



PyTorch Lightning (PL)

Rowel Atienza, PhD

University of the Philippines

github.com/roatienza

2022

Decouple research code from engineering.

– <https://www.pytorchlightning.ai>

Supervised Learning using PL

Experience
Task
Performance



PyTorch Lightning removes boiler plate code so we can focus on **ETP**

PyTorch vs PyTorch Lightning

PyTorch

Define train/val/test functions and for loops
Call optimizer/scheduler routines
Define dataset/dataloader class
Move model and data to/from device
Compute, accumulate & display performance metrics & losses

PyTorch Lightning

None

Installation

```
pip install pytorch-lightning
```

```
pip install torchmetrics
```

```
from pytorch_lightning import LightningModule, Trainer  
from pytorch_lightning.loggers import WandbLogger  
from torchmetrics.functional import accuracy
```

PyTorch → PL

Model, Data and Performance Measure Inside a `LightningModule` (LM)

An LM has a Model

```
class LitMNISTModel(LightningModule):
    def __init__(self, num_classes=10, lr=0.001, batch_size=32):
        super().__init__()
        self.save_hyperparameters()
        self.model = torchvision.models.resnet18(num_classes=num_classes)
        self.model.conv1 = torch.nn.Conv2d(1, 64, kernel_size=7,
                                             stride=2, padding=3, bias=False)
        self.loss = torch.nn.CrossEntropyLoss()

    def forward(self, x):
        return self.model(x)
```

```
# this is called during fit()
def training_step(self, batch, batch_idx):
    x, y = batch
    y_hat = self.forward(x)
    loss = self.loss(y_hat, y)
    return {"loss": loss}
```

Train/Test
Functions are **LM**
methods



```
def train_data_loader(self):
    return torch.utils.data.DataLoader(
        torchvision.datasets.MNIST(
            "./data", train=True, download=True,
            transform=torchvision.transforms.ToTensor()
        ),
        batch_size=self.hparams.batch_size,
        shuffle=True,
        num_workers=48,
        pin_memory=True,
    )
```

```
def test_data_loader(self):
    return torch.utils.data.DataLoader(
        torchvision.datasets.MNIST(
            "./data", train=False, download=True,
            transform=torchvision.transforms.ToTensor()
        ),
        batch_size=self.hparams.batch_size,
        shuffle=False,
        num_workers=48,
        pin_memory=True,
    )
```

Dataset/dataloader
all in LM

LightningDataModule

– Advanced data handling

```
class KWSDDataModule(LightningDataModule):
    def __init__(self, path, batch_size=128, num_workers=0, n_fft=512,
                 n_mels=128, win_length=None, hop_length=256, class_dict={},
                 **kwargs):
        super().__init__(**kwargs)
        self.path = path
        self.batch_size = batch_size
        self.num_workers = num_workers
        self.n_fft = n_fft
        self.n_mels = n_mels
        self.win_length = win_length
        self.hop_length = hop_length
        self.class_dict = class_dict
```

Trainer for training/testing

```
# wandb is a great way to debug and visualize this model
wandb_logger = WandbLogger(project="pl-mnist")

trainer = Trainer(accelerator=args.accelerator,
                  devices=args.devices,
                  max_epochs=args.max_epochs,
                  logger=wandb_logger if not args.no_wandb else None,
                  callbacks=[WandbCallback() if not args.no_wandb else None])
trainer.fit(model)
trainer.test(model)

wandb.finish()
```

Automatically use right device. Automatic call to optimizer/scheduler. Easy logging of performance metrics

Code demo is next