Machine Learning Project - Assignmnet 11

Convolutional Neural Network for the classification task on MNIST

CAUCSE senior 20151145 Kim Jekyun

Computing Area

0. Preset

```
## Import required libraries
import torch
from torch import nn, optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision import transforms
import matplotlib.pyplot as plt
import numpy as np
import random
import pandas as pd
import math
%matplotlib inline
```

1. Data

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

```
data_path = './MNIST'

testing_set = datasets.MNIST(root = data_path, train= True, download=True, transform= trans
training_set = datasets.MNIST(root = data_path, train= False, download=True, transform= tra

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

```
the number of your training data (must be 10,000) =  10000
the number of your testing data (must be 60,000) =  60000
```

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
        # ********************************************************************
        self.number_class   = num_classes
        self.size_kernel    = size_kernel

        # ********************************************************************
        # feature layer
        # ********************************************************************
        self.conv1          = nn.Conv2d(1, 6, kernel_size=size_kernel, stride=1, bias=True)
        self.conv2          = nn.Conv2d(6, 16, kernel_size=size_kernel, stride=1, bias=True)
        self.conv_layer1    = nn.Sequential(self.conv1, nn.ReLU(), nn.MaxPool2d(kernel_size=
        self.conv_layer2    = nn.Sequential(self.conv2, nn.ReLU(), nn.MaxPool2d(kernel_size=

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

        # ********************************************************************
        # classifier layer
        # ********************************************************************
        self.fc1        = nn.Linear(16*4*4, 120, bias=True)
        self.fc2        = nn.Linear(120, 84, bias=True)
        self.fc3        = nn.Linear(84, num_classes)
        self.fc_layer1  = nn.Sequential(self.fc1, nn.ReLU(True))
        self.fc_layer2  = nn.Sequential(self.fc2, nn.ReLU(True))

        self.classifier = nn.Sequential(self.fc_layer1, self.fc_layer2, self.fc3)
        self._initialize_weight()

        if USE_CUDA:
            self.feature = self.feature.cuda()
            self.classifier = self.classifier.cuda()


    def _initialize_weight(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.xavier_uniform_(m.weight, gain=math.sqrt(2))
                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))
                if m.bias is not None:
                    m.bias.data.zero_()

    def forward(self, x):

        x = self.feature(x)
        dim = 1
        for i in x.size()[1:]:
            dim = i*dim
        x = x.view(-1, dim)
        x = self.classifier(x)
        x = F.softmax(x, dim=1)

        return x


# Definition of hyper parameters
learning_rate_value = 0.0005
batch_size = 64
```

```
epochs = 175

USE_CUDA = torch.cuda.is_available()
device = torch.device("cuda" if USE_CUDA else "cpu")

random.seed(777)
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)

num_train = len(training_set)
num_test = len(testing_set)
```

3. Loss function

```
criterion = nn.CrossEntropyLoss().to(device)
```

4. Optimization

```
# Dataloader & Optimizer
training_loader = DataLoader(training_set, batch_size=batch_size, shuffle=True)
testing_loader = DataLoader(testing_set, batch_size=batch_size, shuffle=False)

classifier = MyModel().to(device)
optimizer = optim.Adam(classifier.parameters(), lr=learning_rate_value)
```

```
# Training - Gradient Descent
train_loss = []
train_acc = []
test_loss = []
test_acc = []

for epoch in range(epochs):
    train_loss_tmp = 0
    train_acc_tmp = 0
    n = 0

    classifier.train()
    for data, target in training_loader:
        data, target = data.to(device), target.to(device)
        # Zero the parameter gradients
        optimizer.zero_grad()
        # Forward
        output = classifier(data)
        loss = criterion(output, target)
        # Backword
        loss.backward()
        # Loss
        train_loss_tmp += loss
        # Update
        optimizer.step()
        n += 1
        # Accuracy
        result = output.argmax(dim=1, keepdim=True)
        accuracy = result.eq(target.view_as(result)).sum()
        train_acc_tmp += accuracy

```

```
    train_loss.append(train_loss_tmp / n)
    train_acc.append(train_acc_tmp / num_train)
    if epoch%10 == 0:
        print('Training - Epoch : {}, Loss : {}, Accuracy : {}'.format(epoch, train_loss[-1]

    test_loss_tmp = 0
    test_acc_tmp = 0
    n = 0

    classifier.eval()
    with torch.no_grad():
        for data, target in testing_loader:
            data, target = data.to(device), target.to(device)
            # Forward
            output = classifier(data).to(device)
            loss = criterion(output, target).to(device)
            # Loss
            test_loss_tmp += loss
            n += 1
            # Accuracy
            result = output.argmax(dim=1, keepdim=True)
            accuracy = result.eq(target.view_as(result)).sum()
            test_acc_tmp += accuracy

    test_loss.append(test_loss_tmp / n)
    test_acc.append(test_acc_tmp / num_test)
    if epoch%25 == 0:
        print('Testing - Epoch : {}, Loss : {}, Accuracy : {}'.format(epoch, test_loss[-1],
```

```
Training - Epoch : 0, Loss : 1.9075461626052856, Accuracy : 0.5636999607086182
Testing - Epoch : 0, Loss : 1.7267872095108032, Accuracy : 0.7402166724205017
Training - Epoch : 10, Loss : 1.4774219989776611, Accuracy : 0.9850999712944031
Training - Epoch : 20, Loss : 1.4667549133300781, Accuracy : 0.9948999881744385
Testing - Epoch : 25, Loss : 1.4910619258880615, Accuracy : 0.9705666899681091
Training - Epoch : 30, Loss : 1.4661331176757812, Accuracy : 0.9955999851226807
Training - Epoch : 40, Loss : 1.4648830890655518, Accuracy : 0.9965999722480774
Training - Epoch : 50, Loss : 1.4636873006820679, Accuracy : 0.9976999759674072
Testing - Epoch : 50, Loss : 1.486364722251892, Accuracy : 0.9749000072479248
Training - Epoch : 60, Loss : 1.463984489440918, Accuracy : 0.9972999691963196
Training - Epoch : 70, Loss : 1.4625582695007324, Accuracy : 0.998699963092804
Testing - Epoch : 75, Loss : 1.4861027002334595, Accuracy : 0.9749667048454285
Training - Epoch : 80, Loss : 1.462244987487793, Accuracy : 0.9988999962806702
Training - Epoch : 90, Loss : 1.4622467756271362, Accuracy : 0.9988999962806702
Training - Epoch : 100, Loss : 1.4622459411621094, Accuracy : 0.9988999962806702
Testing - Epoch : 100, Loss : 1.4848207235336304, Accuracy : 0.9763000011444092
Training - Epoch : 110, Loss : 1.4654611349105835, Accuracy : 0.9957000017166138
Training - Epoch : 120, Loss : 1.4631671905517578, Accuracy : 0.9978999495506287
Testing - Epoch : 125, Loss : 1.4845560789108276, Accuracy : 0.9765333533287048
Training - Epoch : 130, Loss : 1.4644423723220825, Accuracy : 0.9967999458312988
Training - Epoch : 140, Loss : 1.4620469808578491, Accuracy : 0.9990999698638916
Training - Epoch : 150, Loss : 1.4618477821350098, Accuracy : 0.9993000030517578
Testing - Epoch : 150, Loss : 1.4846627712249756, Accuracy : 0.9764500260353088
Training - Epoch : 160, Loss : 1.4618476629257202, Accuracy : 0.9993000030517578
Training - Epoch : 170, Loss : 1.4618477821350098, Accuracy : 0.9993000030517578
```

```
train_loss_tot = train_loss[-1]
test_loss_tot = test_loss[-1]
train_acc_tot = train_acc[-1]
test_acc_tot = test_acc[-1]
```

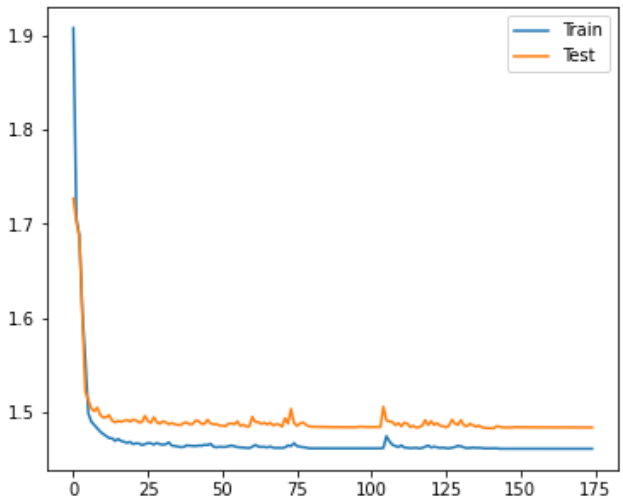
```
ind = ['training', 'testing']
con_loss = {'loss':ind, '':[round(train_loss_tot.item(), 5), round(test_loss_tot.item(), 5)]
con_acc = {'accuracy':ind, '':[round(train_acc_tot.item(), 5), round(test_acc_tot.item(), 5]
```

```
5
6 tot_loss = pd.DataFrame(con_loss).set_index('loss')
7 tot_acc = pd.DataFrame(con_acc).set_index('accuracy')
```

Result Area

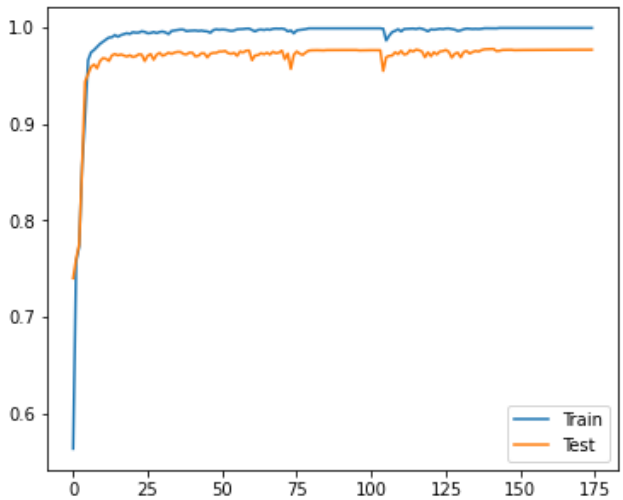
1. Plot the training and testing losses over epochs

```
1 plt.figure(figsize=(6,5))
2 plt.plot(train_loss)
3 plt.plot(test_loss)
4 plt.legend(['Train', 'Test'])
5 plt.show()
```



2. Plot the training and testing accuracies over epochs

```
1 plt.figure(figsize=(6,5))
2 plt.plot(train_acc)
3 plt.plot(test_acc)
4 plt.legend(['Train', 'Test'])
5 plt.show()
```



3. Print the final training and testing losses at convergence

```
1 tot_loss
```

	loss
training	1.46185
testing	1.48437

4. Print the final training and testing accuracies at convergence

```
1 tot_acc
```

	accuracy
training	0.99930
testing	0.97677

5. Print the testing accuracies within the last 10 epochs

```
1 for j in range(10):
2     print('[epoch = {0:03d}] {1:0.5f}'.format(epochs-(9-j), test_acc[-10+j].item()))
```

```
[epoch = 166] 0.97668
[epoch = 167] 0.97668
[epoch = 168] 0.97668
[epoch = 169] 0.97672
[epoch = 170] 0.97668
[epoch = 171] 0.97672
[epoch = 172] 0.97673
[epoch = 173] 0.97673
[epoch = 174] 0.97675
[epoch = 175] 0.97677
```