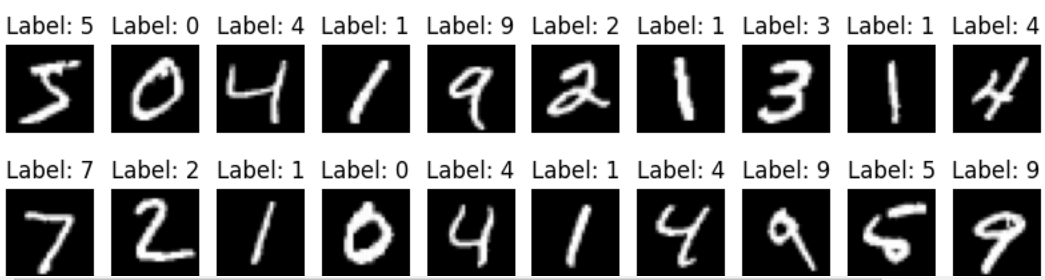
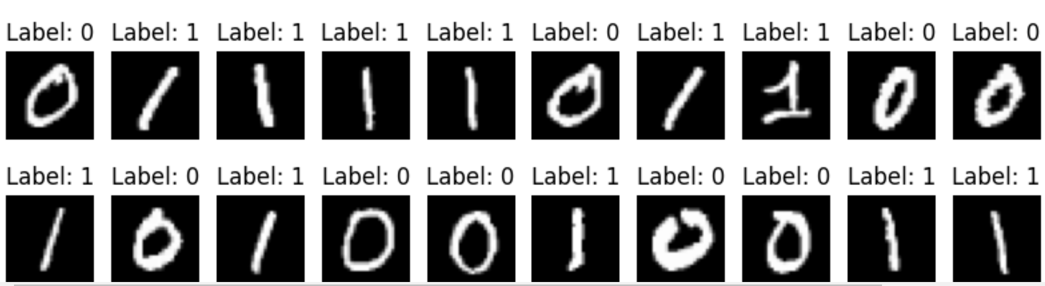
TA Rename notebook)42112

```
# Import libraries
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from sklearn.model_selection import train_test_split

# Load MNIST dataset
(X_train_full, y_train_full), (X_test_full, y_test_full) = mnist.load_data()
```

```
# Display a few images from the dataset
def display_images(X, y, n=10):
    plt.figure(figsize=(10, 2))
    for i in range(n):
        plt.subplot(1, n, i + 1)
        plt.imshow(X[i], cmap='gray')
        plt.title(f"Label: {y[i]}")
        plt.axis('off')
    plt.show()

# Show first 10 images in the training set
display_images(X_train_full, y_train_full, n=10)

# Show first 10 images in the test set
display_images(X_test_full, y_test_full, n=10)
```




```
# Filter to only include images of digits 0 and 1
def filter_binary_data(X, y):
    binary_filter = np.where((y == 0) | (y == 1))
    X_binary = X[binary_filter]
    y_binary = y[binary_filter]
    return X_binary, y_binary

X_train, y_train = filter_binary_data(X_train_full, y_train_full)
X_test, y_test = filter_binary_data(X_test_full, y_test_full)

display_images(X_train, y_train, n=10)
display_images(X_test, y_test, n=10)
```




```
print(f'X_train shape: {X_train.shape}')
print(f'y_train shape: {y_train.shape}')
```

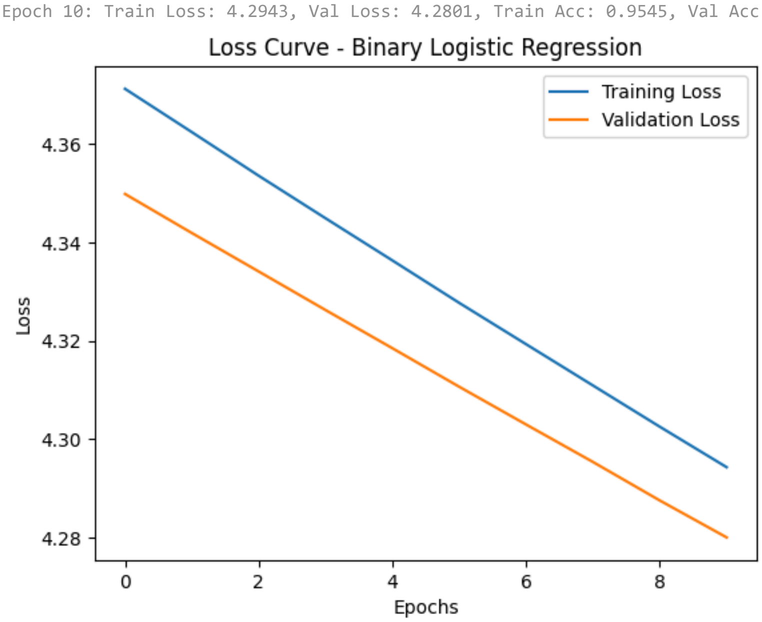
```
print(X_train[0].shape)
```

```
X_tr Rename notebook 565, 28, 28)
y_train shape: (12665,)
(28, 28)
```

```
# Normalize pixel values (0-255 to 0-1 range)
X_train = X_train / 255.0
X_test = X_test / 255.0

# Flatten images (28x28 -> 784) for logistic regression input
X_train_flat = X_train.reshape(X_train.shape[0], -1).astype(np.float32)  # Explicit cast to float32
X_test_flat = X_test.reshape(X_test.shape[0], -1).astype(np.float32)    # Explicit cast to float32

# Split the training data into training and validation sets
X_train_flat, X_val_flat, y_train, y_val = train_test_split(X_train_flat, y_train, test_size=0.2, random_state=42)

# Ensure labels are also in float32
y_train = y_train.astype(np.float32)
y_val = y_val.astype(np.float32)
y_test = y_test.astype(np.float32)


# Define a custom logistic regression model
class LogisticRegressionModelBinary(tf.Module):
    def __init__(self, input_dim):
        # Initialize weights and bias (weights shape is [input_dim, 1], bias is scalar)
        self.w = tf.Variable(tf.random.normal(shape=(input_dim, 1), dtype=tf.float32), trainable=True)
        self.b = tf.Variable(tf.zeros(1, dtype=tf.float32), trainable=True)

    def __call__(self, X):
        # Logistic regression hypothesis: h(X) = sigmoid(X @ w + b)
        logits = tf.matmul(X, self.w) + self.b
        return tf.sigmoid(logits)


# Initialize logistic regression model
input_dim = X_train_flat.shape[1]  # 784 features (28x28 image)
binary_model = LogisticRegressionModelBinary(input_dim)

# Define binary cross-entropy loss with clipping to avoid log(0)
def binary_cross_entropy_with_regularization(y_true, y_pred, model, lambda_reg=0.01):
    y_pred = tf.clip_by_value(y_pred, 1e-7, 1 - 1e-7)  # Avoid log(0)
    cross_entropy = -tf.reduce_mean(y_true * tf.math.log(y_pred) + (1 - y_true) * tf.math.log(1 - y_pred))

    # L2 Regularization term
    l2_loss = tf.nn.l2_loss(model.w)  # Sum of squares of weights
    return cross_entropy + lambda_reg * l2_loss  # Combine cross-entropy and regularization

# Define accuracy metric
def compute_accuracy_binary(y_true, y_pred):
    predictions = tf.cast(y_pred > 0.5, dtype=tf.float32)
    return tf.reduce_mean(tf.cast(tf.equal(predictions, y_true), dtype=tf.float32))


# Training parameters
learning_rate = 0.001  # Reduced learning rate to avoid large updates
epochs = 10
lambda_reg = 0.01  # Regularization strength

# Optimizer (using Gradient Descent)
optimizer = tf.optimizers.Adam(learning_rate=learning_rate)

# To store the losses for plotting
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

# Training loop
for epoch in range(epochs):
    with tf.GradientTape() as tape:
        # Forward pass: compute predictions and loss for training set
        y_train_pred = binary_model(X_train_flat)
        train_loss = binary_cross_entropy_with_regularization(
            tf.reshape(y_train, (-1, 1)), y_train_pred, binary_model, lambda_reg
        )
```

```
    # Compute gradients and update weights
    gradients = tape.gradient(train_loss, [binary_model.w, binary_model.b])
    optim Rename notebook lients(zip(gradients, [binary_model.w, binary_model.b]))

    # Compute validation loss and accuracy
    y_val_pred = binary_model(X_val_flat)
    val_loss = binary_cross_entropy_with_regularization(
        tf.reshape(y_val, (-1, 1)), y_val_pred, binary_model, lambda_reg
    )
    val_acc = compute_accuracy_binary(tf.reshape(y_val, (-1, 1)), y_val_pred).numpy()

    # Compute training accuracy
    train_acc = compute_accuracy_binary(tf.reshape(y_train, (-1, 1)), y_train_pred).numpy()

    # Store losses and accuracies
    train_losses.append(train_loss.numpy())
    val_losses.append(val_loss.numpy())
    train_accuracies.append(train_acc)
    val_accuracies.append(val_acc)

    print(f"Epoch {epoch+1}: Train Loss: {train_loss.numpy():.4f}, Val Loss: {val_loss.numpy():.4f},
    Train Acc: {train_acc:.4f}, Val Acc: {val_acc:.4f}")

# Plot the loss curve
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.title('Loss Curve - Binary Logistic Regression')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the model on the test set
y_test_pred = binary_model(X_test_flat)
test_loss = binary_cross_entropy_with_regularization(
    tf.reshape(y_test, (-1, 1)), y_test_pred, binary_model, lambda_reg
).numpy()
test_acc = compute_accuracy_binary(tf.reshape(y_test, (-1, 1)), y_test_pred).numpy()
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_acc * 100:.2f}%")
```

```
Epoch 1: Train Loss: 4.3712, Val Loss: 4.3498, Train Acc: 0.9489, Val Acc: 0.9593
Epoch 2: Train Loss: 4.3624, Val Loss: 4.3419, Train Acc: 0.9496, Val Acc: 0.9593
Epoch 3: Train Loss: 4.3536, Val Loss: 4.3341, Train Acc: 0.9502, Val Acc: 0.9597
Epoch 4: Train Loss: 4.3449, Val Loss: 4.3263, Train Acc: 0.9507, Val Acc: 0.9597
Epoch 5: Train Loss: 4.3363, Val Loss: 4.3185, Train Acc: 0.9514, Val Acc: 0.9597
Epoch 6: Train Loss: 4.3277, Val Loss: 4.3107, Train Acc: 0.9518, Val Acc: 0.9597
Epoch 7: Train Loss: 4.3193, Val Loss: 4.3030, Train Acc: 0.9525, Val Acc: 0.9605
Epoch 8: Train Loss: 4.3110, Val Loss: 4.2954, Train Acc: 0.9535, Val Acc: 0.9601
Epoch 9: Train Loss: 4.3026, Val Loss: 4.2876, Train Acc: 0.9542, Val Acc: 0.9601
Epoch 10: Train Loss: 4.2943, Val Loss: 4.2801, Train Acc: 0.9545, Val Acc: 0.9601
```




```
Test Loss: 4.2622, Test Accuracy: 95.32%
```

Rename notebook