



Pytorch Lightning

Quentin Febvre, PhD student





```
n = 30
model 1 = MLP()
model 1.to(device=device)
optimizer = torch.optim.SGD(model 1.parameters(),lr = 0.01)
criterion = nn.CrossEntropyLoss()
train losses, valid losses = [], []
valid loss min = np.Inf
for epoch in range(n epochs):
    train loss, valid loss = \theta, \theta
    model.train()
    for data, label in train loader:
        data = data.to(device=device, dtype=torch.float32)
        label = label.to(device=device, dtvpe=torch.long)
        optimizer.zero grad()
        output = model(data)
        loss = criterion(output, label)
        loss, backward()
        optimizer.step()
        train loss += loss.item() * data.size(0)
    model.eval()
    for data, label in valid loader:
        data = data.to(device=device, dtype=torch.float32)
        label = label.to(device=device, dtype=torch.long)
        with torch.no grad():
            output = model(data)
        loss = criterion(output, label)
        valid loss += loss.item() * data.size(θ)
    train loss /= len(train loader.sampler)
    valid loss /= len(valid loader.sampler)
    train losses.append(train loss)
    valid losses.append(valid loss)
    print('epoch: {} \ttraining Loss: {:.6f} \tvalidation Loss: {:.6f}'.format(
        epoch+1, train loss, valid loss))
    if valid loss <= valid loss min:
        print('validation loss decreased ({:.6f} --> {:.6f}). Saving model ...'.format(
        valid loss min,
        valid loss))
        torch.save(model.state dict(), 'model.pt')
        valid loss min = valid loss
```

10

12

14

15

16

17

18

19

20

21

22

23

25

26

28

29

30

31 32 33

34

35

36

37 38

39

41

42

44

```
"Simple" training routine walkthrough
```

```
optimizer = torch.optim.SGD(model 1.parameters(),lr = 0.01)
    criterion = nn.CrossEntropyLoss()
    train losses, valid losses = [], []
    valid loss min = np.Inf
    for epoch in range(n epochs):
10
11
        train loss, valid loss = \theta, \theta
12
13
        model.train()
14
        for data, label in train loader:
15
            data = data.to(device=device, dtype=torch.float32)
            label = label.to(device=device, dtype=torch.long)
16
17
            optimizer.zero grad()
18
            output = model(data)
19
            loss = criterion(output, label)
20
             loss.backward()
             optimizer.step()
22
            train loss += loss.item() * data.size(0)
23
24
        model.eval()
25
        for data, label in valid loader:
26
            data = data.to(device=device, dtype=torch.float32)
27
            label = label.to(device=device, dtype=torch.long)
            with torch.no grad():
28
                output = model(data)
30
            loss = criterion(output, label)
            valid loss += loss.item() * data.size(0)
33
        train loss /= len(train loader.sampler)
        valid loss /= len(valid loader.sampler)
34
35
        train losses.append(train loss)
        valid losses.append(valid loss)
36
37
38
        print('epoch: {} \ttraining Loss: {:.6f} \tvalidation Loss: {:.6f}'.format(
39
            epoch+1, train loss, valid loss))
40
41
        if valid loss <= valid loss min:
            print('validation loss decreased ({:.6f} --> {:.6f}). Saving model ...'.format(
42
43
            valid loss min,
            valid loss))
            torch.save(model.state dict(), 'model.pt')
45
            valid loss min = valid loss
```

n_epochs = 30 model 1 = MLP()

model 1.to(device=device)

"Simple" learning routine contains many considerations intertwined:

What do I optimize for ?

Training logic:

- How do I compute it?
 - How do I optimize it?

```
optimizer = torch.optim.SGD(model 1.parameters(), lr = 0.01)
    criterion = nn.CrossEntropyLoss()
    train losses, valid losses = [], []
    valid loss min = np.Inf
10
    for epoch in range(n epochs):
        train loss, valid loss = \theta, \theta
        model.train()
        for data, label in train loader:
            data = data.to(device=device, dtype=torch.float32)
            label = label.to(device=device, dtype=torch.long)
            optimizer.zero grad()
            output = model(data)
            loss = criterion(output, label)
            loss.backward()
            optimizer.step()
            train loss += loss.item() * data.size(0)
        model.eval()
24
        for data, label in valid loader:
            data = data.to(device=device, dtype=torch.float32)
            label = label.to(device=device, dtype=torch.long)
            with torch.no grad():
                output = model(data)
            loss = criterion(output, label)
            valid loss += loss.item() * data.size(0)
33
        train loss /= len(train loader.sampler)
34
        valid loss /= len(valid loader.sampler)
        train losses.append(train loss)
        valid losses.append(valid loss)
        print('epoch: {} \ttraining Loss: {:.6f} \tvalidation Loss: {:.6f}'.format(
            epoch+1, train loss, valid loss))
        if valid loss <= valid loss min:
            print('validation loss decreased ({:.6f} --> {:.6f}). Saving model ...'.format(
            valid loss min,
            valid loss))
            torch.save(model.state dict(), 'model.pt')
            valid loss min = valid loss
```

n = 0 epochs = 30 model 1 = MLP()

11

12

13

14

15

16

17

18

19

20

22

23

26

27

28

30

31

35

36 37 38

39

40 41

42 43

45

model 1.to(device=device)

"Simple" learning routine contains many considerations intertwined:

- Training logic: What do I optimize for ?
 - How do I compute it?
 - How do I optimize it?

 - Training flow: Loop for each epoch
 - Iterate over each minibatch
- Autograd/Pytorch flow
 - train/eval mode
 - Backprop + parameter update
 - No grad (disable grad computation)

```
model 1 = MLP()
model 1.to(device=device)
optimizer = torch.optim.SGD(model 1.parameters(),lr = 0.01)
criterion = nn.CrossEntropyLoss()
train losses, valid losses = [], []
valid loss min = np.Inf
for epoch in range(n epochs):
    train loss, valid loss = \theta, \theta
    model.train()
    for data, label in train loader:
        data = data.to(device=device, dtype=torch.float32)
        label = label.to(device=device, dtype=torch.long)
        optimizer.zero grad()
        output = model(data)
        loss = criterion(output, label)
        loss.backward()
        optimizer.step()
       train loss += loss.item() * data.size(0)
    model.eval()
    for data, label in valid loader:
        data = data.to(device=device, dtype=torch.float32)
        label = label.to(device=device, dtype=torch.long)
        with torch.no grad():
            output = model(data)
        loss = criterion(output, label)
        valid loss += loss.item() * data.size(0)
    train loss /= len(train loader.sampler)
    valid loss /= len(valid loader.sampler)
    train losses.append(train loss)
    valid losses.append(valid loss)
   print('epoch: {} \ttraining Loss: {:.6f} \tvalidation Loss: {:.6f}'.format(
        epoch+1, train loss, valid loss))
    if valid loss <= valid loss min:
        print('validation loss decreased ({:.6f} --> {:.6f}). Saving model ...'.format(
        valid loss min,
        valid loss))
        torch.save(model.state dict(), 'model.pt')
        valid loss min = valid loss
```

n = 0 epochs = 30

12

14

15 16

17

18

19

22

23

24

25

26

27

28

31

32

33

34

35

36

38

40

42

45

"Simple" learning routine contains many considerations intertwined:

- Training logic:What do I optimize for ?
- How do I compute it?
- How do I optimize it ?
- Training flow:
 - Loop for each epoch
- Iterate over each minibatch
- Autograd/Pytorch flowtrain/eval mode
 - Backprop + parameter update
 - Backprop + parameter update
 No grad (disable grad computation)
 - Engineering flow / Plugins
 - Training hardware (.to(device))
 - Logging training metrics
 - Saving best model

```
model 1 = MLP()
model 1.to(device=device)
optimizer = torch.optim.SGD(model 1.parameters(), lr = 0.01)
criterion = nn.CrossEntropyLoss()
train losses, valid losses = [], []
valid loss min = np.Inf
for epoch in range(n epochs):
    train loss, valid loss = \theta, \theta
    model.train()
    for data, label in train loader:
        data = data.to(device=device, dtype=torch.float32)
        label = label.to(device=device, dtype=torch.long)
        optimizer.zero grad()
        output = model(data)
        loss = criterion(output, label)
        loss.backward()
        optimizer.step()
        train loss += loss.item() * data.size(0)
    model.eval()
    for data, label in valid loader:
        data = data.to(device=device, dtype=torch.float32)
        label = label.to(device=device, dtype=torch.long)
        with torch.no grad():
            output = model(data)
        loss = criterion(output, label)
        valid loss += loss.item() * data.size(0)
    train loss /= len(train loader.sampler)
    valid loss /= len(valid loader.sampler)
    train losses.append(train loss)
    valid losses.append(valid loss)
   print('epoch: {} \ttraining Loss: {:.6f} \tvalidation Loss: {:.6f}'.format(
        epoch+1, train loss, valid loss))
    if valid loss <= valid loss min:
        print('validation loss decreased ({:.6f} --> {:.6f}). Saving model ...'.format(
        valid loss min.
        valid loss))
        torch.save(model.state dict(), 'model.pt')
        valid loss min = valid loss
```

n = 0 epochs = 30

10

11

12

14

15 16

17

19

22

23

24

25

26

27

28

31

32

34

35

36

38

39

40

42

43

45

"Simple" learning routine contains many considerations intertwined:

Training logic:
What do I optimize for?
How do I compute it?
How do I optimize it?

Training flow:

Autograd/Pytorch flow

- Loop for each epoch
- Iterate over each minibatch
- train/eval mode
- Backprop + parameter update
 No. grad (disable grad computed)
- No_grad (disable grad computation)
- Engineering flow / Plugins
 - Training hardware (.to(device))
 - Logging training/validation metrics
 - Saving best model

Configure

Abstract

```
output = model(data)
                                                                                       37
   loss = criterion(output, label)
                                                                                                def configure optimizers(self):
   loss.backward()
                                                                                                     return torch.optim.SGD(self.parameters(),lr = 0.01)
   optimizer.step()
   train loss += loss.item() * data.size(0)
model.eval()
for data, label in valid loader:
   data = data.to(device=device, dtype=torch.float32)
   label = label.to(device=device, dtvpe=torch.long)
   with torch.no grad():
       output = model(data)
   loss = criterion(output, label)
   valid loss += loss.item() * data.size(0)
train loss /= len(train loader.sampler)
valid loss /= len(valid loader.sampler)
train losses.append(train loss)
                                                                                 Training logic extraction:
valid losses.append(valid loss)
print('epoch: {} \ttraining Loss: {:.6f} \tvalidation Loss: {:.6f}'.format(
                                                                                         Change the nn.Module to
   epoch+1, train loss, valid loss))
                                                                                          pytorch lightning.LightningModule
if valid loss <= valid loss min:
   print('validation loss decreased ({:.6f} --> {:.6f}). Saving model ...'.format(
                                                                                         Compute and return the optimization objective in the
    valid loss min.
   valid loss))
                                                                                          training and validation step method
   torch.save(model.state dict(), 'model.pt')
   valid loss min = valid loss
```

19 >

27

28

29

31

32

33

34

36

class LitMLP(pl.LightningModule):

data, label = batch

output = self(data)

data, label = batch

output = self(data)

def training step(self, batch, batch idx):

return F.cross entropy(output, label)

def validation step(self, batch, batch idx):

return F.cross entropy(output, label)

def init (self): --

def forward(self,x): --

n = 30model 1 = MLP()

10

12

13

14

15

16

17

18

19

20

21

22

23 24

25

26

27 28

29

30

31

32

33

34 35

36 37

38

39

40 41

42

43

45

46

model 1.to(device=device)

valid loss min = np.Inf

model.train()

for epoch in range(n epochs):

criterion = nn.CrossEntropyLoss()

train losses, valid losses = [], []

train loss, valid loss = θ , θ

for data, label in train loader:

optimizer.zero grad()

optimizer = torch.optim.SGD(model 1.parameters(), lr = 0.01)

data = data.to(device=device, dtype=torch.float32)

label = label.to(device=device, dtype=torch.long)

```
32
13
        model.train()
        for data, label in train loader:
                                                                                                  33
14
                                                                                                           def validation step(self, batch, batch idx):
           data = data.to(device=device, dtype=torch.ftsat32)
15
                                                                                                  34
                                                                                                                data, label = batch
16
           label = label.to(device=device, dtype=torch.long)
                                                                                                  35
                                                                                                                output = self(data)
17
           optimizer.zero grad()
                                                                                                  36
                                                                                                                return F.cross entropy(output, label)
           output = model(data)
18
                                                                                                  37
19
            loss = criterion(output, label)
                                                                                                  38
                                                                                                           def configure optimizers(self):
20
            loss.backward()
                                                                                                  39
                                                                                                                return torch.optim.SGD(self.parameters(),lr = 0.01)
21
           optimizer.step()
           train loss += loss.item() * data.size(0)
22
23
24
        model.eval()
25
        for data, label in valid loader:
           data = data.to(device=device, dtype=torch.float32)
26
           label = label.to(device=device, dtype=torch.long)
27
28
           with torch.no grad():
29
               output = model(data)
                                                                                                       model = LitMLP()
30
           loss = criterion(output, label)
                                                                                                       trainer = pl.Trainer(max epochs=1)
31
           valid loss += loss.item() * data.size(0)
                                                                                                        trainer.fit(
32
                                                                                                           model, train dataloaders=train loader, val dataloaders=valid loader)
        train loss /= len(train loader.sampler)
33
34
        valid loss /= len(valid loader.sampler)
35
        train losses.append(train loss)
                                                                                           Training flow abstraction:
        valid losses.append(valid loss)
36
37
        print('epoch: {} \ttraining Loss: {:.6f} \tvalidation Loss: {:.6f}'.format(
38
39
           epoch+1, train loss, valid loss))
                                                                                                    Instantiate the pytorch lightning. Trainer object
40
41
        if valid loss <= valid loss min:
42
           print('validation loss decreased ({:.6f} --> {:.6f}). Saving model ...'.format(
                                                                                                    Pass the Lightning module and the training data to the fit
43
           valid loss min,
                                                                                                    method to perform the optimization procedure
           valid loss))
45
           torch.save(model.state dict(), 'model.pt')
46
           valid loss min = valid loss
```

class LitMLP(pl.LightningModule):

data, label = batch

output = self(data)

def training step(self, batch, batch idx):

return F.cross entropy(output, label)

def init (self): --

def forward(self,x): --

19 >

27

28

29

30

31

n_epochs = 30 model 1 = MLP()

10

11

12

model 1.to(device=device)

valid loss min = np.Inf

for epoch in range(n epochs):

criterion = nn.CrossEntropyLoss()

train losses, valid losses = [], []

train loss, valid loss = θ , θ

optimizer = torch.optim.SGD(model 1.parameters(),lr = 0.01)

```
n = 30
                                                                                                      class LitMLP(pl.LightningModule):
    model 1 = MLP()
                                                                                                         def init (self): --
    model 1.to(device=device)--
                                                                                                 18
    optimizer = torch.optim.SGD(model l.parameters(), lr = 0.01)
                                                                                                          def forward(self,x): --
                                                                                                 19 >
    criterion = nn.CrossEntropyLoss()
                                                                                                 27
                                                                                                          def training step(self, batch, batch idx):
                                                                                                 28
    train losses, valid losses = [], []
                                                                                                  29
                                                                                                              data, label = batch
    valid loss min = np.Inf
                                                                                                  30
                                                                                                              output = self(data)
                                                                                                  31
                                                                                                              loss = F.cross entropy(output, label)
10
    for epoch in range(n epochs):
                                                                                                              self.log('train loss', loss)
                                                                                                  32
        train loss, valid loss = \theta, \theta
11
                                                                                                  33
                                                                                                              return loss
12
                                                                                                  34
13
        model.train()
                                                                                                  35
                                                                                                          def validation step(self, batch, batch idx):
        for data, label in train loader:
14
                                                                                                 36
                                                                                                              data, label = batch
            data = data.to(device=device, dtvpe=torch.float32)
15
                                                                                                 37
                                                                                                              output = self(data)
16
            label = label.to(device=device, dtype=torch.long)
                                                                                                 38
                                                                                                              loss = F.cross entropy(output, label)
17
            optimizer.zero grad()
                                                                                                              self.log('val loss', loss)
                                                                                                  39
            output = model(data)
                                                                                                  40
                                                                                                              return loss
19
            loss = criterion(output, label)
                                                                                                  41
20
            loss.backward()
                                                                                                  42
                                                                                                          def configure optimizers(self):
            optimizer.step()
                                                                                                              return torch.optim.SGD(self.parameters(),lr = 0.01)
                                                                                                  43
22
            train loss += loss.item() * data.size(0)
23
24
        model.eval()
                                                                                                   model = LitMLP()
25
        for data, label in valid loader:
                                                                                                   model checkpoint = pl.callbacks.ModelCheckpoint(monitor='val loss')
            data = data.to(device=device, dtype=torch.float32)
26
                                                                                                   logger = pl.loggers.CSVLogger('logs', name='mlp mnist')
27
            label = label.to(device=device, dtype=torch.long)
                                                                                                   trainer = pl.Trainer(
28
            with torch.no grad():
                                                                                                       max epochs=10,
29
                output = model(data)
                                                                                                     ■ apus=1.
30
            loss = criterion(output, label)
                                                                                                     callbacks=[model checkpoint],
31
            valid loss += loss.item() * data.size(0)
                                                                                                      logger=logger
32
        train loss /= len(train loader.sampler)
33
        valid loss /= len(valid loader.sampler)
34
                                                                                                   trainer.fit(model, train dataloaders=train loader, val dataloaders=valid loader)
35
        train losses.append(train loss)
        valid losses.append(valid loss)
36
37
                                                                                              Engineering/Plugins configuration:
        print('epoch: {} \ttraining Loss: {:.6f} \tvalidation Loss: {:.6f}'.format(
38
39
            epoch+1, train loss, valid loss))
40
41
        if valid loss <= valid loss min:
                                                                                                        Instantiate the logger and model checkpoint callback
42
            print('validation loss decreased ({:.6f} --> {:.6f}). Saving model ...'.format(
43
            valid loss min.
            valid loss))
                                                                                                        Pass as parameter to the trainer, as well as other flags
45
            torch.save(model.state dict(), 'model.pt')
                                                                                                        (gpus...)
46
            valid loss min = valid loss
```

Vanilla Pytorch:



VS

Pytorch Lightning:



- + All the logic is exposed sequentially: easier to write and reason about
- + Fewer classes/concept to handle

- Code harder to read
- Reinvent the wheel each time

- + Code is organized : easier to read/share
- + Less code
- + Benefit from SOTA implementations
- Additional concepts to understand: Trainer, Logger, ModelCheckpointCallback
- Some modifications are less straight-forward (find the method, the parameter...)

Give it a try: notebooks/notebook_PytorchLightning_MNIST_CNN_students.ipynb