0. Setting

```
# import library
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
import math
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
import torch.nn.functional as F
import math
```

```
torch.__version__
```

```
'1.7.0+cu101'
```

```
# using gpu
use_cuda = True
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(torch.cuda.is_available())
print(device)
```

```
True
cuda
```

1. Data

```
from torchvision import transforms, datasets

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,),(0.3081,)),  # mean value = 0.1307, standard deviation v
])
```

```
data_path = './MNIST'

data_test   = datasets.MNIST(root = data_path, train= True, download=True, transform= trans
data_train  = datasets.MNIST(root = data_path, train= False, download=True, transform= tran

print("the number of your training data (must be 10,000) = ", data_train.__len__())
print("hte number of your testing data (must be 60,000) = ", data_test.__len__())
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./MNIST/MNIST/raw/
                                        9920512/? [00:03<00:00, 2643255.26it/s]
Extracting ./MNIST/MNIST/raw/train-images-idx3-ubyte.gz to ./MNIST/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./MNIST/MNIST/raw/
                                        32768/? [00:01<00:00, 16690.68it/s]
Extracting ./MNIST/MNIST/raw/train-labels-idx1-ubyte.gz to ./MNIST/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./MNIST/MNIST/raw/t
                                        1654784/? [00:01<00:00, 1049065.26it/s]
Extracting ./MNIST/MNIST/raw/t10k-images-idx3-ubyte.gz to ./MNIST/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./MNIST/MNIST/raw/t

0%                                      0/4542 [00:00<?, ?it/s]
Extracting ./MNIST/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./MNIST/MNIST/raw
Processing...
Done!
the number of your training data (must be 10,000) =  10000
hte number of your testing data (must be 60,000) =  60000
/usr/local/lib/python3.6/dist-packages/torchvision/datasets/mnist.py:480: UserWarning: The gi
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
```

2. Model

```
class MyModel(nn.Module):

    def __init__(self, num_classes=10, size_kernel=5):

        super(MyModel, self).__init__()

        # *********************************************************************
        # input parameter
        #
        # data size:
        #   mnist   : 28 * 28
        #
        # the tensor given to the model should be of shape [batch_size, 1, height, width]
        # because first convolution has in_channels = 1
        # *********************************************************************
        self.number_class   = num_classes
        self.size_kernel    = size_kernel

        # *********************************************************************
        # feature layer
        # *********************************************************************
        self.conv1          = nn.Conv2d(1, 20, kernel_size=size_kernel, stride=1, padding=i
        self.conv2          = nn.Conv2d(20, 50, kernel_size=size_kernel, stride=1, padding=
```

```
        self.conv_layer1    = nn.Sequential(self.conv1, nn.MaxPool2d(kernel_size=2), nn.ReL
        self.conv_layer2    = nn.Sequential(self.conv2, nn.MaxPool2d(kernel_size=2), nn.ReL

        self.feature        = nn.Sequential(self.conv_layer1, self.conv_layer2)

        # ******************************************************************
        # classifier layer
        # ******************************************************************
        self.fc1        = nn.Linear(50*7*7, 50, bias=True)
        self.fc2        = nn.Linear(50, num_classes, bias=True)

        self.fc_layer1  = nn.Sequential(self.fc1, nn.ReLU(True))
        self.fc_layer2  = nn.Sequential(self.fc2, nn.LogSoftmax(dim=1))


        self.classifier = nn.Sequential(self.fc_layer1, self.fc_layer2)


        # ******************************************************************
        # dropout
        # ******************************************************************
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)

        self._initialize_weight()


    def _initialize_weight(self):

        for m in self.modules():

            if isinstance(m, nn.Conv2d):

                #nn.init.xavier_uniform_(m.weight, gain=math.sqrt(2))
                nn.init.kaiming_normal_(m.weight.data, a=0, mode='fan_in')

                if m.bias is not None:

                    m.bias.data.zero_()

            elif isinstance(m, nn.Linear):

                #nn.init.xavier_uniform_(m.weight, gain=math.sqrt(2))
                nn.init.kaiming_normal_(m.weight.data, a=0, mode='fan_in')

                if m.bias is not None:

                    m.bias.data.zero_()


    def forward(self, x):
        x = x.to(device)
        x = self.feature(x)
        x = self.dropout1(x)
        x = x.view(x.size(0), -1)
```

```python
        x = self.dropout2(x)
        x = self.classifier(x)

        return x
```

3. Loss Function

```python
model = MyModel(10, 5).to(device)
criterion = nn.CrossEntropyLoss()
train_y_pred = model.forward(data_train.data.unsqueeze(dim=1).float())
train_y = data_train.targets.to(device)
temp_loss = criterion(train_y_pred, train_y)
print(temp_loss.data.item())
```

```
776.919677734375
```

4. Optimization

Define Train Function

```python
def train(model, criterion, train_loader, optimizer, batch_size):

  model.train()
  loss_sum = 0
  acc_sum = 0
  iteration = 0
  for xs, ts in iter(train_loader):

    iteration = iteration + 1
    optimizer.zero_grad()
    y_pred = model(xs)
    ts = ts.to(device)
    loss = criterion(y_pred, ts)
    loss.backward()
    optimizer.step()

    loss_sum = loss_sum + float(loss)
    zs = y_pred.max(1, keepdim=True)[1] # first column has actual prob
    acc_sum = acc_sum + zs.eq(ts.view_as(zs)).sum().item()/batch_size

  loss_avg = round(loss_sum/iteration, 5)
  acc_avg = round(acc_sum/iteration, 5)

  return loss_avg, acc_avg
```

Define Test Function

```
def test(model,criterion, test_loader, batch_size):
  model.eval()
  loss_sum = 0
  acc_sum = 0
  iteration = 0
  with torch.no_grad():
    for xs, ts in iter(test_loader):
      iteration = iteration + 1
      ts = ts.to(device)
      y_pred = model(xs)
      loss_sum = loss_sum + criterion(y_pred, ts).data.item()
      zs = y_pred.max(1, keepdim=True)[1]
      acc_sum = acc_sum + zs.eq(ts.view_as(zs)).sum().item()/batch_size

  loss_avg = round(loss_sum/iteration, 5)
  acc_avg = round(acc_sum/iteration, 5)

  return loss_avg, acc_avg
```

Define Gradient Descent Fucntion

```
def gradient_descent(model, optimizer, criterion, batch_size, num_epochs):

  # batching
  train_loader = torch.utils.data.DataLoader(
      data_train,
      batch_size=batch_size,
      num_workers=4,
      shuffle=True,
      drop_last=True)

  test_loader = torch.utils.data.DataLoader(
      data_test,
      batch_size=batch_size,
      num_workers=4,
      shuffle=False,
      drop_last=True)


  # return variables
  train_loss_list, train_acc_list = [], []
  test_loss_list, test_acc_list = [], []


  # run training & testing
  for epoch in range(num_epochs + 1):

    train_loss_avg, train_acc_avg = train(model, criterion, train_loader, optimizer, batch_
    test_loss_avg, test_acc_avg = test(model, criterion, test_loader, batch_size)

    # add loss and accuracy data
    train_loss_list.append(train_loss_avg)
```

```
    train_loss_list.append(train_loss_avg)
    train_acc_list.append(train_acc_avg)
    test_loss_list.append(test_loss_avg)
    test_acc_list.append(test_acc_avg)

    # print
    if epoch % 10 != 0 :
      continue

    print("epoch : ", epoch, " -------------------------------------- ")
    print("train loss : {}      accuracy = {}".format(train_loss_avg, train_acc_avg))
    print("test loss : {}       accuracy = {}".format(test_loss_avg, test_acc_avg))


  return train_loss_list, train_acc_list, test_loss_list, test_acc_list



def gradient_descent_with_scheduler(scheduler, model, optimizer, criterion, batch_size, num

  # batching
  train_loader = torch.utils.data.DataLoader(
      data_train,
      batch_size=batch_size,
      num_workers=4,
      shuffle=True,
      drop_last=True)

  test_loader = torch.utils.data.DataLoader(
      data_test,
      batch_size=batch_size,
      num_workers=4,
      shuffle=False,
      drop_last=True)


  # return variables
  #train_loss_list, train_acc_list = [], []
  #test_loss_list, test_acc_list = [], []


  # run training & testing
  for epoch in range(num_epochs + 1):

    train_loss_avg, train_acc_avg = train(model, criterion, train_loader, optimizer, batch_
    test_loss_avg, test_acc_avg = test(model, criterion, test_loader, batch_size)
    scheduler.step(train_loss_avg)

    # add loss and accuracy data
    train_loss_list.append(train_loss_avg)
    train_acc_list.append(train_acc_avg)
    test_loss_list.append(test_loss_avg)
    test_acc_list.append(test_acc_avg)

    # print
    if epoch % 10 != 0 :
```

```
      continue

    print("epoch : ", epoch, " ---------------------------------------- ")
    print("train loss : {}      accuracy = {}".format(train_loss_avg, train_acc_avg))
    print("test loss : {}       accuracy = {}".format(test_loss_avg, test_acc_avg))
```

5. Plot Function

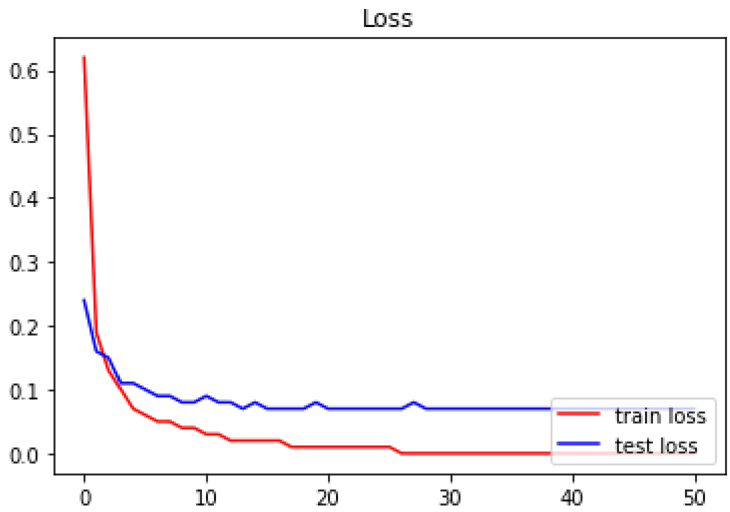
```
def plot_loss(train_loss_list, test_loss_list):
  plt.title("Loss")
  plt.plot(train_loss_list, c = 'red', label = 'train loss')
  plt.plot(test_loss_list, c = 'blue', label = 'test loss')
  plt.legend(loc = 'upper right')
  plt.show()


def plot_accuracy(train_acc_list, test_acc_list):
  plt.title("Accuracy")
  plt.plot(train_acc_list, c = 'red', label = 'train accuracy')
  plt.plot(test_acc_list, c = 'blue', label = 'test accuracy')
  plt.legend(loc = 'lower right')
  plt.show()
```

6. Run

```
# model
num_classes=10
size_kernel=5
model1 = MyModel(num_classes, size_kernel).to(device)

# mini-batch size
batch_size = 32

# num of epochs
num_epochs = 50

# learning rate
learning_rate = 0.01

# optimizer
optimizer = torch.optim.SGD(model1.parameters(), lr = learning_rate, weight_decay=0.0001)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', factor=0.5, patien

# loss function
criterion = nn.CrossEntropyLoss()

# run
```

```
train_loss_list1, train_acc_list1, test_loss_list1, test_acc_list1 = [], [], [], []
gradient_descent_with_scheduler(scheduler, model1, optimizer, criterion, batch_size, num_ep

# plot
plot_loss(train_loss_list1, test_loss_list1)
```

```
epoch :  0  ---------------------------------------
train loss : 0.62      accuracy = 0.8
test loss : 0.24       accuracy = 0.92
epoch :  10  ---------------------------------------
train loss : 0.03      accuracy = 0.98
test loss : 0.09       accuracy = 0.97
epoch :  20  ---------------------------------------
train loss : 0.01      accuracy = 0.99
test loss : 0.07       accuracy = 0.97
Epoch    24: reducing learning rate of group 0 to 5.0000e-03.
epoch :  30  ---------------------------------------
train loss : 0.0      accuracy = 0.99
test loss : 0.07       accuracy = 0.97
Epoch    33: reducing learning rate of group 0 to 2.5000e-03.
Epoch    39: reducing learning rate of group 0 to 1.2500e-03.
epoch :  40  ---------------------------------------
train loss : 0.0      accuracy = 0.99
test loss : 0.07       accuracy = 0.97
Epoch    45: reducing learning rate of group 0 to 6.2500e-04.
Epoch    51: reducing learning rate of group 0 to 3.1250e-04.
epoch :  50  ---------------------------------------
train loss : 0.0      accuracy = 0.99
test loss : 0.07       accuracy = 0.97
```




```
# model
num_classes=10
size_kernel=5
model2 = MyModel(num_classes, size_kernel).to(device)

# mini-batch size
batch_size = 32

# num of epochs
num_epochs = 100

# learning rate
learning_rate = 0.01
```

```
learning_rate = 0.01

# optimizer
optimizer = torch.optim.SGD(model2.parameters(), lr = learning_rate, weight_decay=0.0001)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', factor=0.5, patien

# loss function
criterion = nn.CrossEntropyLoss()

# run
train_loss_list2, train_acc_list2, test_loss_list2, test_acc_list2 = [], [], [], []
gradient_descent_with_scheduler(scheduler, model2, optimizer, criterion, batch_size, num_ep

# plot
plot_loss(train_loss_list2, test_loss_list2)
```

```
epoch :  0  -------------------------------------
train loss : 0.84726      accuracy = 0.72436
test loss : 0.26487       accuracy = 0.91923
epoch :  10  -------------------------------------
train loss : 0.0737      accuracy = 0.97646
test loss : 0.0795       accuracy = 0.97597
epoch :  20  -------------------------------------
train loss : 0.04198      accuracy = 0.98718
test loss : 0.06594      accuracy = 0.9801
Epoch    30: reducing learning rate of group 0 to 5.0000e-03.
epoch :  30  -------------------------------------
```

```
# model
num_classes=10
size_kernel=7
model3 = MyModel(num_classes, size_kernel).to(device)

# mini-batch size
batch_size = 32

# num of epochs
num_epochs = 100

# learning rate
learning_rate = 0.01

# optimizer
optimizer = torch.optim.SGD(model3.parameters(), lr = learning_rate, weight_decay=0.0001)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', factor=0.5, patien

# loss function
criterion = nn.CrossEntropyLoss()

# run
train_loss_list3, train_acc_list3, test_loss_list3, test_acc_list3 = [], [], [], []
gradient_descent_with_scheduler(scheduler, model3, optimizer, criterion, batch_size, num_ep

# plot
plot_loss(train_loss_list3, test_loss_list3)
```

```
epoch :  0  --------------------------------------
train loss : 0.87445      accuracy = 0.70833
test loss : 0.24402       accuracy = 0.93017
epoch :  10  --------------------------------------
train loss : 0.07214      accuracy = 0.97486
test loss : 0.0816       accuracy = 0.97485
epoch :  20  --------------------------------------
train loss : 0.03971      accuracy = 0.98688
test loss : 0.07057       accuracy = 0.97907
epoch :  30  --------------------------------------
train loss : 0.02691      accuracy = 0.99219
test loss : 0.07332       accuracy = 0.97922
epoch :  40  --------------------------------------
train loss : 0.01742      accuracy = 0.99439
test loss : 0.07023       accuracy = 0.98115
epoch :  50  --------------------------------------
train loss : 0.0164      accuracy = 0.99479
test loss : 0.0702       accuracy = 0.98198
epoch :  60  --------------------------------------
train loss : 0.01049      accuracy = 0.997
test loss : 0.06953       accuracy = 0.9824
Epoch    66: reducing learning rate of group 0 to 5.0000e-03.
epoch :  70  --------------------------------------
train loss : 0.00873      accuracy = 0.9973
test loss : 0.06852       accuracy = 0.983
epoch :  80  --------------------------------------
train loss : 0.00945      accuracy = 0.9974
test loss : 0.06948       accuracy = 0.9827
Epoch    85: reducing learning rate of group 0 to 2.5000e-03.
epoch :  90  --------------------------------------
train loss : 0.00486      accuracy = 0.999
test loss : 0.06863       accuracy = 0.9832
Epoch    97: reducing learning rate of group 0 to 1.2500e-03.
epoch :  100  --------------------------------------
```

```
# num of epochs
num_epochs = 10

gradient_descent_with_scheduler(scheduler, model3, optimizer, criterion, batch_size, num_ep
```

```
epoch :  0  --------------------------------------
train loss : 0.00588      accuracy = 0.9986
test loss : 0.06838       accuracy = 0.98342
Epoch   103: reducing learning rate of group 0 to 6.2500e-04.
Epoch   109: reducing learning rate of group 0 to 3.1250e-04.
epoch :  10  --------------------------------------
train loss : 0.00612      accuracy = 0.9981
test loss : 0.06843       accuracy = 0.98352
```

```
# num of epochs
num_epochs = 50

gradient_descent_with_scheduler(scheduler, model3, optimizer, criterion, batch_size, num_ep
```

```
epoch :  0  --------------------------------------
train loss : 0.00628      accuracy = 0.9982
test loss : 0.06825       accuracy = 0.9835
```

```
Epoch   115: reducing learning rate of group 0 to 1.5625e-04.
Epoch   121: reducing learning rate of group 0 to 7.8125e-05.
epoch :  10 ---------------------------------------
train loss : 0.0081      accuracy = 0.997
test loss : 0.06801       accuracy = 0.98352
Epoch   127: reducing learning rate of group 0 to 3.9063e-05.
Epoch   133: reducing learning rate of group 0 to 1.9531e-05.
epoch :  20 ---------------------------------------
train loss : 0.00721      accuracy = 0.9978
test loss : 0.06792       accuracy = 0.98347
Epoch   139: reducing learning rate of group 0 to 9.7656e-06.
epoch :  30 ---------------------------------------
train loss : 0.00603      accuracy = 0.9984
test loss : 0.06793       accuracy = 0.9835
Epoch   145: reducing learning rate of group 0 to 4.8828e-06.
Epoch   151: reducing learning rate of group 0 to 2.4414e-06.
epoch :  40 ---------------------------------------
train loss : 0.00484      accuracy = 0.999
test loss : 0.06793       accuracy = 0.9835
Epoch   159: reducing learning rate of group 0 to 1.2207e-06.
epoch :  50 ---------------------------------------
train loss : 0.00594      accuracy = 0.9984
test loss : 0.06793       accuracy = 0.98348
```

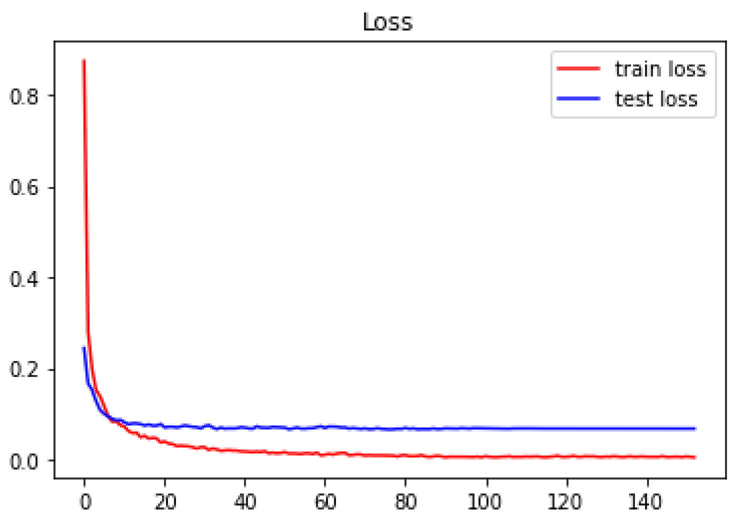
```
length = len(test_acc_list3)-10
print(length)
```

```
153
```

7. Output

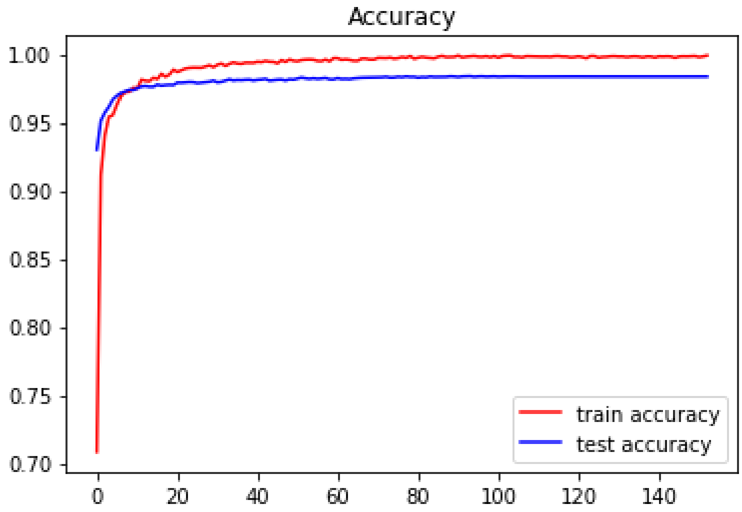
1. Plot the training and testing losses over epochs [2pt]

```
plot_loss(train_loss_list3[0:length], test_loss_list3[0:length])
```



2. Plot the training and testing accuracies over epochs [2pt]

```
plot_accuracy(train_acc_list3[0:length], test_acc_list3[0:length])
```



3. Print the final training and testing losses at convergence [2pt]

```
data1 = {'' : [train_loss_list3[length-1], test_loss_list3[length-1]]}
index1 = ['training', 'testing']
frame1 = DataFrame(data1, index = index1)
frame1.columns.name = 'loss'
frame1
```

loss	
training	0.00484
testing	0.06793

4. Print the final training and testing accuracies at convergence [20pt]

```
data2 = {'' : [str(train_acc_list3[length-1]) + '00', str(test_acc_list3[length-1]) + '0']}
index2 = ['training', 'testing']
frame2 = DataFrame(data2, index = index2)
frame2.columns.name = 'accuracy'
frame2
```

accuracy	
training	0.99900
testing	0.98350

5. Print the testing accuracies within the last 10 epochs [5pt]

```
for i in range(10):
```

```
for i in range(10):
  idx = length - 10 + i
  print("[epoch = {0}] {1:0.5f}".format(idx, test_acc_list3[idx]))
```

```
    [epoch = 143] 0.98350
    [epoch = 144] 0.98350
    [epoch = 145] 0.98350
    [epoch = 146] 0.98350
    [epoch = 147] 0.98350
    [epoch = 148] 0.98350
    [epoch = 149] 0.98350
    [epoch = 150] 0.98350
    [epoch = 151] 0.98350
    [epoch = 152] 0.98350
```