trainingNN_practice_pytorch_answer

December 20, 2021

1 Assignment 2: Training neural network: Learning rate, dropout, activation function

Trong bài thực hành này, chúng ta sẽ tìm hiểu các vấn đề về learning rate, dropout, activation function thông qua bài toán phân loại chữ số viết tay trên bộ dữ liệu MNIST.

[ ]:
```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import matplotlib.pyplot as plt
import cv2
import keras
from keras import  backend as K

import torch
import os
from torch import nn
import torch.nn.functional as F
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor

from torch.utils.tensorboard import SummaryWriter

%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

1.1 Phần 1: Quan sát dữ liệu

MNIST là tập dữ liệu ảnh đen trắng các chữ số viết tay, có cùng kích thước 28x28. Chữ số trong ảnh đã được căn chỉnh vào tâm. Đây là tập dữ liệu rất phù hợp cho việc thử nghiệm các kỹ thuật huấn luyện và nhận dạng mẫu mà không đòi hỏi quá nhiều công sức tiền xử lý.

Bộ dữ liệu MNIST được chia sẵn thành 2 phần: tập dữ liệu huấn luyện gồm 60.000 ảnh, tập dữ liệu kiểm thử gồm 10.000 ảnh. Các ảnh trong bộ dữ liệu thuộc về một trong 10 lớp: 0, 1, 2,..., 9.

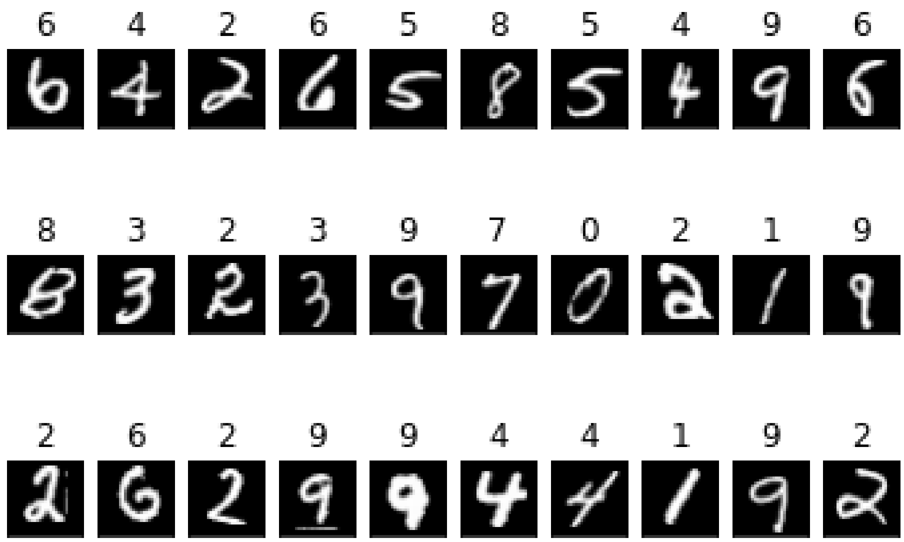
Thư viện PyTorch đã cung cấp sẵn một module để tải về MNIST:

[ ]:
```
train_data = datasets.MNIST(
    root = 'data',
    train = True,
    download = True
)
test_data = datasets.MNIST(
    root = 'data',
    train = False
)
(x_train, y_train) = train_data.data[:].detach().numpy(), train_data.targets[:].
 ↪detach().numpy()
(x_test, y_test) = test_data.data[:].detach().numpy(), test_data.targets[:].
 ↪detach().numpy()
print('Training image: ', x_train.shape)
print('Testing image: ', x_test.shape)
print('Training label: ', y_train.shape)
print('Testing label: ', y_test.shape)
```

```
Training image:  (60000, 28, 28)
Testing image:  (10000, 28, 28)
Training label:  (60000,)
Testing label:  (10000,)
```

Để dễ hình dung về dữ liệu, có thể sử dụng thư viện matplotlib quan sát một vài mẫu dữ liệu:

[ ]:
```
for i in range(30):
    idx = np.random.randint(0, x_train.shape[0])
    image = x_train[idx]
    plt.subplot(3, 10, i + 1), plt.imshow(image, cmap='gray')
    plt.title(y_train[idx]), plt.xticks([]), plt.yticks([])
plt.show()
```



[ ]:
```
batch_size = 64
num_classes = 10
epochs = 20
# input image dimensions
img_rows, img_cols = 28, 28

# Backend is Tensorflow which accepts input as Batch, Number Channels, height,
 ↪width
# So we have to reshape the input to that format.

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

Trước khi đưa vào mô hình, cần chuẩn hoá giá trị điểm ảnh về trong khoảng [0,1] nhằm giúp các thuật toán tối ưu hội tụ nhanh hơn:

[ ]:
```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

[ ]:
```
print(x_train.shape, y_train.shape)
```

```
(60000, 28, 28, 1) (60000,)
```

1.2 Phần 2: Xây dựng mô hình phân loại

Trong phần này, chúng ta sẽ xây dựng một mô hình phân loại đơn giản cho bài toán nhận diện chữ số viết tay sử dụng thư viện keras. Từ mô hình này ta sẽ thử nghiệm tác động của các yếu tố như learning rate, dropout, activation function đến quá trình huấn luyện mạng.

[ ]:
```
class DeepModel(nn.Module):
    def __init__(self, dropout_rate):
        super(DeepModel, self).__init__()
        self.conv_1 = nn.Conv2d(1, 32, kernel_size = 3, stride = (1, 1),␣
 ↪padding=0, bias=False, dilation=1)
        self.conv_2 = nn.Conv2d(32, 64, kernel_size = 3, stride = (1, 1),␣
 ↪padding=0, bias=False, dilation=1)

        self.dropout = nn.Dropout(dropout_rate)

        self.dense_1 = nn.Linear(12*12*64, 512)
        self.dense_2 = nn.Linear(512, 10)

    def forward(self, x):
        x = x.permute(0, 3, 1, 2)
        x = self.conv_1(x)
        x = nn.ReLU(inplace=True)(x)
        x = self.conv_2(x)
        x = nn.ReLU(inplace=True)(x)
        x = nn.MaxPool2d(kernel_size=2)(x)

        x = x.reshape(x.size(0), -1)
        x = self.dense_1(x)
        x = nn.ReLU(inplace=True)(x)
        x = self.dropout(x)

        output = F.log_softmax(self.dense_2(x))

        return output
```

1.2.1 Kiểm tra số lượng tham số

[ ]:
```
simple_model = DeepModel(dropout_rate = 0.5).cuda()
for param in simple_model.parameters():
    if param.requires_grad:
        print('param autograd')
        break

input = torch.rand(2, 28, 28, 1).cuda()
```

```
output = simple_model(input)  # type: torch.Tensor

model_parameters = filter(lambda p: p.requires_grad, simple_model.parameters())
params = sum([np.prod(p.size()) for p in model_parameters])
print('Number of parameter:', params)
```

```
param autograd
Number of parameter: 4742954
```

1.2.2 Khởi tạo Generator

```
[ ]: class Generator(Dataset):
         def __init__(self, images, labels):
             self.images = images
             self.labels = labels

         def __len__(self):
             return self.images.shape[0]

         def __getitem__(self, idx):
             return self.images[idx], self.labels[idx]
```

```
[ ]: training_data = Generator(x_train, y_train)
     train_dataloader = DataLoader(training_data, batch_size=32, shuffle=True)
```

```
[ ]: test_data = Generator(x_test, y_test)
     test_dataloader = DataLoader(test_data, batch_size=32, shuffle=True)
```

1.2.3 Phần 2.1: Vai trò của Dropout

Kỹ thuật dropout tắt đi một số kết nối một cách ngẫu nhiên trong mỗi lượt huấn luyện, giúp tránh hiện tượng đồng thích nghi (co-adaptation). Điều đó giúp mô hình bị quá khớp. Việc quá khớp thể hiện ở việc lỗi huấn luyện là rất nhỏ nhưng lỗi kiểm thử lại lớn. Phần thực hành tiếp theo nhằm minh hoạ cho tác dụng của kỹ thuật này.

- Dropout rate = 0

```
[ ]: use_cuda = torch.cuda.is_available()  #GPU cuda
     best_loss = float('inf')

     model = DeepModel(dropout_rate = 0)

     optimizer = torch.optim.Adam(model.parameters())
     if use_cuda:
         model = torch.nn.parallel.DataParallel(model.cuda())   # , device_ids=[0,␣
      ↪1, 2, 3]
         torch.backends.cudnn.benchmark = True
```

```
[ ]: def train(model, optimizer, epoch, writer):
        print('\n ############################## Train phase, Epoch: {}␣
     ↪##############################'.format(epoch))
        model.train()
        train_loss = 0
        running_loss = 0
        print('\nLearning rate at this epoch is: ', optimizer.
     ↪param_groups[0]['lr'], '\n')
        for (batch_idx, target_tuple) in enumerate(train_dataloader):
            if use_cuda:
                target_tuple = [target_tensor.cuda(non_blocking=True) for␣
     ↪target_tensor in target_tuple]

            images, labels = target_tuple
            # Convert label to long type pytorch
            labels = torch.tensor(labels,dtype=torch.long)

            optimizer.zero_grad()  # zero the gradient buff
            output_tuple = model(images)

            loss = F.nll_loss(output_tuple, labels).cuda()

            loss.backward()  # retain_graph=True
            optimizer.step()

            train_loss += loss.item()  # loss
            running_loss += loss.item()
            if batch_idx % 50 == 49:
                writer.add_scalar('training loss', running_loss/50, epoch *␣
     ↪len(train_dataloader) + batch_idx)
                running_loss = 0
            #print('########################### Epoch:', epoch, ', --  batch:', ␣
     ↪batch_idx, '/', len(train_dataloader), ',   ',
            #      'Train loss: %.3f, accumulated average loss: %.3f␣
     ↪##############################' % (loss.item(), train_loss / (batch_idx +␣
     ↪1)))
```

```
[ ]: def test(model, epoch, writer):
        print('\n ############################## Test phase, Epoch: {}␣
     ↪##############################'.format(epoch))
        model.eval()
        with torch.no_grad():
            test_loss = 0
            correct = 0
            for (batch_idx, target_tuple) in enumerate(test_dataloader):
                if use_cuda:
```

```
                target_tuple = [target_tensor.cuda(non_blocking=True) for␣
↪target_tensor in target_tuple]

            images, labels = target_tuple
            # Convert label to long type pytorch
            labels = torch.tensor(labels,dtype=torch.long)
            output_tuple = model(images)
            #print(output_tuple.shape)

            _, predicted = torch.max(output_tuple.data, 1)
            correct += (predicted == labels).sum().item()

            loss = F.nll_loss(output_tuple, labels).cuda()

            test_loss += loss.item()  # loss
            #print('########################### Epoch:', epoch, ', --  batch:',␣
↪ batch_idx, '/', len(test_dataloader), ',    ',
            #     'Test loss: %.3f, accumulated average loss: %.3f␣
↪##############################' % (loss.item(), test_loss / (batch_idx + 1)))
        acc = correct*100/len(test_data)
        print('Accuracy: ', acc)
        writer.add_scalar('test accuracy', acc, epoch)
```

[ ]:

```
def train_and_test(model, optimizer, epoch_num = 5, summary_path='runs/
↪mnist_experiment_dropout'):
    writer = SummaryWriter(summary_path)
    for epoch in range(epoch_num):
        train(model, optimizer, epoch, writer)
        test(model, epoch, writer)
```

[ ]:

```
train_and_test(model, optimizer, 5, 'runs/mnist_experiment_dropout=0')
```

```
 ############################## Train phase, Epoch: 0
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 0
##############################
Accuracy:  98.5

 ############################## Train phase, Epoch: 1
##############################

Learning rate at this epoch is:  0.001
```

```
 ############################## Test phase, Epoch: 1
##############################
Accuracy:  99.04

 ############################## Train phase, Epoch: 2
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 2
##############################
Accuracy:  98.48

 ############################## Train phase, Epoch: 3
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 3
##############################
Accuracy:  99.04

 ############################## Train phase, Epoch: 4
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 4
##############################
Accuracy:  98.87
```

[ ]:
```
%tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 163), started 0:17:16 ago. (Use '!kill␣
↪163' to kill it.)
```

```
<IPython.core.display.Javascript object>
```

- Dropout rate = 0.5

[ ]:
```
use_cuda = torch.cuda.is_available()  #GPU cuda
best_loss = float('inf')

model = DeepModel(dropout_rate = 0.5)
```

```
optimizer = torch.optim.Adam(model.parameters())
if use_cuda:
    model = torch.nn.parallel.DataParallel(model.cuda())   # , device_ids=[0,␣
↪1, 2, 3]
    torch.backends.cudnn.benchmark = True
```

[ ]:
```
train_and_test(model, optimizer, 5, 'runs/mnist_experiment_dropout=0.5')
```

```
 ############################## Train phase, Epoch: 0
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 0
##############################
Accuracy:  98.81

 ############################## Train phase, Epoch: 1
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 1
##############################
Accuracy:  98.97

 ############################## Train phase, Epoch: 2
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 2
##############################
Accuracy:  98.93

 ############################## Train phase, Epoch: 3
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 3
##############################
```

```
Accuracy:  99.18

 ############################## Train phase, Epoch: 4
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 4
##############################
Accuracy:  99.14
```

[ ]:
```
%tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 163), started 0:18:49 ago. (Use '!kill
↪163' to kill it.)
```

```
<IPython.core.display.Javascript object>
```

1.2.4 Phần 2.2: Vai trò của activation function

[ ]:
```
class SimpleModel(nn.Module):
    def __init__(self, dropout_rate =0.5, activation = None):
        super(SimpleModel, self).__init__()
        self.dense_1 = nn.Linear(28*28, 512)
        self.dropout = nn.Dropout(dropout_rate)
        self.dense_2 = nn.Linear(512, 10)
        self.activation = activation

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.dense_1(x)
        if self.activation is not None:
          x = self.activation(x)
        x = self.dropout(x)
        x = self.dense_2(x)
        output = F.log_softmax(x)

        return output
```

1.2.5 Kiểm tra số lượng tham số

[ ]:
```
simple_model = SimpleModel().cuda()
for param in simple_model.parameters():
    if param.requires_grad:
        print('param autograd')
        break
```

```
input = torch.rand(1, 28, 28).cuda()
output = simple_model(input)  # type: torch.Tensor

model_parameters = filter(lambda p: p.requires_grad, simple_model.parameters())
params = sum([np.prod(p.size()) for p in model_parameters])
print('Number of parameter:', params)
```

```
param autograd
Number of parameter: 407050
```

Trước tiên hãy thử xem, sẽ thế nào nếu không sử dụng hàm kích hoạt cho lớp ẩn:

[ ]:
```
# YOUR CODE HERE
simple_model = SimpleModel(dropout_rate = 0.5, activation = None).cuda()
optimizer = torch.optim.Adam(simple_model.parameters())
# YOUR CODE HERE
train_and_test(simple_model, optimizer, 5, 'runs/mnist_none_activation')
```

```
 ############################## Train phase, Epoch: 0
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 0
##############################
Accuracy:  91.99

 ############################## Train phase, Epoch: 1
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 1
##############################
Accuracy:  90.19

 ############################## Train phase, Epoch: 2
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 2
##############################
Accuracy:  91.98
```

```
 ############################## Train phase, Epoch: 3
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 3
##############################
Accuracy:  92.01

 ############################## Train phase, Epoch: 4
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 4
##############################
Accuracy:  91.48
```

Kết quả đạt được là tương đối tệ trên tập dữ liệu MNIST. Tiếp theo chúng ta sẽ thử nghiệm và so sánh kết quả khi sử dụng các hàm kích hoạt khác nhau:

[ ]:
```python
# YOUR CODE HERE
for activation in [None, nn.Sigmoid(), nn.Tanh(), nn.ReLU()]:
    print('Activation: ', str(activation))
    simple_model = SimpleModel(dropout_rate = 0.5, activation = activation).
 ↪cuda()
    optimizer = torch.optim.Adam(simple_model.parameters())
    # YOUR CODE HERE
    train_and_test(simple_model, optimizer, 5, 'runs/mnist_' + str(activation))
```

```
Activation:  None

 ############################## Train phase, Epoch: 0
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 0
##############################
Accuracy:  91.06

 ############################## Train phase, Epoch: 1
##############################

Learning rate at this epoch is:  0.001
```

```
 ############################# Test phase, Epoch: 1
#############################
Accuracy:  92.0

 ############################# Train phase, Epoch: 2
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 2
#############################
Accuracy:  92.0

 ############################# Train phase, Epoch: 3
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 3
#############################
Accuracy:  90.91

 ############################# Train phase, Epoch: 4
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 4
#############################
Accuracy:  91.59
Activation:  Sigmoid()

 ############################# Train phase, Epoch: 0
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 0
#############################
Accuracy:  93.41

 ############################# Train phase, Epoch: 1
```

```
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 1
#############################
Accuracy:  95.49

 ############################# Train phase, Epoch: 2
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 2
#############################
Accuracy:  96.52

 ############################# Train phase, Epoch: 3
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 3
#############################
Accuracy:  97.06

 ############################# Train phase, Epoch: 4
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 4
#############################
Accuracy:  97.36
Activation:  Tanh()

 ############################# Train phase, Epoch: 0
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 0
#############################
```

```
Accuracy:  94.19

 ############################# Train phase, Epoch: 1
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 1
#############################
Accuracy:  95.75

 ############################# Train phase, Epoch: 2
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 2
#############################
Accuracy:  96.42

 ############################# Train phase, Epoch: 3
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 3
#############################
Accuracy:  96.73

 ############################# Train phase, Epoch: 4
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 4
#############################
Accuracy:  97.16
Activation:  ReLU()

 ############################# Train phase, Epoch: 0
#############################

Learning rate at this epoch is:  0.001
```

```
 ############################## Test phase, Epoch: 0
##############################
Accuracy:  96.44

 ############################## Train phase, Epoch: 1
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 1
##############################
Accuracy:  97.4

 ############################## Train phase, Epoch: 2
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 2
##############################
Accuracy:  97.53

 ############################## Train phase, Epoch: 3
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 3
##############################
Accuracy:  97.87

 ############################## Train phase, Epoch: 4
##############################

Learning rate at this epoch is:  0.001


 ############################## Test phase, Epoch: 4
##############################
Accuracy:  97.96
```

[ ]:
```
%tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 163), started 0:04:05 ago. (Use '!kill␣
↪163' to kill it.)
```

```
<IPython.core.display.Javascript object>
```

1.2.6 Phần 2.3: Vai trò của hệ số học learning rate

Ta tiếp tục quan sát tác động của learning rate đến quá trình học của mạng:

[ ]:
```
# YOUR CODE HERE
learning_rates = [1E-0, 1E-1, 1E-2, 1E-3, 1E-4, 1E-5, 1E-6, 1E-7]
for lr in learning_rates:
    # YOUR CODE HERE
    print('Learning rate = %f' % lr)
    simple_model = SimpleModel(dropout_rate = 0.5, activation = nn.ReLU()).
↪cuda()
    optimizer = torch.optim.SGD(simple_model.parameters(), lr = lr, momentum =␣
↪0.9)
    # YOUR CODE HERE
    train_and_test(simple_model, optimizer, 5, 'runs/lnr_' + str(lr))
# YOUR CODE HERE
```

```
Learning rate = 1.000000

 ############################## Train phase, Epoch: 0
##############################

Learning rate at this epoch is:  1.0


 ############################## Test phase, Epoch: 0
##############################
Accuracy:  9.82

 ############################## Train phase, Epoch: 1
##############################

Learning rate at this epoch is:  1.0


 ############################## Test phase, Epoch: 1
##############################
Accuracy:  11.35

 ############################## Train phase, Epoch: 2
##############################

Learning rate at this epoch is:  1.0
```

```
 ############################# Test phase, Epoch: 2
#############################
Accuracy:  9.82

 ############################# Train phase, Epoch: 3
#############################

Learning rate at this epoch is:  1.0


 ############################# Test phase, Epoch: 3
#############################
Accuracy:  9.74

 ############################# Train phase, Epoch: 4
#############################

Learning rate at this epoch is:  1.0


 ############################# Test phase, Epoch: 4
#############################
Accuracy:  11.35
Learning rate = 0.100000

 ############################# Train phase, Epoch: 0
#############################

Learning rate at this epoch is:  0.1


 ############################# Test phase, Epoch: 0
#############################
Accuracy:  92.66

 ############################# Train phase, Epoch: 1
#############################

Learning rate at this epoch is:  0.1


 ############################# Test phase, Epoch: 1
#############################
Accuracy:  93.22

 ############################# Train phase, Epoch: 2
#############################
```

```
Learning rate at this epoch is:  0.1


 ############################# Test phase, Epoch: 2
#############################
Accuracy:  93.03

 ############################# Train phase, Epoch: 3
#############################

Learning rate at this epoch is:  0.1


 ############################# Test phase, Epoch: 3
#############################
Accuracy:  93.77

 ############################# Train phase, Epoch: 4
#############################

Learning rate at this epoch is:  0.1


 ############################# Test phase, Epoch: 4
#############################
Accuracy:  94.6
Learning rate = 0.010000

 ############################# Train phase, Epoch: 0
#############################

Learning rate at this epoch is:  0.01


 ############################# Test phase, Epoch: 0
#############################
Accuracy:  95.13

 ############################# Train phase, Epoch: 1
#############################

Learning rate at this epoch is:  0.01


 ############################# Test phase, Epoch: 1
#############################
Accuracy:  96.58
```

```
 ############################# Train phase, Epoch: 2
#############################

Learning rate at this epoch is:  0.01


 ############################# Test phase, Epoch: 2
#############################
Accuracy:  97.26

 ############################# Train phase, Epoch: 3
#############################

Learning rate at this epoch is:  0.01


 ############################# Test phase, Epoch: 3
#############################
Accuracy:  97.53

 ############################# Train phase, Epoch: 4
#############################

Learning rate at this epoch is:  0.01


 ############################# Test phase, Epoch: 4
#############################
Accuracy:  97.75
Learning rate = 0.001000

 ############################# Train phase, Epoch: 0
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 0
#############################
Accuracy:  89.14

 ############################# Train phase, Epoch: 1
#############################

Learning rate at this epoch is:  0.001
```

```
 ############################# Test phase, Epoch: 1
#############################
Accuracy:  91.23

 ############################# Train phase, Epoch: 2
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 2
#############################
Accuracy:  92.17

 ############################# Train phase, Epoch: 3
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 3
#############################
Accuracy:  92.98

 ############################# Train phase, Epoch: 4
#############################

Learning rate at this epoch is:  0.001


 ############################# Test phase, Epoch: 4
#############################
Accuracy:  93.58
Learning rate = 0.000100

 ############################# Train phase, Epoch: 0
#############################

Learning rate at this epoch is:  0.0001


 ############################# Test phase, Epoch: 0
#############################
Accuracy:  73.82

 ############################# Train phase, Epoch: 1
#############################
```

```
Learning rate at this epoch is:  0.0001


 ############################# Test phase, Epoch: 1
#############################
Accuracy:  80.12

 ############################# Train phase, Epoch: 2
#############################

Learning rate at this epoch is:  0.0001


 ############################# Test phase, Epoch: 2
#############################
Accuracy:  84.03

 ############################# Train phase, Epoch: 3
#############################

Learning rate at this epoch is:  0.0001


 ############################# Test phase, Epoch: 3
#############################
Accuracy:  85.76

 ############################# Train phase, Epoch: 4
#############################

Learning rate at this epoch is:  0.0001


 ############################# Test phase, Epoch: 4
#############################
Accuracy:  86.94
Learning rate = 0.000010

 ############################# Train phase, Epoch: 0
#############################

Learning rate at this epoch is:  1e-05


 ############################# Test phase, Epoch: 0
#############################
Accuracy:  33.16
```

```
 ############################# Train phase, Epoch: 1
#############################

Learning rate at this epoch is:  1e-05


 ############################# Test phase, Epoch: 1
#############################
Accuracy:  51.05

 ############################# Train phase, Epoch: 2
#############################

Learning rate at this epoch is:  1e-05


 ############################# Test phase, Epoch: 2
#############################
Accuracy:  58.94

 ############################# Train phase, Epoch: 3
#############################

Learning rate at this epoch is:  1e-05


 ############################# Test phase, Epoch: 3
#############################
Accuracy:  63.89

 ############################# Train phase, Epoch: 4
#############################

Learning rate at this epoch is:  1e-05


 ############################# Test phase, Epoch: 4
#############################
Accuracy:  66.88
Learning rate = 0.000001

 ############################# Train phase, Epoch: 0
#############################

Learning rate at this epoch is:  1e-06


 ############################# Test phase, Epoch: 0
```

```
#############################
Accuracy:  13.24

 ############################# Train phase, Epoch: 1
#############################

Learning rate at this epoch is:  1e-06


 ############################# Test phase, Epoch: 1
#############################
Accuracy:  14.79

 ############################# Train phase, Epoch: 2
#############################

Learning rate at this epoch is:  1e-06


 ############################# Test phase, Epoch: 2
#############################
Accuracy:  16.33

 ############################# Train phase, Epoch: 3
#############################

Learning rate at this epoch is:  1e-06


 ############################# Test phase, Epoch: 3
#############################
Accuracy:  17.47

 ############################# Train phase, Epoch: 4
#############################

Learning rate at this epoch is:  1e-06


 ############################# Test phase, Epoch: 4
#############################
Accuracy:  18.9
Learning rate = 0.000000

 ############################# Train phase, Epoch: 0
#############################

Learning rate at this epoch is:  1e-07
```

```
 ############################## Test phase, Epoch: 0
##############################
Accuracy:  10.28

 ############################## Train phase, Epoch: 1
##############################

Learning rate at this epoch is:  1e-07


 ############################## Test phase, Epoch: 1
##############################
Accuracy:  10.41

 ############################## Train phase, Epoch: 2
##############################

Learning rate at this epoch is:  1e-07


 ############################## Test phase, Epoch: 2
##############################
Accuracy:  10.52

 ############################## Train phase, Epoch: 3
##############################

Learning rate at this epoch is:  1e-07


 ############################## Test phase, Epoch: 3
##############################
Accuracy:  10.61

 ############################## Train phase, Epoch: 4
##############################

Learning rate at this epoch is:  1e-07


 ############################## Test phase, Epoch: 4
##############################
Accuracy:  10.71
```

[ ]:
```
%tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 163), started 0:07:12 ago. (Use '!kill␣
↪163' to kill it.)
```

```
<IPython.core.display.Javascript object>
```

- Giảm learning rate theo cơ chế: lnr = init_lnr / (1 + decay * step)

[ ]:
```
def train(model, optimizer, scheduler, epoch, writer):
    print('\n ############################# Train phase, Epoch: {}␣
↪#############################'.format(epoch))
    model.train()
    train_loss = 0
    running_loss = 0
    print('\nLearning rate at this epoch is: ', scheduler.get_last_lr(), '\n')
    for (batch_idx, target_tuple) in enumerate(train_dataloader):
        if use_cuda:
            target_tuple = [target_tensor.cuda(non_blocking=True) for␣
↪target_tensor in target_tuple]

        images, labels = target_tuple
        # Convert label to long type pytorch
        labels = torch.tensor(labels,dtype=torch.long)

        optimizer.zero_grad()  # zero the gradient buff
        output_tuple = model(images)

        loss = F.nll_loss(output_tuple, labels).cuda()

        loss.backward()  # retain_graph=True
        optimizer.step()


        train_loss += loss.item()  # loss
        running_loss += loss.item()
        if batch_idx % 50 == 49:
            writer.add_scalar('training loss', running_loss/50, epoch *␣
↪len(train_dataloader) + batch_idx)
            running_loss = 0
        #print('########################### Epoch:', epoch, ', --  batch:', ␣
↪batch_idx, '/', len(train_dataloader), ',    ',
        #      'Train loss: %.3f, accumulated average loss: %.3f␣
↪#############################' % (loss.item(), train_loss / (batch_idx +␣
↪1)))
    scheduler.step()
```

[ ]:
```
def train_and_test(model, optimizer, scheduler, epoch_num = 5,␣
↪summary_path='runs/mnist_experiment_dropout'):
    writer = SummaryWriter(summary_path)
```

```
    for epoch in range(epoch_num):
        train(model, optimizer, scheduler, epoch, writer)
        test(model, epoch, writer)
```

[ ]:
```
simple_model = SimpleModel(dropout_rate = 0.5, activation = nn.ReLU()).cuda()
optimizer = torch.optim.SGD(simple_model.parameters(), lr = 0.01, momentum = 0.
 ↪9)
scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.9)
# YOUR CODE HERE
train_and_test(simple_model, optimizer, scheduler, 20, 'runs/exponentialLR')
```

```
 ############################## Train phase, Epoch: 0
##############################

Learning rate at this epoch is:  [0.01]


 ############################## Test phase, Epoch: 0
##############################
Accuracy:  95.08

 ############################## Train phase, Epoch: 1
##############################

Learning rate at this epoch is:  [0.009000000000000001]


 ############################## Test phase, Epoch: 1
##############################
Accuracy:  96.41

 ############################## Train phase, Epoch: 2
##############################

Learning rate at this epoch is:  [0.008100000000000001]


 ############################## Test phase, Epoch: 2
##############################
Accuracy:  97.04

 ############################## Train phase, Epoch: 3
##############################

Learning rate at this epoch is:  [0.007290000000000001]
```

```
 ############################# Test phase, Epoch: 3
#############################
Accuracy:  97.39

 ############################# Train phase, Epoch: 4
#############################

Learning rate at this epoch is:  [0.006561000000000002]


 ############################# Test phase, Epoch: 4
#############################
Accuracy:  97.55

 ############################# Train phase, Epoch: 5
#############################

Learning rate at this epoch is:  [0.005904900000000002]


 ############################# Test phase, Epoch: 5
#############################
Accuracy:  97.63

 ############################# Train phase, Epoch: 6
#############################

Learning rate at this epoch is:  [0.005314410000000002]


 ############################# Test phase, Epoch: 6
#############################
Accuracy:  97.79

 ############################# Train phase, Epoch: 7
#############################

Learning rate at this epoch is:  [0.004782969000000002]


 ############################# Test phase, Epoch: 7
#############################
Accuracy:  97.93

 ############################# Train phase, Epoch: 8
#############################

Learning rate at this epoch is:  [0.004304672100000002]
```

```
 ############################## Test phase, Epoch: 8
##############################
Accuracy:  97.95

 ############################## Train phase, Epoch: 9
##############################

Learning rate at this epoch is:  [0.003874204890000002]


 ############################## Test phase, Epoch: 9
##############################
Accuracy:  97.99

 ############################## Train phase, Epoch: 10
##############################

Learning rate at this epoch is:  [0.003486784401000002]


 ############################## Test phase, Epoch: 10
##############################
Accuracy:  98.01

 ############################## Train phase, Epoch: 11
##############################

Learning rate at this epoch is:  [0.003138105960900002]


 ############################## Test phase, Epoch: 11
##############################
Accuracy:  98.06

 ############################## Train phase, Epoch: 12
##############################

Learning rate at this epoch is:  [0.0028242953648100018]


 ############################## Test phase, Epoch: 12
##############################
Accuracy:  98.07

 ############################## Train phase, Epoch: 13
##############################
```

```
Learning rate at this epoch is:  [0.0025418658283290017]


 ############################## Test phase, Epoch: 13
##############################
Accuracy:  98.09

 ############################## Train phase, Epoch: 14
##############################

Learning rate at this epoch is:  [0.0022876792454961017]


 ############################## Test phase, Epoch: 14
##############################
Accuracy:  98.1

 ############################## Train phase, Epoch: 15
##############################

Learning rate at this epoch is:  [0.0020589113209464917]


 ############################## Test phase, Epoch: 15
##############################
Accuracy:  98.12

 ############################## Train phase, Epoch: 16
##############################

Learning rate at this epoch is:  [0.0018530201888518425]


 ############################## Test phase, Epoch: 16
##############################
Accuracy:  98.12

 ############################## Train phase, Epoch: 17
##############################

Learning rate at this epoch is:  [0.0016677181699666583]


 ############################## Test phase, Epoch: 17
##############################
Accuracy:  98.15
```

```
 ############################## Train phase, Epoch: 18
##############################

Learning rate at this epoch is:  [0.0015009463529699924]


 ############################## Test phase, Epoch: 18
##############################
Accuracy:  98.15

 ############################## Train phase, Epoch: 19
##############################

Learning rate at this epoch is:  [0.0013508517176729932]


 ############################## Test phase, Epoch: 19
##############################
Accuracy:  98.14
```

[ ]:
```
%tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 163), started 0:08:33 ago. (Use '!kill␣
↪163' to kill it.)

<IPython.core.display.Javascript object>
```

[ ]:
```
```