```
import numpy as np
import torch
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt
from torchvision import transforms

import torch.nn as nn       # torch.nn module, contains classes and functions to he

import torch.optim as optim # provides various optimization algorithms, such as SGD
from torch.utils.data import DataLoader, TensorDataset # Dataloader - helps to loac

from scipy.special import softmax
from sklearn.metrics import confusion_matrix, accuracy_score
```

Downloading the MNIST digit datasets

```
# Ensuring one-hot format
def one_hot_encoder(x):
  temp_array = np.zeros(10, dtype=float) # numpy arrays of zeros with length 10, 0
  temp_array[x] = 1 # element at index x in the temp array set to 1
  return temp_array

# To normalize the input
def transform(x):
  return np.array(x)/255.0
```

```
train_data = datasets.MNIST(root='./data', train = True , download=True, transform=
test_data = datasets.MNIST(root='./data', train = False ,download=True, transform=t
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ul
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ul
100%|██████████| 9912422/9912422 [00:02<00:00, 4518298.03it/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ul
```

```
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-u
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ul
100%|██████████| 28881/28881 [00:00<00:00, 56805.19it/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-uby
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-uby
100%|██████████| 1648877/1648877 [00:01<00:00, 1245651.84it/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-uby
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-uby
100%|██████████| 4542/4542 [00:00<00:00, 4021644.24it/s]Extracting ./data/MNIS
```

```
len(train_data)
```

```
60000
```

```
len(test_data)
```

```
10000
```

```
# Visualizing the data
fig, axes = plt.subplots(2, 5, figsize=(6, 4))  # 2 rows, 5 columns

for i in range(10):           # Loop through the first 10 images
  ax = axes[i // 5, i % 5]  # Determine the position of the subplot (row, column)

  ax.imshow(train_data.data[i], cmap='gray') # Display each image in grayscale
  ax.set_title(f"Index: {i}\nLabel: {train_data.targets[i].item()}")
  ax.axis('off')

plt.tight_layout() # Adjust layout to prevent overlap of titles
plt.show()
```


```
# organize the data in batches
# want to pass samples in “minibatches”, reshuffle the data at every epoch to red
train_dataloader = DataLoader(train_data, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64, shuffle=True)
```

```
len(train_dataloader)
```

    938

```
len(test_dataloader)
```

    157

Code from scratch

```
input_layer = train_data.data[i].flatten().shape[0]
hidden1_layer = 500
hidden2_layer = 250
hidden3_layer = 100
out_layer = train_data.train_labels.unique().shape[0]

layers_dims = [input_layer, hidden1_layer, hidden2_layer, hidden3_layer, out_laye
```

```
def initialize_parameters(layer_dimensions, initial):
  parameters = {}
  num_layers = len(layer_dimensions) # number of layers in the network

  for layer in range(1, num_layers):
    if initial == "glorot":  # glorot intitalization
      M = np.sqrt(6*(1/(layer_dimensions[layer]+layer_dimensions[layer-1])))
      parameters['W' + str(layer)] = np.random.uniform(low = -M, high = M, size =
```

```
      parameters['b' + str(layer)] = np.zeros((layer_dimensions[layer], 1))

    elif initial == "random": # Random Initialization
      parameters['W' + str(layer)] = np.random.randn(layer_dimensions[layer], lay
      parameters['b' + str(layer)] = np.zeros((layer_dimensions[layer], 1))

    else: # Zero Initialization
      parameters['W' + str(layer)] = np.zeros((layer_dimensions[layer], layer_dim
      parameters['b' + str(layer)] = np.zeros((layer_dimensions[layer], 1))

    assert(parameters['W' + str(layer)].shape == (layer_dimensions[layer], layer_
    assert(parameters['b' + str(layer)].shape == (layer_dimensions[layer], 1))

  return parameters
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/datasets/mnist.py:66: Use
  warnings.warn("train_labels has been renamed targets")
```

Activation Function

```
# Sigmoid activation function
def sigmoid(x):
  return 1.0 / (1.0 + np.exp(-x))

# Softmax function for output probabilities
# def softmax(x):
#     exps = np.exp(x - np.max(x, axis=0))
#     return exps / exps.sum(axis=0)

# def softmax(Z, axis=None):
#   exp_Z = np.exp(Z - np.max(Z))  # Subtract max for numerical stability
#   return exp_Z / np.sum(exp_Z, axis=axis, keepdims=True)

# ReLu
def relu(x):
  return np.maximum(x, 0)

# tanh activation function
def tanh(x):
  return (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x))
```

Forward Propagation

```
def forward_propagation(input_data, parameters, activation_function):
```

```
def forward_propagation(input_data, parameters, activation_function):
    forward_propagation = {}
    num_layers = int(len(parameters) / 2)  # Total number of layers (excluding in

    forward_propagation['Z1'] = np.dot(parameters['W1'], input_data) + parameters

    # Loop through layers 2 to (num_layers - 1) (hidden layers)
    for layer in range(2, num_layers):
        forward_propagation['A' + str(layer - 1)] = activation_function(forward_p

        # Linear transformation for the current layer
        forward_propagation['Z' + str(layer)] = np.dot(parameters['W' + str(layer

    # final layer's activation using softmax
    forward_propagation['A' + str(num_layers - 1)] = activation_function(forward_
    forward_propagation['Z' + str(num_layers)] = np.dot(parameters['W' + str(num_

    forward_propagation['A' + str(num_layers)] = softmax(forward_propagation['Z'
    cache = (forward_propagation, parameters) # Store forward pass results and pa

    return forward_propagation['A' + str(num_layers)], cache
```

Backpropagation

```
def back_propagation(input, labels, cache):

  num_examples = input.shape[1]  # Number of examples in the batch (m)
  forward_propagation, parameters = cache # Extract activations and parameters fr
  num_layers = len(parameters) // 2  # Number of layers (assuming W1, b1, ..., WL

  # Initialize a dictionary to store gradients
  grads = {}
  grads['dZ' + str(num_layers)] = forward_propagation['A' + str(num_layers)] - la

  # Backpropagate through all hidden layers (in reverse order)
  for layer in range(num_layers-1, 0, -1):

    # Compute gradients for weights and biases
    grads['dW' + str(layer+1)] = (1. / num_examples) * np.dot(grads['dZ' + str(la
    grads['db' + str(layer+1)] = (1. / num_examples) * np.sum(grads['dZ' + str(la
    grads['dA'+ str(layer)] = np.dot(parameters['W'+ str(layer+1)].T, grads['dZ'+
    grads['dZ' + str(layer)] = grads['dA' + str(layer)] * forward_propagation['A'

  grads['dW1'] = 1./num_examples * np.dot(grads['dZ1'], input.T)
  grads['db1'] = 1./num_examples * np.sum(grads['dZ1'], axis=1, keepdims = True)
```

```
  return grads
```

Update parameters

```
def update_parameters(parameters, grads, learning_rate, lambd=0):
  num_layers = len(parameters) // 2  # Number of layers in the network

  for layer in range(num_layers):
    # Update weights with regularization (if lambd > 0)
    parameters["W" + str(layer + 1)] -= (learning_rate * (grads["dW" + str(layer

    # Update biases (biases are not regularized)
    parameters["b" + str(layer + 1)] -= (learning_rate * grads["db" + str(layer +

  return parameters
```

Cost Funtion

```
def cross_entropy_cost(predictions, labels):
  num_examples = labels.shape[1]

  # Compute cross-entropy loss
  loss_per_example = np.multiply(-np.log(predictions), labels) + np.multiply(-np.
  average_cost = 1. / num_examples * np.sum(loss_per_example)

  return average_cost
```

```
train_dataloader = DataLoader(train_data, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64, shuffle=True)
```

Accuracy and Confusion matrix

```
def accuracy(parameter, test_data, function):
  size = test_data.data.shape[0] # total number of test samples
  img_size = test_data.data.shape[1] * test_data.data.shape[2]

  test_dataloader = next(iter(DataLoader(test_data, batch_size=size, shuffle=True
  X = np.swapaxes(np.array(test_dataloader[0]),0,2).reshape(img_size, size) # Res
```

```
  X = np.swapaxes(np.array(test_dataloader[0]),0,2).reshape(img_size, size) # Res

  pred = np.swapaxes(forward_propagation(X, parameter, function)[0], 0, 1) # forw
  Y = np.array(test_dataloader[1]) # true labels

  # Compute accuracy by comparing the predicted and true labels
  accuracy = accuracy_score(np.argmax(Y, axis=1), np.argmax(pred, axis=1))
  return accuracy


def confusion_mat(parameter, test_data, function):
  size = test_data.data.shape[0] # total number of test samples
  img_size = test_data.data.shape[1] * test_data.data.shape[2]

  test_dataloader = next(iter(DataLoader(test_data, batch_size=size, shuffle=True
  X = np.swapaxes(np.array(test_dataloader[0]),0,2).reshape(img_size, size)

  pred = np.swapaxes(forward_propagation(X, parameter, function)[0], 0, 1)
  Y = np.array(test_dataloader[1])

  confu_matrix = confusion_matrix(np.argmax(Y, axis=1), np.argmax(pred, axis=1))
  return confu_matrix
```

Training the model

```
def model(train_dataloader, test_data, batch_size=64, learning_rate=0.01, epoch=1
    grads = {}
    train_costs = []  # To store training costs
    test_costs = []   # To store test costs
    layers_dims = [input_layer, hidden1_layer, hidden2_layer, hidden3_layer, out_
    parameters = initialize_parameters(layers_dims, initial)
    count = 0

    for i in range(epoch):
        for (batch_idx, batch) in enumerate(train_dataloader):
            batch_x, batch_y = batch
            X = np.swapaxes(np.array(batch_x), 0, 2).reshape(batch_x.shape[1]*bat
            Y = np.swapaxes(np.array(batch_y), 0, 1)

            # Forward propagation
            a3, cache = forward_propagation(X, parameters, function)
            train_cost = cross_entropy_cost(a3, Y)

            # Backward propagation and parameter update
            grads = back_propagation(X, Y, cache)
            parameters = update_parameters(parameters, grads, learning_rate, lamb
```

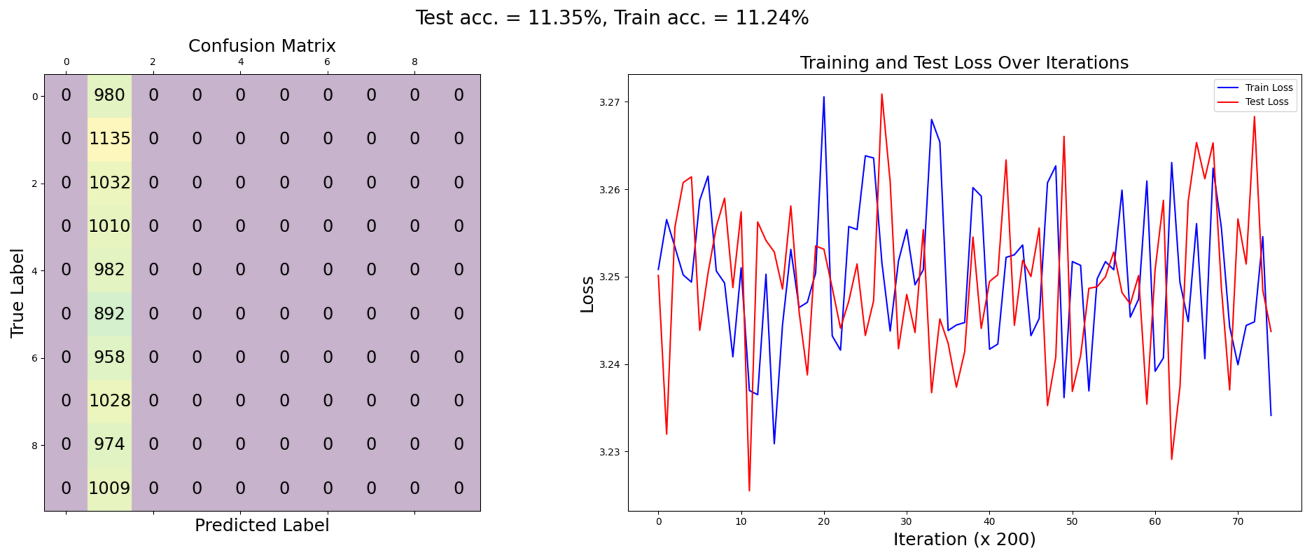
```
            if batch_idx % 200 == 0:
                train_costs.append(train_cost)

                # Calculate test loss at every 200th batch
                test_dataloader = next(iter(DataLoader(test_data, batch_size=batc
                test_x = np.swapaxes(np.array(test_dataloader[0]), 0, 2).reshape(
                test_y = np.swapaxes(np.array(test_dataloader[1]), 0, 1)
                test_a3, _ = forward_propagation(test_x, parameters, function)
                test_cost = cross_entropy_cost(test_a3, test_y)
                test_costs.append(test_cost)

            if print_cost and batch_idx % 200 == 0:
                print(f"Cost after epoch {i}, iteration {batch_idx}: Train Cost:

    return parameters, train_costs, test_costs



def plotting(parameters, test_data, train_data, function):
    # test and train accuracy, passing the 'function' parameter
    test_acc = accuracy(parameters[0], test_data, function)
    train_acc = accuracy(parameters[0], train_data, function)

    conf_matrix = confusion_mat(parameters[0], test_data, function) # confusion m

    fig, (ax, bx) = plt.subplots(1, 2, figsize=(20, 8)) # two subplots: for the c

    # confusion matrix
    ax.matshow(conf_matrix, cmap='viridis', alpha=0.3)
    for i in range(conf_matrix.shape[0]):
        for j in range(conf_matrix.shape[1]):
            ax.text(x=j, y=i, s=conf_matrix[i, j], va='center', ha='center', size

    ax.set_xlabel('Predicted Label', fontsize=18)
    ax.set_ylabel('True Label', fontsize=18)
    ax.set_title('Confusion Matrix', fontsize=18)

    # cost curve over iterations (training and test)
    bx.plot(range(0, len(parameters[1])), parameters[1], label='Train Loss', colo
    bx.plot(range(0, len(parameters[2])), parameters[2], label='Test Loss', color

    bx.set_xlabel('Iteration (x 200)', fontsize=18)
    bx.set_ylabel('Loss', fontsize=18)
    bx.set_title('Training and Test Loss Over Iterations', fontsize=18)
    bx.legend()

    # Combine test and train accuracy in a label for the plot
    label = f"Test acc. = {test_acc * 100:.2f}%, Train acc. = {train_acc * 100:.2
    plt.suptitle(label, fontsize=20)

    plt.tight_layout()
```

```
    plt.show()
```

```
# Dictionary to store learned parameters for different models
learned_parameters = {}

learning_rate = 0.01
lambd = 0
epoch = 15
batch_size = 64
initial = "zero"
train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True)

#  model name (key) based on training parameters
model_name = "Epoch=" + str(epoch) + ",alpha=" + str(learning_rate) + ",Regulariz
print("Model Key: " + model_name)

# Train the model and store the learned parameters
learned_parameters[model_name] = model(train_dataloader, test_data, batch_size=ba

# Find the model with 'zero' initialization dynamically
i = [key for key in learned_parameters.keys() if "Initilization=zero" in key][0]

# Plotting the losses and confusion matrix
plotting(learned_parameters[i], test_data, train_data, sigmoid)
```

```
Model Key: Epoch=15,alpha=0.01,Regularization=0,Batch=64,Initilization=zero
Cost after epoch 0, iteration 0: Train Cost: 3.250829733914482, Test Cost: 3.
Cost after epoch 0, iteration 200: Train Cost: 3.2565329875540816, Test Cost:
Cost after epoch 0, iteration 400: Train Cost: 3.25339618727332, Test Cost: 3
Cost after epoch 0, iteration 600: Train Cost: 3.2502085364189286, Test Cost:
Cost after epoch 0, iteration 800: Train Cost: 3.249378139533564, Test Cost:
Cost after epoch 1, iteration 0: Train Cost: 3.2587339809720453, Test Cost: 3
Cost after epoch 1, iteration 200: Train Cost: 3.261504600220028, Test Cost:
Cost after epoch 1, iteration 400: Train Cost: 3.2506611852155576, Test Cost:
Cost after epoch 1, iteration 600: Train Cost: 3.2492749719657006, Test Cost:
Cost after epoch 1, iteration 800: Train Cost: 3.240837990512092, Test Cost:
Cost after epoch 2, iteration 0: Train Cost: 3.2510219206202167, Test Cost: 3
Cost after epoch 2, iteration 200: Train Cost: 3.2369782424003755, Test Cost:
Cost after epoch 2, iteration 400: Train Cost: 3.2365079589847077, Test Cost:
Cost after epoch 2, iteration 600: Train Cost: 3.2502773810978045, Test Cost:
Cost after epoch 2, iteration 800: Train Cost: 3.2308956296547304, Test Cost:
Cost after epoch 3, iteration 0: Train Cost: 3.244505698716834, Test Cost: 3.
Cost after epoch 3, iteration 200: Train Cost: 3.253117419061061, Test Cost:
Cost after epoch 3, iteration 400: Train Cost: 3.246474807415987, Test Cost:
Cost after epoch 3, iteration 600: Train Cost: 3.247069954261385, Test Cost:
Cost after epoch 3, iteration 800: Train Cost: 3.250439553604092, Test Cost:
Cost after epoch 4, iteration 0: Train Cost: 3.2705736412651394, Test Cost: 3
Cost after epoch 4, iteration 200: Train Cost: 3.2432364572929075, Test Cost:
Cost after epoch 4, iteration 400: Train Cost: 3.2415968525848053, Test Cost:
```

```
Cost after epoch 4, iteration 600: Train Cost: 3.2557381385384803, Test Cost:
Cost after epoch 4, iteration 800: Train Cost: 3.2553914203856937, Test Cost:
Cost after epoch 5, iteration 0: Train Cost: 3.2638263538159835, Test Cost: 3
Cost after epoch 5, iteration 200: Train Cost: 3.263572930310024, Test Cost:
Cost after epoch 5, iteration 400: Train Cost: 3.2516063762733536, Test Cost:
Cost after epoch 5, iteration 600: Train Cost: 3.24378557158859, Test Cost: 3
Cost after epoch 5, iteration 800: Train Cost: 3.2517871678898347, Test Cost:
Cost after epoch 6, iteration 0: Train Cost: 3.2553969192385055, Test Cost: 3
Cost after epoch 6, iteration 200: Train Cost: 3.249067322358443, Test Cost:
Cost after epoch 6, iteration 400: Train Cost: 3.2507997251688963, Test Cost:
Cost after epoch 6, iteration 600: Train Cost: 3.2679872141356405, Test Cost:
Cost after epoch 6, iteration 800: Train Cost: 3.2654015156477487, Test Cost:
Cost after epoch 7, iteration 0: Train Cost: 3.243842265610249, Test Cost: 3.
Cost after epoch 7, iteration 200: Train Cost: 3.2444746297080544, Test Cost:
Cost after epoch 7, iteration 400: Train Cost: 3.2447571975594056, Test Cost:
Cost after epoch 7, iteration 600: Train Cost: 3.2601863861281783, Test Cost:
Cost after epoch 7, iteration 800: Train Cost: 3.2592054671778254, Test Cost:
Cost after epoch 8, iteration 0: Train Cost: 3.2417023514385246, Test Cost: 3
Cost after epoch 8, iteration 200: Train Cost: 3.2422991014027702, Test Cost:
Cost after epoch 8, iteration 400: Train Cost: 3.2522057213969284, Test Cost:
Cost after epoch 8, iteration 600: Train Cost: 3.2525065806625526, Test Cost:
Cost after epoch 8, iteration 800: Train Cost: 3.253627265661895, Test Cost:
Cost after epoch 9, iteration 0: Train Cost: 3.2432537834416446, Test Cost: 3
Cost after epoch 9, iteration 200: Train Cost: 3.245224139193847, Test Cost:
Cost after epoch 9, iteration 400: Train Cost: 3.260735751163527, Test Cost:
Cost after epoch 9, iteration 600: Train Cost: 3.2626711805087245, Test Cost:
Cost after epoch 9, iteration 800: Train Cost: 3.2361713721684056, Test Cost:
Cost after epoch 10, iteration 0: Train Cost: 3.251730131868171, Test Cost: 3
Cost after epoch 10, iteration 200: Train Cost: 3.2512979415397245, Test Cost
Cost after epoch 10, iteration 400: Train Cost: 3.236940366863079, Test Cost:
Cost after epoch 10, iteration 600: Train Cost: 3.249757300942335, Test Cost:
Cost after epoch 10, iteration 800: Train Cost: 3.2517237467195237, Test Cost
Cost after epoch 11, iteration 0: Train Cost: 3.250793497878294, Test Cost: 3
Cost after epoch 11, iteration 200: Train Cost: 3.2598988205724284, Test Cost
Cost after epoch 11, iteration 400: Train Cost: 3.245367522396654, Test Cost:
Cost after epoch 11, iteration 600: Train Cost: 3.247420548133018, Test Cost:
Cost after epoch 11, iteration 800: Train Cost: 3.260941535924095, Test Cost:
Cost after epoch 12, iteration 0: Train Cost: 3.2391822508067234, Test Cost:
Cost after epoch 12, iteration 200: Train Cost: 3.240718085810979, Test Cost:
Cost after epoch 12, iteration 400: Train Cost: 3.2630533951546132, Test Cost
Cost after epoch 12, iteration 600: Train Cost: 3.2493459488996237, Test Cost
Cost after epoch 12, iteration 800: Train Cost: 3.2448709053659766, Test Cost
Cost after epoch 13, iteration 0: Train Cost: 3.256068252002303 4, Test Cost:
Cost after epoch 13, iteration 200: Train Cost: 3.240626786081916, Test Cost:
Cost after epoch 13, iteration 400: Train Cost: 3.262408943066613, Test Cost:
Cost after epoch 13, iteration 600: Train Cost: 3.2556142647300397, Test Cost
Cost after epoch 13, iteration 800: Train Cost: 3.2442697201850006, Test Cost
Cost after epoch 14, iteration 0: Train Cost: 3.239932910908842, Test Cost: 3
Cost after epoch 14, iteration 200: Train Cost: 3.2444265194616166, Test Cost
Cost after epoch 14, iteration 400: Train Cost: 3.2448425828577507, Test Cost
Cost after epoch 14, iteration 600: Train Cost: 3.2545820267057937, Test Cost
Cost after epoch 14, iteration 800: Train Cost: 3.234142592868748, Test Cost:
```



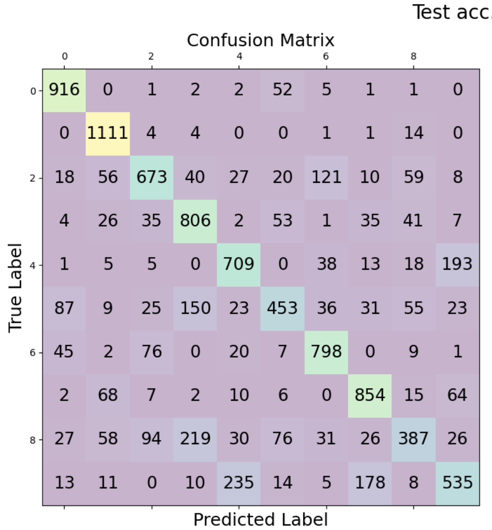

```
print("Available model keys:", learned_parameters.keys())


# Dictionary to store learned parameters for different models
learned_parameters = {}

learning_rate = 0.01
lambd = 0
epoch = 15
batch_size = 64
initial = "random"
train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True)

# Create a model name (key) based on training parameters
model_name = "Epoch=" + str(epoch) + ",alpha=" + str(learning_rate) + ",Regulariz
print("Model Key: " + model_name)
```

```
print("Model Key: " + model_name)

# Train the model and store the learned parameters
learned_parameters[model_name] = model(train_dataloader, test_data, batch_size=ba

# Find the model with 'zero' initialization dynamically
i = [key for key in learned_parameters.keys() if "Initilization=random" in key][0

# Plotting the losses and confusion matrix
plotting(learned_parameters[i], test_data, train_data, sigmoid)
```

```
Model Key: Epoch=15,alpha=0.01,Regularization=0,Batch=64,Initilization=random
Cost after epoch 0, iteration 0: Train Cost: 3.519360144013942, Test Cost: 3.
Cost after epoch 0, iteration 200: Train Cost: 3.2592092895664875, Test Cost:
Cost after epoch 0, iteration 400: Train Cost: 3.2743739861694334, Test Cost:
Cost after epoch 0, iteration 600: Train Cost: 3.240629678421011, Test Cost:
Cost after epoch 0, iteration 800: Train Cost: 3.247208632769028, Test Cost:
Cost after epoch 1, iteration 0: Train Cost: 3.251322893857357, Test Cost: 3.
Cost after epoch 1, iteration 200: Train Cost: 3.248305123739125, Test Cost:
Cost after epoch 1, iteration 400: Train Cost: 3.2402493965780113, Test Cost:
Cost after epoch 1, iteration 600: Train Cost: 3.2403541162958276, Test Cost:
Cost after epoch 1, iteration 800: Train Cost: 3.2649508866171786, Test Cost:
Cost after epoch 2, iteration 0: Train Cost: 3.2351723504900116, Test Cost: 3
Cost after epoch 2, iteration 200: Train Cost: 3.255519698991239, Test Cost:
Cost after epoch 2, iteration 400: Train Cost: 3.2400170213512696, Test Cost:
Cost after epoch 2, iteration 600: Train Cost: 3.2236533755368066, Test Cost:
Cost after epoch 2, iteration 800: Train Cost: 3.2384612170484663, Test Cost:
Cost after epoch 3, iteration 0: Train Cost: 3.250774829401215, Test Cost: 3.
Cost after epoch 3, iteration 200: Train Cost: 3.243591663732648, Test Cost:
Cost after epoch 3, iteration 400: Train Cost: 3.2605030750862265, Test Cost:
Cost after epoch 3, iteration 600: Train Cost: 3.240769044081449, Test Cost:
Cost after epoch 3, iteration 800: Train Cost: 3.2300699680217884, Test Cost:
Cost after epoch 4, iteration 0: Train Cost: 3.22949180444846, Test Cost: 3.2
Cost after epoch 4, iteration 200: Train Cost: 3.221596645078888, Test Cost:
Cost after epoch 4, iteration 400: Train Cost: 3.216273728328467, Test Cost:
Cost after epoch 4, iteration 600: Train Cost: 3.206606276219917, Test Cost:
Cost after epoch 4, iteration 800: Train Cost: 3.218671107929225, Test Cost:
Cost after epoch 5, iteration 0: Train Cost: 3.2276504423580072, Test Cost: 3
Cost after epoch 5, iteration 200: Train Cost: 3.200078459126675, Test Cost:
Cost after epoch 5, iteration 400: Train Cost: 3.198845430652609, Test Cost:
Cost after epoch 5, iteration 600: Train Cost: 3.2214099413411716, Test Cost:
Cost after epoch 5, iteration 800: Train Cost: 3.194122192510529, Test Cost:
Cost after epoch 6, iteration 0: Train Cost: 3.219181911403849, Test Cost: 3.
Cost after epoch 6, iteration 200: Train Cost: 3.199737977403298, Test Cost:
Cost after epoch 6, iteration 400: Train Cost: 3.2141775208711447, Test Cost:
Cost after epoch 6, iteration 600: Train Cost: 3.184978186477246, Test Cost:
Cost after epoch 6, iteration 800: Train Cost: 3.17694463172506, Test Cost: 3
Cost after epoch 7, iteration 0: Train Cost: 3.192697750015851, Test Cost: 3.
Cost after epoch 7, iteration 200: Train Cost: 3.1747465394554055, Test Cost:
Cost after epoch 7, iteration 400: Train Cost: 3.1616540068459016, Test Cost:
Cost after epoch 7, iteration 600: Train Cost: 3.1468068769118576, Test Cost:
```

```
Cost after epoch 7, iteration 800: Train Cost: 3.102612807470501, Test Cost:
Cost after epoch 8, iteration 0: Train Cost: 3.0960598950060203, Test Cost: 3
Cost after epoch 8, iteration 200: Train Cost: 3.129777733313836, Test Cost:
Cost after epoch 8, iteration 400: Train Cost: 3.0781711663219706, Test Cost:
Cost after epoch 8, iteration 600: Train Cost: 3.065260948288906, Test Cost:
Cost after epoch 8, iteration 800: Train Cost: 2.9411704134188845, Test Cost:
Cost after epoch 9, iteration 0: Train Cost: 3.0870631932936936, Test Cost: 3
Cost after epoch 9, iteration 200: Train Cost: 2.9115998923804276, Test Cost:
Cost after epoch 9, iteration 400: Train Cost: 2.8625525908072564, Test Cost:
Cost after epoch 9, iteration 600: Train Cost: 2.9378300535248845, Test Cost:
Cost after epoch 9, iteration 800: Train Cost: 2.7744459179220398, Test Cost:
Cost after epoch 10, iteration 0: Train Cost: 2.7370973450113256, Test Cost:
Cost after epoch 10, iteration 200: Train Cost: 2.654605503744629, Test Cost:
Cost after epoch 10, iteration 400: Train Cost: 2.7476577669045894, Test Cost
Cost after epoch 10, iteration 600: Train Cost: 2.3432344845185815, Test Cost
Cost after epoch 10, iteration 800: Train Cost: 2.6261285499693496, Test Cost
Cost after epoch 11, iteration 0: Train Cost: 2.5325927034579996, Test Cost:
Cost after epoch 11, iteration 200: Train Cost: 2.6143919967563867, Test Cost
Cost after epoch 11, iteration 400: Train Cost: 2.3483270246061085, Test Cost
Cost after epoch 11, iteration 600: Train Cost: 2.2434750206051755, Test Cost
Cost after epoch 11, iteration 800: Train Cost: 2.3002819066392264, Test Cost
Cost after epoch 12, iteration 0: Train Cost: 2.23326635011668, Test Cost: 2.
Cost after epoch 12, iteration 200: Train Cost: 2.1951309924787927, Test Cost
Cost after epoch 12, iteration 400: Train Cost: 2.2179730790806413, Test Cost
Cost after epoch 12, iteration 600: Train Cost: 2.085555556381551, Test Cost:
Cost after epoch 12, iteration 800: Train Cost: 1.9908395439627058, Test Cost
Cost after epoch 13, iteration 0: Train Cost: 1.9660988933585444, Test Cost:
Cost after epoch 13, iteration 200: Train Cost: 1.7572401071468056, Test Cost
Cost after epoch 13, iteration 400: Train Cost: 1.8489943509496218, Test Cost
Cost after epoch 13, iteration 600: Train Cost: 1.7787855668157846, Test Cost
Cost after epoch 13, iteration 800: Train Cost: 1.8060532651806351, Test Cost
Cost after epoch 14, iteration 0: Train Cost: 1.866086059309159, Test Cost: 1
Cost after epoch 14, iteration 200: Train Cost: 1.711832784158902, Test Cost:
Cost after epoch 14, iteration 400: Train Cost: 1.4451838014382679, Test Cost
Cost after epoch 14, iteration 600: Train Cost: 1.6388443038835072, Test Cost
Cost after epoch 14, iteration 800: Train Cost: 1.6470055813176687, Test Cost
```

Test acc. = 72.42%, Train acc. = 71.92%



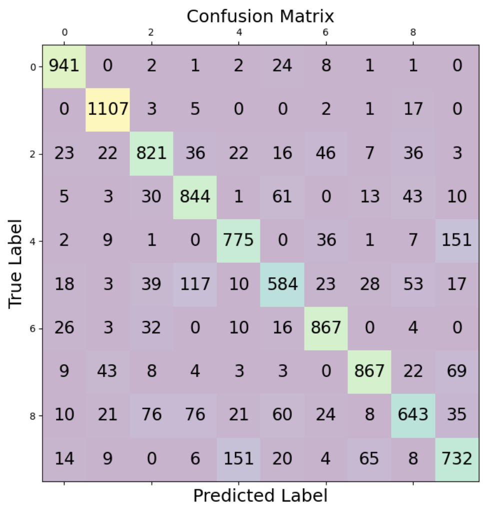
```
# Dictionary to store learned parameters for different models
learned_parameters = {}

learning_rate = 0.01
lambd = 0
epoch = 15
batch_size = 64
initial = "glorot"
train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True)

# Create a model name (key) based on training parameters
model_name = "Epoch=" + str(epoch) + ",alpha=" + str(learning_rate) + ",Regulariz
print("Model Key: " + model_name)

# Train the model and store the learned parameters
learned_parameters[model_name] = model(train_dataloader, test_data, batch_size=ba

# Find the model with 'zero' initialization dynamically
i = [key for key in learned_parameters.keys() if "Initilization=glorot" in key][0

# Plotting the losses and confusion matrix
plotting(learned_parameters[i], test_data, train_data, sigmoid)
```

```
Model Key: Epoch=15,alpha=0.01,Regularization=0,Batch=64,Initilization=glorot
Cost after epoch 0, iteration 0: Train Cost: 3.364798570254992, Test Cost: 3.
Cost after epoch 0, iteration 200: Train Cost: 3.243369220322021, Test Cost:
Cost after epoch 0, iteration 400: Train Cost: 3.239952774484645, Test Cost:
Cost after epoch 0, iteration 600: Train Cost: 3.260996398545833, Test Cost:
Cost after epoch 0, iteration 800: Train Cost: 3.2458382898675375, Test Cost:
Cost after epoch 1, iteration 0: Train Cost: 3.236380140797255, Test Cost: 3.
```

```
Cost after epoch 1, iteration 200: Train Cost: 3.2324376071011782, Test Cost:
Cost after epoch 1, iteration 400: Train Cost: 3.226601522035077, Test Cost: 3
Cost after epoch 1, iteration 600: Train Cost: 3.2205031569197335, Test Cost:
Cost after epoch 1, iteration 800: Train Cost: 3.2238699755178204, Test Cost:
Cost after epoch 2, iteration 0: Train Cost: 3.2414518167961512, Test Cost: 3
Cost after epoch 2, iteration 200: Train Cost: 3.2321656846832334, Test Cost:
Cost after epoch 2, iteration 400: Train Cost: 3.2239726115364276, Test Cost:
Cost after epoch 2, iteration 600: Train Cost: 3.2150790699035756, Test Cost:
Cost after epoch 2, iteration 800: Train Cost: 3.175178192896444, Test Cost: 3
Cost after epoch 3, iteration 0: Train Cost: 3.220662568149917, Test Cost: 3.1
Cost after epoch 3, iteration 200: Train Cost: 3.220254106918895, Test Cost: 3
Cost after epoch 3, iteration 400: Train Cost: 3.172929211916992, Test Cost: 3
Cost after epoch 3, iteration 600: Train Cost: 3.2039107472329187, Test Cost:
Cost after epoch 3, iteration 800: Train Cost: 3.2133469359946707, Test Cost:
Cost after epoch 4, iteration 0: Train Cost: 3.150886398712264, Test Cost: 3.1
Cost after epoch 4, iteration 200: Train Cost: 3.1625915069611383, Test Cost:
Cost after epoch 4, iteration 400: Train Cost: 3.122943879667715, Test Cost: 3
Cost after epoch 4, iteration 600: Train Cost: 3.1258681806245296, Test Cost:
Cost after epoch 4, iteration 800: Train Cost: 3.102560800918038, Test Cost: 3
Cost after epoch 5, iteration 0: Train Cost: 3.1139502303168825, Test Cost: 3
Cost after epoch 5, iteration 200: Train Cost: 3.071085648157956, Test Cost: 3
Cost after epoch 5, iteration 400: Train Cost: 3.026216486670541, Test Cost: 3
Cost after epoch 5, iteration 600: Train Cost: 2.9897686777214165, Test Cost:
Cost after epoch 5, iteration 800: Train Cost: 2.98471148072211, Test Cost: 2
Cost after epoch 6, iteration 0: Train Cost: 2.9841444740728136, Test Cost: 2
Cost after epoch 6, iteration 200: Train Cost: 2.923937104057952, Test Cost: 2
Cost after epoch 6, iteration 400: Train Cost: 2.7635043082198227, Test Cost:
Cost after epoch 6, iteration 600: Train Cost: 2.837491940652071, Test Cost: 2
Cost after epoch 6, iteration 800: Train Cost: 2.5737259451193095, Test Cost:
Cost after epoch 7, iteration 0: Train Cost: 2.5773732118465196, Test Cost: 2
Cost after epoch 7, iteration 200: Train Cost: 2.489964664775587, Test Cost: 2
Cost after epoch 7, iteration 400: Train Cost: 2.6214275383675503, Test Cost:
Cost after epoch 7, iteration 600: Train Cost: 2.3277629915088554, Test Cost:
Cost after epoch 7, iteration 800: Train Cost: 2.411285462875298, Test Cost: 2
Cost after epoch 8, iteration 0: Train Cost: 2.1187750411800987, Test Cost: 2
Cost after epoch 8, iteration 200: Train Cost: 2.188796742313521, Test Cost: 2
Cost after epoch 8, iteration 400: Train Cost: 2.122692765029715, Test Cost: 2
Cost after epoch 8, iteration 600: Train Cost: 2.2269349602806363, Test Cost:
Cost after epoch 8, iteration 800: Train Cost: 2.2665076690909496, Test Cost:
Cost after epoch 9, iteration 0: Train Cost: 1.819630981909926, Test Cost: 1.8
Cost after epoch 9, iteration 200: Train Cost: 1.8358508026996296, Test Cost:
Cost after epoch 9, iteration 400: Train Cost: 1.7865112827998202, Test Cost:
Cost after epoch 9, iteration 600: Train Cost: 1.689358425038149, Test Cost: 1
Cost after epoch 9, iteration 800: Train Cost: 1.6243746188647665, Test Cost:
Cost after epoch 10, iteration 0: Train Cost: 1.5695059311531983, Test Cost: 1
Cost after epoch 10, iteration 200: Train Cost: 1.6733570875096553, Test Cost
Cost after epoch 10, iteration 400: Train Cost: 1.6424057412743682, Test Cost
Cost after epoch 10, iteration 600: Train Cost: 1.590765012652721, Test Cost:
Cost after epoch 10, iteration 800: Train Cost: 1.3487691377994646, Test Cost
Cost after epoch 11, iteration 0: Train Cost: 1.2426443406924768, Test Cost: 1
Cost after epoch 11, iteration 200: Train Cost: 1.6385245855110568, Test Cost
Cost after epoch 11, iteration 400: Train Cost: 1.5341111799404428, Test Cost
Cost after epoch 11, iteration 600: Train Cost: 1.381173233100007, Test Cost:
```

```
Cost after epoch 11, iteration 600: Train Cost: 1.38117525100007, Test Cost:
Cost after epoch 11, iteration 800: Train Cost: 1.3948349617377138, Test Cost
Cost after epoch 12, iteration 0: Train Cost: 1.184987143879167, Test Cost: 1
Cost after epoch 12, iteration 200: Train Cost: 1.3971677164942475, Test Cost
Cost after epoch 12, iteration 400: Train Cost: 1.389407376157979, Test Cost:
Cost after epoch 12, iteration 600: Train Cost: 1.4541029384296784, Test Cost
Cost after epoch 12, iteration 800: Train Cost: 1.369913115201828, Test Cost:
Cost after epoch 13, iteration 0: Train Cost: 1.1003248927952103, Test Cost: 
Cost after epoch 13, iteration 200: Train Cost: 1.1441898208988162, Test Cost
Cost after epoch 13, iteration 400: Train Cost: 1.1762899301131453, Test Cost
Cost after epoch 13, iteration 600: Train Cost: 1.2444027641091537, Test Cost
Cost after epoch 13, iteration 800: Train Cost: 1.0348782177123392, Test Cost
Cost after epoch 14, iteration 0: Train Cost: 1.0915221993517819, Test Cost: 
Cost after epoch 14, iteration 200: Train Cost: 0.9763705686334501, Test Cost
Cost after epoch 14, iteration 400: Train Cost: 1.0164122596383742, Test Cost
Cost after epoch 14, iteration 600: Train Cost: 1.1955217304877415, Test Cost
Cost after epoch 14, iteration 800: Train Cost: 1.0699530915245175, Test Cost
```

Test acc. = 81.81%, Train acc. = 81.38%

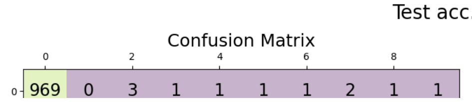


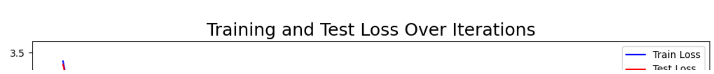
```
# Dictionary to store learned parameters for different models
learned_parameters = {}

learning_rate = 0.3
lambd = 0
epoch = 15
batch_size = 64
initial = "glorot"
train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True)

# Create a model name (key) based on training parameters
model_name = "Epoch=" + str(epoch) + ",alpha=" + str(learning_rate) + ",Regulariz
print("Model Key: " + model_name)

# Train the model and store the learned parameters
learned_parameters[model_name] = model(train_dataloader, test_data, batch_size=ba

# Find the model with 'zero' initialization dynamically
i = [key for key in learned_parameters.keys() if "Initilization=glorot" in key][0

# Plotting the losses and confusion matrix
plotting(learned_parameters[i], test_data, train_data, sigmoid)
```

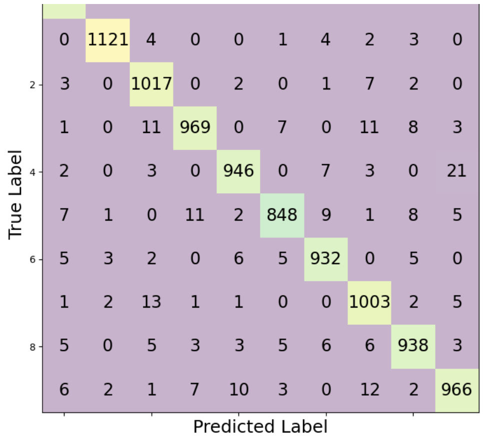
```
Model Key: Epoch=15,alpha=0.3,Regularization=0,Batch=64,Initilization=glorot
Cost after epoch 0, iteration 0: Train Cost: 3.4279112610278846, Test Cost: 3
Cost after epoch 0, iteration 200: Train Cost: 3.043075569907403, Test Cost:
Cost after epoch 0, iteration 400: Train Cost: 1.7837047317592867, Test Cost:
Cost after epoch 0, iteration 600: Train Cost: 0.8526466934907384, Test Cost:
Cost after epoch 0, iteration 800: Train Cost: 0.8478836011458577, Test Cost:
Cost after epoch 1, iteration 0: Train Cost: 1.0859324942424613, Test Cost: 0
Cost after epoch 1, iteration 200: Train Cost: 0.9667986712697354, Test Cost:
Cost after epoch 1, iteration 400: Train Cost: 0.7646010544192154, Test Cost:
Cost after epoch 1, iteration 600: Train Cost: 0.4107641543362994, Test Cost:
Cost after epoch 1, iteration 800: Train Cost: 0.5678795383436114, Test Cost:
Cost after epoch 2, iteration 0: Train Cost: 0.4655348639689104, Test Cost: 0
Cost after epoch 2, iteration 200: Train Cost: 0.33402107341090653, Test Cost
Cost after epoch 2, iteration 400: Train Cost: 0.5039842598969551, Test Cost:
Cost after epoch 2, iteration 600: Train Cost: 0.48713596697442196, Test Cost
Cost after epoch 2, iteration 800: Train Cost: 0.22633804356418114, Test Cost
Cost after epoch 3, iteration 0: Train Cost: 0.44198466352383264, Test Cost: (
Cost after epoch 3, iteration 200: Train Cost: 0.5313649585431595, Test Cost:
Cost after epoch 3, iteration 400: Train Cost: 0.7306898132739741, Test Cost:
Cost after epoch 3, iteration 600: Train Cost: 0.35640128003699745, Test Cost
Cost after epoch 3, iteration 800: Train Cost: 0.16010918656429562, Test Cost
Cost after epoch 4, iteration 0: Train Cost: 0.6085857082890895, Test Cost: 0
Cost after epoch 4, iteration 200: Train Cost: 0.4083426203292443, Test Cost:
Cost after epoch 4, iteration 400: Train Cost: 0.10303285171253318, Test Cost
Cost after epoch 4, iteration 600: Train Cost: 0.7217904662954933, Test Cost:
Cost after epoch 4, iteration 800: Train Cost: 0.2111157782297961, Test Cost:
Cost after epoch 5, iteration 0: Train Cost: 0.45499542851650193, Test Cost: (
```

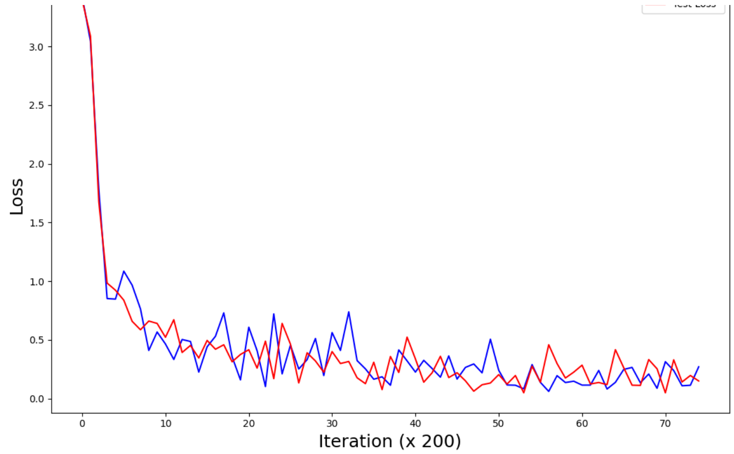
```
Cost after epoch 5, iteration 0: Train Cost: 0.43499342851650193, Test Cost:
Cost after epoch 5, iteration 200: Train Cost: 0.2527122198790355, Test Cost:
Cost after epoch 5, iteration 400: Train Cost: 0.3323603782082074, Test Cost:
Cost after epoch 5, iteration 600: Train Cost: 0.5123375609977856, Test Cost:
Cost after epoch 5, iteration 800: Train Cost: 0.1967377438991776, Test Cost:
Cost after epoch 6, iteration 0: Train Cost: 0.5629835479963595, Test Cost: 0
Cost after epoch 6, iteration 200: Train Cost: 0.41085905668935163, Test Cost
Cost after epoch 6, iteration 400: Train Cost: 0.739675194315569, Test Cost:
Cost after epoch 6, iteration 600: Train Cost: 0.3242626144014, Test Cost: 0.
Cost after epoch 6, iteration 800: Train Cost: 0.25370817747095453, Test Cost
Cost after epoch 7, iteration 0: Train Cost: 0.16541803231069235, Test Cost:
Cost after epoch 7, iteration 200: Train Cost: 0.1864007304092983, Test Cost
Cost after epoch 7, iteration 400: Train Cost: 0.11433620088875754, Test Cost
Cost after epoch 7, iteration 600: Train Cost: 0.41531647408489836, Test Cost
Cost after epoch 7, iteration 800: Train Cost: 0.3219148411020107, Test Cost
Cost after epoch 8, iteration 0: Train Cost: 0.2264875125575308, Test Cost: 0
Cost after epoch 8, iteration 200: Train Cost: 0.32641103938214144, Test Cost
Cost after epoch 8, iteration 400: Train Cost: 0.25704469141556385, Test Cost
Cost after epoch 8, iteration 600: Train Cost: 0.18331492573632843, Test Cost
Cost after epoch 8, iteration 800: Train Cost: 0.36403704239326284, Test Cost
Cost after epoch 9, iteration 0: Train Cost: 0.16684915511478385, Test Cost:
Cost after epoch 9, iteration 200: Train Cost: 0.2660924267098522, Test Cost:
Cost after epoch 9, iteration 400: Train Cost: 0.29578180981310176, Test Cost
Cost after epoch 9, iteration 600: Train Cost: 0.22013058138237349, Test Cost
Cost after epoch 9, iteration 800: Train Cost: 0.5062187529919551, Test Cost:
Cost after epoch 10, iteration 0: Train Cost: 0.2413533852917663, Test Cost:
Cost after epoch 10, iteration 200: Train Cost: 0.11664147108362725, Test Cos
Cost after epoch 10, iteration 400: Train Cost: 0.11459160940609103, Test Cos
Cost after epoch 10, iteration 600: Train Cost: 0.08395675303468819, Test Cos
Cost after epoch 10, iteration 800: Train Cost: 0.29013218441464744, Test Cos
Cost after epoch 11, iteration 0: Train Cost: 0.13868324681729272, Test Cost:
Cost after epoch 11, iteration 200: Train Cost: 0.061830460381866195, Test Co
Cost after epoch 11, iteration 400: Train Cost: 0.19603326446697655, Test Cos
Cost after epoch 11, iteration 600: Train Cost: 0.13739929124884723, Test Cos
Cost after epoch 11, iteration 800: Train Cost: 0.14890399085668682, Test Cos
Cost after epoch 12, iteration 0: Train Cost: 0.11549667283436582, Test Cost:
Cost after epoch 12, iteration 200: Train Cost: 0.11571769382259159, Test Cos
Cost after epoch 12, iteration 400: Train Cost: 0.24028019627100194, Test Cos
Cost after epoch 12, iteration 600: Train Cost: 0.08223351536711809, Test Cos
Cost after epoch 12, iteration 800: Train Cost: 0.13907117650352216, Test Cos
Cost after epoch 13, iteration 0: Train Cost: 0.24871844365086865, Test Cost:
Cost after epoch 13, iteration 200: Train Cost: 0.2663262414183262, Test Cost
Cost after epoch 13, iteration 400: Train Cost: 0.13544166203678693, Test Cos
Cost after epoch 13, iteration 600: Train Cost: 0.21010861261069874, Test Cos
Cost after epoch 13, iteration 800: Train Cost: 0.08828236798685289, Test Cos
Cost after epoch 14, iteration 0: Train Cost: 0.3148605364071575, Test Cost:
Cost after epoch 14, iteration 200: Train Cost: 0.23992172609144685, Test Cos
Cost after epoch 14, iteration 400: Train Cost: 0.1099304897206666, Test Cost
Cost after epoch 14, iteration 600: Train Cost: 0.11429649892827659, Test Cos
Cost after epoch 14, iteration 800: Train Cost: 0.2714569385849436, Test Cost
```

Test acc. = 97.09%, Train acc. = 97.82%






0 1121 4 0 0 1 4 2 3 0
3 0 1017 0 2 0 1 7 2 0
1 0 11 969 0 7 0 11 8 3
2 0 3 0 946 0 7 3 0 21
7 1 0 11 2 848 9 1 8 5
5 3 2 0 6 5 932 0 5 0
1 2 13 1 1 0 0 1003 2 5
5 0 5 3 3 5 6 6 938 3
6 2 1 7 10 3 0 12 2 966
True Label
Predicted Label


Loss
Iteration (x 200)