# CNN识别MNIST

2023年8月8日 19:45

```
importtorch
fromtorchvisionimporttransforms
fromtorchvisionimportdatasets
fromtorch.utils.dataimportDataLoader
importtorch.optimasoptim
importtorch.nn.functionalasF
importmatplotlib.pyplotasplt
```

库的导入

```
batch_size=64

transform=transforms.Compose([
transforms.ToTensor(),
transforms.Normalize((0.1307),(0.3081))#两个参数，平均值和标准差

])

train_dataset=datasets.MNIST(
root='.科研\胡小永老师组\MNIST数据集',
train=True,
download=True,
transform=transform
)

train_loader=DataLoader(train_dataset,
shuffle=True,
batch_size=batch_size)

test_dataset=datasets.MNIST(
root='.科研\胡小永老师组\MNIST数据集',
train=False,
download=True,
transform=transform
)

test_loder=DataLoader(test_dataset,
shuffle=True,
batch_size=batch_size)
```

这段代码是一个用于加载MNIST数据集并进行数据预处理的代码。下面是对代码的解释：

1. 首先定义了一个`batch_size`变量，表示每个批次中的样本数量。

2. `transform`是一个数据预处理的操作序列，将图像数据转换为张量，并进行归一化处理。

3. 使用`datasets.MNIST`函数加载MNIST训练集和测试集数据。
   - `root`参数指定数据集存储的路径。
   - `train=True`表示加载训练集数据。
   - `train=False`表示加载测试集数据。
   - `download=True`表示如果数据集不存在时需要从网络下载数据。
   - `transform=transform`表示对数据应用预处理操作。

4. 创建训练集和测试集的`DataLoader`对象，用于批量加载数据。
   - `DataLoader`是一个用于自动加载数据的工具类。
   - `shuffle=True`表示在每个epoch中对数据进行随机洗牌。
   - `batch_size`参数指定每个批次中的样本数量。

```
classNet(torch.nn.Module):
def__init__(self):
super(Net,self).__init__()
self.conv1=torch.nn.Conv2d(in_channels=1,out_channels=10,kernel_size=3)
self.conv2=torch.nn.Conv2d(in_channels=10,out_channels=20,kernel_size=3)
self.conv3=torch.nn.Conv2d(in_channels=20,out_channels=40,kernel_size=3)
self.pooling1=torch.nn.MaxPool2d(kernel_size=2)
self.pooling2=torch.nn.MaxPool2d(kernel_size=2)
self.pooling3=torch.nn.MaxPool2d(kernel_size=2)
self.linear1=torch.nn.Linear(40,32)#想确定40这个值？是和
self.linear2=torch.nn.Linear(32,10)

defforward(self,x):
x=self.conv1(x)
x=F.relu(x)
x=self.pooling1(x)
x=self.conv2(x)
x=F.relu(x)
x=self.pooling2(x)
x=self.conv3(x)
x=F.relu(x)
x=self.pooling3(x)
x=x.view(x.size(0),-1)#Flatten改变张量形状
#print(x.size(-1))
#此时x.sixe()[64,40]对应liner1中的40，具体linear1的40读者可以算出来，也可以采用偷懒的方法，运行
代码，由print(x.size(-1))确定
x=self.linear1(x)
x=self.linear2(x)
returnx#最后一层不做激活，因为下一步输入到交叉损失函数中，交叉熵包含了激活层
```

这段代码定义了一个名为`Net`的神经网络模型，该模型用于对图像进行分类。下面是对代码的解释：

1. `Net`类继承自`torch.nn.Module`类，它是PyTorch中构建神经网络模型的基类。

2. 在`__init__`方法中，定义了神经网络的各个层：
   - `self.conv1`是一个卷积层，输入通道数为1，输出通道数为10，卷积核大小为3。
   - `self.conv2`是另一个卷积层，输入通道数为10，输出通道数为20，卷积核大小为3。
   - `self.conv3`是第三个卷积层，输入通道数为20，输出通道数为40，卷积核大小为3。
   - `self.pooling1`、`self.pooling2`和`self.pooling3`是最大池化层，池化核大小为2。
   - `self.linear1`是一个全连接层，输入大小为40，输出大小为32。
   - `self.linear2`是最后一个全连接层，输入大小为32，输出大小为10（对应10个类别）。

3. 在`forward`方法中，定义了前向传播过程：
   - `x`表示输入的图像数据。
   - `x`依次经过卷积层、ReLU激活函数和最大池化层的操作。
   - `x`经过Flatten操作将其形状从[batch_size, channels, height, width]改变为[batch_size, flatten_size]。
   - `x`经过两个全连接层，得到最终的输出结果。

这段代码的作用是定义了一个用于图像分类的神经网络模型，包含了卷积层、池化层和全连接层，并定义了网络的前向传播过程。

```
model=Net()
#有GPU就使用GPU，没有就是用CPU
device=torch.device("cuda"iftorch.cuda.is_available()else"cpu")
model.to(device)

criterion=torch.nn.CrossEntropyLoss()
optimizer=optim.SGD(model.parameters(),lr=0.01,momentum=0.5)
```

```
deftrain(epoch):
total=0
running_loss=0.0
train_loss=0.0#记录每次epoch的损失
accuracy=0#记录每次epoch的accuracy
forbatch_id,datainenumerate(train_loader,0):
inputs,target=data
inputs,target=inputs.to(device),target.to(device)
optimizer.zero_grad()
#forword+backward+update
outputs=model(inputs)
loss=criterion(outputs,target)
```

这段代码定义了一个用于训练神经网络模型的函数`train(epoch)`，下面是对代码的解释：

1. `total = 0`：用于记录训练样本的总数。
2. `running_loss = 0.0`：记录每个batch的累计损失。
3. `train_loss = 0.0`：记录每个epoch的损失，即每个epoch中所有batch的累计损失。
4. `accuracy = 0`：记录每个epoch的准确率。
5. `for batch_id, data in enumerate(train_loader, 0):`：遍历训练集数据，其中`train_loader`是一个数据加载器对象，用于批量加载训练数据。
6. `inputs, target = data`：将输入数据和目标标签从`data`中分离出来。
7. `inputs, target = inputs.to(device), target.to(device)`：将输入数据和目标标签移动到指定的计算设备上。

```
_,predicted=torch.max(outputs.data,dim=1)
accuracy+=(predicted==target).sum().item()
total+=target.size(0)

loss.backward()
optimizer.step()

running_loss+=loss.item()
train_loss=running_loss
#每迭代300次，求一下这三百次迭代的平均
ifbatch_id%300==299:
print('[%d,%5d]loss:%.3f'%(epoch+1,batch_id+1,running_loss/300))
running_loss=0.0
print('第%depoch的Accuracyontrainset:%d%%,Lossontrainset:%f'%(
epoch+1,100*accuracy/total,train_loss))

#返回acc和loss
return1.0*accuracy/total,train_loss
```

7. `inputs, target = inputs.to(device), target.to(device)`：将输入数据和目标标签移动到指定的计算设备上。
8. `optimizer.zero_grad()`：梯度清零，用于每个batch更新前将之前的梯度置零。
9. `outputs = model(inputs)`：将输入数据输入到模型中，得到模型的输出。
10. `loss = criterion(outputs, target)`：计算模型输出与目标标签之间的损失。
11. `_, predicted = torch.max(outputs.data, dim=1)`：根据模型的输出结果得到预测的类别。
12. `accuracy += (predicted == target).sum().item()`：计算预测准确的数量，并累加到`accuracy`中。
13. `total += target.size(0)`：更新训练样本的总数。
14. `loss.backward()`：反向传播，计算梯度。
15. `optimizer.step()`：更新模型参数。
16. `running_loss += loss.item()`：累加每个batch的损失。
17. `train_loss = running_loss`：将累计损失作为该epoch的损失。
18. `if batch_id % 300 == 299:`：每迭代300次，打印一次损失平均值。
19. `print('[%d, %5d] loss: %.3f' % (epoch+1, batch_id+1, running_loss / 300))`：打印该epoch的训练状态，包括epoch数、batch数和平均损失。
20. `print('第 %d epoch的 Accuracy on train set: %d %%, Loss on train set: %f' % (epoch + 1, 100 * accuracy / total, train_loss))`：打印该epoch的训练准确率和损失。
21. `return 1.0 * accuracy / total, train_loss`：返回训练准确率和损失。

这段代码的作用是定义了一个用于训练神经网络模型的函数，它会通过遍历训练数据集的所有样本进行模型训练，并输出训练过程中的损失和准确率。

```
defvalidation(epoch):
correct=0
total=0
val_loss=0.0
withtorch.no_grad():
fordataintest_loder:
images,target=data
images,target=images.to(device),target.to(device)
outputs=model(images)
loss=criterion(outputs,target)
val_loss+=loss.item()
_,predicted=torch.max(outputs.data,dim=1)
total+=target.size(0)
correct+=(predicted==target).sum().item()
print('第%depoch的Accuracyonvalidationset:%d%%,Lossonvalidationset:%f'%(epoch+1,100
*correct/total,val_loss))

#返回acc和loss
return1.0*correct/total,val_loss
```

这段代码定义了一个用于验证神经网络模型的函数`validation(epoch)`，下面是对代码的解释：

1. `correct = 0`：用于记录验证集中预测正确的样本数量。
2. `total = 0`：用于记录验证集的总样本数量。
3. `val_loss = 0.0`：记录验证集的累计损失。
4. `with torch.no_grad():`：在验证过程中不计算梯度，用于减少内存消耗和加快计算速度。
5. `for data in test_loader:`：遍历验证集数据，其中`test_loader`是一个数据加载器对象，用于批量加载验证数据。
6. `images, target = data`：将输入数据和目标标签从`data`中分离出来。
7. `images, target = images.to(device), target.to(device)`：将输入数据和目标标签移动到指定的计算设备上。
8. `outputs = model(images)`：将输入数据输入到模型中，得到模型的输出。
9. `loss = criterion(outputs, target)`：计算模型输出与目标标签之间的损失。
10. `val_loss += loss.item()`：累加每个样本的损失。
11. `_, predicted = torch.max(outputs.data, dim=1)`：根据模型的输出结果得到预测的类别。
12. `total += target.size(0)`：更新验证样本的总数。
13. `correct += (predicted == target).sum().item()`：计算预测准确的数量，并累加到`correct`中。
14. `print('第 %d epoch的 Accuracy on validation set: %d %%, Loss on validation set: %f' %(epoch+1,100 *correct / total, val_loss))`：打印该epoch的验证准确率和损失。
15. `return 1.0 * correct / total, val_loss`：返回验证准确率和损失。

这段代码的作用是定义了一个用于验证神经网络模型的函数，它会通过遍历验证数据集的所有样本进行模型验证，并输出验证过程中的损失和准确率。

```
defdraw_in_one(list,epoch):
#x_axix，train_pn_dis这些都是长度相同的list()
#开始画图
x_axix=[xforxinrange(1,epoch+1)]#把ranage转化为list
train_acc=list[0]
train_loss=list[1]
val_acc=list[2]
val_loss=list[3]
#sub_axix=filter(lambdax:x%200==0,x_axix)
plt.title('ResultAnalysis')
plt.plot(x_axix,train_acc,color='green',label='trainingaccuracy')
plt.plot(x_axix,train_loss,color='red',label='trainingloss')
plt.plot(x_axix,val_acc,color='skyblue',label='valaccuracy')
plt.plot(x_axix,val_loss,color='blue',label='valloss')
plt.legend()#显示图例
plt.xlabel('epochtimes')
plt.ylabel('rate')
plt.show()
```

这段代码定义了一个函数`draw_in_one(list, epoch)`，用于绘制训练过程中的准确率和损失曲线。

解释如下：

1. `x_axix = [x for x in range(1, epoch+1)]`：生成一个包含从1到`epoch`的列表，用作横轴坐标。
2. `train_acc = list[0]`：获取传入参数`list`中的训练准确率数据。
3. `train_loss = list[1]`：获取传入参数`list`中的训练损失数据。
4. `val_acc = list[2]`：获取传入参数`list`中的验证准确率数据。
5. `val_loss = list[3]`：获取传入参数`list`中的验证损失数据。
6. `plt.title('Result Analysis')`：设置图表标题。
7. `plt.plot(x_axix, train_acc, color='green', label='training accuracy')`：绘制训练准确率曲线，以绿色表示，添加图例标签。
8. `plt.plot(x_axix, train_loss, color='red', label='training loss')`：绘制训练损失曲线，以红色表示，添加图例标签。
9. `plt.plot(x_axix, val_acc, color='skyblue', label='val accuracy')`：绘制验证准确率曲线，以天蓝色表示，添加图例标签。
10. `plt.plot(x_axix, val_loss, color='blue', label='val loss')`：绘制验证损失曲线，以蓝色表示，添加图例标签。
11. `plt.legend()`：显示图例。
12. `plt.xlabel('epoch times')`：设置横轴标签。
13. `plt.ylabel('rate')`：设置纵轴标签。
14. `plt.show()`：显示绘制的图像。

此代码函数用于将传入的训练集和验证集的准确率与损失数据绘制成一张折线图，用于分析模型的训练进展和性能。

```
if__name__=='__main__':

train_loss=[]
train_acc=[]

val_loss=[]
val_acc=[]
epoches=10
list=[]
forepochinrange(epoches):
acc1,loss1=train(epoch)

train_loss.append(loss1)
train_acc.append(acc1)

acc2,loss2=validation(epoch)

val_loss.append(loss2)
val_acc.append(acc2)
#四幅图合并绘制
list.append(train_acc)
list.append(train_loss)
list.append(val_acc)
list.append(val_loss)
draw_in_one(list,epoches)
```

这段代码的意思是：

1. 首先定义了一些空的列表，分别用于存储训练过程中的损失和准确率。

2. 然后设置了总共的训练轮数`epoches`为10，并初始化了一个空的`list`用于存储这些数据。

3. 在一个循环中，从0到`epoches-1`遍历，每次执行以下步骤：
   - 调用`train(epoch)`函数进行训练，并获取训练准确率和损失值，分别存入`acc1`和`loss1`中。
   - 将训练准确率和损失值分别添加到对应的列表`train_acc`和`train_loss`中。
   - 调用`validation(epoch)`函数进行验证，并获取验证准确率和损失值，分别存入`acc2`和`loss2`中。
   - 将验证准确率和损失值分别添加到对应的列表`val_acc`和`val_loss`中。

4. 最后，将训练集和验证集的准确率和损失值按顺序存入`list`中，然后调用`draw_in_one(list, epoches)`函数将这些数据绘制为一张折线图显示出来。

总的来说，这段代码的目的是执行训练过程和验证过程，并将结果的准确率和损失值存储起来，最后以折线图的形式展示出来，方便分析模型的训练和验证效果。