第三章 lightningModule的使用

作业布置 & 参考资料

- 作业：
  - 1. 实践作业：改lightningModule代码，在training_step中添加自定义损失函数。配置Trainer使用双GPU训练，并记录训练时间与准确率变化。
  - 2. 思考题：解释ModelCheckpoint回调函数在训练过程中的作用，并说明如何配置其保存最优模型的策略。
- 参考资料：
  - 1. PyTorch Lightning官方文档
  - 2. 教程《用PyTorch+Lightning优化神经网络代码》

背景

- 前面深度学习的代码就是一个很通用的流程，最基本的
- 掌握前面的流程，就已经具备了可以自己设计模型，训练模型的技能
- 当我们看别人的深度学习模型代码的时候，会发现怎么完全对应不上，我知道该怎么训练，可是看不懂别人的代码逻辑
- 因为很多代码用LightningModule包装起来了，底层还是右边代码这一套，但是“语法”不同，我们就需要学习这个LightningModule的语法,或者说是使用规范

```
save_metrics_csv = "./metrics.csv"
with open(save_metrics_csv,"w") as f:
    f.write(f"{','.join(['epoch', 'train_loss', 'valid_loss', 'train_accuracy', 'valid_accuracy'])}\n")
for epoch in range(num_epoch):
    train_loss = []
    train_pred = []
    train_real = []
    valid_loss = []
    valid_pred = []
    valid_real = []
    for X,Y in train_dl:
        X = X.float()
        Y = Y.long()
        optimizer.zero_grad()
        y_pred = net(X)
        loss = criterion(y_pred,Y)
        loss.backward()
        optimizer.step()
        train_loss.append(loss.item())
        train_pred.append(y_pred.argmax(dim = 1).numpy())
        train_real.append(Y.numpy())
    with torch.no_grad():
        # 一个epoch训练完成以后，在验证集上测试一下
        for X,Y in valid_dl:
            X = X.float()
            Y = Y.long()
            y_pred = net(X)
            loss = criterion(y_pred,Y)
            valid_loss.append(loss.item())
            valid_pred.append(y_pred.argmax(-1).numpy())
            valid_real.append(Y.numpy())
    # 根据训练过程保存的数据计算指标，打印并且记录
    get_metrics(train_loss,valid_loss,train_pred,valid_pred,train_real,valid_real)
```

lightningModule是什么

- lightningModule 就是一个将深度学习模型**训练过程** 进行一个**高度抽象**的一个python包。
  - 它是一个类，定义了一些函数，比如 training_step (定义每一个训练step的逻辑), on_train_epoch_end (定义每一个训练epoch结束以后的处理逻辑) on_validation_epoch_end(定义每一个验证epoch结束以后的处理逻辑)。用户通过覆写这每一个阶段的函数，然后lightningModule就能够将这些组合起来，更高效便捷地训练模型；
- 通过使用lightningModule对训练过程进行包装
  - 1. 可以让代码的可读性更高，更加符合规范 (如果理解了lightningModule的设计的话)
  - 2. 更加关注模型训练的逻辑，帮助用户忽略一些麻烦的细节
    - 自动帮我们将数据和模型 迁移到同一个gpu(不需要用户自己写to(“cuda”))；
    - 帮我们进行进程通信，将数据集，模型分配给多张卡进行训练，汇总loss进行梯度更新；
    - 帮我们监控训练流程，保存最优模型
    - 。。。
  - 3. 现在很多大模型，新一点的模型，多是用lightningModule进行包装的，要想看懂他们的代码，要想大改他们的代码，必须学会lightningModule的使用

https://lightning.ai/docs/pytorch/stable/

解析普通代码

一个train_epoch

一个train_step

一个validation_epoch

一个validation_step

```
save_metrics_csv = "./metrics.csv"
with open(save_metrics_csv,"w") as f:
    f.write(f"{','.join(['epoch', 'train_loss', 'valid_loss', 'train_accuracy', 'valid_accuracy'])}\n")
for epoch in range(num_epoch):
    train_loss = []
    train_pred = []
    train_real = []
    valid_loss = []
    valid_pred = []
    valid_real = []
    for X,Y in train_dl:
        X = X.float()
        Y = Y.long()
        optimizer.zero_grad()
        y_pred = net(X)
        loss = criterion(y_pred,Y)
        loss.backward()
        optimizer.step()
        train_loss.append(loss.item())
        train_pred.append(y_pred.argmax(dim = 1).numpy())
        train_real.append(Y.numpy())
    with torch.no_grad():
        # 一个epoch训练完成以后，在验证集上测试一下
        for X,Y in valid_dl:
            X = X.float()
            Y = Y.long()
            y_pred = net(X)
            loss = criterion(y_pred,Y)
            valid_loss.append(loss.item())
            valid_pred.append(y_pred.argmax(-1).numpy())
            valid_real.append(Y.numpy())
    # 根据训练过程保存的数据计算指标，打印并且记录
    get_metrics(train_loss,valid_loss,train_pred,valid_pred,train_real,valid_real)
```

lightningModule的生命周期

红色框框框出来的，就是我们用lightningModule包装模型的时候经常要覆写(定义)的方法

```
def train():
    for epoch in [0..num_epochs]:
        on_train_epoch_start()
        on_before_batch_transfer()
        transfer_batch_to_device()
        on_after_batch_transfer()
        out = training_step()
        on_before_zero_grad()
        optimizer_zero_grad()
        on_before_backward()
        backward()
        on_after_backward()
        on_before_optimizer_step()
        configure_gradient_clipping()
        optimizer_step()
        on_train_batch_end(out, batch, batch_idx)

        if should_check_val:
            on_validation_model_eval()  # calls `model.eval()`
            torch.set_grad_enabled(False)
            on_validation_start()
            on_validation_epoch_start()
            for batch_idx, batch in enumerate(val_dataloader()):
                on_validation_batch_start(batch, batch_idx)
                batch = on_before_batch_transfer(batch)
                batch = transfer_batch_to_device(batch)
                batch = on_after_batch_transfer(batch)
                out = validation_step(batch, batch_idx)
                on_validation_batch_end(out, batch, batch_idx)
            on_validation_epoch_end()
            on_validation_end()
        on_train_epoch_end()
```

```python
save_metrics_csv = "./metrics.csv"
with open(save_metrics_csv,"w") as f:
    f.write(f"{','.join(['epoch', 'train_loss', 'valid_loss', 'train_accuracy', 'valid_accuracy'])}\n")
for epoch in range(num_epoch):
    train_loss = []
    train_pred = []
    train_real = []
    for X,Y in train_dl:
        X = X.float()
        Y = Y.long()
        y_pred = net(X)
        loss = criterion(y_pred,Y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_loss.append(loss.item())
        train_pred.append(y_pred.argmax(dim = 1).numpy())
        train_real.append(Y.numpy())
    with torch.no_grad():
        valid_loss = []
        valid_pred = []
        valid_real = []
        # 一个epoch训练完成以后，在验证集上测试一下
        for X,Y in valid_dl:
            X = X.float()
            Y = Y.long()
            y_pred = net(X)
            loss = criterion(y_pred,Y)
            valid_loss.append(loss.item())
            valid_pred.append(y_pred.argmax(-1).numpy())
            valid_real.append(Y.numpy())
    # 根据训练过程保存的数据计算指标，打印并且记录
    get_metrics(train_loss,valid_loss,train_pred,valid_pred,train_real,valid_real)
```

```python
def train():
    for epoch in [0..num_epochs]:
        on_train_epoch_start()
        on_before_batch_transfer()
        transfer_batch_to_device()
        on_after_batch_transfer()
        out = training_step()
        on_before_zero_grad()
        optimizer_zero_grad()
        on_before_backward()
        backward()
        on_after_backward()
        on_before_optimizer_step()
        configure_gradient_clipping()
        optimizer_step()
        on_train_batch_end(out, batch, batch_idx)

        if should_check_val:
            on_validation_model_eval()  # calls `model.eval()`
            torch.set_grad_enabled(False)
            on_validation_start()
            on_validation_epoch_start()
            for batch_idx, batch in enumerate(val_dataloader()):
                on_validation_batch_start(batch, batch_idx)
                batch = on_before_batch_transfer(batch)
                batch = transfer_batch_to_device(batch)
                batch = on_after_batch_transfer(batch)
                out = validation_step(batch, batch_idx)
                on_validation_batch_end(out, batch, batch_idx)
            on_validation_epoch_end()
            on_validation_end()
        on_train_epoch_end()
```

使用lightningModule将Iris分类器包装起来

```
class Lit_iris_classifier(pl.LightningModule):
    def __init__(self, n_feature,n_hidden,n_output,lr):
        super().__init__()
        self.save_hyperparameters()
        self.lr = lr
        self.model = Iris_classifier(n_feature,n_hidden,n_output)
        self.criterion = torch.nn.CrossEntropyLoss()

    def on_train_epoch_start(self):
        self.train_loss = []
        self.train_pred = []
        self.train_real = []

    def training_step(self, batch, batch_idx):
        X,Y = batch
        X = X.float()
        Y = Y.long()
        y_pred = self.model(X)
        loss = self.criterion(y_pred,Y)
        self.train_loss.append(loss.item())
        self.train_pred.append(y_pred.argmax(dim = 1).cpu().numpy())
        self.train_real.append(Y.cpu().numpy())
        # loss必须返回，不然无法反向传播，梯度更新
        return loss
```

```
    def on_validation_epoch_start(self):
        self.valid_loss = []
        self.valid_pred = []
        self.valid_real = []
    def validation_step(self, batch, batch_idx):
        X,Y = batch
        X = X.float()
        Y = Y.long()
        y_pred = self.model(X)
        loss = self.criterion(y_pred,Y)
        self.valid_loss.append(loss.item())
        self.valid_pred.append(y_pred.argmax(dim = 1).cpu().numpy())
        self.valid_real.append(Y.cpu().numpy())
    def on_train_epoch_end(self):
        train_loss = np.mean(self.train_loss).item()
        valid_loss = np.mean(self.valid_loss).item()
        train_pred =np.concatenate(self.train_pred)
        train_real = np.concatenate(self.train_real)
        valid_pred =np.concatenate(self.valid_pred)
        valid_real = np.concatenate(self.valid_real)
        train_accuracy = get_accuracy(train_pred,train_real)
        valid_accuracy = get_accuracy(valid_pred,valid_real)
        save_metrics =
{'epoch':self.trainer.current_epoch,"step":self.trainer.global_step,'train_loss':train_loss,'valid_loss':valid_loss,'train_accuracy':trai
n_accuracy,'valid_accuracy':valid_accuracy}
        # 通过lightningModel的log函数保存指标
        self.log_dict(save_metrics, batch_size=1)
        # 这里为什么要写batch_size为1呢? 指标都已经算完了，不需要pl推断batch_size有多少然后计算均值，所以直接设置 batch_size为1

    def configure_optimizers(self):
        optimizer = torch.optim.SGD(self.parameters(), lr=self.lr)
        return optimizer
```

```python
class Lit_iris_classifier(pl.LightningModule):
    def __init__(self, n_feature,n_hidden,n_output,lr):
        super().__init__()
        self.save_hyperparameters()
        self.lr = lr
        self.model = Iris_classifier(n_feature,n_hidden,n_output)
        self.criterion = torch.nn.CrossEntropyLoss()

    def on_train_epoch_start(self):...

    def training_step(self, batch, batch_idx):...

    def on_validation_epoch_start(self):...

    def validation_step(self, batch, batch_idx):...

    def on_train_epoch_end(self):...

    def configure_optimizers(self):...
```

```python
def train():
    for epoch in [0..num_epochs]:
        on_train_epoch_start()
        on_before_batch_transfer()
        transfer_batch_to_device()
        on_after_batch_transfer()
        out = training_step()
        on_before_zero_grad()
        optimizer_zero_grad()
        on_before_backward()
        backward()
        on_after_backward()
        on_before_optimizer_step()
        configure_gradient_clipping()
        optimizer_step()
        on_train_batch_end(out, batch, batch_idx)

        if should_check_val:
            on_validation_model_eval()  # calls `model.eval()`
            torch.set_grad_enabled(False)
            on_validation_start()
            on_validation_epoch_start()
            for batch_idx, batch in enumerate(val_dataloader()):
                on_validation_batch_start(batch, batch_idx)
                batch = on_before_batch_transfer(batch)
                batch = transfer_batch_to_device(batch)
                batch = on_after_batch_transfer(batch)
                out = validation_step(batch, batch_idx)
                on_validation_batch_end(out, batch, batch_idx)
            on_validation_epoch_end()
            on_validation_end()
        on_train_epoch_end()
```

其他的没有定义的 函数呢?

比如 反向传播，参数更新，这些通用逻辑，都自动帮我们处理好了，没有其他特殊的处理逻辑，那么就不需要覆写这些函数

如何训练LightningModule

- 将Iris分类器用lightningModule包装好以后，如何对这个包装后的模型进行训练?
  - 训练逻辑已经写在每一个接口中了
  - 我只需要定义一个 训练器 Trainer 对象，将这个包装好的模型传递进去，然后调用 trainer.fit 就能够完成模型的训练

```python
model_save_dir = "./pl_train_output"
n_feature = 4 # 输入X的特征数目为4
n_hidden = 16 # 隐藏层的神经元数目
n_output = 3 # 输出3种类别的logits
max_epoch = 100
batch_size = 8
learning_rate = 0.01
monitor = "valid_accuracy"
check_val_every_n_epoch = 1

model = Lit_iris_classifier(n_feature = n_feature, n_hidden = n_hidden, n_output = n_output,lr = learning_rate)

trainer = pl.Trainer(
        default_root_dir=model_save_dir,
        accelerator="gpu" if torch.cuda.is_available() else 'auto',
        devices = [0],
        max_epochs = max_epoch,
        callbacks=[
            ModelCheckpoint(
                dirpath=model_save_dir,
                save_top_k=1,
                save_last=True,
                monitor=monitor,
                save_on_train_epoch_end=True,
                mode = "max",
                # 必须指定mode 为max或者min，不然不知道该如何保存最优模型，然后就只会保存第一个
                filename="best_model_{epoch:02d}-{valid_accuracy:.2f}"
            )
        ],
        logger=True,
        check_val_every_n_epoch = check_val_every_n_epoch
    )
```

trainer参数解析

default_root_dir: 模型和评估指标保存的位置

accelerator: 用gpu训练

device：指定要用的gpu，如果有多块，会自动帮我们多卡并行

callbacks: 回调函数，通过回调让LightningModule保存最优模型

check_val_every_n_epoch: 每每n个训练epoch，就用验证集评估一次

如何训练LightningModule

```
# 定义训练集 和 验证集 的DataLoader对象
train_ds = Iris_Dataset(df_train)
valid_ds = Iris_Dataset(df_valid)
train_dl = torch.utils.data.DataLoader(train_ds, shuffle = True, batch_size = batch_size,num_workers = 4)
valid_dl = torch.utils.data.DataLoader(valid_ds, shuffle = True, batch_size = batch_size,num_workers = 4)
# 训练模型
trainer.fit(model, train_dataloaders = train_dl, val_dataloaders = valid_dl)
```

trainer.fit，将包装好的lightning model，训练集的dataloader，验证集的dataloader传入即可完成训练

关于训练结果

```
(base) [MiWiFi-RD03-srv zyd ~/workspace/data12T/deeplearning_lab/zyd]$ tree ./pl_train_output/
./pl_train_output/
├── best_model_epoch=20-valid_accuracy=0.97.ckpt
├── last.ckpt
└── lightning_logs
    └── version_0
        ├── hparams.yaml
        └── metrics.csv
```

Delimiter: ,

	epoch	step	train_accuracy	train_loss	valid_accuracy	valid_loss
1	0.0	12	0.3333333432674408	1.1060184240341187	0.3333333432674408	1.0403215885162354
2	1.0	24	0.3222222328186035	1.037888765335083	0.6666666865348816	0.9977619647979736
3	2.0	36	0.4555555582046509	1.0026252269744873	0.6666666865348816	0.9816060662269592
4	3.0	48	0.6777777671813965	0.9844319224357605	0.5666666626930237	0.9428871273994446
5	4.0	60	0.5888888835906982	0.9485476613044739	0.6666666865348816	0.9098227024078369
6	5.0	72	0.6777777671813965	0.9176263809204102	0.6666666865348816	0.9176682233810425
7	6.0	84	0.6666666865348816	0.8783360123634338	0.6666666865348816	0.8482908606529236
8	7.0	96	0.6666666865348816	0.8718663454055786	0.6666666865348816	0.840820848941803
9	8.0	108	0.6666666865348816	0.8285675644874573	0.9333333373069763	0.8008905649185181
10	9.0	120	0.7333333492279053	0.8046047687530518	0.6666666865348816	0.8027387857437134
11	10.0	132	0.7222222089767456	0.7817032933235168	0.6666666865348816	0.7241725325584412
12	11.0	144	0.699999988079071	0.7473838329315186	0.6666666865348816	0.7347923517227173
13	12.0	156	0.699999988079071	0.7194215059280396	0.8999999761581421	0.6867293119430542
14	13.0	168	0.8333333134651184	0.6754886507987976	0.6666666865348816	0.6758612990379333
15	14.0	180	0.699999988079071	0.6617293357849121	0.8999999761581421	0.6226417422294617
16	15.0	192	0.7444444298744202	0.6230769753456116	0.8333333134651184	0.6048460006713867
17	16.0	204	0.8444444537162781	0.5892831683158875	0.6666666865348816	0.5842357277870178

如何载入最优模型进行预测

```
best_model_ckpt = "./pl_train_output/best_model_epoch=20-valid_accuracy=0.97.ckpt"
n_feature = 4 # 输入X的特征数目为4
n_hidden = 16 # 隐藏层的神经元数目
n_output = 3 # 输出3种类别的logits
learning_rate = 0.01
best_pl_model = Lit_iris_classifier.load_from_checkpoint(best_model_ckpt,
                                                         n_feature = n_feature,
                                                         n_hidden = n_hidden,
                                                         n_output = n_output,
                                                         lr = learning_rate)

test_ds = Iris_Dataset(df_test)
test_dl = torch.utils.data.DataLoader(test_ds, shuffle = False, batch_size = 14)
pred = []
real = []
with torch.no_grad():
    for X,Y in test_dl:
        X = X.float()
        Y = Y.long()
        X = X.to("cuda")
        Y = Y.to("cuda")
        y_pred = best_pl_model.model(X).argmax(-1)
        real.append(Y.cpu().numpy())
        pred.append(y_pred.cpu().numpy())
real = np.concatenate(real)
pred = np.concatenate(pred)
print(f"模型训练后,测试集的准确率:{get_accuracy(pred,real)}")
```

模型训练后,测试集的准确率:1.0

总结

- 1. 学会了如何使用lightningModule来让模型训练更加规范方便快捷
- 2. 知道了LightningModule的整个生命周期，学会了如何在LightningModule的框架下，“定制”自己的训练逻辑(就是我该重新定义LightningModule中的哪一些函数，定制自己的训练流程)
- 3. 学会了如何在训练过程保存在验证集上评估效果最好的模型