Instructions for Classification Algorithm with PyTorch

**Zian Gu**

**05/18/2020**

In project 2, I use the PyTorch to make classification algorithm. The dataset I used to make classification is MNIST. The training set contains 60,000 images and the test set contains 10,000 images. Each image is in grayscale. Thus, it’s easy to handle. The procedures and instructions are as follows:

1. Get train data by torchvision.datavision.MNIST(), save as train\_data. Set train as True, then it will create data form “training.pt”. For processing easier, I transform the data to cuda tensors and make normalization with data’s mean and standard deviation.
2. Get test data by torchvision.datavision.MNIST(), save as test\_data. Set train as False, then it will create data form “test.pt”. For processing easier, I transform the data to tensor and make normalization with data’s mean and standard deviation, too.
3. Use torch.utils.data.DataLoader() to load the train data and the test data. At the same time, I set batch size as 32, and open the shuffle, which means drawing the samples randomly to train or test the neural network model. Do it like this can make the samples more stable and make training easier. I set pin\_memory as True to accelerate the speed the training and test.
4. Define the train function. Transfer the data and target to GPU, and get the prediction. Use the prediction and target to calculate the loss. Thus, we get a loss function. Our aim is to decrease the loss function. So, I use SGD here to make gradient descent. Every time the algorithm makes 100 interactions of forward and backward, output the loss.
5. Define the test function. Transfer the data and target to GPU. Get the max one of the trained outputs. Finally, calculate the loss and accuracy of each epoch.
6. Define class Net() to create a neural network model. I created two 2-D convolutions. In forward function, my activation function is ReLU. After transferring the convolution to ReLU, I pooling them, and repeat the operation twice. Thus, the data will transform from 1\*28\*28 to 4\*4. Finally, output the result with the softmax activation functions.
7. Config the optimizer (SGD algorithm).
8. Start training and testing.
9. Save the results as "mnist\_cnn.pt".

**Result：**

Train epoch:0,interaction:0,Loss:2.289472818374634

Train epoch:0,interaction:100,Loss:0.6407032012939453

Train epoch:0,interaction:200,Loss:0.3204149603843689

Train epoch:0,interaction:300,Loss:0.10340282320976257

Train epoch:0,interaction:400,Loss:0.20435121655464172

Train epoch:0,interaction:500,Loss:0.06960785388946533

Train epoch:0,interaction:600,Loss:0.16397224366664886

Train epoch:0,interaction:700,Loss:0.054308533668518066

Train epoch:0,interaction:800,Loss:0.023791640996932983

Train epoch:0,interaction:900,Loss:0.05123170465230942

Train epoch:0,interaction:1000,Loss:0.2288484424352646

Train epoch:0,interaction:1100,Loss:0.08851347863674164

Train epoch:0,interaction:1200,Loss:0.1602393239736557

Train epoch:0,interaction:1300,Loss:0.16796903312206268

Train epoch:0,interaction:1400,Loss:0.07510010898113251

Train epoch:0,interaction:1500,Loss:0.07629606127738953

Train epoch:0,interaction:1600,Loss:0.08288237452507019

Train epoch:0,interaction:1700,Loss:0.006779968738555908

Train epoch:0,interaction:1800,Loss:0.08625960350036621

Test loss:0.05738990838527679，Accuracy:98.13

Train epoch:1,interaction:0,Loss:0.06285350769758224

Train epoch:1,interaction:100,Loss:0.2574132978916168

Train epoch:1,interaction:200,Loss:0.025152862071990967

Train epoch:1,interaction:300,Loss:0.0012662112712860107

Train epoch:1,interaction:400,Loss:0.16512173414230347

Train epoch:1,interaction:500,Loss:0.16722336411476135

Train epoch:1,interaction:600,Loss:0.028076812624931335

Train epoch:1,interaction:700,Loss:0.005281195044517517

Train epoch:1,interaction:800,Loss:0.006769746541976929

Train epoch:1,interaction:900,Loss:0.05106577277183533

Train epoch:1,interaction:1000,Loss:0.014953359961509705

Train epoch:1,interaction:1100,Loss:0.12080337107181549

Train epoch:1,interaction:1200,Loss:0.05194117873907089

Train epoch:1,interaction:1300,Loss:0.018856287002563477

Train epoch:1,interaction:1400,Loss:0.06751567125320435

Train epoch:1,interaction:1500,Loss:0.009818866848945618

Train epoch:1,interaction:1600,Loss:0.050221703946590424

Train epoch:1,interaction:1700,Loss:0.00966520607471466

Train epoch:1,interaction:1800,Loss:0.027836665511131287

Test loss:0.043577573537826535，Accuracy:98.54

Train epoch:2,interaction:0,Loss:0.04561026394367218

Train epoch:2,interaction:100,Loss:0.0069593340158462524

Train epoch:2,interaction:200,Loss:0.0009650588035583496

Train epoch:2,interaction:300,Loss:0.00883011519908905

Train epoch:2,interaction:400,Loss:0.0032729506492614746

Train epoch:2,interaction:500,Loss:0.0035886764526367188

Train epoch:2,interaction:600,Loss:0.0554439052939415

Train epoch:2,interaction:700,Loss:0.01086321473121643

Train epoch:2,interaction:800,Loss:0.18476802110671997

Train epoch:2,interaction:900,Loss:0.029095515608787537

Train epoch:2,interaction:1000,Loss:0.0054645538330078125

Train epoch:2,interaction:1100,Loss:0.04616127908229828

Train epoch:2,interaction:1200,Loss:0.03324700891971588

Train epoch:2,interaction:1300,Loss:0.0023902207612991333

Train epoch:2,interaction:1400,Loss:0.014314711093902588

Train epoch:2,interaction:1500,Loss:0.00792171061038971

Train epoch:2,interaction:1600,Loss:0.0014771819114685059

Train epoch:2,interaction:1700,Loss:0.042154744267463684

Train epoch:2,interaction:1800,Loss:0.02960243821144104

Test loss:0.04635755715370178，Accuracy:98.4

Train epoch:3,interaction:0,Loss:0.022247061133384705

Train epoch:3,interaction:100,Loss:0.005449756979942322

Train epoch:3,interaction:200,Loss:0.0018336176872253418

Train epoch:3,interaction:300,Loss:0.039770856499671936

Train epoch:3,interaction:400,Loss:0.004088446497917175

Train epoch:3,interaction:500,Loss:0.0057825446128845215

Train epoch:3,interaction:600,Loss:0.015076041221618652

Train epoch:3,interaction:700,Loss:0.07334397733211517

Train epoch:3,interaction:800,Loss:0.001173853874206543

Train epoch:3,interaction:900,Loss:0.005761280655860901

Train epoch:3,interaction:1000,Loss:0.02849668264389038

Train epoch:3,interaction:1100,Loss:0.0032801032066345215

Train epoch:3,interaction:1200,Loss:0.11731186509132385

Train epoch:3,interaction:1300,Loss:0.024792462587356567

Train epoch:3,interaction:1400,Loss:0.001034468412399292

Train epoch:3,interaction:1500,Loss:0.013856709003448486

Train epoch:3,interaction:1600,Loss:0.010026663541793823

Train epoch:3,interaction:1700,Loss:0.007746607065200806

Train epoch:3,interaction:1800,Loss:0.0009859204292297363

Test loss:0.031498560333251956，Accuracy:98.94

Train epoch:4,interaction:0,Loss:0.0015056133270263672

Train epoch:4,interaction:100,Loss:0.0005371570587158203

Train epoch:4,interaction:200,Loss:0.002870023250579834

Train epoch:4,interaction:300,Loss:0.00020578503608703613

Train epoch:4,interaction:400,Loss:0.00250302255153656

Train epoch:4,interaction:500,Loss:0.017371445894241333

Train epoch:4,interaction:600,Loss:0.0073190778493881226

Train epoch:4,interaction:700,Loss:0.0037694573402404785

Train epoch:4,interaction:800,Loss:0.0015969276428222656

Train epoch:4,interaction:900,Loss:0.03371620178222656

Train epoch:4,interaction:1000,Loss:0.0012650489807128906

Train epoch:4,interaction:1100,Loss:0.018079489469528198

Train epoch:4,interaction:1200,Loss:0.0005759596824645996

Train epoch:4,interaction:1300,Loss:0.027456670999526978

Train epoch:4,interaction:1400,Loss:0.01012471318244934

Train epoch:4,interaction:1500,Loss:0.0003636777400970459

Train epoch:4,interaction:1600,Loss:0.002034813165664673

Train epoch:4,interaction:1700,Loss:0.002942889928817749

Train epoch:4,interaction:1800,Loss:0.0007604658603668213

Test loss:0.030772854399681093，Accuracy:99.05000000000001

**Conclusion:**

The MNIST is a classic dataset for image classification with neural network. It may be easy to process. After 4 epochs, the accuracy of neural network like this can up to 99%. In fact, we have a more complicated MNIST, called fashion-MNIST. It might be difficult to find the proper prediction function. According to my learning material, if I still use the simple neural network like above, the accuracy can decrease to 83%. In this case, AlexNet might be a good choice.

**Source code:**

#Classification Algorithm with PyTorch

#by Zian Gu

#05/18/2020

#导入所需的包

import torch

import torchvision

import sys

import os

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

from PIL import Image

import numpy as np

#print(torch.cuda.is\_available());

local\_file = "C:\\Users\\gu573\\Documents\\GitHub\\Computer Vision\\classification algorithm"

batch\_size = 32 #批大小

learning\_rate = 0.015 #学习率

momentum = 0.5 #用于随机梯度下降算法的参数，可以起到加速效果

num\_epochs = 5 #迭代次数

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

def img\_show(img):

pil\_img = Image.fromarray(np.uint8(img))

pil\_img.show()

def train(model,device,train\_loader,optimizer,epoch):

model.train()

for index,(data,target) in enumerate(train\_loader):

data,target = data.to(device),target.to(device)

prediction = model(data)

loss = F.nll\_loss(prediction,target)

#SGD

optimizer.zero\_grad()

loss.backward()

optimizer.step()

if index % 100 == 0:

print("Train epoch:{},interaction:{},Loss:{}".format(epoch,index,loss.item()))

#测试

def test(model,device,test\_loader):

model.eval()

total\_loss = 0.

correct = 0.

with torch.no\_grad():

for index,(data,target) in enumerate(test\_loader):

data,target = data.to(device),target.to(device)

output = model(data)

total\_loss += F.nll\_loss(output,target,reduction="sum").item()

prediction = output.argmax(dim=1)

correct+=prediction.eq(target.view\_as(prediction)).sum().item()

total\_loss/=len(test\_loader.dataset)

accuracy = correct / len(test\_loader.dataset) \* 100.

print("Test loss:{}，Accuracy:{}\n".format(total\_loss,accuracy))

#神经网络模型

class Net(nn.Module):

def \_\_init\_\_(self):

super(Net,self).\_\_init\_\_()

self.conv1 = nn.Conv2d(1,20,5,1)

self.conv2 = nn.Conv2d(20,50,5,1)

self.fc1 = nn.Linear(4 \* 4 \* 50,500)

self.fc2 = nn.Linear(500,10)

def forward(self,x):

x = F.relu(self.conv1(x))

x = F.max\_pool2d(x,2,2)

x = F.relu(self.conv2(x))

x = F.max\_pool2d(x,2,2)

x = x.view(-1,4 \* 4 \* 50)

x = F.relu(self.fc1(x))

x = self.fc2(x)

return F.log\_softmax(x,dim=1)

#Data set

train\_data = torchvision.datasets.MNIST(root=local\_file,

train=True,

transform=torchvision.transforms.Compose([torchvision.transforms.ToTensor(),torchvision.transforms.Normalize((0.1307,), (0.3081,))]))

test\_data = torchvision.datasets.MNIST(root=local\_file,

train=False,

transform=torchvision.transforms.Compose([torchvision.transforms.ToTensor(),torchvision.transforms.Normalize((0.1307,), (0.3081,))]))

# Data loader

train\_loader = torch.utils.data.DataLoader(dataset=train\_data,

batch\_size=batch\_size,

shuffle=True,

pin\_memory=True)

test\_loader = torch.utils.data.DataLoader(dataset=test\_data,

batch\_size=batch\_size,

shuffle=True,

pin\_memory=True)

#img\_show(train\_data[223][0].reshape(28,28))#显示图片

model = Net().to(device)

optimizer = torch.optim.SGD(model.parameters(),lr=learning\_rate,momentum=momentum)#随机梯度下降算法，momentum用于加速

for epoch in range(num\_epochs):

train(model,device,train\_loader,optimizer,epoch)

test(model,device,test\_loader)

torch.save(model.state\_dict(),"mnist\_cnn.pt")