实操教程｜Pytorch-lightning的使用

CV开发者都爱看的 极市平台 2021-05-17 22:00:00 手机阅读

作者｜Caliber@知乎（已授权）

来源｜https://zhuanlan.zhihu.com/p/370185203

编辑｜极市平台

极市导读

Pytorch-lightning可以非常简洁得构建深度学习代码。但是其实大部分人用不到很多复杂得功能，并且用的时候稍微有一些不灵活。本文作者分享了自己在使用时的一些心得，附有代码链接。 >>加入极市CV技术交流群，走在计算机视觉的最前沿

Pytorch-lightning(以下简称pl)可以非常简洁得构建深度学习代码。但是其实大部分人用不到很多复杂得功能。而pl有时候包装得过于深了，用的时候稍微有一些不灵活。通常来说，在你的模型搭建好之后，大部分的功能都会被封装在一个叫trainer的类里面。一些比较麻烦但是需要的功能通常如下：

1. 保存checkpoints
2. 输出log信息
3. resume training 即重载训练，我们希望可以接着上一次的epoch继续训练
4. 记录模型训练的过程(通常使用tensorboard)
5. 设置seed，即保证训练过程可以复制

好在这些功能在pl中都已经实现。

由于doc上的很多解释并不是很清楚，而且网上例子也不是特别多。下面分享一点我自己的使用心得。

首先关于设置全局的种子：

```
from pytorch_lightning import seed_everything

# Set seed
seed = 42
seed_everything(seed)
```

只需要import如上的seed_everything函数即可。它应该和如下的函数是等价的：

```
def seed_all(seed_value):
    random.seed(seed_value) # Python
    np.random.seed(seed_value) # cpu vars
    torch.manual_seed(seed_value) # cpu vars

    if torch.cuda.is_available():
        print ('CUDA is available')
        torch.cuda.manual_seed(seed_value)
        torch.cuda.manual_seed_all(seed_value) # gpu vars
        torch.backends.cudnn.deterministic = True  #needed
        torch.backends.cudnn.benchmark = False

seed=42
seed_all(seed)
```

但经过我的测试，好像pl的seed_everything函数应该更全一点。

下面通过一个具体的例子来说明一些使用方法：

先下载、导入必要的包和下载数据集：

```
!pip install pytorch-lightning
!wget https://download.pytorch.org/tutorial/hymenoptera_data.zip
```

```
!unzip -q hymenoptera_data.zip
!rm hymenoptera_data.zip

import pytorch_lightning as pl
import os
import numpy as np
import random
import matplotlib.pyplot as plt

import torch
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
```

以下代码种加入!的代码是在terminal中运行的。在google colab中运行linux命令需要在之前加!

如果是使用google colab，由于它创建的是一个虚拟机，不能及时保存，所以如果需要保存，挂载自己google云盘也是有必要的。使用如下的代码：

```
from google.colab import drive
drive.mount('./content/drive')

import os
os.chdir("/content/drive/My Drive/")
```

先如下定义如下的LightningModule和main函数。

```
class CoolSystem(pl.LightningModule):

    def __init__(self, hparams):
        super(CoolSystem, self).__init__()

        self.params = hparams

        self.data_dir = self.params.data_dir
```

```
        self.num_classes = self.params.num_classes

        ########## define the model ##########
        arch = torchvision.models.resnet18(pretrained=True)
        num_ftrs = arch.fc.in_features

        modules = list(arch.children())[:-1] # ResNet18 has 10 children
        self.backbone = torch.nn.Sequential(*modules) # [bs, 512, 1, 1]
        self.final = torch.nn.Sequential(
               torch.nn.Linear(num_ftrs, 128),
               torch.nn.ReLU(inplace=True),
               torch.nn.Linear(128, self.num_classes),
               torch.nn.Softmax(dim=1))

    def forward(self, x):
        x = self.backbone(x)
        x = x.reshape(x.size(0), -1)
        x = self.final(x)

        return x

    def configure_optimizers(self):
        # REQUIRED
        optimizer = torch.optim.SGD([
                {'params': self.backbone.parameters()},
                {'params': self.final.parameters(), 'lr': 1e-2}
            ], lr=1e-3, momentum=0.9)

        exp_lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, st

        return [optimizer], [exp_lr_scheduler]

    def training_step(self, batch, batch_idx):
        # REQUIRED
        x, y = batch
        y_hat = self.forward(x)

        loss = F.cross_entropy(y_hat, y)

        _, preds = torch.max(y_hat, dim=1)
```

```
        acc = torch.sum(preds == y.data) / (y.shape[0] * 1.0)

        self.log('train_loss', loss)
        self.log('train_acc', acc)

        return {'loss': loss, 'train_acc': acc}


    def validation_step(self, batch, batch_idx):
        # OPTIONAL
        x, y = batch
        y_hat = self.forward(x)
        loss = F.cross_entropy(y_hat, y)
        _, preds = torch.max(y_hat, 1)
        acc = torch.sum(preds == y.data) / (y.shape[0] * 1.0)

        self.log('val_loss', loss)
        self.log('val_acc', acc)

        return {'val_loss': loss, 'val_acc': acc}


    def test_step(self, batch, batch_idx):
        # OPTIONAL
        x, y = batch
        y_hat = self.forward(x)
        loss = F.cross_entropy(y_hat, y)
        _, preds = torch.max(y_hat, 1)
        acc = torch.sum(preds == y.data) / (y.shape[0] * 1.0)

        return {'test_loss': loss, 'test_acc': acc}


    def train_dataloader(self):
        # REQUIRED

        transform = transforms.Compose([
                                transforms.RandomResizedCrop(224),
                                transforms.RandomHorizontalFlip(),
                                transforms.ToTensor(),
```

```
                                transforms.Normalize([0.485, 0.456, 0.40
                                ])

        train_set = torchvision.datasets.ImageFolder(os.path.join(self.d
        train_loader = torch.utils.data.DataLoader(train_set, batch_size

        return train_loader

    def val_dataloader(self):
        transform = transforms.Compose([
                                transforms.Resize(256),
                                transforms.CenterCrop(224),
                                transforms.ToTensor(),
                                transforms.Normalize([0.485, 0.456, 0.40
                                ])

        val_set = torchvision.datasets.ImageFolder(os.path.join(self.data_
        val_loader = torch.utils.data.DataLoader(val_set, batch_size=32, s

        return val_loader

    def test_dataloader(self):
        transform = transforms.Compose([
                                transforms.Resize(256),
                                transforms.CenterCrop(224),
                                transforms.ToTensor(),
                                transforms.Normalize([0.485, 0.456, 0.406]
                                ])

        val_set = torchvision.datasets.ImageFolder(os.path.join(self.data_
        val_loader = torch.utils.data.DataLoader(val_set, batch_size=8, sh

        return val_loader




def main(hparams):
    model = CoolSystem(hparams)
```

```
    trainer = pl.Trainer(
        max_epochs=hparams.epochs,
        gpus=1,
        accelerator='dp'
    )

    trainer.fit(model)

```

下面是run的部分：

```
from argparse import Namespace

args = {
    'num_classes': 2,
    'epochs': 5,
    'data_dir': "/content/hymenoptera_data",
}

hyperparams = Namespace(**args)


if __name__ == '__main__':
    main(hyperparams)
```

如果希望重载训练的话，可以按如下方式：

```
# resume training

RESUME = True

if RESUME:
    resume_checkpoint_dir = './lightning_logs/version_0/checkpoints/'
    checkpoint_path = os.listdir(resume_checkpoint_dir)[0]
    resume_checkpoint_path = resume_checkpoint_dir + checkpoint_path

```

```
    args = {
    'num_classes': 2,
    'data_dir': "/content/hymenoptera_data"}

    hparams = Namespace(**args)

    model = CoolSystem(hparams)


    trainer = pl.Trainer(gpus=1,
                max_epochs=10,
                accelerator='dp',
                resume_from_checkpoint = resume_checkpoint_path)

    trainer.fit(model)
```

如果我们想要从checkpoint加载模型，并进行使用可以按如下操作来：

```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image
def imshow(inp):
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    plt.show()

classes = ['ants', 'bees']

checkpoint_dir = 'lightning_logs/version_1/checkpoints/'
checkpoint_path = checkpoint_dir + os.listdir(checkpoint_dir)[0]

checkpoint = torch.load(checkpoint_path)
```

```
model_infer = CoolSystem(hparams)
model_infer.load_state_dict(checkpoint['state_dict'])

try_dataloader = model_infer.test_dataloader()

inputs, labels = next(iter(try_dataloader))

# print images and ground truth
imshow(torchvision.utils.make_grid(inputs))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range

# inference
outputs = model_infer(inputs)

_, preds = torch.max(outputs, dim=1)
# print (preds)
print (torch.sum(preds == labels.data) / (labels.shape[0] * 1.0))

print('Predicted: ', ' '.join('%5s' % classes[preds[j]] for j in range(8)
```

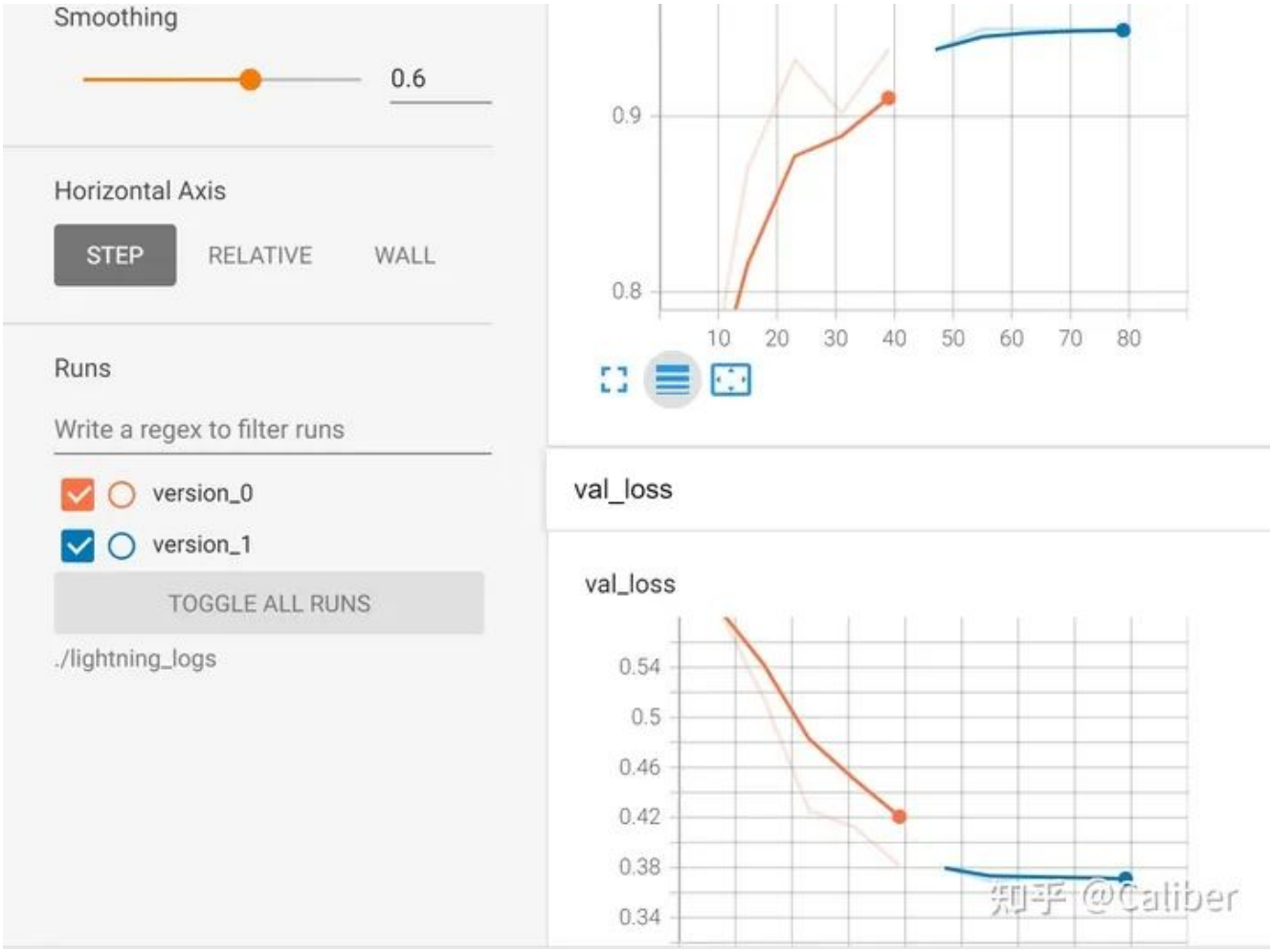

```
GroundTruth:   bees  ants  ants  bees  ants  bees  bees  bees
tensor(1.)
Predicted:   bees  ants  ants  bees  ants  bees  bees  bees
```


预测结果如上。

如果希望检测训练过程（第一部分+重载训练的部分），如下：

```
# tensorboard

%load_ext tensorboard
%tensorboard --logdir = ./lightning_logs
```



训练过程在tensorboard里面记录，version0是第一次的训练，version1是重载后的结果。

完整的code在这里.

https://colab.research.google.com/gist/calibertytz/a9de31175ce15f384dead94c2a9fad4d/pl_tutorials_1.ipynb

如果觉得有用，就请分享到朋友圈吧！

YOLO教程：一文读懂YOLO V5 与 YOLO V4｜大盘点｜YOLO 系目标检测算法总览｜全面解析YOLO V4网络结构

实操教程：PyTorch vs LibTorch：网络推理速度谁更快？｜只用两行代码，我让Transformer推理加速了50倍｜PyTorch AutoGrad C++层实现

算法技巧（trick）：深度学习训练tricks总结（有实验支撑）｜深度强化学习调参Tricks合集｜长尾识别中的Tricks汇总（AAAI2021）

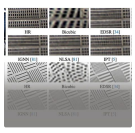
最新CV竞赛：2021 高通人工智能应用创新大赛｜CVPR 2021 | Short-video Face Parsing Challenge｜3D人体目标检测与行为分析竞赛开赛，奖池7万+，数据集达16671张！

阅读原文

喜欢此内容的人还喜欢

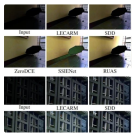
ICCV 2023｜南开程明明团队提出适用于SR任务的新颖注意力机制（已开源）

极市平台



ICCV23｜将隐式神经表征用于低光增强，北大张健团队提出NeRCo

极市平台



ICCV 2023｜Pixel-based MIM: 简单高效的多级特征融合自监督方法

极市平台