A = os.getenv(“INPUT_PATH”, “/data”)
-> A = os.environ[“INPUT_PATH”] if os.environ[“INPUT_PATH”] else “/data”

isinstance(obj, class)
-> type(obj) in [class, parent class, …]

str.strip() -> remove prefix and suffix “ ”
str.strip(“abc”) -> remove all the characters from prefix and suffix

Any(iterable) -> for or (bool)
All(iterable) -> for and (bool)

functools.reduce(lambda a, b: a+b, [1,2,3]) -> 6

operator.lt(a, b) operator.le(a, b) operator.eq(a, b) operator.ne(a, b) operator.ge(a, b) operator.gt(a, b) operator.__lt__(a, b) operator.__le__(a, b) operator.__eq__(a, b) operator.__ne__(a, b) operator.__ge__(a, b) operator.__gt__(a, b)

~~~~~~

torch.zeros_like(A) -> zero matrix that has same shape as A

from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter()
writer.add_scalar("Loss/train", loss, epoch)
! tensorboard --logdir=runs
# more: https://pytorch.org/docs/stable/tensorboard.html

Ansible_vault: a package that can encrypt and decrypt files.

For i in tqdm(range(10)) -> same as no tqdm + progress bar

Pip install logging

Object	Class	角色	Description
Logger	<class ‘logging.Logger’>	大腦	Logger 負責把 log 事件記在腦海中
file handler	<class ‘logging.FileHandler’>	手	負責把 Logger 記在腦海的 log 記錄到 log 日誌文件
~~~~~~

stream handler	<class 'logging.StreamHandler'>	手	負責把 Logger 記在腦海的 log 輸出到螢幕控制台

```
logger          = logging.getLogger("main") -> Initialize a logger
handler         = logging.FileHandler(loggerPath) -> Initialize a handler
consoleHandler = logging.StreamHandler()

log_format = logging.Formatter(fmt, datefmt) -> log information.
logging.setLevel(level)
        Level: logging.DEBUG<INFO<WARNING<ERROR
logger.addHandler(handler) -> sync
logger.addHandler(consoleHandler) -> sync
```

—-----------------
[PT lightning]
—-----------------

+ Model

Class Model(nn.Module): …	Class Model(pl.LightningModule): …

model = Model()
model.load_state_dict(torch.load(PATH))
	model = Model.load_from_checkpoint(PATH)

+ Data

transform = transform.Compose(...)
trainloader = DataLoader(...)
valloader = DataLoader(...)
Testloader = DataLoader(...)
	Class Data(pl.LightningDataModule)
	Def prepare_data(self): … # no duplicate for GPUs
	Def train_dataloader(self): ...
	Def val_dataloader(self): …
	Def test_dataloder(self): …
	# lightning allows DataLoader as well

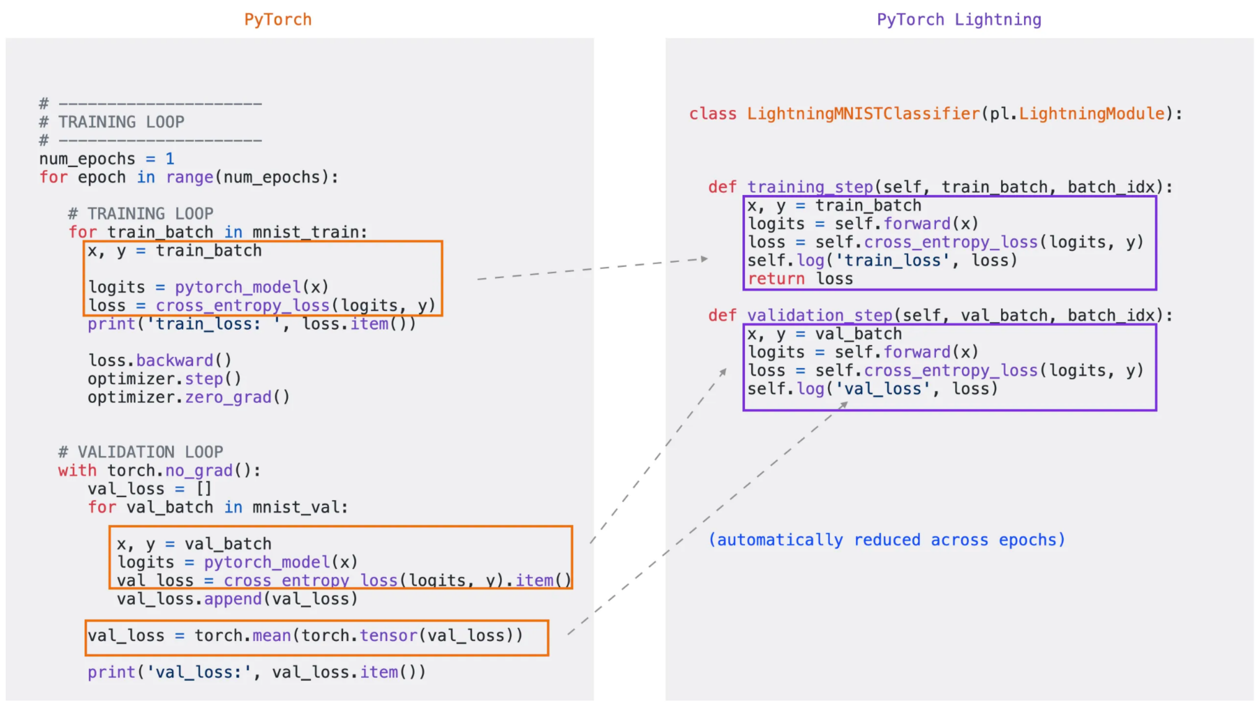
+ Optimizer

optimizer=torch.optim.Adam(model.parameters, lr=1e-3)	# under class Model Def configure_optimizer(self): Return torch.optim.Adam(self.parameters, lr=1e-3) # You can return multiple optimizer as well

+ Loss

lossFunc **=** nn**.**MSELoss()	# under class Model Def lossFunc(self, logits, labels): Return nn.MSELoss(logits, labels)

- Training / validation Loop



```
# ---------------------
# TRAINING LOOP
# ---------------------
num_epochs = 1
for epoch in range(num_epochs):

   # TRAINING LOOP
   for train_batch in mnist_train:
      x, y = train_batch

      logits = pytorch_model(x)
      loss = cross_entropy_loss(logits, y)
      print('train_loss: ', loss.item())

      loss.backward()
      optimizer.step()
      optimizer.zero_grad()


  # VALIDATION LOOP
  with torch.no_grad():
     val_loss = []
     for val_batch in mnist_val:

        x, y = val_batch
        logits = pytorch_model(x)
        val_loss = cross_entropy_loss(logits, y).item()
        val_loss.append(val_loss)

     val_loss = torch.mean(torch.tensor(val_loss))

     print('val_loss:', val_loss.item())
```

```
class LightningMNISTClassifier(pl.LightningModule):


  def training_step(self, train_batch, batch_idx):
      x, y = train_batch
      logits = self.forward(x)
      loss = self.cross_entropy_loss(logits, y)
      self.log('train_loss', loss)
      return loss

  def validation_step(self, val_batch, batch_idx):
      x, y = val_batch
      logits = self.forward(x)
      loss = self.cross_entropy_loss(logits, y)
      self.log('val_loss', loss)



  (automatically reduced across epochs)
```

->
trainer = pl.trainer()
trainer.fit(model, data)

- More on pl.trainer:
  - instantiation:
    - accelerator: cpu, gpu, tpu, auto
    - strategy: ddp
    - devices: list[int]
    - num_nodes: 1
    - devices=1
    - accumulate_grad_batches=1
    - check_val_every_n_epochs=1
    - max_epochs=1000
    - max_steps=-1
    - deterministic=False
    - default_root_dir=os.getcwd()
    - callbacks=None
    - log_every_n_steps=50
  - fit:
    - model
    - train_dataloaders=None
    - val_dataloaders=None
    - datamodule=None
  - validate:

        + model
        + dataloaders=None
        + datamodule=None
    + predict
        + model=None
        + dataloaders=None
        + datamodule=None

+ Callbacks:

    +
      ```
      from lightning.pytorch.callbacks import Callback
      class PrintCallback(Callback):
          def on_train_start(self, trainer, pl_module):
              print("Training is started!")
          def on_train_end(self, trainer, pl_module):
              print("Training is done.")
      trainer(callbacks=PrintCallBack)
      ```

+ More:
    + Customize backward pass
    + Customize optimizer updating

+ Tensorboard usage is same