```
import torch
import torch.nn as nn
from torch.optim import Optimizer
from torch.utils.data import DataLoader

# torchvision: popular datasets, model architectures, and common image transformations for computer
vision.
from torchvision import datasets
from torchvision.transforms import transforms

from random import randint
from random import shuffle
import numpy as np
import matplotlib.pyplot as plt
import random

# fixing seed
random_seed = 42
torch.manual_seed(random_seed)
torch.cuda.manual_seed(random_seed)
torch.cuda.manual_seed_all(random_seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(random_seed)
random.seed(random_seed)



# Use data with only 4 and 9 as labels: which is hardest to classify
label_1, label_2 = 4, 9

# MNIST training data
train_set = datasets.MNIST(root='~/Downloads/mnist_data/', train=True,
transform=transforms.ToTensor(), download=True)

# Use data with two labels
idx = (train_set.targets == label_1) + (train_set.targets == label_2)
train_set.data = train_set.data[idx]
train_set.targets = train_set.targets[idx]
train_set.targets[train_set.targets == label_1] = -1
train_set.targets[train_set.targets == label_2] = 1

# MNIST testing data
test_set = datasets.MNIST(root='~/Downloads/mnist_data/', train=False,
transform=transforms.ToTensor())

# Use data with two labels
idx = (test_set.targets == label_1) + (test_set.targets == label_2)
test_set.data = test_set.data[idx]
test_set.targets = test_set.targets[idx]
test_set.targets[test_set.targets == label_1] = -1
test_set.targets[test_set.targets == label_2] = 1
```

```
'''
Step 2: Define the neural network class
'''
class LR(nn.Module) :

    # MNIST data is 28x28 images
    def __init__(self, name, input_dim=28*28) :
        super().__init__()
        self.name = name
        self.linear = nn.Linear(input_dim, 1, bias=True)

    ''' forward given input x '''
    def forward(self, x) :
        return self.linear(x.float().view(-1, 28*28))/255


'''
Step 3: Create the model, specify loss function and optimizer.
'''

def logistic_loss(output, target):
    return -torch.nn.functional.logsigmoid(target*output)

'''
Step 4: Train model with SGD
'''

max_iter = 25000

def training(model, loss, iter):
    optimizer = torch.optim.SGD(model.parameters(), lr=255*1e-4)   # specify SGD with learning rate

    for _ in range(iter) :
        # Sample a random data for training
        ind = randint(0, len(train_set.data)-1)
        image, label = train_set.data[ind], train_set.targets[ind]

        # Clear previously computed gradient
        optimizer.zero_grad()

        # then compute gradient with forward and backward passes
        train_loss = loss(model(image), label.float())
        # if _ % 100 == 1: print(model(image),label.float(), train_loss)
        train_loss.backward()

        #(This syntax will make more sense once we learn about minibatches)

        # perform SGD step (parameter update)
        optimizer.step()

    # print(model(test_set.data))
    # print(test_set.targets)
                                        print(f'{model.name}          accuracy(%)          :'
,(model(test_set.data).reshape(-1)*test_set.targets>0).sum().item()/len(test_set.targets)*100)

    return torch.nonzero((model(test_set.data).reshape(-1)*test_set.targets<0).float()).squeeze().tolist()

LR_model = LR('KL loss')

misclassified_ind_1 = training(LR_model, logistic_loss, max_iter)
print('misclassified label:',misclassified_ind_1)


MSE_model = LR('MSE loss')

def MSE_loss(output, target):
    ## using 1-sigmoid(-z) = sigmoid(z)
                                                                                                                        return
(1-target)*torch.nn.functional.sigmoid(output)**2+(1+target)*torch.nn.functional.sigmoid(-output)**2


misclassified_ind = training(MSE_model, MSE_loss, max_iter)
print('misclassified label:',misclassified_ind)
```

Result

```
$ python Non_CE_loss.py
KL loss accuracy(%) : 96.98643897538926
misclassified label: [22, 124, 125, 149, 156, 182, 248, 253, 254,
281, 287, 288, 292, 341, 343, 392, 418, 421, 433, 434, 453, 487,
498, 527, 560, 562, 579, 593, 699, 748, 755, 767, 779, 804, 809,
815, 827, 835, 889, 904, 937, 962, 967, 983, 996, 1000, 1067,
1191, 1197, 1204, 1206, 1327, 1432, 1502, 1504, 1691, 1712,
1918, 1956, 1958]
MSE loss accuracy(%) : 96.98643897538926
misclassified label: [22, 124, 149, 156, 181, 182, 196, 248, 253,
254, 281, 287, 292, 341, 392, 418, 421, 433, 434, 437, 487, 498,
520, 527, 555, 560, 562, 579, 593, 699, 748, 755, 764, 767, 779,
804, 815, 827, 835, 889, 904, 935, 937, 967, 983, 1000, 1010,
1067, 1197, 1327, 1378, 1432, 1435, 1502, 1691, 1712, 1847,
1918, 1958, 1965]
```

설명

25000 iteration 기준, 두 loss에 대해 같은 accuarcy가 나왔다. (60개의 오답)
misclassified label을 모두 출력해보았으나 똑같은 데이터에 대해 오답을 반환하지는 않았다.