0.0124 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 42/100 | Train Loss: 0.0083 | Val Loss: 0.0123 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 43/100 | Train Loss: 0.0082 | Val Loss: 0.0123 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 44/100 | Train Loss: 0.0081 | Val Loss: 0.0122 | Train Acc: 0.9984 | Val Acc: 0.9961
Epoch 45/100 | Train Loss: 0.0080 | Val Loss: 0.0121 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 46/100 | Train Loss: 0.0079 | Val Loss: 0.0120 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 47/100 | Train Loss: 0.0078 | Val Loss: 0.0119 | Train Acc: 0.9983 | Val Acc: 0.9961
Epoch 48/100 | Train Loss: 0.0077 | Val Loss: 0.0118 | Train Acc: 0.9983 | Val Acc: 0.9961
```

```
Epoch 49/100 | Train Loss: 0.0077 | Val Loss: 0.0118 | Train Acc: 0.9983 | Val Acc: 0.9964
Epoch 50/100 | Train Loss: 0.0076 | Val Loss: 0.0117 | Train Acc: 0.9983 | Val Acc: 0.9964
Epoch 51/100 | Train Loss: 0.0076 | Val Loss: 0.0116 | Train Acc: 0.9983 | Val Acc: 0.9964
Epoch 52/100 | Train Loss: 0.0074 | Val Loss: 0.0115 | Train Acc: 0.9983 | Val Acc: 0.9964
Epoch 53/100 | Train Loss: 0.0073 | Val Loss: 0.0115 | Train Acc: 0.9983 | Val Acc: 0.9964
Epoch 54/100 | Train Loss: 0.0073 | Val Loss: 0.0114 | Train Acc: 0.9984 | Val Acc: 0.9964
Epoch 55/100 | Train Loss: 0.0072 | Val Loss: 0.0114 | Train Acc: 0.9984 | Val Acc: 0.9964
Epoch 56/100 | Train Loss: 0.0071 | Val Loss: 0.0113 | Train Acc: 0.9984 | Val Acc: 0.9964
Epoch 57/100 | Train Loss: 0.0071 | Val Loss: 0.0112 | Train Acc: 0.9986 | Val Acc: 0.9964
Epoch 58/100 | Train Loss: 0.0070 | Val Loss: 0.0112 | Train Acc: 0.9986 | Val Acc: 0.9964
```

```
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()

best_logistic_accuracy = max(val_accuracies)
```



Soft Max Regression

```
transform = transforms.Compose([
    transforms.ToTensor()  # Convert to tensor and scale 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)
X = torch.stack([img.view(-1) for img, _ in mnist_train])
y = torch.tensor([label for _, label in mnist_train])
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]}")
batch_size = 64
train_loader = DataLoader(TensorDataset(
    X_train, y_train), batch_size=batch_size, shuffle=True)
val_loader = DataLoader(TensorDataset(X_val, y_val), batch_size=batch_size)
test_loader = DataLoader(TensorDataset(X_test, y_test), batch_size=batch_size)
```

```
Train set size: 36000
Validation set size: 12000
Test set size: 12000
```

```
def softmax(z):
    exp_z = torch.exp(z - torch.max(z, dim=1, keepdim=True).values)
    return exp_z / exp_z.sum(dim=1, keepdim=True)


def cross_entropy_loss(y_hat, y_true):
    eps = 1e-15
    y_hat = torch.clamp(y_hat, eps, 1 - eps)
    one_hot = torch.nn.functional.one_hot(y_true, num_classes=10).float()
    return -torch.mean(torch.sum(one_hot * torch.log(y_hat), dim=1))


n_features = 784
n_classes = 10
W = torch.zeros((n_features, n_classes), requires_grad=True)
b = torch.zeros(n_classes, requires_grad=True)
learning_rate = 0.01
epochs = 100
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []
```

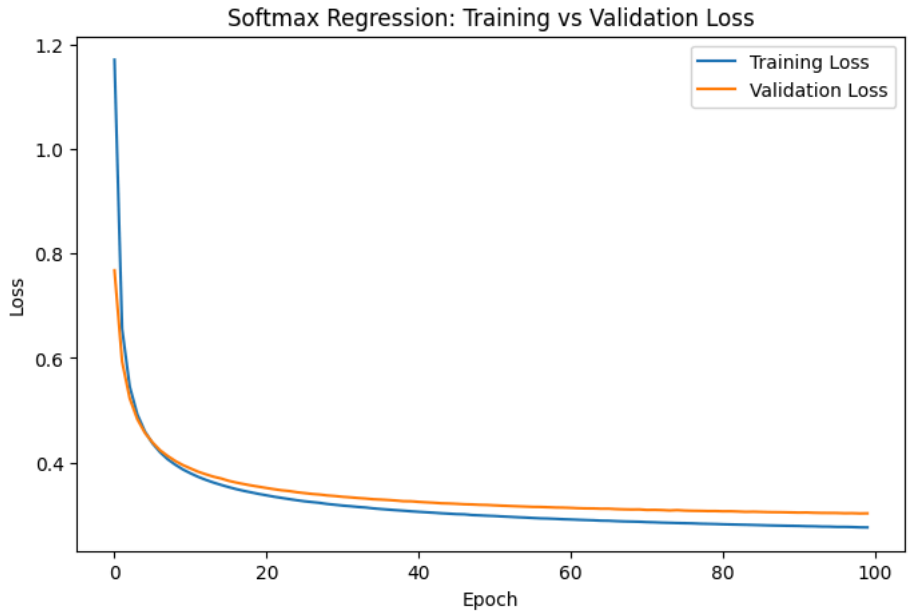
```
# --- Training loop ---
start_time = time.time()
for epoch in range(epochs):
    epoch_loss = 0.0
    correct_train = 0
    total_train = 0

    # ----- TRAINING PHASE -----
    for X_batch, y_batch in train_loader:
        # Forward pass
        y_pred = softmax(X_batch @ W + b)
        loss = cross_entropy_loss(y_pred, y_batch)

        # 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.argmax(dim=1)
        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_acc = correct_train / total_train
    train_losses.append(avg_train_loss)
    train_accuracies.append(train_acc)

    # ----- 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 = softmax(X_val_batch @ W + b)
            val_loss_total += cross_entropy_loss(y_val_pred,
                                                 y_val_batch).item()

            # Compute accuracy
            y_val_pred_label = y_val_pred.argmax(dim=1)
            correct += (y_val_pred_label == y_val_batch).sum().item()
```

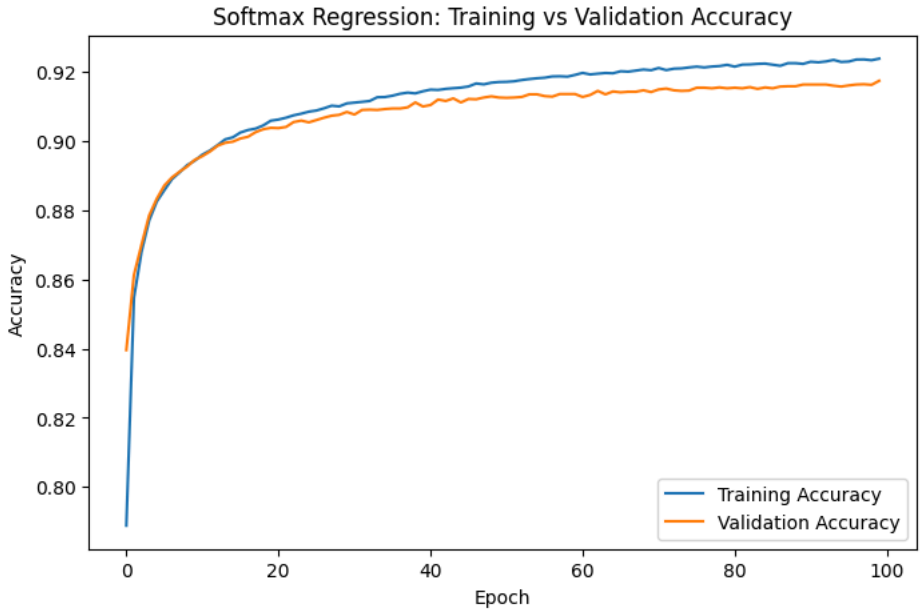
```
            total += y_val_batch.size(0)

        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}")

end_time = time.time()
sm_run_time = end_time - start_time
```

```
Epoch 1/100 | Train Loss: 1.1717 | Val Loss: 0.7680 | Train Acc: 0.7889 | Val Acc: 0.8396
Epoch 2/100 | Train Loss: 0.6561 | Val Loss: 0.5932 | Train Acc: 0.8547 | Val Acc: 0.8612
Epoch 3/100 | Train Loss: 0.5455 | Val Loss: 0.5229 | Train Acc: 0.8680 | Val Acc: 0.8700
Epoch 4/100 | Train Loss: 0.4921 | Val Loss: 0.4828 | Train Acc: 0.8770 | Val Acc: 0.8784
Epoch 5/100 | Train Loss: 0.4590 | Val Loss: 0.4566 | Train Acc: 0.8824 | Val Acc: 0.8832
Epoch 6/100 | Train Loss: 0.4364 | Val Loss: 0.4382 | Train Acc: 0.8858 | Val Acc: 0.8871
Epoch 7/100 | Train Loss: 0.4194 | Val Loss: 0.4231 | Train Acc: 0.8889 | Val Acc: 0.8894
Epoch 8/100 | Train Loss: 0.4059 | Val Loss: 0.4121 | Train Acc: 0.8909 | Val Acc: 0.8911
Epoch 9/100 | Train Loss: 0.3954 | Val Loss: 0.4026 | Train Acc: 0.8930 | Val Acc: 0.8926
Epoch 10/100 | Train Loss: 0.3863 | Val Loss: 0.3947 | Train Acc: 0.8943 | Val Acc: 0.8945
Epoch 11/100 | Train Loss: 0.3789 | Val Loss: 0.3883 | Train Acc: 0.8960 | Val Acc: 0.8956
Epoch 12/100 | Train Loss: 0.3722 | Val Loss: 0.3820 | Train Acc: 0.8972 | Val Acc: 0.8969
Epoch 13/100 | Train Loss: 0.3665 | Val Loss: 0.3771 | Train Acc: 0.8988 | Val Acc: 0.8986
Epoch 14/100 | Train Loss: 0.3614 | Val Loss: 0.3726 | Train Acc: 0.9004 | Val Acc: 0.8994
Epoch 15/100 | Train Loss: 0.3568 | Val Loss: 0.3691 | Train Acc: 0.9010 | Val Acc: 0.8998
Epoch 16/100 | Train Loss: 0.3526 | Val Loss: 0.3647 | Train Acc: 0.9024 | Val Acc: 0.9007
Epoch 17/100 | Train Loss: 0.3487 | Val Loss: 0.3613 | Train Acc: 0.9031 | Val Acc: 0.9012
Epoch 18/100 | Train Loss: 0.3453 | Val Loss: 0.3583 | Train Acc: 0.9035 | Val Acc: 0.9025
Epoch 19/100 | Train Loss: 0.3424 | Val Loss: 0.3556 | Train Acc: 0.9044 | Val Acc: 0.9033
Epoch 20/100 | Train Loss: 0.3394 | Val Loss: 0.3532 | Train Acc: 0.9058 | Val Acc: 0.9038
Epoch 21/100 | Train Loss: 0.3367 | Val Loss: 0.3508 | Train Acc: 0.9061 | Val Acc: 0.9037
Epoch 22/100 | Train Loss: 0.3341 | Val Loss: 0.3485 | Train Acc: 0.9067 | Val Acc: 0.9040
Epoch 23/100 | Train Loss: 0.3318 | Val Loss: 0.3464 | Train Acc: 0.9074 | Val Acc: 0.9054
Epoch 24/100 | Train Loss: 0.3295 | Val Loss: 0.3449 | Train Acc: 0.9078 | Val Acc: 0.9058
Epoch 25/100 | Train Loss: 0.3273 | Val Loss: 0.3427 | Train Acc: 0.9084 | Val Acc: 0.9053
Epoch 26/100 | Train Loss: 0.3254 | Val Loss: 0.3410 | Train Acc: 0.9087 | Val Acc: 0.9060
Epoch 27/100 | Train Loss: 0.3239 | Val Loss: 0.3394 | Train Acc: 0.9094 | Val Acc: 0.9067
Epoch 28/100 | Train Loss: 0.3223 | Val Loss: 0.3382 | Train Acc: 0.9101 | Val Acc: 0.9073
Epoch 29/100 | Train Loss: 0.3202 | Val Loss: 0.3365 | Train Acc: 0.9099 | Val Acc: 0.9075
Epoch 30/100 | Train Loss: 0.3187 | Val Loss: 0.3354 | Train Acc: 0.9108 | Val Acc: 0.9083
Epoch 31/100 | Train Loss: 0.3171 | Val Loss: 0.3340 | Train Acc: 0.9110 | Val Acc: 0.9076
Epoch 32/100 | Train Loss: 0.3160 | Val Loss: 0.3329 | Train Acc: 0.9113 | Val Acc: 0.9089
Epoch 33/100 | Train Loss: 0.3145 | Val Loss: 0.3317 | Train Acc: 0.9115 | Val Acc: 0.9090
Epoch 34/100 | Train Loss: 0.3135 | Val Loss: 0.3306 | Train Acc: 0.9126 | Val Acc: 0.9089
Epoch 35/100 | Train Loss: 0.3118 | Val Loss: 0.3293 | Train Acc: 0.9126 | Val Acc: 0.9092
Epoch 36/100 | Train Loss: 0.3105 | Val Loss: 0.3287 | Train Acc: 0.9130 | Val Acc: 0.9093
Epoch 37/100 | Train Loss: 0.3094 | Val Loss: 0.3278 | Train Acc: 0.9136 | Val Acc: 0.9093
Epoch 38/100 | Train Loss: 0.3083 | Val Loss: 0.3268 | Train Acc: 0.9139 | Val Acc: 0.9097
Epoch 39/100 | Train Loss: 0.3073 | Val Loss: 0.3255 | Train Acc: 0.9137 | Val Acc: 0.9111
Epoch 40/100 | Train Loss: 0.3062 | Val Loss: 0.3254 | Train Acc: 0.9143 | Val Acc: 0.9099
Epoch 41/100 | Train Loss: 0.3052 | Val Loss: 0.3242 | Train Acc: 0.9148 | Val Acc: 0.9103
Epoch 42/100 | Train Loss: 0.3043 | Val Loss: 0.3234 | Train Acc: 0.9147 | Val Acc: 0.9119
Epoch 43/100 | Train Loss: 0.3034 | Val Loss: 0.3225 | Train Acc: 0.9150 | Val Acc: 0.9115
Epoch 44/100 | Train Loss: 0.3024 | Val Loss: 0.3215 | Train Acc: 0.9152 | Val Acc: 0.9123
Epoch 45/100 | Train Loss: 0.3016 | Val Loss: 0.3212 | Train Acc: 0.9154 | Val Acc: 0.9111
Epoch 46/100 | Train Loss: 0.3008 | Val Loss: 0.3205 | Train Acc: 0.9157 | Val Acc: 0.9121
Epoch 47/100 | Train Loss: 0.3004 | Val Loss: 0.3197 | Train Acc: 0.9166 | Val Acc: 0.9120
Epoch 48/100 | Train Loss: 0.2990 | Val Loss: 0.3193 | Train Acc: 0.9163 | Val Acc: 0.9125
Epoch 49/100 | Train Loss: 0.2984 | Val Loss: 0.3186 | Train Acc: 0.9168 | Val Acc: 0.9128
Epoch 50/100 | Train Loss: 0.2977 | Val Loss: 0.3183 | Train Acc: 0.9170 | Val Acc: 0.9125
Epoch 51/100 | Train Loss: 0.2971 | Val Loss: 0.3175 | Train Acc: 0.9170 | Val Acc: 0.9124
Epoch 52/100 | Train Loss: 0.2963 | Val Loss: 0.3169 | Train Acc: 0.9172 | Val Acc: 0.9125
Epoch 53/100 | Train Loss: 0.2955 | Val Loss: 0.3163 | Train Acc: 0.9176 | Val Acc: 0.9127
Epoch 54/100 | Train Loss: 0.2948 | Val Loss: 0.3157 | Train Acc: 0.9178 | Val Acc: 0.9134
Epoch 55/100 | Train Loss: 0.2942 | Val Loss: 0.3152 | Train Acc: 0.9181 | Val Acc: 0.9134
Epoch 56/100 | Train Loss: 0.2934 | Val Loss: 0.3146 | Train Acc: 0.9182 | Val Acc: 0.9129
Epoch 57/100 | Train Loss: 0.2927 | Val Loss: 0.3145 | Train Acc: 0.9186 | Val Acc: 0.9127
Epoch 58/100 | Train Loss: 0.2923 | Val Loss: 0.3140 | Train Acc: 0.9186 | Val Acc: 0.9135
```

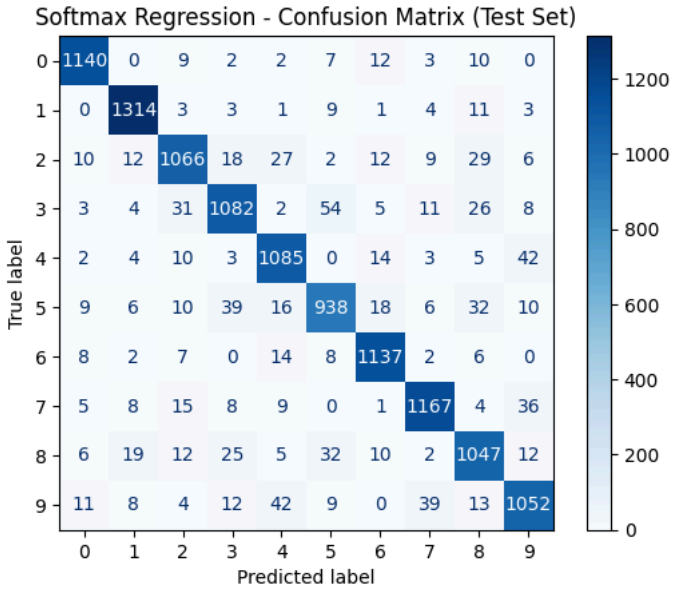
```
plt.figure(figsize=(8, 5))
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

```
plt.title('Softmax Regression: 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('Softmax Regression: Training vs Validation Accuracy')
plt.legend()
plt.show()

with torch.no_grad():
    logits_test = X_test @ W + b
    y_test_pred = softmax(logits_test)
    y_test_pred_label = y_test_pred.argmax(dim=1)

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

cm = confusion_matrix(y_test, y_test_pred_label)
ConfusionMatrixDisplay(cm).plot(cmap='Blues')
plt.title("Softmax Regression - Confusion Matrix (Test Set)")
plt.show()
```



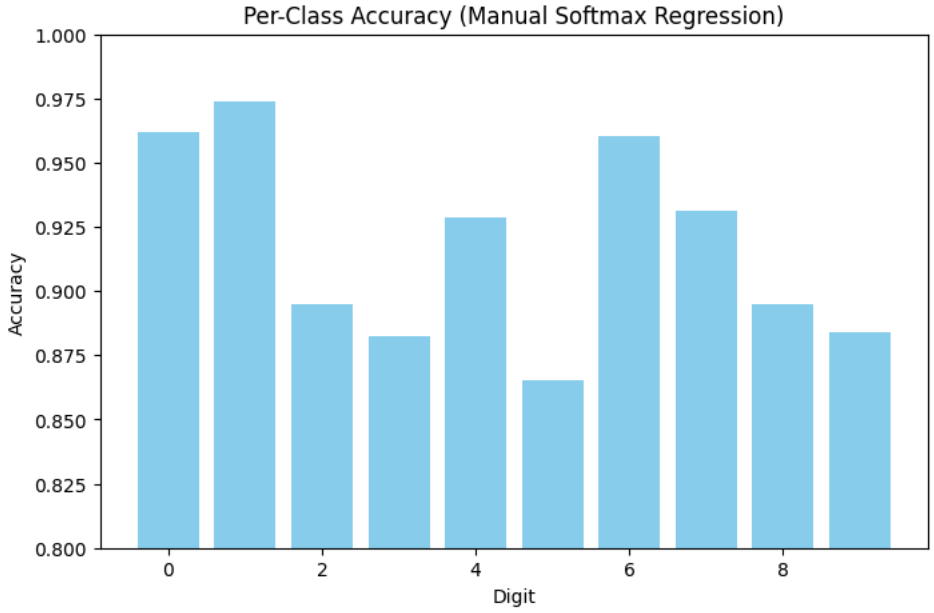



```
Final Test Accuracy: 0.9190
```



```
classes = list(range(10))
per_class_acc = []
for c in classes:
    mask = (y_test == c)
    acc_c = (y_test_pred_label[mask] == y_test[mask]).float().mean().item()
    per_class_acc.append(acc_c)
    print(f"Accuracy for digit {c}: {acc_c:.4f}")

plt.figure(figsize=(8, 5))
plt.bar(classes, per_class_acc, color='skyblue')
plt.xlabel('Digit')
plt.ylabel('Accuracy')
plt.title('Per-Class Accuracy (Manual Softmax Regression)')
plt.ylim(0.8, 1.0)
plt.show()
```

```
Accuracy for digit 0: 0.9620
Accuracy for digit 1: 0.9741
Accuracy for digit 2: 0.8950
Accuracy for digit 3: 0.8825
Accuracy for digit 4: 0.9289
Accuracy for digit 5: 0.8653
Accuracy for digit 6: 0.9603
Accuracy for digit 7: 0.9314
Accuracy for digit 8: 0.8949
Accuracy for digit 9: 0.8840
```




```
model_torch = nn.Sequential(
    nn.Linear(784, 10)
)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model_torch.parameters(), lr=0.01)

epochs_builtin = 100
for epoch in range(epochs_builtin):
    total_loss = 0
    correct = 0
    total = 0
    for X_batch, y_batch in train_loader:
        outputs = model_torch(X_batch)
        loss = criterion(outputs, y_batch)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        preds = outputs.argmax(dim=1)
        correct += (preds == y_batch).sum().item()
        total += y_batch.size(0)
```

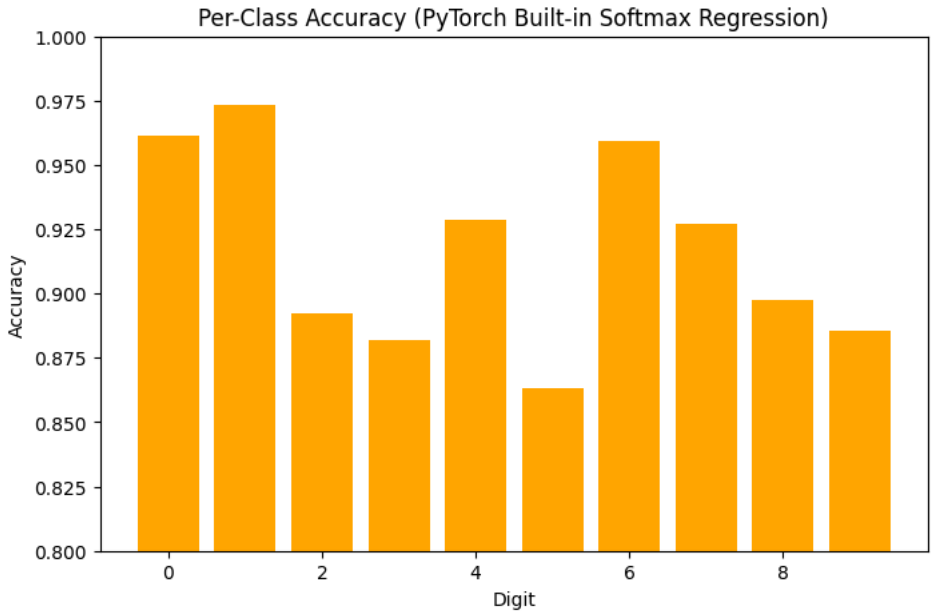
```
    print(f"[PyTorch Model] Epoch {epoch+1}/{epochs_builtin} | "
          f"Loss: {total_loss/len(train_loader):.4f} | Train Acc: {correct/total:.4f}")


with torch.no_grad():
    outputs_test = model_torch(X_test)
    preds_test = outputs_test.argmax(dim=1)
    acc_torch_test = (preds_test == y_test).float().mean().item()

print(
    f"\nPyTorch Built-in Softmax Regression - Test Accuracy: {acc_torch_test:.4f}")
```

```
[PyTorch Model] Epoch 1/100 | Loss: 1.1765 | Train Acc: 0.7682
[PyTorch Model] Epoch 2/100 | Loss: 0.6594 | Train Acc: 0.8532
[PyTorch Model] Epoch 3/100 | Loss: 0.5473 | Train Acc: 0.8674
[PyTorch Model] Epoch 4/100 | Loss: 0.4933 | Train Acc: 0.8761
[PyTorch Model] Epoch 5/100 | Loss: 0.4601 | Train Acc: 0.8811
[PyTorch Model] Epoch 6/100 | Loss: 0.4375 | Train Acc: 0.8846
[PyTorch Model] Epoch 7/100 | Loss: 0.4203 | Train Acc: 0.8876
[PyTorch Model] Epoch 8/100 | Loss: 0.4068 | Train Acc: 0.8903
[PyTorch Model] Epoch 9/100 | Loss: 0.3961 | Train Acc: 0.8926
[PyTorch Model] Epoch 10/100 | Loss: 0.3872 | Train Acc: 0.8943
[PyTorch Model] Epoch 11/100 | Loss: 0.3795 | Train Acc: 0.8958
[PyTorch Model] Epoch 12/100 | Loss: 0.3727 | Train Acc: 0.8972
[PyTorch Model] Epoch 13/100 | Loss: 0.3669 | Train Acc: 0.8983
[PyTorch Model] Epoch 14/100 | Loss: 0.3619 | Train Acc: 0.8998
[PyTorch Model] Epoch 15/100 | Loss: 0.3571 | Train Acc: 0.9005
[PyTorch Model] Epoch 16/100 | Loss: 0.3533 | Train Acc: 0.9016
[PyTorch Model] Epoch 17/100 | Loss: 0.3491 | Train Acc: 0.9027
[PyTorch Model] Epoch 18/100 | Loss: 0.3459 | Train Acc: 0.9037
[PyTorch Model] Epoch 19/100 | Loss: 0.3426 | Train Acc: 0.9042
[PyTorch Model] Epoch 20/100 | Loss: 0.3395 | Train Acc: 0.9051
[PyTorch Model] Epoch 21/100 | Loss: 0.3369 | Train Acc: 0.9059
[PyTorch Model] Epoch 22/100 | Loss: 0.3343 | Train Acc: 0.9061
[PyTorch Model] Epoch 23/100 | Loss: 0.3321 | Train Acc: 0.9072
[PyTorch Model] Epoch 24/100 | Loss: 0.3296 | Train Acc: 0.9080
[PyTorch Model] Epoch 25/100 | Loss: 0.3279 | Train Acc: 0.9079
[PyTorch Model] Epoch 26/100 | Loss: 0.3257 | Train Acc: 0.9084
[PyTorch Model] Epoch 27/100 | Loss: 0.3240 | Train Acc: 0.9090
[PyTorch Model] Epoch 28/100 | Loss: 0.3225 | Train Acc: 0.9097
[PyTorch Model] Epoch 29/100 | Loss: 0.3204 | Train Acc: 0.9106
[PyTorch Model] Epoch 30/100 | Loss: 0.3191 | Train Acc: 0.9106
[PyTorch Model] Epoch 31/100 | Loss: 0.3173 | Train Acc: 0.9113
[PyTorch Model] Epoch 32/100 | Loss: 0.3161 | Train Acc: 0.9111
[PyTorch Model] Epoch 33/100 | Loss: 0.3148 | Train Acc: 0.9118
[PyTorch Model] Epoch 34/100 | Loss: 0.3131 | Train Acc: 0.9124
[PyTorch Model] Epoch 35/100 | Loss: 0.3122 | Train Acc: 0.9123
[PyTorch Model] Epoch 36/100 | Loss: 0.3108 | Train Acc: 0.9129
[PyTorch Model] Epoch 37/100 | Loss: 0.3098 | Train Acc: 0.9129
[PyTorch Model] Epoch 38/100 | Loss: 0.3087 | Train Acc: 0.9132
[PyTorch Model] Epoch 39/100 | Loss: 0.3077 | Train Acc: 0.9137
[PyTorch Model] Epoch 40/100 | Loss: 0.3064 | Train Acc: 0.9142
[PyTorch Model] Epoch 41/100 | Loss: 0.3055 | Train Acc: 0.9139
[PyTorch Model] Epoch 42/100 | Loss: 0.3044 | Train Acc: 0.9149
[PyTorch Model] Epoch 43/100 | Loss: 0.3035 | Train Acc: 0.9148
[PyTorch Model] Epoch 44/100 | Loss: 0.3027 | Train Acc: 0.9148
[PyTorch Model] Epoch 45/100 | Loss: 0.3017 | Train Acc: 0.9156
[PyTorch Model] Epoch 46/100 | Loss: 0.3008 | Train Acc: 0.9160
[PyTorch Model] Epoch 47/100 | Loss: 0.3001 | Train Acc: 0.9158
[PyTorch Model] Epoch 48/100 | Loss: 0.2994 | Train Acc: 0.9161
[PyTorch Model] Epoch 49/100 | Loss: 0.2985 | Train Acc: 0.9166
[PyTorch Model] Epoch 50/100 | Loss: 0.2976 | Train Acc: 0.9170
[PyTorch Model] Epoch 51/100 | Loss: 0.2968 | Train Acc: 0.9169
[PyTorch Model] Epoch 52/100 | Loss: 0.2962 | Train Acc: 0.9174
[PyTorch Model] Epoch 53/100 | Loss: 0.2956 | Train Acc: 0.9173
[PyTorch Model] Epoch 54/100 | Loss: 0.2948 | Train Acc: 0.9176
[PyTorch Model] Epoch 55/100 | Loss: 0.2942 | Train Acc: 0.9181
[PyTorch Model] Epoch 56/100 | Loss: 0.2935 | Train Acc: 0.9180
[PyTorch Model] Epoch 57/100 | Loss: 0.2930 | Train Acc: 0.9187
[PyTorch Model] Epoch 58/100 | Loss: 0.2922 | Train Acc: 0.9186
```

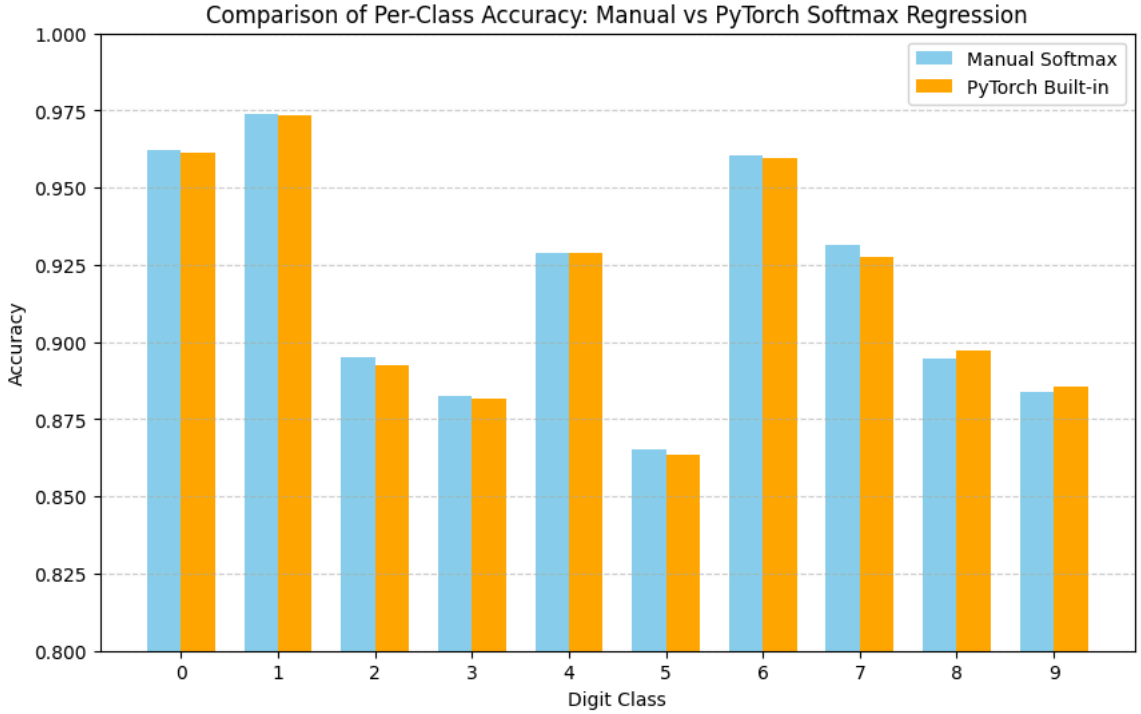
```
import matplotlib.pyplot as plt
classes = list(range(10))
per_class_acc_torch = []

for c in classes:
    mask = (y_test == c)
    acc_c = (preds_test[mask] == y_test[mask]).float().mean().item()
    per_class_acc_torch.append(acc_c)
    print(f"Digit {c}: {acc_c:.4f}")

# Optional: visualize as bar chart
```

```
plt.figure(figsize=(8, 5))
plt.bar(classes, per_class_acc_torch, color='orange')
plt.xlabel('Digit')
plt.ylabel('Accuracy')
plt.title('Per-Class Accuracy (PyTorch Built-in Softmax Regression)')
plt.ylim(0.8, 1.0)
plt.show()
```

```
Digit 0: 0.9612
Digit 1: 0.9733
Digit 2: 0.8925
Digit 3: 0.8817
Digit 4: 0.9289
Digit 5: 0.8635
Digit 6: 0.9595
Digit 7: 0.9274
Digit 8: 0.8974
Digit 9: 0.8857
```




```
manual_acc = np.array(per_class_acc)
torch_acc = np.array(per_class_acc_torch)

# --- Plot comparison ---
plt.figure(figsize=(10, 6))
bar_width = 0.35
x = np.arange(len(classes))

plt.bar(x - bar_width/2, manual_acc, width=bar_width,
        label='Manual Softmax', color='skyblue')
plt.bar(x + bar_width/2, torch_acc, width=bar_width,
        label='PyTorch Built-in', color='orange')

plt.xlabel('Digit Class')
plt.ylabel('Accuracy')
plt.title('Comparison of Per-Class Accuracy: Manual vs PyTorch Softmax Regression')
plt.xticks(x, classes)
plt.ylim(0.8, 1.0)
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.show()

# --- Print numerical summary ---
print("Digit | Manual Acc | PyTorch Acc | Difference")
print("-------------------------------------------")
for c in range(10):
    diff = abs(manual_acc[c] - torch_acc[c])
    print(
        f"  {c:>2}   |   {manual_acc[c]:.4f}   |   {torch_acc[c]:.4f}   |   {diff:.4f}")

best_softmax_accuracy = max(val_accuracies)
```




```
Digit | Manual Acc | PyTorch Acc | Difference
-----------------------------------------------
   0   |   0.9620   |   0.9612    |   0.0008
   1   |   0.9741   |   0.9733    |   0.0007
   2   |   0.8950   |   0.8925    |   0.0025
   3   |   0.8825   |   0.8817    |   0.0008
   4   |   0.9289   |   0.9289    |   0.0000
   5   |   0.8653   |   0.8635    |   0.0018
   6   |   0.9603   |   0.9595    |   0.0008
   7   |   0.9314   |   0.9274    |   0.0040
   8   |   0.8949   |   0.8974    |   0.0026
   9   |   0.8840   |   0.8857    |   0.0017
```

neural network

```
class ManualFeedForward(nn.Module):
    def __init__(self, layers):
        super().__init__()
        self.layers = nn.ModuleList()
        for i in range(len(layers) - 1):
            l = nn.Linear(layers[i], layers[i + 1])
            self.layers.append(l)

        for i, l in enumerate(self.layers):
            if i < len(self.layers) - 1:
                init.kaiming_uniform_(l.weight, nonlinearity='relu')
            else:
                init.xavier_uniform_(l.weight)
            init.zeros_(l.bias)

    def forward(self, x):
        if x.dim() > 2:
            x = x.view(x.size(0), -1)
        self.activations = [x]
        self.z_values = []
        out = x
        for i, layer in enumerate(self.layers):
            z = out @ layer.weight.T + layer.bias
            self.z_values.append(z)
            if i < len(self.layers) - 1:
                out = torch.relu(z)
            else:
                out = z
            self.activations.append(out)
        return out
```

```
    def manual_backward(self, y_true, lr):
        batch_size = y_true.size(0)
        logits = self.activations[-1]

        exp_logits = torch.exp(logits - logits.max(dim=1, keepdim=True)[0])
        probs = exp_logits / exp_logits.sum(dim=1, keepdim=True)
        # ensure same device as logits
        y_onehot = torch.zeros_like(probs, device=probs.device)
        y_onehot[torch.arange(batch_size, device=probs.device), y_true] = 1.0
        loss = -torch.sum(y_onehot * torch.log(probs + 1e-9)) / batch_size

        delta = (probs - y_onehot) / batch_size

        for i in reversed(range(len(self.layers))):
            a_prev = self.activations[i]
            z = self.z_values[i]
            dW = delta.T @ a_prev
            db = delta.sum(dim=0)
            self.layers[i].weight.grad = dW
            self.layers[i].bias.grad = db

            if i > 0:
                delta = delta @ self.layers[i].weight
                z_prev = self.z_values[i - 1]
                delta = delta * (z_prev > 0)

        with torch.no_grad():
            for layer in self.layers:
                layer.weight -= lr * layer.weight.grad
                layer.bias -= lr * layer.bias.grad

        return loss.item()
```

```
def train_manual(model, dataloader, lr, device):
    model.train()
    total_loss, total_correct, N = 0, 0, 0
    for X, y in dataloader:
        X, y = X.to(device), y.to(device)
        logits = model.forward(X)
        loss = model.manual_backward(y, lr)
        preds = logits.argmax(dim=1)
        total_correct += (preds == y).sum().item()
        total_loss += loss * X.size(0)
        N += X.size(0)
    return total_loss / N, total_correct / N
```

```
def validate_manual(model, dataloader, device):
    model.eval()
    total_loss, total_correct, N = 0, 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            logits = model.forward(X)
            exp_logits = torch.exp(logits - logits.max(dim=1, keepdim=True)[0])
            probs = exp_logits / exp_logits.sum(dim=1, keepdim=True)
            loss = -torch.log(probs[torch.arange(y.size(0)), y] + 1e-9).mean()
            preds = logits.argmax(dim=1)
            total_correct += (preds == y).sum().item()
            total_loss += loss.item() * X.size(0)
            N += X.size(0)
    return total_loss / N, total_correct / N
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
INPUT_SIZE = 784
HIDDEN1 = 512
HIDDEN2 = 128
OUTPUT = 10

learning_rates = [0.001, 0.01, 0.1, 1.0]
BATCH_SIZE = 64
PATIENCE = 3
CHECKPOINT_PATH = "best_manual_model.pth"
SEED = 42
```

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
full_train_dataset = datasets.MNIST(
    './data', train=True, download=True, transform=transform)
train_ds, val_ds = random_split(full_train_dataset, [50000, 10000])
train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_ds, batch_size=BATCH_SIZE, shuffle=False)

test_dataset = TensorDataset(X_test, y_test)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)

model = ManualFeedForward([INPUT_SIZE, HIDDEN1, HIDDEN2, OUTPUT]).to(device)
train_losses, val_losses, train_accs, val_accs = [], [], [], []
best_val_loss = float('inf')
epochs_no_improve = 0
best_model_state = None
epoch = 0
best_epoch = 0
```

```
def train_manual_model(model, train_loader, val_loader, lr, patience=3, device='cpu'):
    model.to(device)
    train_losses, val_losses, train_accs, val_accs = [], [], [], []
    best_val_loss = float('inf')
    epochs_no_improve = 0
    best_model_state = None
    epoch = 0
    best_epoch = 0

    # get run time
    start_time = time.time()
    while epoch < 40:
        epoch += 1
        train_loss, train_acc = train_manual(model, train_loader, lr, device)
        val_loss, val_acc = validate_manual(model, val_loader, device)

        train_losses.append(train_loss)
        val_losses.append(val_loss)
        train_accs.append(train_acc)
        val_accs.append(val_acc)

        print(f"Epoch {epoch:02d} | Train Loss: {train_loss:.4f} | Train Acc: {train_acc*100:.2f}% | "
              f"Val Loss: {val_loss:.4f} | Val Acc: {val_acc*100:.2f}%")

        # Early stopping
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            best_model_state = {k: v.clone()
                                for k, v in model.state_dict().items()}
            best_epoch = epoch
            epochs_no_improve = 0
        else:
            epochs_no_improve += 1
            if epochs_no_improve >= patience:
                print(f"Early stopping triggered after {epoch} epochs.")
                break

    end_time = time.time()
    run_time = end_time - start_time
    # Restore best model
    if best_model_state is not None:
        model.load_state_dict(best_model_state)

    return {
        "train_losses": train_losses,
        "val_losses": val_losses,
        "train_accs": train_accs,
        "val_accs": val_accs,
        "best_val_loss": best_val_loss,
        "best_model_state": best_model_state,
        "best_epoch": best_epoch,
        "best_run_time": run_time
    }
```

```
learning_rates = [0.001, 0.01, 0.1, 1.0]

lr_results = {}

for LR in learning_rates:
    print(f"\n--- Training with LR={LR} ---")
    model = ManualFeedForward([INPUT_SIZE, HIDDEN1, HIDDEN2, OUTPUT])
    result = train_manual_model(
        model, train_loader, val_loader, lr=LR, patience=PATIENCE, device=device)
    lr_results[LR] = result

# find best learning rate
best_lr = 0.01
print(best_lr)
best_lr_val_error = lr_results[best_lr]["best_val_loss"]
print(
    f"\nBest Learning Rate: {best_lr} with Val Loss: {lr_results[best_lr]['best_val_loss']:.4f}")
print(
    f"meet with batch size {BATCH_SIZE}, number of layers 3, number of neurons [{HIDDEN1}, {HIDDEN2}]")

# Plot learning curves
plt.figure(figsize=(10, 5))
for LR in learning_rates:
    plt.plot(lr_results[LR]["train_losses"], label=f'Train Loss LR={LR}')
    plt.plot(lr_results[LR]["val_losses"], '--', label=f'Val Loss LR={LR}')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Learning Rate Analysis')
plt.legend()
plt.show()

best_nn_accuracy = max(lr_results[best_lr]["val_accs"])
best_nn_run_time = lr_results[best_lr]["best_run_time"]
```

```
--- Training with LR=0.001 ---
Epoch 01 | Train Loss: 1.1035 | Train Acc: 67.30% | Val Loss: 0.6466 | Val Acc: 81.61%
Epoch 02 | Train Loss: 0.5327 | Train Acc: 85.08% | Val Loss: 0.4666 | Val Acc: 86.46%
Epoch 03 | Train Loss: 0.4167 | Train Acc: 88.19% | Val Loss: 0.3943 | Val Acc: 88.40%
Epoch 04 | Train Loss: 0.3606 | Train Acc: 89.73% | Val Loss: 0.3531 | Val Acc: 89.66%
Epoch 05 | Train Loss: 0.3256 | Train Acc: 90.68% | Val Loss: 0.3250 | Val Acc: 90.44%
Epoch 06 | Train Loss: 0.3008 | Train Acc: 91.36% | Val Loss: 0.3051 | Val Acc: 90.91%
Epoch 07 | Train Loss: 0.2816 | Train Acc: 91.91% | Val Loss: 0.2888 | Val Acc: 91.46%
Epoch 08 | Train Loss: 0.2662 | Train Acc: 92.33% | Val Loss: 0.2752 | Val Acc: 92.02%
Epoch 09 | Train Loss: 0.2533 | Train Acc: 92.73% | Val Loss: 0.2647 | Val Acc: 92.21%
Epoch 10 | Train Loss: 0.2422 | Train Acc: 93.04% | Val Loss: 0.2544 | Val Acc: 92.70%
Epoch 11 | Train Loss: 0.2327 | Train Acc: 93.36% | Val Loss: 0.2467 | Val Acc: 92.83%
Epoch 12 | Train Loss: 0.2241 | Train Acc: 93.65% | Val Loss: 0.2387 | Val Acc: 93.17%
Epoch 13 | Train Loss: 0.2164 | Train Acc: 93.86% | Val Loss: 0.2325 | Val Acc: 93.41%
Epoch 14 | Train Loss: 0.2093 | Train Acc: 94.04% | Val Loss: 0.2263 | Val Acc: 93.58%
Epoch 15 | Train Loss: 0.2029 | Train Acc: 94.21% | Val Loss: 0.2207 | Val Acc: 93.82%
Epoch 16 | Train Loss: 0.1970 | Train Acc: 94.38% | Val Loss: 0.2162 | Val Acc: 93.89%
Epoch 17 | Train Loss: 0.1915 | Train Acc: 94.56% | Val Loss: 0.2108 | Val Acc: 94.06%
Epoch 18 | Train Loss: 0.1864 | Train Acc: 94.73% | Val Loss: 0.2065 | Val Acc: 94.21%
Epoch 19 | Train Loss: 0.1817 | Train Acc: 94.84% | Val Loss: 0.2024 | Val Acc: 94.31%
Epoch 20 | Train Loss: 0.1771 | Train Acc: 95.01% | Val Loss: 0.1987 | Val Acc: 94.39%
Epoch 21 | Train Loss: 0.1730 | Train Acc: 95.10% | Val Loss: 0.1963 | Val Acc: 94.38%
Epoch 22 | Train Loss: 0.1689 | Train Acc: 95.25% | Val Loss: 0.1919 | Val Acc: 94.40%
Epoch 23 | Train Loss: 0.1651 | Train Acc: 95.36% | Val Loss: 0.1891 | Val Acc: 94.57%
Epoch 24 | Train Loss: 0.1616 | Train Acc: 95.47% | Val Loss: 0.1858 | Val Acc: 94.73%
Epoch 25 | Train Loss: 0.1582 | Train Acc: 95.56% | Val Loss: 0.1823 | Val Acc: 94.84%
Epoch 26 | Train Loss: 0.1549 | Train Acc: 95.62% | Val Loss: 0.1797 | Val Acc: 94.92%
Epoch 27 | Train Loss: 0.1517 | Train Acc: 95.74% | Val Loss: 0.1770 | Val Acc: 94.97%
Epoch 28 | Train Loss: 0.1487 | Train Acc: 95.84% | Val Loss: 0.1753 | Val Acc: 94.96%
Epoch 29 | Train Loss: 0.1459 | Train Acc: 95.92% | Val Loss: 0.1719 | Val Acc: 95.14%
Epoch 30 | Train Loss: 0.1431 | Train Acc: 95.98% | Val Loss: 0.1700 | Val Acc: 95.17%
Epoch 31 | Train Loss: 0.1405 | Train Acc: 96.07% | Val Loss: 0.1674 | Val Acc: 95.36%
Epoch 32 | Train Loss: 0.1379 | Train Acc: 96.12% | Val Loss: 0.1654 | Val Acc: 95.39%
Epoch 33 | Train Loss: 0.1355 | Train Acc: 96.22% | Val Loss: 0.1635 | Val Acc: 95.45%
Epoch 34 | Train Loss: 0.1331 | Train Acc: 96.29% | Val Loss: 0.1613 | Val Acc: 95.51%
Epoch 35 | Train Loss: 0.1308 | Train Acc: 96.39% | Val Loss: 0.1599 | Val Acc: 95.57%
Epoch 36 | Train Loss: 0.1286 | Train Acc: 96.46% | Val Loss: 0.1582 | Val Acc: 95.52%
Epoch 37 | Train Loss: 0.1264 | Train Acc: 96.53% | Val Loss: 0.1563 | Val Acc: 95.59%
Epoch 38 | Train Loss: 0.1243 | Train Acc: 96.59% | Val Loss: 0.1542 | Val Acc: 95.67%
Epoch 39 | Train Loss: 0.1223 | Train Acc: 96.63% | Val Loss: 0.1529 | Val Acc: 95.64%
Epoch 40 | Train Loss: 0.1203 | Train Acc: 96.70% | Val Loss: 0.1514 | Val Acc: 95.73%

--- Training with LR=0.01 ---
Epoch 01 | Train Loss: 0.4241 | Train Acc: 87.53% | Val Loss: 0.2612 | Val Acc: 92.44%
Epoch 02 | Train Loss: 0.2110 | Train Acc: 93.81% | Val Loss: 0.2044 | Val Acc: 94.20%
Epoch 03 | Train Loss: 0.1652 | Train Acc: 95.25% | Val Loss: 0.1764 | Val Acc: 94.80%
Epoch 04 | Train Loss: 0.1383 | Train Acc: 96.07% | Val Loss: 0.1601 | Val Acc: 95.35%
Epoch 05 | Train Loss: 0.1185 | Train Acc: 96.58% | Val Loss: 0.1494 | Val Acc: 95.72%
Epoch 06 | Train Loss: 0.1037 | Train Acc: 97.10% | Val Loss: 0.1353 | Val Acc: 95.94%
Epoch 07 | Train Loss: 0.0919 | Train Acc: 97.48% | Val Loss: 0.1279 | Val Acc: 96.29%
```

```
print("\n--- Evaluating best model (LR = 0.01) on test set ---")

best_model = ManualFeedForward([INPUT_SIZE, HIDDEN1, HIDDEN2, OUTPUT])

best_model.load_state_dict(lr_results[0.01]["best_model_state"])
best_model.to(device)
best_model.eval()

total_loss, total_correct, total_samples = 0, 0, 0

with torch.no_grad():
    for X_batch, y_batch in test_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        outputs = best_model(X_batch)

        exp_logits = torch.exp(outputs - outputs.max(dim=1, keepdim=True)[0])
        probs = exp_logits / exp_logits.sum(dim=1, keepdim=True)
        loss = -torch.log(probs[torch.arange(y_batch.size(0)), y_batch] + 1e-9).mean()

        total_loss += loss.item() * X_batch.size(0)
        preds = outputs.argmax(dim=1)
        total_correct += (preds == y_batch).sum().item()
        total_samples += X_batch.size(0)

avg_test_loss = total_loss / total_samples
test_accuracy = 100 * total_correct / total_samples

print(f"Test Loss: {avg_test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy:.2f}%")
```

```
--- Training with LR=0.1 ---
Epoch 01 | Train Loss: 0.2264 | Train Acc: 93.05% | Val Loss: 0.2008 | Val Acc: 94.07%
Epoch 02 | Train Loss: 0.0852 | Train Acc: 97.41% | Val Loss: 0.0906 | Val Acc: 97.15%
Epoch 03 | Train Loss: 0.0497 | Train Acc: 98.52% | Val Loss: 0.0839 | Val Acc: 97.39%
Epoch 04 | Train Loss: 0.0306 | Train Acc: 99.15% | Val Loss: 0.0896 | Val Acc: 97.30%
Epoch 05 | Train Loss: 0.0187 | Train Acc: 99.52% | Val Loss: 0.1119 | Val Acc: 96.71%
--- Evaluating best model (LR = 0.01) on test set ---
Test Loss: 0.3475
Test Accuracy: 97.38%
```

```
batch_sizes = [16, 32, 64, 128]
batch_results = {}

for BS in batch_sizes:
    print(f"\n--- Training with batch size={BS} ---")
    train_loader_bs = DataLoader(train_ds, batch_size=BS, shuffle=True)
    val_loader_bs = DataLoader(val_ds, batch_size=BS, shuffle=False)

    model = ManualFeedForward([INPUT_SIZE, HIDDEN1, HIDDEN2, OUTPUT])
    result = train_manual_model(
        model, train_loader_bs, val_loader_bs, lr=0.01, patience=PATIENCE, device=device)
    batch_results[BS] = result

# Find the best batch size based on validation loss
best_bs = min(batch_results, key=lambda bs: batch_results[bs]["best_val_loss"])
best_bs_val_error = batch_results[best_bs]["best_val_loss"]
print(
    f"\nBest Batch Size: {best_bs} with Val Loss: {batch_results[best_bs]['best_val_loss']:.4f}")
print(
    f"meet with learning rate 0.01, number of layers 3, number of neurons [{HIDDEN1}, {HIDDEN2}]")

# Plot batch size comparison
plt.figure(figsize=(10, 5))
for BS in batch_sizes:
    plt.plot(batch_results[BS]["train_losses"], label=f'Train Loss BS={BS}')
    plt.plot(batch_results[BS]["val_losses"], '--', label=f'Val Loss BS={BS}')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Batch Size Analysis')
plt.legend()
plt.show()
```