

Ứng dụng Trí tuệ nhân tạo trong Nuôi trồng thủy sản

NGUYỄN HẢI TRIỀU¹

¹ Bộ môn Kỹ thuật phần mềm,
Khoa Công nghệ thông tin, Trường ĐH Nha Trang

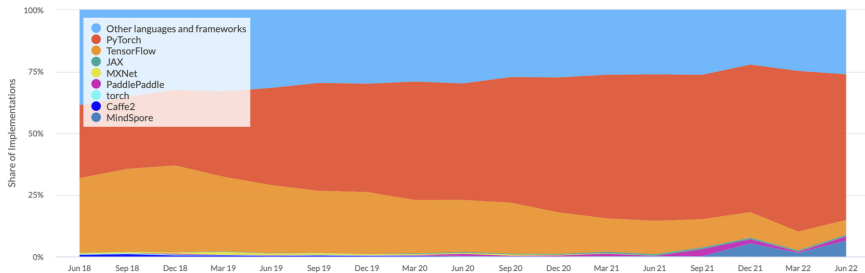
NhaTrang, September 2024

Giới thiệu

PyTorch là Deep Learning (DL) framework mã nguồn mở được sử dụng rất phổ biến hiện nay bởi cộng đồng Researcher do *sự cân bằng giữa khả năng tùy chỉnh và tính thân thiện với người dùng của PyTorch* so với các DL frameworks khác.

Frameworks

Paper Implementations grouped by framework



Tuy nhiên, khi xây dựng mạng Neural network lớn với các tính năng phức tạp hơn như:

- model checkpointing, logging
- data loader
- multi-GPU training
- distributed, parallel computing
- optimize

thì code *PyTorch* thuần thường sẽ dài, lộn xộn → khó quản lý.

Vậy nên, chúng ta sẽ tìm hiểu *PyTorch Lightning*, cũng là một DL framework mã nguồn mở được xem như PyTorch API, giúp chúng ta sắp xếp, quản lý mã nguồn *PyTorch* một cách có hệ thống bằng cách gọi các built-in class.

Mục tiêu

- 1 Giới thiệu tổng quan *PyTorch Lightning*
- 2 So sánh PyTorch Lightning vs PyTorch thuần
- 3 Ví dụ xây dựng một mạng Neural network đơn giản bằng PyTorch Lightning

- 1 Tổng quan PyTorch Lightning
- 2 So sánh huấn luyện mạng MLP sử dụng PyTorch thuần và Lightning
- 3 Một vài tính năng khác của Lightning

Cài đặt môi trường

Pip users

```
1 pip install lightning
```

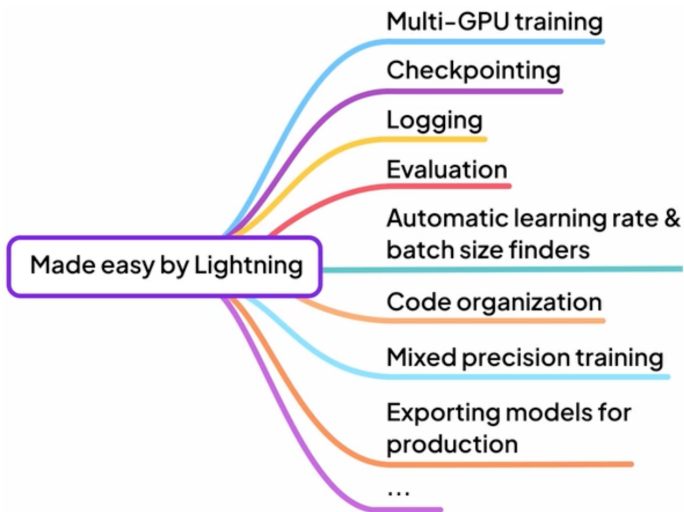
Conda users

```
1 conda install lightning -c conda-forge
```

Tài liệu tham khảo chi tiết của thư viện Lightning

<https://lightning.ai/docs/pytorch/stable/starter/introduction.html>. Để sử dụng ta gọi thư viện:

```
1 import lightning
```



Hình 1: Lightning hỗ trợ

Core API

Trong thư viện lightning, chúng ta có 2 thành phần cần quan tâm là

- LightningModule dùng để xây dựng model (https://lightning.ai/docs/pytorch/stable/common/lightning_module.html)
- Trainer dùng để huấn luyện model (<https://lightning.ai/docs/pytorch/stable/common/trainer.html>)

LightningModule

Nhắc lại cấu trúc một model được xây dựng bằng PyTorch thông thường:

```
1  # Regular PyTorch Module
2  class PyTorchNet(torch.nn.Module):
3      def __init__(self, ...):
4          super().__init__()
5          # Initialize layers ...
6
7      def forward(self, x):
8          # ...
9          return logits
10
```

LightningModule

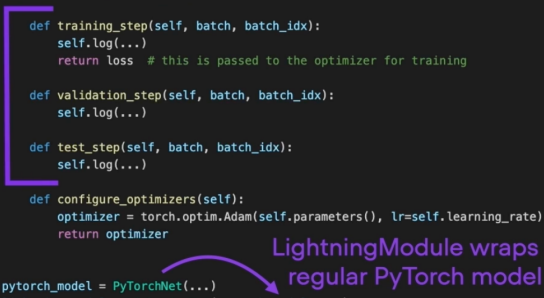
Model được xây dựng bằng pytorch-lightning:

```
1  # LightningModule that receives a PyTorch model as input
2  class LightningNet(LightningModule):
3      def __init__(self, model, ...):
4          super().__init__()
5          self.model = model
6
7      def forward(self, x):
8          return self.model(x)
9
10     def training_step(self, batch, batch_idx):
11         self.log(...)
12         return loss # this is passed to the optimizer for training
13
14     def validation_step(self, batch, batch_idx):
15         self.log(...)
16
17     def test_step(self, batch, batch_idx):
18         self.log(...)
19
20     def configure_optimizers(self):
21         optimizer = torch.optim.Adam(self.parameters(), lr=self.learning_rate)
22         return optimizer
```

LightningModule

Lightning module *nhận đầu vào là Pytorch model thuần và tổ chức các bước training, validation, test... bên trong nó mà không cần viết thủ công từng phương thức như PyTorch thường.*

```
1 # LightningModule that receives a PyTorch model as input
2 class LightningNet(LightningModule):
3     def __init__(self, model, ...):
4         super().__init__()
5         self.model = model
6
7     def forward(self, x):
8         return self.model(x)
9
10    def training_step(self, batch, batch_idx):
11        self.log(...)
12        return loss # this is passed to the optimizer for training
13
14    def validation_step(self, batch, batch_idx):
15        self.log(...)
16
17    def test_step(self, batch, batch_idx):
18        self.log(...)
19
20    def configure_optimizers(self):
21        optimizer = torch.optim.Adam(self.parameters(), lr=self.learning_rate)
22        return optimizer
23
24
25 pytorch_model = PyTorchNet(...)
26 lightning_model = LightningNet(pytorch_model, ...)
```



LightningModule wraps a regular PyTorch model

LightningModule

Ưu điểm

Chúng ta thậm chí không cần gọi *.cuda()* hoặc *.to(device)* khi huấn luyện sử dụng GPU vì Lightning API sẽ làm giúp chúng ta, chỉ cần định nghĩa đúng theo tổ chức code của Lightning. Ví dụ

```
1 # don't do in Lightning
2 x = torch.Tensor(2, 3)
3 x = x.cuda()
4 x = x.to(device)
5
6 # do this instead
7 x = x # leave it alone!
8
9 # or to init a new tensor
10 new_x = torch.Tensor(2, 3)
11 new_x = new_x.to(x)
```

Trainer

Thay vì viết thủ công *training loop trong PyTorch*, ta sử dụng **Trainer API của lightning** rất nhanh chóng và đơn giản

```
1  trainer = Trainer(  
2      max_epochs=NUM_EPOCHS,  
3      accelerator="auto", # Uses GPUs, TPUs etc. if available  
4      devices="auto", # Uses all available GPUs/TPUs if applicable  
5      logger=logger,  
6      deterministic=True,  
7      ...  
8  )  
9  
10 trainer.fit(model=lightning_model, ...)
```

Ta chỉ cần **thiết lập thông số tập trung bên trong khi gọi lớp *Trainer()*** ví dụ như số lượng epochs, huấn luyện trên phần cứng nào, số lượng card đồ họa cần sử dụng, hiển thị log như thế nào?

- 1 Tổng quan PyTorch Lightning
- 2 So sánh huấn luyện mạng MLP sử dụng PyTorch thuần và Lightning
- 3 Một vài tính năng khác của Lightning

PyTorch thuần I

Một vài mẫu code PyTorch thuần huấn luyện MLP trên tập dữ liệu MNIST.

Xây dựng model MLP

```
1 class PyTorchMLP(torch.nn.Module):
2     def __init__(self, num_features, num_classes):
3         super().__init__()
4
5         self.all_layers = torch.nn.Sequential(
6             # 1st hidden layer
7             torch.nn.Linear(num_features, 50),
8             torch.nn.ReLU(),
9             # 2nd hidden layer
10            torch.nn.Linear(50, 25),
11            torch.nn.ReLU(),
12            # output layer
13            torch.nn.Linear(25, num_classes),
```

PyTorch thuần II

```
14         )
15
16     def forward(self, x):
17         x = torch.flatten(x, start_dim=1)
18         logits = self.all_layers(x)
19         return logits
```

Huấn luyện model MLP

```
1 def train(
2     model, optimizer, train_loader, val_loader, num_epochs=10,
3     seed=1, device=None):
4     if device is None:
5         device = torch.device(device)
6
7     torch.manual_seed(seed)
8
9     for epoch in range(num_epochs):
10
11         model = model.train()
```


PyTorch thuần III

```
12     for batch_idx, (features, labels) in enumerate(  
13         train_loader):  
14         features, labels = features.to(device), labels.to(  
15             device)  
16         logits = model(features)  
17  
18         loss = F.cross_entropy(logits, labels)  
19  
20         optimizer.zero_grad()  
21         loss.backward()  
22         optimizer.step()  
23  
24     if not batch_idx % 250:  
25  
26         val_loss = compute_total_loss(model, val_loader,  
27             device=device)  
28         # train_loss = compute_total_loss(model,  
29             train_loader, device=device)
```

PyTorch thuần IV

```
28         # LOGGING
29         print(
30             f"Epoch: {epoch+1:03d}/{num_epochs:03d}"
31             f" | Batch {batch_idx:03d}/{len(train_loader)}"
32             f" | Train Batch Loss: {loss:.4f}"
33             f" | Train Total Loss: {train_loss:.4f}"
34             f" | Val Total Loss: {val_loss:.4f}"
35         )
```

Ước lượng model MLP

```
1 def compute_total_loss(model, dataloader, device=None):
2
3     if device is None:
4         device = torch.device(device)
5
6     model = model.eval()
7     loss = 0.0
8     examples = 0.0
```

PyTorch thuần V

```
9
10     for idx, (features, labels) in enumerate(dataloader):
11
12         features, labels = features.to(device), labels.to(device)
13
14         with torch.no_grad():
15             logits = model(features)
16             batch_loss = F.cross_entropy(logits, labels,
17                                     reduction="sum")
18
19         loss += batch_loss.item()
20         examples += logits.shape[0]
21     return loss / examples
```

Gọi hàm

PyTorch thuần VI

```
1 if __name__ == "__main__":
2
3     print(watermark(packages="torch,lightning", python=True))
4     print("Torch CUDA available?", torch.cuda.is_available())
5     print("Torch MPS Mac ARM available?", torch.backends.mps.is_available())
6     if torch.cuda.is_available():
7         device_gpu = "cuda"
8     elif torch.backends.mps.is_available():
9         device_gpu = "mps"
10    else:
11        device_gpu = "cpu"
12    device = torch.device(device_gpu)
13
14    train_loader, val_loader, test_loader = get_dataset_loaders()
15
16    model = PyTorchMLP(num_features=784, num_classes=10)
17    model.to(device)
18    optimizer = torch.optim.SGD(model.parameters(), lr=0.05)
19
20    train(
```

PyTorch thuần VII

```
21     model ,
22     optimizer ,
23     train_loader=train_loader ,
24     val_loader=val_loader ,
25     num_epochs=10 ,
26     seed=1 ,
27     device=device ,
28 )
```

PyTorch Lightning I

Chúng ta có thể quản lý, tổ chức code gọn gàng hơn bằng PyTorch Lightning như sau:

Xây dựng LightningModule

```
1 class LightningModule(L.LightningModule):
2     def __init__(self, model, learning_rate):
3         super().__init__()
4
5         self.learning_rate = learning_rate
6         self.model = model
7
8     def forward(self, x):
9         return self.model(x)
10
11     def training_step(self, batch, batch_idx):
12         features, true_labels = batch
13         logits = self(features)
```

PyTorch Lightning II

```
14         loss = F.cross_entropy(logits, true_labels)
15         self.log("train_loss", loss)
16         return loss # this is passed to the optimizer for
           training
17
18     def validation_step(self, batch, batch_idx):
19         features, true_labels = batch
20         logits = self(features)
21         loss = F.cross_entropy(logits, true_labels)
22         self.log("val_loss", loss, prog_bar=True)
23
24     def configure_optimizers(self):
25         optimizer = torch.optim.SGD(self.parameters(), lr=self.
           learning_rate)
26         return optimizer
```

Thực thi huấn luyện model

PyTorch Lightning III

```
1 if __name__ == "__main__":
2
3     print(watermark(packages="torch,lightning", python=True))
4
5     train_loader, val_loader, test_loader = get_dataset_loaders()
6
7     pytorch_model = PyTorchMLP(num_features=784, num_classes=10)
8     lightning_model = LightningModel(model=pytorch_model,
9                                     learning_rate=0.05)
9
10    trainer = L.Trainer(
11        max_epochs=10,
12        accelerator="auto", # set to "auto" or "gpu" to use GPUs
13                           if available
14        devices="auto", # Uses all available GPUs if applicable
15    )
16    trainer.fit()
```


PyTorch Lightning IV

```
17     model=lightning_model ,  
18     train_dataloaders=train_loader ,  
19     val_dataloaders=val_loader ,  
20 )  
21  
22 train_acc = compute_accuracy(pytorch_model, train_loader)  
23 val_acc = compute_accuracy(pytorch_model, val_loader)  
24 test_acc = compute_accuracy(pytorch_model, test_loader)
```

- 1 Tổng quan PyTorch Lightning
- 2 So sánh huấn luyện mạng MLP sử dụng PyTorch thuần và Lightning
- 3 Một vài tính năng khác của Lightning

Logger I

Trong quá trình huấn luyện, *lightning module* hỗ trợ tracking log với nhiều định dạng, chúng ta thường dùng *TensorBoardLogger* (mặc định), *CSVLogger*. Việc thiết lập *TensorBoardLogger* trong lightning module hết sức đơn giản. Sau khi cài đặt thư viện mã nguồn mở Tensorboard, trong lớp *LightningModel*, ta bổ sung như code bên dưới

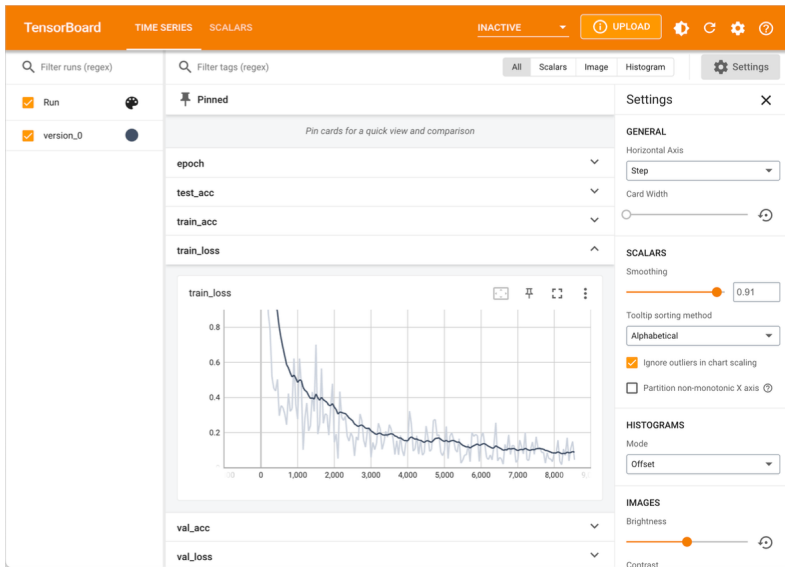
```

1 class LightningModel(L.LightningModule):
2     def __init__(self, model, learning_rate):
3         super().__init__()
4
5         self.learning_rate = learning_rate
6         self.model = model
7         ...
8
9     def training_step(self, batch, batch_idx):
```

Logger II

```
10     ...
11     self.log(
12         "train\_acc", self.train\_acc, prog\_bar=True, on\
13         _epoch=True, on\_step=False
14     )
15     return loss
```

Logger III



Lưu checkpoints

Lightning module **hỗ trợ sử dụng callbacks function để lưu checkpoints** bằng cách định nghĩa callbacks cho checkpoint và bổ sung vào lớp *Trainer()*

```

1 from lightning.pytorch.callbacks import ModelCheckpoint
2 ...
3 callbacks = [
4     ModelCheckpoint(save_top_k=1, mode="max", monitor="val_acc", save_last=True)
5 ]
6 ...
7 trainer = L.Trainer(
8     callbacks=callbacks,
9     ...
10 )

```

Hỗ trợ tìm các Hyperparameters I

Lightning module cũng hỗ trợ tìm tự động giá trị các siêu tham số của model bằng cách sử dụng *Tuner*. Ví dụ bên dưới đi tìm giá trị tối ưu của *learning rate*

```
1 from lightning.pytorch.tuner import Tuner
2
3 trainer = L.Trainer(
4     max_epochs=100,
5     accelerator="cpu",
6     devices="auto",
7     logger=CSVLogger(save_dir="logs/", name="my-model"),
8     deterministic=True,
9 )
10
11 # Create a Tuner
12 tuner = Tuner(trainer)
13
```

Hỗ trợ tìm các Hyperparameters II

```
14 # finds learning rate automatically  
15 # sets hparams.lr or hparams.learning_rate to that learning rate  
16 lr_finder = tuner.lr_find(lightning_model, datamodule=dm)
```


Tài liệu tham khảo



Pytorch-Lightning

Pytorch-Lightning. *https:*

//github.com/Lightning-AI/pytorch-lightning.