1. Pytorch basics

References:

- Dive-into-DL-PyTorch 动手学习深入学习pytorch版
- Welcome to PyTorch Tutorials — PyTorch Tutorials 2.6.0+cu124 documentation Pytorch官方教材
- Pytorch学习 | Myblog 网上的博客

The process of training a model :

Data-> Model->Criterion and optimizer->Train the model ->save

1.1 DataSet and Dataloader

在pytorch中，提供了`torch.utils.data`模块来处理数据；

在该模块中，提供了`Dataset`和`Dataloader`两个类：

```
form torch.utils.data import Dataloader, Dataset
```

Fence 1

我们需要准备`tranin_data` and `test_data`， 分别用于训练和测试。

用到的数据集可以直接从一些库里import，并下载，比如：

```
from torchvision import datasets
# Download training data from open datasets.
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)
# Download test data from open datasets.
test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)
```

Fence 2

从hugging face导入数据集：

```
from datasets import load_dataset

ds = load_dataset("refoundd/NailongClassification")
```

Fence 3

也可以先下载到本地再导入：

```
hf_dataset=load_dataset("parquet","path_to_dataset")
```

Fence 4

通常需要自己写数据类，来对下载的数据集进行一些处理：

```
class MyDataset(Dataset):
    //自定义的dataset类继承自torch里面的`Dataset`
    def __init__(self,size=1000):
        self.inputs= # size个数据
        self.labels= # size个标签
    def  __len__(self):
        # return the size of the dataset
        return
    def __getitem__(self,idx):
        #return the data and label.
        return
```

Fence 5

注意，`__len_`and `__getitem__`需要自己重载，分别返回的是数据集的大小，以及如何根据index得到对应的数据。

在数据处理的时候，`dataset`是一个对象实例，成员变量有inputs(or data)和labels等；

`dataloader`是一个迭代器对象，用于按batch遍历dataset。

```
dataloader= Dataloader(dataset,batch_size=32)
# batch_size的含义是train过程中，每一次送入模型一起训练的数据数量大小
```

Fence 6

迭代的过程：

```
for batch in dataloader: #每一个batch是(X,y)
    inputs,labels=batch  #解包unpacking
    #这里的inputs和labels的形状是(batch_size, ...)，即第一维是batch_size,代表有inputs/labels这个张量有batch_size个数据。之后的维数代表了每一个数据的特征
    #例子：形状为（32，10）的labels说明每一个batch是32个数据，每一个数据的形状是（10），即一个特征数为10的一位向量；
    #例子：形状为（32，3，32，32）的inputs说明了每一个batch是32个数据，每一个数据的形状是（3，32，32）；
    #例子：形状为 （32，）的labels说明了每一个batch是32个数据，数据是一个标量！
    # labels和inputs的形状往往不相同。
```

1.2 Neural Network

在 PyTorch 中，自定义神经网络的核心是继承 `torch.nn.Module` 类，并实现以下两个方法：

1. `__init__` **方法**：定义神经网络的结构。
2. `forward` **方法**：定义前向传播的逻辑（即输入如何通过这些层生成输出）。

一个例子：

```
from torch import torch.nn as nn
class SimpleNet(nn.Module):
    def __init__(self):
        super(SimpleNet, self).__init__()
        self.fc1 = nn.Linear(10, 50)  # 输入维度 10，隐藏层维度 50
        self.dropout = nn.Dropout(p=0.5)  # Dropout 层
        self.fc2 = nn.Linear(50, 1)   # 输出维度 1
        //定义了结构
    def forward(self, x):
        x = torch.relu(self.fc1(x))   # 第一层 + ReLU 激活
        x = self.dropout(x)           # Dropout
        x = torch.sigmoid(self.fc2(x))  # 输出概率值（Sigmoid）
        return x
        //将输入得到输出
  model=SimpleNet()
```

Fence 8

1.3 Train & Test the model

在每一次训练的时候，模型根据输入得到prediction，根据prediction计算loss，再反向传播，更新参数

Pytorch的计算图

在每一个batch的数据送入后，pytorch会创建一个计算图，记录每一步的运算步骤，中间结果，以及各个参数。

forward: 记录操作，记录中间结果，参数等

backward：根据记录的内容，计算梯度，以供优化器（optimizer）调整参数，之后计算图会自动被销毁。

Train the model

我们需要定义一个`criterion` function and `optimizer` function：

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
```

Fence 9

其中optimizer的第一个参数是模型的参数，它可以知道每一个参数对loss的影响（也就是每一个参数的grad属性）从而根据这个来动态地调整每一个参数的大小，来使得loss最小.

定义训练：

```
def train(dataloader,model,criterion,optimizer):
    model.train() # 设置为训练模式
    for datas,labels in dataloader:
        pred=model(datas)  #将输入传入模型，会调用它的forward方法
        loss=criterion(pred,labels) #根据prediction和实际的值label进行基表计算得到
loss
        loss.backward()          #backprapopagate反向传播计算梯度
        optimizer.step()         #根据每一个参数的grad值来调整参数的大小
        optimizer.zero_grad() #清空每一个参数的grad值
```

Fence 10

Test the model

评估模型的好坏，需要用数据来测试：

```
def test(test_dataloader,model,criterion):
    model.eval() # 设置为评估模式
    correct=0
    total=0
    test_loss=0.0
    with torch.no_grad():# 在关闭梯度计算的条件下
        for datas,labels in test_dataloader:
            pred=model(datas)  #将输入传入模型，会调用它的forward方法
            loss=criterion(pred,labels) #根据prediction和实际的值label进行基表计算得
到loss
            test_loss=loss.item()
            total+=labels.size(0) #取出label第0维的size，也就是前面说的batch
            correct+=(predicted==labels).sum().item()
            # 这里的==操作符是被pytorch重载过的，得到的是一个布尔张量，一个一维的、特征为
batch_size的张量，即这个张量的内容是batch_size个true or false，分别代表了predicted和
labels里面每一个张量是否相等。
    test_loss/=len(test_dataloader)
    test_accuracy=correct/total
    print(......)
    return test_loss,test_accuracy
```

Fence 11

1.4 Save & load the model

保存状态字典

模型的参数存储在状态字典`state dict`中，模型重要的是参数，因此我们只需要保存模型的参数，并且在使用的时候导入参数即可。

Save the `state dict`:

```
torch.save(trained_model.state_dict(),"path_to_save")
```

Fence 12

Load the model:

```
class MyNet():
    ...

model= MyNet()
model.load_state_dict(torch.load("path"))
#需要先定义一个结构相同的模型，然后将参数导入
```

Fence 13

保存整个模型

也可以保存整个模型：

```
torch.save(trained_model,"path_to_save")
```

Fence 14

在导入的时候：

```
loaded_model = torch.load("path")
#直接导入整个模型
```

Fence 15