

pytorch常见包

xbZhong

2025-01-24

[本页PDF](#)

Pytorch

pytorch与tensorflow的区别：

pytorch支持动态计算图

- 计算图在代码运行时动态生成
- 可根据运行逻辑实时更改计算图

tensorflow支持静态计算图

- 先定义后执行
- 计算图在编译后无法再修改

常用包

- Dataset(数据集)：提供一种方式去获取数据及其label，并形成编号
 - 需要完成的功能：
 - 如何获取每一个数据及其label
 - 告诉我们总共有多少的数据
 - 所有的数据集都需要去继承Dataset类，所有的子类都要重写’**getitem**’方法（获取每个数据及其label）与’**len**’方法（获得数据集长度）
- **getitem(self, index)**：这个方法让类的实例对象可以像列表一样通过索引来访问元素。当你调用 ants_data[3] 时，本质上是调用了 ants_data.__getitem__(3)。在这个方法中，你定义了如何根据索引来获取数据。在你的代码中，**getitem** 方法根据索引获取图像的文件名，并将图像加载到内存中，同时返回图像和标签
- os库中的listdir方法可以让图片名变成列表，path.join方法可以让前后两个路径用'拼接起来
- Image.open(path)表示打开以path为路径的文件，.show表示展示这个图片
- Dataloader(数据装载器)：为后面的网络提供不同的数据形式

入包

```
torch.utils.data import Dataset
```

为图像处理库

```
PIL import Image
```

库是用来操作文件的

```
to os
```

```
Mydata(Dataset):
```

```
def __init__(self, root_dir, label_dir):
```

```
    self.root_dir = root_dir
```

```
    self.label_dir = label_dir
```

```
    self.path = os.path.join(self.root_dir, self.label_dir)
```

```
    self.img_path = os.listdir(self.path)
```

```
_getitem__(self, index):
```

```
    img_name = self.img_path[index]
```

```
    img_item_path = os.path.join(self.root_dir, self.label_dir, img_name)
```

```
    img = Image.open(img_item_path)
```

```
    label = self.label_dir
```

```
    return img, label
```

```
_len__(self):
```

```
    return len(self.img_path)
```

```
dir = 'C:\study\pytorch'
```

```
_dir = 'ants'
```

```
data = Mydata(root_dir, label_dir)
```

```
label = ants_data[5]
```

```
now()
```

Tensorboard

作为pytorch中的一部分，是一个用于可视化深度学习模型训练过程和结果的工具
SummaryWriter类的使用
* 初始化函数
可以输入一个文件夹名称，使得这个文件被tensorboard解析,使用完后要把对象关掉
* writer.add_scalar()
* tag: 图表标题
* scalar_value: 要保存的数值，对应的是y轴
* global_step: 训练了多少步，对应的是x轴

1. 在pycharm中配置conda环境
2. 设置pytorch虚拟环境
3. 编完代码后在终端输入conda activate pytorch进入pytorch虚拟环境
4. 运行代码并将终端目录调整至生成文件的文件夹的上一级

5. 在终端输入tensorboard --logdir=生成文件的文件夹名

6. 打开网址（端口是默认的，也可以自定义）

```
torch.utils.tensorboard import SummaryWriter
```

建 *SummaryWriter* 并指定日志目录

```
r = SummaryWriter('C:/study/pytorch/logs')
```

设在训练过程中记录损失

```
for i in range(100):  
    r.add_scalar('y=2x', i*2, i)
```

闭 *SummaryWriter*

```
r.close()
```

- writer.add_image()
 - tag: 图像标题
 - img_tensor: 数据类型要么是torch.Tensor, numpy.array, 或者string/blobname
 - 使用opencv库去读取图片, 得到的类型是numpy.array类型
 - 也可以利用numpy.array(), 将PIL图片进行转换, 但要在add_image()中指定shape中每一个数字/维表示的含义
 - 在传输路径时要在路径前加上r
 - step: 步数
 - dataformats: 数据类型, 由通道颜色数, 宽度高度组成, img_np类型是HWC, 即高度, 宽度, 颜色通道数(一般为3)

```
torch.utils.tensorboard import SummaryWriter
from PIL import Image
import numpy
```

建 *SummaryWriter* 并指定日志目录

```
r = SummaryWriter('C:/study/pytorch/logs')
path = r'C:\study\pytorch\ants\0013035.jpg'
img = Image.open(img_path)
img_np = numpy.array(img)

for i in range(100):
    writer.add_scalar('y=2x', i**2, i)
    writer.add_image('test', img_np, 1, dataformats='HWC')
```

闭 *SummaryWriter*

```
writer.close()
```

Transforms(torchvision库里面)

- 一个工具箱：里面由许多类(方法)组成
 - 关注输入和输出类型
 - 看官方文档
 - 关注需要传入什么参数
- 使用时要先实例化对象：`tensor = transforms.ToTensor()`,然后利用对象进行类型转换等操作
- opencv的cv2是把图片变为numpy类型 ##### 常见的Transforms
- PIL:Image.open()
- tensor:ToTensor()
 - `torch.tensor()`可以把数据变成tensor类型
- `narrays:cv.imread()` **`__call__`方法可以让实例对象调用方法时像调用函数一样** ##### compose类 把不同的transforms结合在一起

ToTensor类

- 把PIL或numpy数据类型转换成tensor数据类型 ##### ToPILImage类 把图片类型转换为PIL类型 ##### Normalize类
- 必须要是一个tensor数据类型
- 参数：传入为列表
 - 均值：填三个
 - 标准差：填三个
 - 计算公式：(输入-均值)/标准差

Resize类

- 输入为PIL类型,返回值也是PIL类型

- 重新定义图片大小，传入为元组

Compose类

- Compose()中的参数需要是一个列表，且其数据类型需要时transforms类型
- 本质上是对图片操作的方法变成列表放在compose里，减少了代码

RandomCrop

- 给定一个PIL数据类型，进行随机裁剪
- 传入参数可以是序列也可以是一个整数，整数的话会进行裁剪，裁剪为一个正方形

```
from torchvision import transforms
from PIL import Image
from torch.utils.tensorboard import SummaryWriter

totensor

riter = SummaryWriter('logs')
img_path = r'C:\study\pytorch\bees\16838648_415acd9e3f.jpg'
img = Image.open(img_path)
trans_totensor = transforms.ToTensor()
img_tensor = trans_totensor(img)
riter.add_image('text', img_tensor)
```

normalize

三通道数据: *rgb*(红绿蓝), 因此要有三个均值, 三个标准差

```
trans_norm = transforms.Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])
img_norm = trans_norm(img_tensor)
```

resize

```
trans_resize = transforms.Resize((512,512))
img_resize = trans_resize(img)
```

Compose

```
trans_resize_2 = transforms.Normalize(512)
trans_compose = transforms.Compose([trans.resize_2,trans.totensor])
img_resize_2 = trans_compose(img)
```

RandomCrop

```
trans_random = transforms.RandomCrop(512)
trans_compose_2 = transforms.Compose([trans_random.trans_totensor])
for i in range(10):
    img_crop = trans_compose_2(img)
riter.close()
```

torchvision中的数据集使用

- 提供了许多可用的数据集
- Dataset参数:
 - root: 下载的数据集要存放的位置

- train: true为训练数据集, false为测试数据集
- transform: 对数据集进行处理
- target_transform: 对结果进行处理
- download: true则自动为我们下载, false则不为我们下载
import torchvision from torch.utils.tensorboard
import SummaryWriter

PIL类型转变为tensor类型

```
transform = torchvision.transforms.Compose(  
    [torchvision.transforms.ToTensor()])
```

```
_set = torchvision.datasets.CIFAR10(root='./dataset', train=True, transform=data_transform, download=True)  
set = torchvision.datasets.CIFAR10(root='./dataset', train=False, transform=data_transform, download=True)
```

试集的第一个数据

```
(test_set[0])
```

试集里面的类型

```
(test_set.classes)
```

据包含图片和label,且图片为PIL类型

```
label = test_set[0]
```

```
g.show()
```

```
r = SummaryWriter('runs')  
for i in range(10):  
    img, target = test_set[i]  
    writer.add_image('test_set', img, i)
```

Dataloader的使用

- 一个加载器, 把我们的数据加载到神经网络中, 取多少数据。如何取数据都是取决于Dataloader
- 常见参数:
 - dataset: 数据集
 - batch_size: 每次取的数据个数, 然后进行打包
 - shuffle: True表示每提取一次数据便会打乱数据, False表示不会打乱数据
 - num_workers: 使用单线程还是多线程

- drop_last: 当数据集总数除以batch_size有余数时, True表示舍去这剩下的数据, False表示不舍弃剩下的数据
- 代码中的imgs会作为数据传输到神经网络

```

import torchvision
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
test_set=torchvision.datasets.CIFAR10(root='./dataset',train=False,transform=torchvision.transforms.ToTensor())
test_loader = DataLoader(dataset=test_set,batch_size=64,shuffle=True,num_workers=0,drop_last=False)
writer = SummaryWriter('dataloader')
step=0
for data in test_loader:
    imgs,targets = data

# 要使用add_images方法

writer.add_images('data_test', imgs, step)
step+=1
writer.close()

```

神经网络的搭建

- 最常用的模块: Module类
- 需要继承的父类: nn.Module, 需要导入包:from torch import nn
- forward方法:input经过forward变成output ### 代码框架构建步骤

1. 查看官方文档, 了解参数
2. 传入数据集, 导入必要的包
3. 定义类, 利用super()继承nn.Module的属性
4. 在forward方法中定义将input转变为output的方法
5. 必要时需要用reshape进行batch_size, channel的重定义(可能数据类型不满足要求)
6. 在tensorboard中进行图片的可视化

激活函数

在 `torch.nn.Function` 下

$$ReLU(x) = \max(0, x)$$

- Relu

- 输出范围: $[0, +\infty)$
- 零中心分布, 梯度比Sigmoid更强。
- 梯度消失问题仍存在。

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- **Tanh**

- 输出范围： (-1, 1)
- 计算高效，缓解梯度消失（正区间梯度为1）。
- 稀疏激活（负输入直接置0）。

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Sigmoid**

- 输出范围： (0,1)
- 梯度消失（输入绝对值较大时梯度接近0）。
- 输出非零中心（影响梯度更新效率）。

卷积

- conv2d (二维卷积)

- 主要参数：
 - input: 输入
 - 参数： (N,C,H,W) :分别是batch_size:样本数量；通道数：二维张量通道为1；高度和宽度
 - weight: 卷积核，本质上是多维数组
 - bias: 偏置
 - stride: 步长，可以是单个数：横向移动和纵向移动步数相同。也可以是元组：(纵向移动，横向移动)
 - padding: 在输入图像左右两边进行填充，给定一个数或元组(纵向，横向)，空的地方默认为0

- 卷积：数字一一匹配并相乘，然后相加

```
from torch import nn  
import torch  
  
input与output都是tensor类型
```

```
class My(nn.Module):  
    def __init__(self):  
        super().__init__()  
  
    def forward(self, input):  
        output = input + 1  
        return output  
  
y = My()  
= torch.tensor(1.0)
```

也可以是`my(x)`, 因为`nn.Module`重载了`__call__`方法, 使其可以直接调用`forward`方法

```
= my.forward(input = x)  
rint(y)  
import torch  
import torch.nn.functional as F  
nput = torch.tensor([[1,2,0,3,1],  
                     [0,1,2,3,1],  
                     [1,2,1,0,0],  
                     [5,2,3,1,1],  
                     [2,1,0,1,1]])
```

卷积核

```
ernel = torch.tensor([[1,2,1],  
                      [0,1,0],  
                      [2,1,0]])
```

`size`要求四个参数, 因此用`reshape`

```
nput = torch.reshape(input,(1,1,5,5))  
ernal = torch.reshape(kernel,(1,1,3,3))  
utput = F.conv2d(input,kernal,stride=1)  
rint(output)  
utput_2 = F.conv2d(input,kernal,stride=2)  
rint(output_2)
```

```
utput_3 = F.conv2d(input,kernal,stride=1,padding=1)
rint(output_3)
```

卷积层的使用

- conv2d的参数：
 - in_channels:输入通道数
 - out_channels:输出通道数(也是卷积核的个数)
 - kernel_size:卷积核大小，通常为一个数或元组，用元组定义不规则的
 - stride:步长
 - padding:在输入图像左右两边进行填充，给定一个数或元组(纵向，横向)，空的地方默认为0
 - padding_mode:空的地方填什么
 - dilation:
 - groups:设置为1
 - bias:设置为true

```
at torch
import torchvision
from torch import nn
from torch.utils.data import DataLoader
```

里是卷积层，功能较齐全，和前面的*functional*不一样

```
torch.nn import Conv2d
torch.utils.tensorboard import SummaryWriter

et = torchvision.datasets.CIFAR10(root='./dataset', train=False, transform=torchvision.transforms.ToTensor())
oader = DataLoader(dataset, batch_size=64)
```

```
My(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = Conv2d(3, 6, 3, stride=1, padding=0)
    def forward(self, input):
        output = self.conv1(input)
        return output
```

```
def my():
    step = 0
    r = SummaryWriter('data1')
    for data in dataloader:
        imgs, targets = data
        input = my(imgs)
        writer.add_images('input', imgs, step)
        output = torch.reshape(output, (-1, 3, 30, 30))
        writer.add_images('output', output, step)
        step += 1

    r.close()
```

最大池化的使用（最大池化操作）

- Maxpool2d主要参数：
 - kernel_size：池化核(窗口)大小，可传入一个整数或元组
 - stride：步长，默认值是kernel_size大小
 - padding:在输入图像左右两边进行填充，给定一个数或元组(纵向，横向)，空的地方默认为0
 - dilation：空洞卷积，数字匹配时会岔开一定数量格子，格子数量和dilation有关

- ceil_mode: 设置为true会使用ceil模式(向上取整,池化核平移至输入图像边界外会进行数的保留), false使用mode模式(向下取整, 池化核平移至输入图像边界外不会进行数的保留)
- 最大池化操作: 取匹配到的数字中的最大值
- 作用: 保持数据的特征, 减少数据参数, 大小, 例如1080p转到720p, 就是对其进行池化操作

```
import torch,torchvision
from torch import nn
from torch.nn import MaxPool2d
from torch.utils.data import DataLoader

from torch.utils.tensorboard import SummaryWriter
```

记得修改数据类型为32位浮点数

```
input = torch.tensor([[1,2,0,3,1],
                     [0,1,2,3,1],
                     [1,2,1,0,0],
                     [5,2,3,1,1],
                     [2,1,0,1,1]],dtype=torch.float32)
```

输入数据要求是四个参数

```
input = torch.reshape(input,(-1,1,5,5))
```

```
dataset = torchvision.datasets.CIFAR10(root='./dataset',train=False,transform=torchvision.transforms.ToTensor())
dataloader = DataLoader(dataset,batch_size=64)
writer = SummaryWriter('data2')

class My(nn.Module):
    def __init__(self):
        super().__init__()
        self.maxpool = MaxPool2d(3,ceil_mode=True)

    def forward(self,input):
        output = self.maxpool(input)
        return output

y = My()
step = 0
for data in dataloader:
    imgs,targets = data
    output = y(imgs)
    writer.add_images('input',imgs,step)
    writer.add_images('output',output,step)
    step+=1

writer.close()
```

非线性激活

- RELU: input大于0, output等于input, input小于0, output等于0
 - inplace参数: 假设input为-1, 为true的话input会被替换为0, 为false的话input不会被替换, 仍为-1
- Sigmoid: $y = 1/(1+\exp(x))$

```
import torch
import torchvision
from torch import nn
from torch.utils.data import DataLoader
from torch.nn import ReLU,Sigmoid
from torch.utils.tensorboard import SummaryWriter
input = torch.tensor([[1,-0.5],
                     [-1,3]])
input = torch.reshape(input,(-1,1,2,2))
dataset = torchvision.datasets.CIFAR10('./dataset',train=False,transform=torchvision.transforms.ToTensor())
dataloader = DataLoader(dataset,batch_size=64)
class My(nn.Module):
    def __init__(self):
        super().__init__()
        self.relu = ReLU()
        self.sigmoid = Sigmoid()
    def forward(self,input):
        output = self.sigmoid(input)
        return output
writer = SummaryWriter('data3')
my = My()
step = 0
for data in dataloader:
    imgs,targets = data
    writer.add_images('input',imgs,step)
    output = my(imgs)
    writer.add_images('output',output,step)
    step+=1

riter.close()
```

神经网络-线性层及其它层介绍

- 线性层参数:
 - in_feature: 输入数据大小
 - out_feature: 输出数据大小

- bias:偏置
- 输入维度要和前一层输出维度的最后一个维度相等，线性层只对前一层的最后一个维度做变换 ### Sequential(类似 transforms的compose)

```

t torch
torch import nn
torch.utils.tensorboard import SummaryWriter
torch.nn import Conv2d,MaxPool2d,Flatten,Linear,Sequential

My(nn.Module):
  def __init__(self):
    super().__init__()
    # self.conv1 = Conv2d(3,32,5,padding=2)
    # self.maxpool1 = MaxPool2d(2)
    # self.conv2= Conv2d(32,32,5,padding=2)
    # self.maxpool2 = MaxPool2d(2)
    # self.conv3 = Conv2d(32,64,5,padding=2)
    # self.maxpool3 = MaxPool2d(2)
    # self.flatten = Flatten()
    # self.linear1 = Linear(1020,64)
    # self.linear2 = Linear(64,10)
    self.model1 = Sequential(
      Conv2d(3, 32, 5, padding=2),
      MaxPool2d(2),
      Conv2d(32, 32, 5, padding=2),
      MaxPool2d(2),
      Conv2d(32, 64, 5, padding=2),
      MaxPool2d(2),
      Flatten(),
      Linear(1024, 64),
      Linear(64, 10)
    )

    def forward(self,input):
      input = self.model1(input)
      return input

  my()
  = torch.ones((64,3,32,32))
  t = my(input)
  r = SummaryWriter('data4')
  r.add_graph(my,input)
  r.close()

```

损失函数与反向传播

- 损失函数：计算实际输出与目标之间的差距
- 反向传播：为我们更新输出提供一定的依据 #### L1loss()
- 计算各个位置之间的差，将差累加并除以维度
- 参数：
 - input：可以是任意维度
 - output：大小要和输入相同
 - reduction：为sum表示相加，不除以维度

Mseloss()

平方差：先作差再平方 * 参数为input和target

Crossentropyloss

交叉熵： $-x[\text{class}] + \log(\exp(x[j]))$ 求和 * 参数：要求input有(N,C)，target有(N)

```
t torch
torch.nn import L1Loss
s = torch.tensor([1,2,3],dtype=torch.float32)
ts = torch.tensor([1,2,5],dtype=torch.float32)
s = torch.reshape(inputs,(1,1,1,3))
ts = torch.reshape(targets,(1,1,1,3))
= L1Loss()
t = loss(inputs,targets)
(result)
```

优化器

- 所在库：torch.optim
- 要放入模型参数，lr(学习速率)
- 工作流程：
 1. 输入经过模型得到输出
 2. 根据真实的target得到loss
 3. 调用误差的反向传播得到每个参数对应的梯度
 4. 利用优化器进行优化
 5. 对梯度清零 #### VGG(分类模型) 模型参数：
- pretrained：为true说明模型已经训练好
- progress：为true会显示下载进度条

数据集参数：和前面的差不多