Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [ ]:
```
NAME = ""
COLLABORATORS = ""
```

In [ ]:
```
try:
    import torch as t
    import torch.nn as tnn
except ImportError:
    print("Colab users: pytorch comes preinstalled. Select Change Ru")
    print("Local users: Please install pytorch for your hardware using instr
    print("ACG users: Please follow instructions here: https://vikasdhiman.i

    raise
```

In [ ]:
```
def wget(url, filename):
    """
    Download files using requests package.
    Better than wget command line because this is cross platform.
    """
    try:
        import requests
    except ImportError:
        import subprocess
        subprocess.call("pip install --user requests".split())
        import requests
    r = requests.get(url)
    with open(filename, 'wb') as fd:
        for chunk in r.iter_content():
            fd.write(chunk)
```

In [ ]:
```
# Get training features from MNIST dataset.
wget("https://vikasdhiman.info/ECE490-Neural-Networks/notebooks/05-mlp/zero_
     "zero_one_train_features.npz")
```

In [ ]:
```
def draw_features(ax, zero_features, one_features):
    zf = ax.scatter(zero_features[:, 0], zero_features[:, 1], marker='.', la
    of = ax.scatter(one_features[:, 0], one_features[:, 1], marker='+', labe
    ax.legend()
    ax.set_xlabel('Feature 1: count of pixels')
    ax.set_ylabel('Feature 2: Variance along x-axis')
    return [zf, of] # return list of artists
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```
zero_one_train_features = np.load('zero_one_train_features.npz')
FEATURE_MEAN = zero_one_train_features['mean']
FEATURE_STD = zero_one_train_features['std']
features = zero_one_train_features['normed_features']
labels = zero_one_train_features['labels']

fig, ax = plt.subplots()
draw_features(ax, features[labels > 0, :], features[labels < 0, :])
```

In [ ]:

```
if t.cuda.is_available():
    DEVICE="cuda"
elif t.mps.is_available():
    DEVICE="mps"
else:
    DEVICE="cpu"

DTYPE = t.get_default_dtype()

def loss(predicted_labels, true_labels):
    # Make sure predicted_labels and true_labels have same shape
    y = true_labels[..., None]
    yhat = predicted_labels
    assert y.shape == yhat.shape
    return t.maximum(- y * yhat, t.Tensor([0.]).to(device=DEVICE)).sum() / y

# TODO:
# Define model = ?
model = tnn.Sequential(
    tnn.Linear(2, 5),
    tnn.ReLU(),
    tnn.Linear(5, 1))


def train_by_gradient_descent(model, loss, train_features, train_labels, lr=
    predicted_labels = model(train_features)
    #print(predicted_labels)

    loss_t = loss(predicted_labels, train_labels)
    loss_t.backward()
    loss_t_minus_1 = 2*loss_t  # Fake  value to make the while test pass onc
    niter = 0
    while t.abs(loss_t - loss_t_minus_1) / loss_t > 0.01: # Stopping criteri
        with t.no_grad(): # parameter update needs no gradients
            for param in model.parameters():
                assert param.grad is not None
                param.add_( - lr * param.grad)  # Gradient descent

        model.zero_grad()
        # Recompute the gradients
        predicted_labels = model(train_features)
        loss_t_minus_1 = loss_t
        loss_t = loss(predicted_labels, train_labels)
        loss_t.backward() # Compute gradients for next iteration

        # If loss increased, decrease lr. Works for gradient descent, not fo
        if loss_t > loss_t_minus_1:
```

```
            lr = lr / 2

        ### DEBUGing information
        iswrong = (train_labels * predicted_labels.ravel()) < 0
        misclassified = (iswrong).sum() / iswrong.shape[0]
        print(f"loss: {loss_t:04.04f}, delta loss: {loss_t - loss_t_minus_1:
              f"train misclassified: {misclassified:04.04f}")
        if niter % 20 == 0: # plot every 20th iteration
            train_features_cpu = train_features.cpu()
            predicted_labels_cpu = predicted_labels.cpu()
            fig, ax = plt.subplots(1,1)
            draw_features(ax,
                          train_features_cpu[predicted_labels_cpu.ravel() >
                          train_features_cpu[predicted_labels_cpu.ravel() <


        niter += 1
    return model


trained_model = train_by_gradient_descent(model.to(device=DEVICE),
                                          loss,
                                          t.from_numpy(features).to(device=D
                                          t.from_numpy(labels).to(device=DEV


fig, axes = plt.subplots(1,2)
draw_features(axes[0], features[labels > 0, :], features[labels < 0, :])
axes[0].set_title('Train labels')

predicted_labels = trained_model(t.from_numpy(features).to(device=DEVICE, dt
predicted_labels_cpu = predicted_labels.cpu()
draw_features(axes[1], features[predicted_labels_cpu.ravel() > 0, :],
                  features[predicted_labels_cpu.ravel() < 0, :])
axes[1].set_title('Predicted labels');
```

In [ ]:

```
## Doing it the Pytorch way without using our custom feature extraction

import torch
import torch.nn
import torch.optim
import torchvision
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader

torch.manual_seed(17)

# Getting the dataset, the Pytorch way
all_training_data = torchvision.datasets.MNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = torchvision.datasets.MNIST(
```

```
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

In [ ]:
```
training_data, validation_data = torch.utils.data.random_split(all_training_
```

In [ ]:
```
# Hyper parameters
learning_rate = 1e-3 # controls how fast the
batch_size = 64
epochs = 5
momentum = 0.9

training_dataloader = DataLoader(training_data, shuffle=True, batch_size=bat
validation_dataloader = DataLoader(validation_data,  batch_size=batch_size)
test_dataloader = DataLoader(test_data,  batch_size=batch_size)

loss = torch.nn.CrossEntropyLoss()
# TODO:
# Define model = ?

model = tnn.Sequential(
    torch.nn.Flatten(),
    tnn.Linear(28*28, 10),
    tnn.ReLU(),
    tnn.Linear(10, 10))


# Define optimizer
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=m

def loss_and_accuracy(model, loss, validation_dataloader, device=DEVICE):
        # Validation loop
        validation_size = len(validation_dataloader.dataset)
        num_batches = len(validation_dataloader)
        test_loss, correct = 0, 0

        with torch.no_grad():
            for X, y in validation_dataloader:
                X = X.to(device)
                y = y.to(device)
                pred = model(X)
                test_loss += loss(pred, y).item()
                correct += (pred.argmax(dim=-1) == y).type(DTYPE).sum().item

        test_loss /= num_batches
        correct /= validation_size
        return test_loss, correct

def train(model, loss, training_dataloader, validation_dataloader, device=DE
    model.to(device)
    for t in range(epochs):
        # Train loop
        training_size = len(training_dataloader.dataset)
```

```
        for batch, (X, y) in enumerate(training_dataloader):
            X = X.to(device)
            y = y.to(device)
            # Compute prediction and loss
            pred = model(X)
            loss_t = loss(pred, y)

            # Backpropagation
            optimizer.zero_grad()
            loss_t.backward()
            optimizer.step()

            if batch % 100 == 0:
                loss_t, current = loss_t.item(), (batch + 1) * len(X)
                print(f"loss: {loss_t:>7f}  [{current:>5d}/{training_size:>5
        valid_loss, correct = loss_and_accuracy(model, loss, validation_data
        print(f"Validation Error: \n Accuracy: {(100*correct):>0.1f}%, Avg l
    return model

trained_model = train(model, loss, training_dataloader, validation_dataloade

test_loss, correct = loss_and_accuracy(model, loss, test_dataloader)
print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_los
```

In [ ]:
```
X, _ = next(iter(test_dataloader))
X.shape
```

In [ ]:
```
import matplotlib.pyplot as plt
plt.imshow(X[0, 0])
```

In [ ]:
```
print("The predicted image label is ", model(X.to(DEVICE)).argmax(dim=-1)[0]
```