**实验5基于循环神经网络的Minist数据集分类**

**一、实验目的**

1．掌握使用pytorch构建RNN网络的方法；

2．了解训练RNN的技巧。

**二、实验内容**

1.将Minist数据集转换为RNN能够训练的形态；

2.采用不同深度的RNN训练Minist数据集；

3.分析不同RNN网络结构的速度和精度。

**三、实验指导**

**1. 数据集：**

28\*28的图片，每个输入序列长度（seq\_len）为28，每个输入的元素维度(input\_size)为28，将一张图片的分为28列，为长度28的序列，序列中每个元素为28个元素（即每一列的像素）。

注意，如果batch\_first设置为1，则输出维度out： batch, seq\_len, hidden\_size

**2.方案**

**RNN1:** 输出的时候，输出的维度（hidden\_size）为一个较大值（例如128），然后取最右上角输出，用linear映射到10的类别上面

**RNN2：**直接输出维度（hidden\_size）为10，作为类别，取最右上角输出作为输出，计算loss

由于RNN很难记住很久之前的，为了增强记忆性，想到将前面输出的一起做个全连接，增强记忆性。

**RNN3：**hidden\_size=1 ，将所有输出的28个1维度（28个序列，每个序列1维度），最后 linear-> 10

**RNN4 :** hidden\_size = 128, 每次先对每个序列元素的128个维度 linear-> 1，最后对序列中28个元素 linear-> 10

**RNN5：**hidden\_size=128，对序列中28个128维度的输出，直接torch.cat，合并为1个128\*28维度的tensor，做128\*28 linear-> 10

**结果：**

训练5个epochs

RNN1: 右上角 128 linear-> 10

Test ACC:0.9581

RNN2：右上角直接10维度

Test ACC:0.4454

由于RNN很难记住很久之前的，为了增强记忆性，想到将前面输出的一起做个全连接。

RNN3：hidden\_size=1 将所有输出的28个1维度（28个序列，每个序列1维度） linear-> 10

Test ACC:0.7156

RNN4:hidden\_size = 128,每次先对每个序列元素的128个维度 linear-> 1，最后对序列中28个元素 linear-> 10

Test ACC:0.6606

RNN5：hidden\_size=128，对序列中28个128维度的输出，直接torch.cat，合并为1个128\*28维度的tensor，做128\*28 linear-> 10

Test ACC:0.9789

RNN5精度最高，速度也最慢。

关于RNN与CNN在MNIST上面的精度，具体要看输出RNN输出维度大小，以及RNN的设计，以及CNN的规模。同等计算量下（设计都合理），应该是CNN精度更高，因为RNN将图像当成序列，忽略了图像每个像素与上下左右像素之间的关联（准确来讲是忽略了左右关联），而CNN卷积的时候对一个个像素块进行卷积，更加考虑到了图像的像素块之间的联系（上下左右联系）。

**3.实验对比结果分析：**

hidden\_size越大，抽取的特征越多，结果越准确，所以RNN3精度较低。一开始以为RNN4精度会比较高，结果却比较低，我认为应该是将128->1的过程中丢失了大量的信息，导致精度很低。

RNN2的问题也在于抽取的特征太少（hidden\_size太小）。

所以现在比较流行的做法是RNN1，直接取序列最后一个输出（右上角输出），输出维度（hidden\_size）设置的比较大，然后用一个linear映射到类别数量。速度还可以。之前看到RNN长时间记忆性不行，于是想到对所有的向上输出都拿来linear，人工进行记忆，如RNN5，（RNN4不谈，第一次linear过程丢失了大量信息），RNN5的精度确实高于RNN1，但是，速度慢了很多很多。

**RNN1：**

import torch

import torch.nn as nn

import torchvision

from torchvision import datasets,transforms

from torch.autograd import Variable

from matplotlib import pyplot as plt

device = torch.device('cuda')

class RNN(nn.Module):

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

super().\_\_init\_\_()

self.rnn = nn.RNN(

input\_size = 28,

hidden\_size = 128,

num\_layers = 1,

batch\_first = True,

)

self.Out2Class = nn.Linear(128,10)

def forward(self, input):

output,hn = self.rnn(input,None)

print('hn,shape:{}'.format(hn.shape))

tmp = self.Out2Class(output[:,-1,:]) #output[:,-1,:]是取输出序列中的最后一个，也可以用hn[0,:,:]或者hn.squeeze(0)代替,

# 为什么用hn[0,:,:],而不是hn,因为hn第一个维度为num\_layers \* num\_directions，此处为1，即hn为(1,x,x)，需要去掉1

# 这边将最右上角的输出的128维度映射到10的分类上面去

return tmp

model = RNN()

model = model.to(device)

print(model)

model = model.train()

img\_transform = transforms.Compose([transforms.ToTensor(),

transforms.Normalize(mean = [0.5,0.5,0.5],std = [0.5,0.5,0.5])])

dataset\_train = datasets.MNIST(root = './data',transform = img\_transform,train = True,download = True)

dataset\_test = datasets.MNIST(root = './data',transform = img\_transform,train = False,download = True)

train\_loader = torch.utils.data.DataLoader(dataset = dataset\_train,batch\_size=64,shuffle = True)

test\_loader = torch.utils.data.DataLoader(dataset = dataset\_test,batch\_size=64,shuffle = False)

# images,label = next(iter(train\_loader))

# print(images.shape)

# print(label.shape)

# images\_example = torchvision.utils.make\_grid(images)

# images\_example = images\_example.numpy().transpose(1,2,0)

# mean = [0.5,0.5,0.5]

# std = [0.5,0.5,0.5]

# images\_example = images\_example\*std + mean

# plt.imshow(images\_example)

# plt.show()

def Get\_ACC():

correct = 0

total\_num = len(dataset\_test)

for item in test\_loader:

batch\_imgs,batch\_labels = item

batch\_imgs = batch\_imgs.squeeze(1)

batch\_imgs = Variable(batch\_imgs)

batch\_imgs = batch\_imgs.to(device)

batch\_labels = batch\_labels.to(device)

out = model(batch\_imgs)

\_,pred = torch.max(out.data,1)

correct += torch.sum(pred==batch\_labels)

# print(pred)

# print(batch\_labels)

correct = correct.data.item()

acc = correct/total\_num

print('correct={},Test ACC:{:.5}'.format(correct,acc))

optimizer = torch.optim.Adam(model.parameters())

loss\_f = nn.CrossEntropyLoss()

Get\_ACC()

for epoch in range(10):

print('epoch:{}'.format(epoch))

cnt = 0

for item in train\_loader:

batch\_imgs ,batch\_labels = item

batch\_imgs = batch\_imgs.squeeze(1)

# print(batch\_imgs.shape)

batch\_imgs,batch\_labels = Variable(batch\_imgs),Variable(batch\_labels)

batch\_imgs = batch\_imgs.to(device)

batch\_labels = batch\_labels.to(device)

out = model(batch\_imgs)

# print(out.shape)

loss = loss\_f(out,batch\_labels)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

if(cnt%100==0):

print\_loss = loss.data.item()

print('epoch:{},cnt:{},loss:{}'.format(epoch,cnt,print\_loss))

cnt+=1

Get\_ACC()

torch.save(model,'model')

**RNN2：**

import torch

import torch.nn as nn

import torchvision

from torchvision import datasets,transforms

from torch.autograd import Variable

from matplotlib import pyplot as plt

import sys

device = torch.device('cuda')

class RNN(nn.Module):

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

super().\_\_init\_\_()

self.rnn = nn.RNN(

input\_size = 28,

hidden\_size = 10,

num\_layers = 1,

batch\_first = True,

)

def forward(self, input):

output,hn = self.rnn(input,None)

hn = hn[0,:,:]

# print(hn.shape)

# last = output[0,-1,:]

# print('outlast:{}'.format(last))

# tmp\_hn = hn[0,:]

# print('hn:{}'.format(tmp\_hn))

# print(hn.shape)

# hn = hn.squeeze(0)

# print('hn,shape:{}'.format(hn.shape))

return hn

model = RNN()

model = model.to(device)

print(model)

model = model.train()

img\_transform = transforms.Compose([transforms.ToTensor(),

transforms.Normalize(mean = [0.5,0.5,0.5],std = [0.5,0.5,0.5])])

dataset\_train = datasets.MNIST(root = './data',transform = img\_transform,train = True,download = True)

dataset\_test = datasets.MNIST(root = './data',transform = img\_transform,train = False,download = True)

train\_loader = torch.utils.data.DataLoader(dataset = dataset\_train,batch\_size=64,shuffle = True)

test\_loader = torch.utils.data.DataLoader(dataset = dataset\_test,batch\_size=64,shuffle = False)

# images,label = next(iter(train\_loader))

# print(images.shape)

# print(label.shape)

# images\_example = torchvision.utils.make\_grid(images)

# images\_example = images\_example.numpy().transpose(1,2,0)

# mean = [0.5,0.5,0.5]

# std = [0.5,0.5,0.5]

# images\_example = images\_example\*std + mean

# plt.imshow(images\_example)

# plt.show()

def Get\_ACC():

correct = 0

total\_num = len(dataset\_test)

for item in test\_loader:

batch\_imgs,batch\_labels = item

batch\_imgs = batch\_imgs.squeeze(1)

batch\_imgs = Variable(batch\_imgs)

batch\_imgs = batch\_imgs.to(device)

batch\_labels = batch\_labels.to(device)

out = model(batch\_imgs)

\_,pred = torch.max(out.data,1)

correct += torch.sum(pred==batch\_labels)

# print(pred)

# print(batch\_labels)

correct = correct.data.item()

acc = correct/total\_num

print('correct={},Test ACC:{:.5}'.format(correct,acc))

optimizer = torch.optim.Adam(model.parameters())

loss\_f = nn.CrossEntropyLoss()

Get\_ACC()

for epoch in range(10):

print('epoch:{}'.format(epoch))

cnt = 0

for item in train\_loader:

batch\_imgs ,batch\_labels = item

batch\_imgs = batch\_imgs.squeeze(1)

# print(batch\_imgs.shape)

batch\_imgs,batch\_labels = Variable(batch\_imgs),Variable(batch\_labels)

batch\_imgs = batch\_imgs.to(device)

batch\_labels = batch\_labels.to(device)

out = model(batch\_imgs)

# print(out.shape)

loss = loss\_f(out,batch\_labels)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

if(cnt%100==0):

print\_loss = loss.data.item()

print('epoch:{},cnt:{},loss:{}'.format(epoch,cnt,print\_loss))

cnt+=1

Get\_ACC()

torch.save(model,'model')

**RNN3:**

import torch

import torch.nn as nn

import torchvision

from torchvision import datasets,transforms

from torch.autograd import Variable

from matplotlib import pyplot as plt

device = torch.device('cuda')

class RNN(nn.Module):

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

super().\_\_init\_\_()

self.rnn = nn.RNN(

input\_size = 28,

hidden\_size = 1,

num\_layers = 1,

batch\_first = True,

)

self.Out2Class = nn.Linear(28,10)

def forward(self, input):

output,hn = self.rnn(input,None)

# print('hn,shape:{}'.format(hn.shape))

outreshape = output[:,:,0]

# print(outreshape.shape)

tmp = self.Out2Class(outreshape)

# print(tmp.shape)

return tmp

model = RNN()

model = model.to(device)

print(model)

model = model.train()

img\_transform = transforms.Compose([transforms.ToTensor(),

transforms.Normalize(mean = [0.5,0.5,0.5],std = [0.5,0.5,0.5])])

dataset\_train = datasets.MNIST(root = './data',transform = img\_transform,train = True,download = True)

dataset\_test = datasets.MNIST(root = './data',transform = img\_transform,train = False,download = True)

train\_loader = torch.utils.data.DataLoader(dataset = dataset\_train,batch\_size=64,shuffle = True)

test\_loader = torch.utils.data.DataLoader(dataset = dataset\_test,batch\_size=64,shuffle = False)

# images,label = next(iter(train\_loader))

# print(images.shape)

# print(label.shape)

# images\_example = torchvision.utils.make\_grid(images)

# images\_example = images\_example.numpy().transpose(1,2,0)

# mean = [0.5,0.5,0.5]

# std = [0.5,0.5,0.5]

# images\_example = images\_example\*std + mean

# plt.imshow(images\_example)

# plt.show()

def Get\_ACC():

correct = 0

total\_num = len(dataset\_test)

for item in test\_loader:

batch\_imgs,batch\_labels = item

batch\_imgs = batch\_imgs.squeeze(1)

batch\_imgs = Variable(batch\_imgs)

batch\_imgs = batch\_imgs.to(device)

batch\_labels = batch\_labels.to(device)

out = model(batch\_imgs)

\_,pred = torch.max(out.data,1)

correct += torch.sum(pred==batch\_labels)

# print(pred)

# print(batch\_labels)

correct = correct.data.item()

acc = correct/total\_num

print('correct={},Test ACC:{:.5}'.format(correct,acc))

optimizer = torch.optim.Adam(model.parameters())

loss\_f = nn.CrossEntropyLoss()

Get\_ACC()

for epoch in range(5):

print('epoch:{}'.format(epoch))

cnt = 0

for item in train\_loader:

batch\_imgs ,batch\_labels = item

batch\_imgs = batch\_imgs.squeeze(1)

# print(batch\_imgs.shape)

batch\_imgs,batch\_labels = Variable(batch\_imgs),Variable(batch\_labels)

batch\_imgs = batch\_imgs.to(device)

batch\_labels = batch\_labels.to(device)

out = model(batch\_imgs)

# print(out.shape)

loss = loss\_f(out,batch\_labels)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

if(cnt%100==0):

print\_loss = loss.data.item()

print('epoch:{},cnt:{},loss:{}'.format(epoch,cnt,print\_loss))

cnt+=1

Get\_ACC()

torch.save(model,'model')

**RNN4:**

import torch

import torch.nn as nn

import torchvision

from torchvision import datasets,transforms

from torch.autograd import Variable

from matplotlib import pyplot as plt

import sys

device = torch.device('cpu')

class RNN(nn.Module):

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

super().\_\_init\_\_()

self.rnn = nn.RNN(

input\_size = 28,

hidden\_size = 128,

num\_layers = 1,

batch\_first = True,

)

self.hidden2one\_list = []

for i in range(28):

self.hidden2one\_list.append(nn.Linear(128,1))

self.Out2Class = nn.Linear(28,10)

def forward(self, input):

output,hn = self.rnn(input,None)

hidden2one\_res = []

for i in range(28):

tmp\_res = self.hidden2one\_list[i](output[:,i,:])

# print(tmp\_res.shape)

hidden2one\_res.append(tmp\_res.data)

hidden2one\_res = torch.cat(hidden2one\_res,dim=1) #或者先对hidden2one\_res中的元素squeeze(1),再用torch.stack

# print(hidden2one\_res.shape) #torch.Size([64, 28])

res = self.Out2Class(hidden2one\_res)

return res

model = RNN()

model = model.to(device)

print(model)

model = model.train()

img\_transform = transforms.Compose([transforms.ToTensor(),

transforms.Normalize(mean = [0.5,0.5,0.5],std = [0.5,0.5,0.5])])

dataset\_train = datasets.MNIST(root = './data',transform = img\_transform,train = True,download = True)

dataset\_test = datasets.MNIST(root = './data',transform = img\_transform,train = False,download = True)

train\_loader = torch.utils.data.DataLoader(dataset = dataset\_train,batch\_size=64,shuffle = True)

test\_loader = torch.utils.data.DataLoader(dataset = dataset\_test,batch\_size=64,shuffle = False)

# images,label = next(iter(train\_loader))

# print(images.shape)

# print(label.shape)

# images\_example = torchvision.utils.make\_grid(images)

# images\_example = images\_example.numpy().transpose(1,2,0)

# mean = [0.5,0.5,0.5]

# std = [0.5,0.5,0.5]

# images\_example = images\_example\*std + mean

# plt.imshow(images\_example)

# plt.show()

def Get\_ACC():

correct = 0

total\_num = len(dataset\_test)

for item in test\_loader:

batch\_imgs,batch\_labels = item

batch\_imgs = batch\_imgs.squeeze(1)

batch\_imgs = Variable(batch\_imgs)

batch\_imgs = batch\_imgs.to(device)

batch\_labels = batch\_labels.to(device)

out = model(batch\_imgs)

\_,pred = torch.max(out.data,1)

correct += torch.sum(pred==batch\_labels)

# print(pred)

# print(batch\_labels)

correct = correct.data.item()

acc = correct/total\_num

print('correct={},Test ACC:{:.5}'.format(correct,acc))

optimizer = torch.optim.Adam(model.parameters())

loss\_f = nn.CrossEntropyLoss()

Get\_ACC()

for epoch in range(5):

print('epoch:{}'.format(epoch))

cnt = 0

for item in train\_loader:

batch\_imgs ,batch\_labels = item

batch\_imgs = batch\_imgs.squeeze(1)

# print(batch\_imgs.shape)

batch\_imgs,batch\_labels = Variable(batch\_imgs),Variable(batch\_labels)

batch\_imgs = batch\_imgs.to(device)

batch\_labels = batch\_labels.to(device)

out = model(batch\_imgs)

# print(out.shape)

loss = loss\_f(out,batch\_labels)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

if(cnt%100==0):

print\_loss = loss.data.item()

print('epoch:{},cnt:{},loss:{}'.format(epoch,cnt,print\_loss))

cnt+=1

Get\_ACC()

torch.save(model,'model')

**RNN5：**

import torch

import torch.nn as nn

import torchvision

from torchvision import datasets,transforms

from torch.autograd import Variable

from matplotlib import pyplot as plt

import sys

device = torch.device('cuda')

class RNN(nn.Module):

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

super().\_\_init\_\_()

self.rnn = nn.RNN(

input\_size = 28,

hidden\_size = 128,

num\_layers = 1,

batch\_first = True,

)

self.Out2Class = nn.Linear(128\*28,10)

def forward(self, input):

output,hn = self.rnn(input,None)

hidden2one\_res = []

for i in range(28):

hidden2one\_res.append(output[:,i,:])

hidden2one\_res = torch.cat(hidden2one\_res,dim=1)

# print(hidden2one\_res.shape)

res = self.Out2Class(hidden2one\_res)

return res

model = RNN()

model = model.to(device)

print(model)

model = model.train()

img\_transform = transforms.Compose([transforms.ToTensor(),

transforms.Normalize(mean = [0.5,0.5,0.5],std = [0.5,0.5,0.5])])

dataset\_train = datasets.MNIST(root = './data',transform = img\_transform,train = True,download = True)

dataset\_test = datasets.MNIST(root = './data',transform = img\_transform,train = False,download = True)

train\_loader = torch.utils.data.DataLoader(dataset = dataset\_train,batch\_size=64,shuffle = True)

test\_loader = torch.utils.data.DataLoader(dataset = dataset\_test,batch\_size=64,shuffle = False)

# images,label = next(iter(train\_loader))

# print(images.shape)

# print(label.shape)

# images\_example = torchvision.utils.make\_grid(images)

# images\_example = images\_example.numpy().transpose(1,2,0)

# mean = [0.5,0.5,0.5]

# std = [0.5,0.5,0.5]

# images\_example = images\_example\*std + mean

# plt.imshow(images\_example)

# plt.show()

def Get\_ACC():

correct = 0

total\_num = len(dataset\_test)

for item in test\_loader:

batch\_imgs,batch\_labels = item

batch\_imgs = batch\_imgs.squeeze(1)

batch\_imgs = Variable(batch\_imgs)

batch\_imgs = batch\_imgs.to(device)

batch\_labels = batch\_labels.to(device)

out = model(batch\_imgs)

\_,pred = torch.max(out.data,1)

correct += torch.sum(pred==batch\_labels)

# print(pred)

# print(batch\_labels)

correct = correct.data.item()

acc = correct/total\_num

print('correct={},Test ACC:{:.5}'.format(correct,acc))

optimizer = torch.optim.Adam(model.parameters())

loss\_f = nn.CrossEntropyLoss()

Get\_ACC()

for epoch in range(5):

print('epoch:{}'.format(epoch))

cnt = 0

for item in train\_loader:

batch\_imgs ,batch\_labels = item

batch\_imgs = batch\_imgs.squeeze(1)

# print(batch\_imgs.shape)

batch\_imgs,batch\_labels = Variable(batch\_imgs),Variable(batch\_labels)

batch\_imgs = batch\_imgs.to(device)

batch\_labels = batch\_labels.to(device)

out = model(batch\_imgs)

# print(out.shape)

loss = loss\_f(out,batch\_labels)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

if(cnt%100==0):

print\_loss = loss.data.item()

print('epoch:{},cnt:{},loss:{}'.format(epoch,cnt,print\_loss))

cnt+=1

Get\_ACC()

torch.save(model,'model')