Scaling PyTorch Model Training With Minimal Code Changes

@rasbt

sebastian@lightning.ai

https://sebastianraschka.com

Lightning AI

https://lightning.ai

It's 2023, training from scratch often doesn't make sense anymore

Training a vision transformer from scratch

```
Epoch: 0001/0010 | Batch 0000/2812 | Loss: 2.5971
Epoch: 0001/0010 | Batch 0300/2812 | Loss: 1.9497
Epoch: 0001/0010 | Batch 0600/2812 | Loss: 1.6241
...
Epoch: 0001/0010 | Batch 2700/2812 | Loss: 1.5693
Epoch: 0001/0010 | Train acc.: 34.81% | Val acc.: 47.22%
Epoch: 0002/0010 | Batch 0000/2812 | Loss: 1.0045
...
Epoch: 0004/0010 | Batch 2700/2812 | Loss: 1.1540
Epoch: 0004/0010 | Train acc.: 59.33% | Val acc.: 59.14%
...
Epoch: 0005/0010 | Train acc.: 62.04% | Val acc.: 60.06%
...
Epoch: 0006/0010 | Train acc.: 64.22% | Val acc.: 61.80%
...
Epoch: 0007/0010 | Train acc.: 66.12% | Val acc.: 61.14%
Epoch: 0008/0010 | Batch 0000/2812 | Loss: 0.5717
...

Epoch: 0010/0010 | Batch 2700/2812 | Loss: 0.6933
Epoch: 0010/0010 | Train acc.: 71.16% | Val acc.: 62.80%
Time elapsed 61.48 min
Test accuracy 62.85%
```

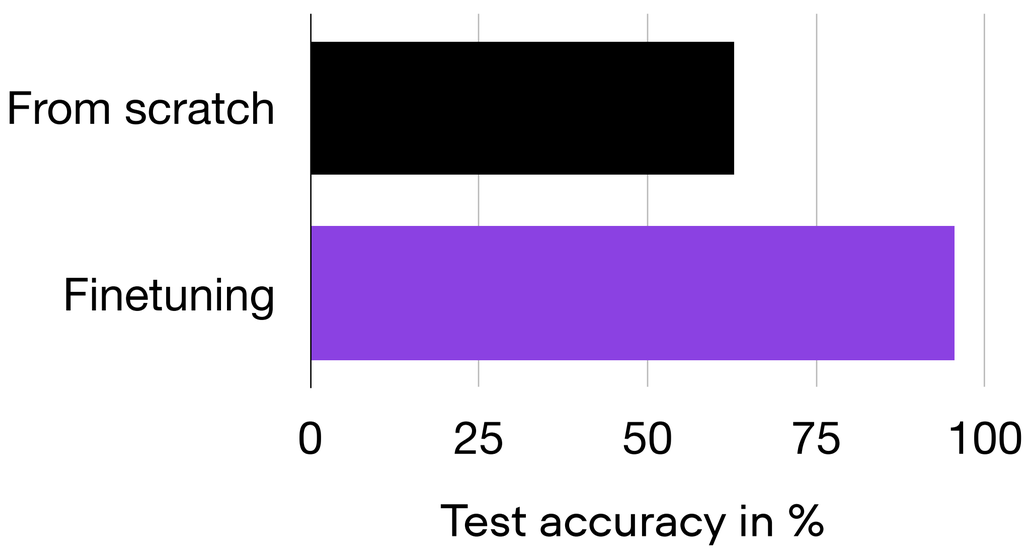
Training a vision transformer from scratch

```
Epoch: 0001/0010 | Batch 0000/2812 | Loss: 2.5971
Epoch: 0001/0010 | Batch 0300/2812 | Loss: 1.9497
Epoch: 0001/0010 | Batch 0600/2812 | Loss: 1.6241
...
Epoch: 0001/0010 | Batch 2700/2812 | Loss: 1.5693
Epoch: 0001/0010 | Train acc.: 34.81% | Val acc.: 47.22%
Epoch: 0002/0010 | Batch 0000/2812 | Loss: 1.0045
...
Epoch: 0004/0010 | Batch 2700/2812 | Loss: 1.1540
Epoch: 0004/0010 | Train acc.: 59.33% | Val acc.: 59.14%
...
Epoch: 0005/0010 | Train acc.: 62.04% | Val acc.: 60.06%
...
Epoch: 0006/0010 | Train acc.: 64.22% | Val acc.: 61.80%
...
Epoch: 0007/0010 | Train acc.: 66.12% | Val acc.: 61.14%
Epoch: 0008/0010 | Batch 0000/2812 | Loss: 0.5717
...

Epoch: 0010/0010 | Batch 2700/2812 | Loss: 0.6933
Epoch: 0010/0010 | Train acc.: 71.16% | Val acc.: 62.80%
Time elapsed 61.48 min
Test accuracy 62.85%
```

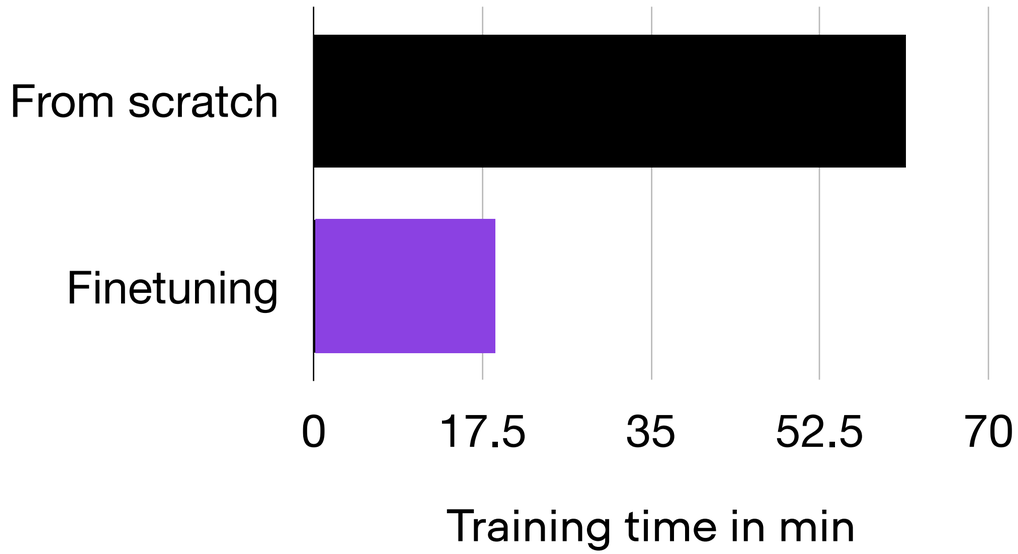
Finetuning a pretrained vision transformer

```
Epoch: 0001/0003 | Batch 0000/2812 | Loss: 2.4934
Epoch: 0001/0003 | Batch 0300/2812 | Loss: 0.0954
Epoch: 0001/0003 | Batch 0600/2812 | Loss: 0.0981
Epoch: 0001/0003 | Batch 0900/2812 | Loss: 0.2078
Epoch: 0001/0003 | Batch 1200/2812 | Loss: 0.3588
Epoch: 0001/0003 | Batch 1500/2812 | Loss: 0.0104
Epoch: 0001/0003 | Batch 1800/2812 | Loss: 0.1560
Epoch: 0001/0003 | Batch 2100/2812 | Loss: 0.0474
Epoch: 0001/0003 | Batch 2400/2812 | Loss: 0.4250
Epoch: 0001/0003 | Batch 2700/2812 | Loss: 0.4414
Epoch: 0001/0003 | Train acc.: 92.40% | Val acc.: 94.12%
Epoch: 0002/0003 | Batch 0000/2812 | Loss: 0.0912
Epoch: 0002/0003 | Batch 0300/2812 | Loss: 0.0337
Epoch: 0002/0003 | Batch 0600/2812 | Loss: 0.1545
Epoch: 0002/0003 | Batch 0900/2812 | Loss: 0.0478
...
Epoch: 0003/0003 | Batch 2100/2812 | Loss: 0.0060
Epoch: 0003/0003 | Batch 2400/2812 | Loss: 0.1395
Epoch: 0003/0003 | Batch 2700/2812 | Loss: 0.1128
Epoch: 0003/0003 | Train acc.: 97.24% | Val acc.: 95.74%
Time elapsed 18.70 min
Test accuracy 95.37%
```

Accuracy (larger is better)



Time to convergence (lower is better)



🎮 https://github.com/rasbt/cvpr2023

How can we accelerate the training even further?

Fabric is the fast and lightweight way to scale PyTorch models without boilerplate code

Suppose we have the following plain PyTorch code

```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset


class PyTorchModel(nn.Module):
    # ...


class PyTorchDataset(Dataset):
    # ...


num_epochs = 10
loss_fn = torch.nn.functional.cross_entropy()

device = "cuda" if torch.cuda.is_available() else "cpu"
model = PyTorchModel(...)
optimizer = torch.optim.SGD(model.parameters())

dataloader = DataLoader(PyTorchDataset(...), ...)

model.train()

for epoch in range(num_epochs):
    for batch in dataloader:
        input, target = input.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(input)
        loss = loss_fn(output, target)
        loss.backward()
        optimizer.step()


```

Fabric* brings advanced functionality without restructuring

Fabric* brings advanced functionality without restructuring

* 100% free an open source

PyTorch

```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset


class PyTorchModel(nn.Module):
    # ...

class PyTorchDataset(Dataset):
    # ...

loss_fn = torch.nn.functional.cross_entropy()
device = "cuda" if torch.cuda.is_available() else "cpu"
model = PyTorchModel(...)
optimizer = torch.optim.SGD(model.parameters())

dataloader = DataLoader(PyTorchDataset(...), ...)

model.train()

for epoch in range(num_epochs):
    for batch in dataloader:
        input, target = input.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(input)
        loss = loss_fn(output, target)
        loss.backward()
        optimizer.step()
```

PyTorch + Fabric

```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset

from lightning.fabric import Fabric

class PyTorchModel(nn.Module):
    # ...

class PyTorchDataset(Dataset):
    # ...

fabric = Fabric(accelerator="cuda", devices=8, strategy="ddp")
fabric.launch()

loss_fn = torch.nn.functional.cross_entropy()

model = PyTorchModel(...)
optimizer = torch.optim.SGD(model.parameters())
model, optimizer = fabric.setup(model, optimizer)
dataloader = DataLoader(PyTorchDataset(...), ...)
dataloader = fabric.setup_dataloaders(dataloader)
model.train()

for epoch in range(num_epochs):
    for batch in dataloader:
        input, target = batch
        optimizer.zero_grad()
        output = model(input)
        loss = loss_fn(output, target)
        fabric.backward(loss)
        optimizer.step()
```

Convert PyTorch to Fabric Step by Step

```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset

from lightning.fabric import Fabric

class PyTorchModel(nn.Module):
    # ...

class PyTorchDataset(Dataset):
    # ...

fabric = Fabric(accelerator="cuda", devices=8, strategy="ddp")
fabric.launch()

loss_fn = torch.nn.functional.cross_entropy()
model = PyTorchModel(...)
optimizer = torch.optim.SGD(model.parameters())
dataloader = DataLoader(PyTorchDataset(...), ...)

model, optimizer = fabric.setup(model, optimizer)
dataloader = fabric.setup_dataloaders(dataloader)

model.train()

for epoch in range(num_epochs):
    for batch in dataloader:
        input, target = batch
        optimizer.zero_grad()
        output = model(input)
        loss = loss_fn(output, target)
        fabric.backward(loss)
        optimizer.step()

```

```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset

from lightning.fabric import Fabric

class PyTorchModel(nn.Module):
    # ...

class PyTorchDataset(Dataset):
    # ...

fabric = Fabric(accelerator="cuda", devices=8, strategy="ddp")
fabric.launch()

loss_fn = torch.nn.functional.cross_entropy()
model = PyTorchModel(...)
optimizer = torch.optim.SGD(model.parameters())
dataloader = DataLoader(PyTorchDataset(...), ...)

model, optimizer = fabric.setup(model, optimizer)
dataloader = fabric.setup_dataloaders(dataloader)

model.train()

for epoch in range(num_epochs):
    for batch in dataloader:
        input, target = batch
        optimizer.zero_grad()
        output = model(input)
        loss = loss_fn(output, target)
        fabric.backward(loss)
        optimizer.step()

```

Import, instantiate, and launch Fabric

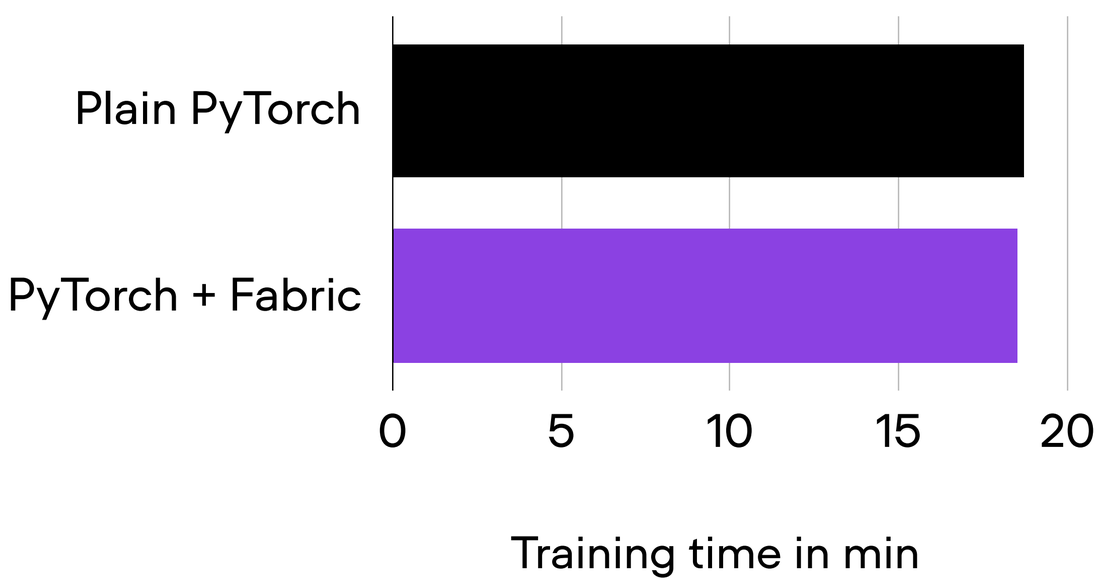
```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset

from lightning.fabric import Fabric

class PyTorchModel(nn.Module):
    # ...

class PyTorchDataset(Dataset):
    # ...

fabric = Fabric(accelerator="cuda", devices=8, strategy="ddp")
fabric.launch()

loss_fn = torch.nn.functional.cross_entropy()
model = PyTorchModel(...)
optimizer = torch.optim.SGD(model.parameters())
dataloader = DataLoader(PyTorchDataset(...), ...)

model, optimizer = fabric.setup(model, optimizer)
dataloader = fabric.setup_dataloaders(dataloader)

model.train()

for epoch in range(num_epochs):
    for batch in dataloader:
        input, target = batch
        optimizer.zero_grad()
        output = model(input)
        loss = loss_fn(output, target)
        fabric.backward(loss)
        optimizer.step()

```

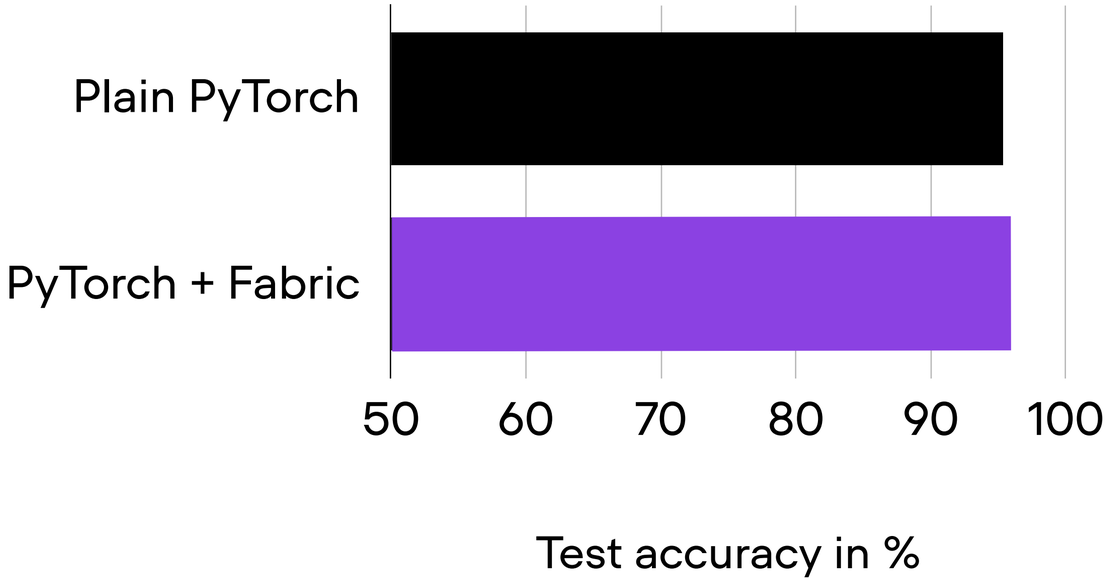
Set up model, optimizer, and data loader

```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset

from lightning.fabric import Fabric

class PyTorchModel(nn.Module):
    # ...

class PyTorchDataset(Dataset):
    # ...

fabric = Fabric(accelerator="cuda", devices=8, strategy="ddp")
fabric.launch()

loss_fn = torch.nn.functional.cross_entropy()
model = PyTorchModel(...)
optimizer = torch.optim.SGD(model.parameters())
dataloader = DataLoader(PyTorchDataset(...), ...)

model, optimizer = fabric.setup(model, optimizer)
dataloader = fabric.setup_dataloaders(dataloader)

model.train()

for epoch in range(num_epochs):
    for batch in dataloader:
        input, target = batch
        optimizer.zero_grad()
        output = model(input)
        loss = loss_fn(output, target)
        fabric.backward(loss)
        optimizer.step()

```

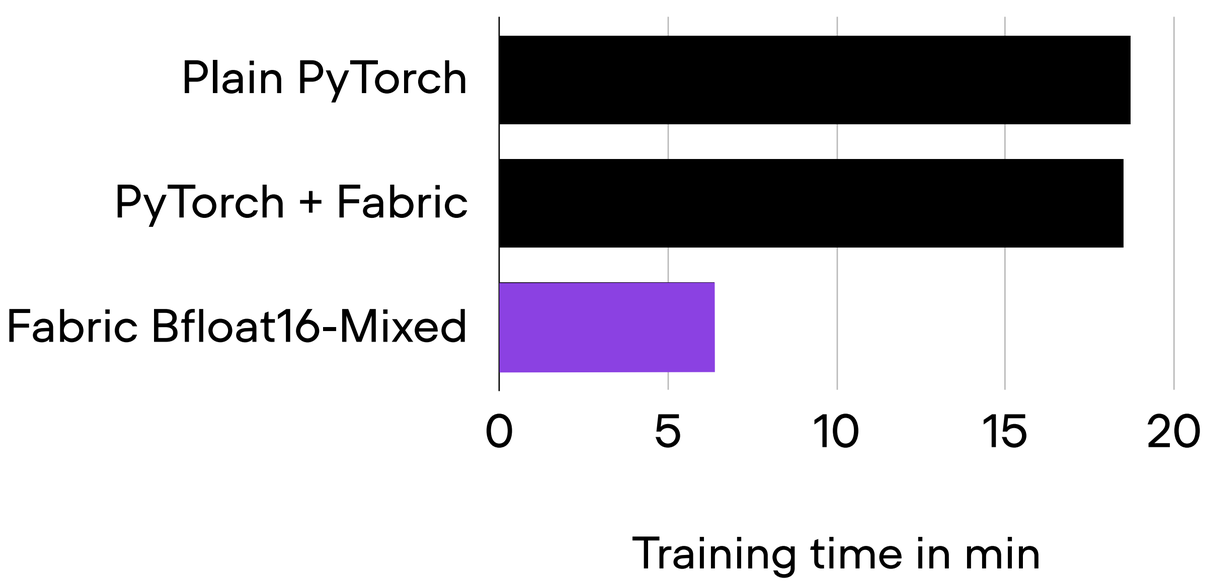
Use fabric.backward(loss)

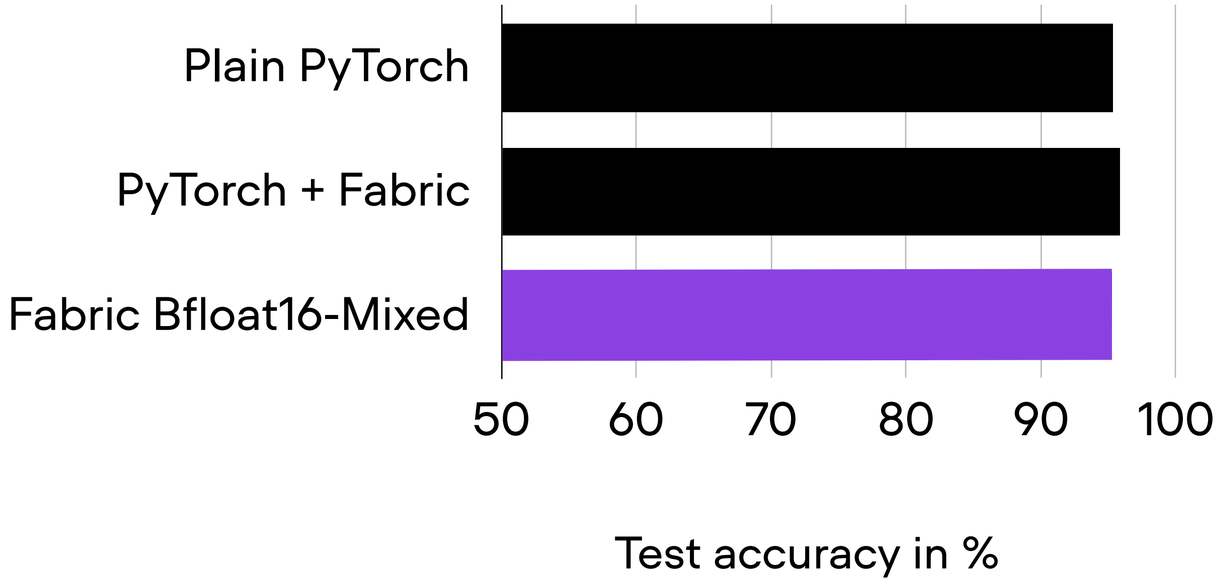
A Performance Baseline





Using Mixed-Precision Training

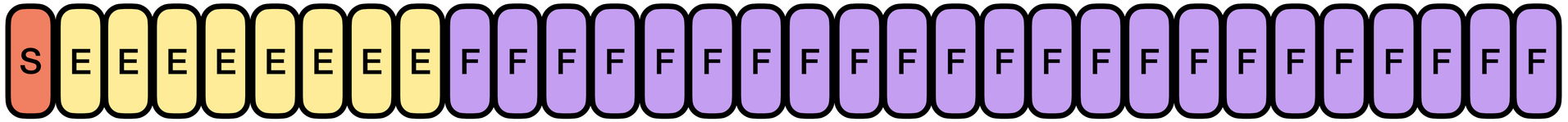




Required Change: 1/2 Lines of Code

```
fabric = Fabric(accelerator="cuda", devices=1, precision="bf16-mixed")
```

Float 32

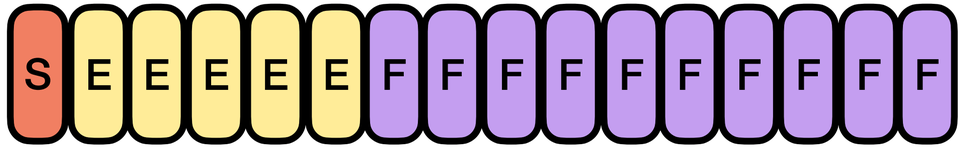


Sign
(1 bit)

Exponent
(8 bits)

Fraction
(23 bits)

Float 16 ("half" precision)

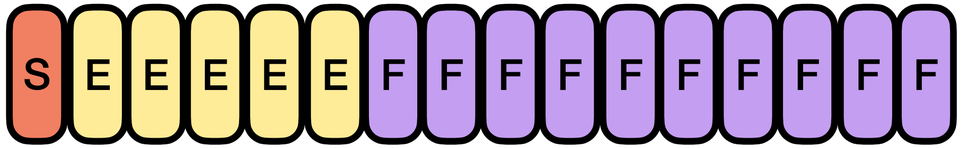


Sign
(1 bit)

Exponent
(5 bits)

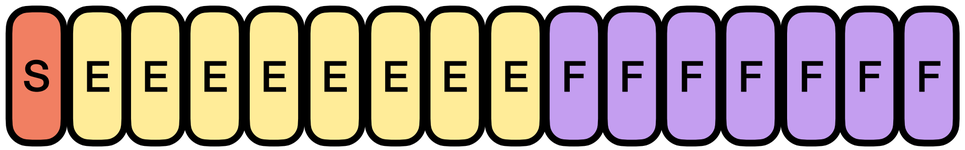
Fraction
(10 bits)

Float 16 ("half" precision)



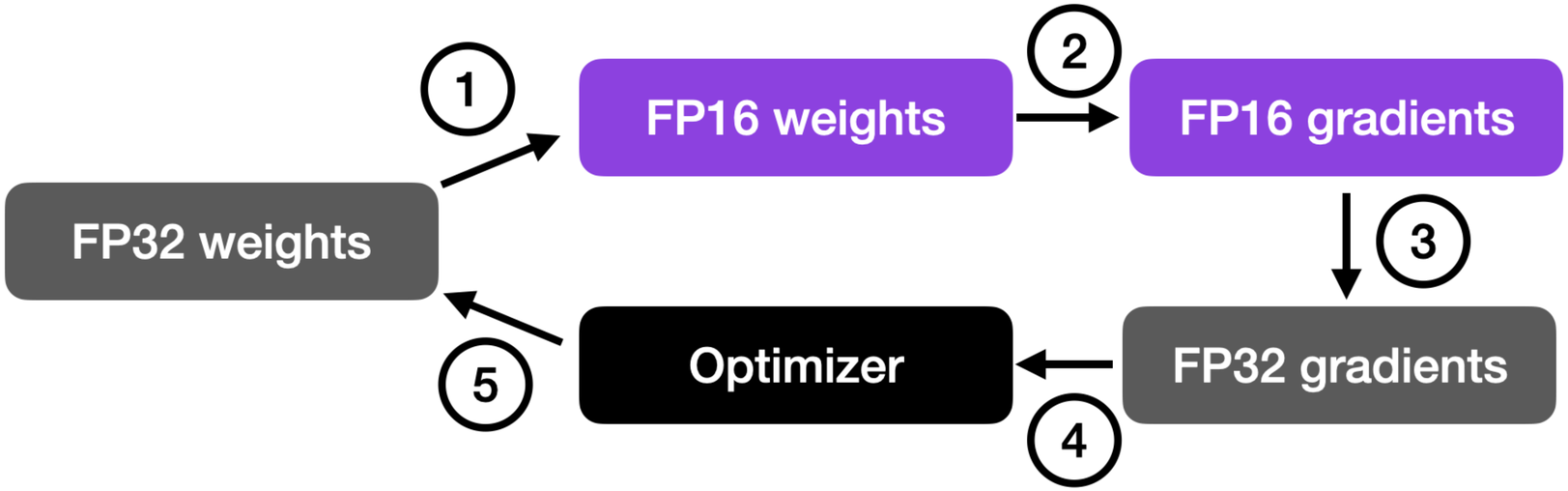
Sign (1 bit) | Exponent (5 bits) | Fraction (10 bits)

Bfloat 16 ("brain" floating point, more "dynamic range" like float 32)



Sign (1 bit) | Exponent (8 bits) | Fraction (7 bits)

Using Mixed-Precision Training



Multi-GPU Training with
Fully Sharded Data Parallelism

Broad Categories of Parallelism for Multi-GPU Training

Model parallelism

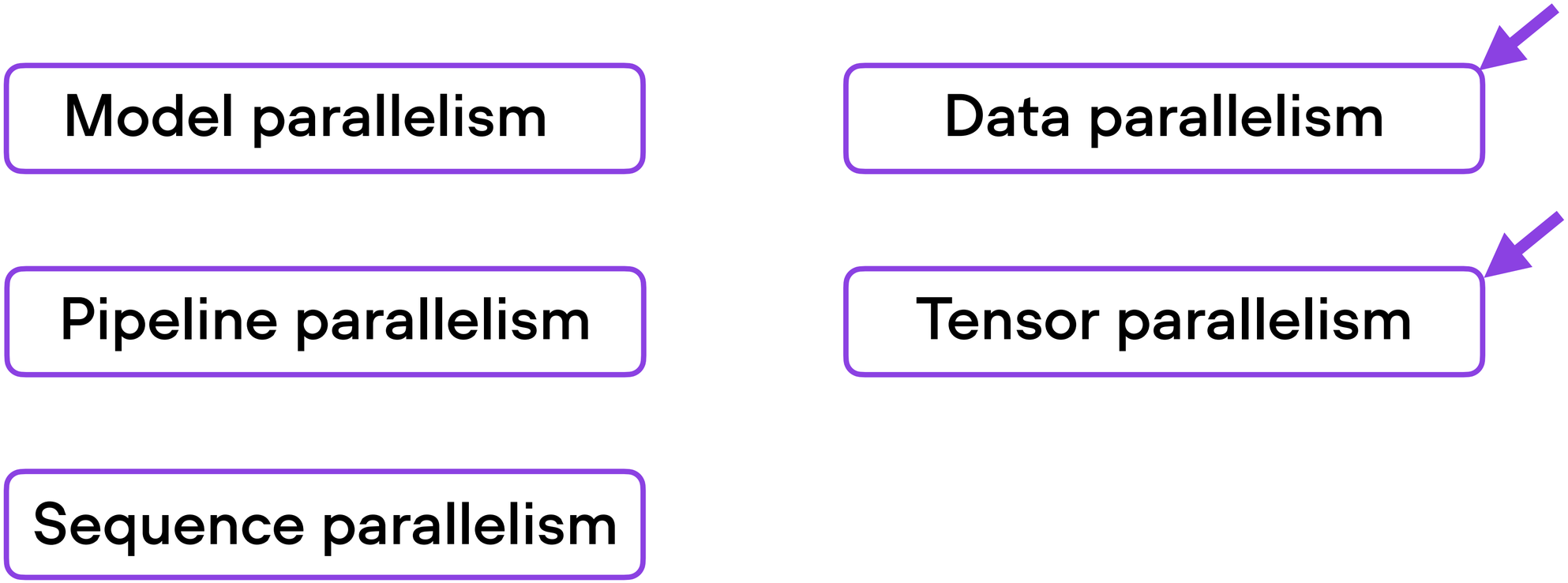
Data parallelism

Pipeline parallelism

Tensor parallelism

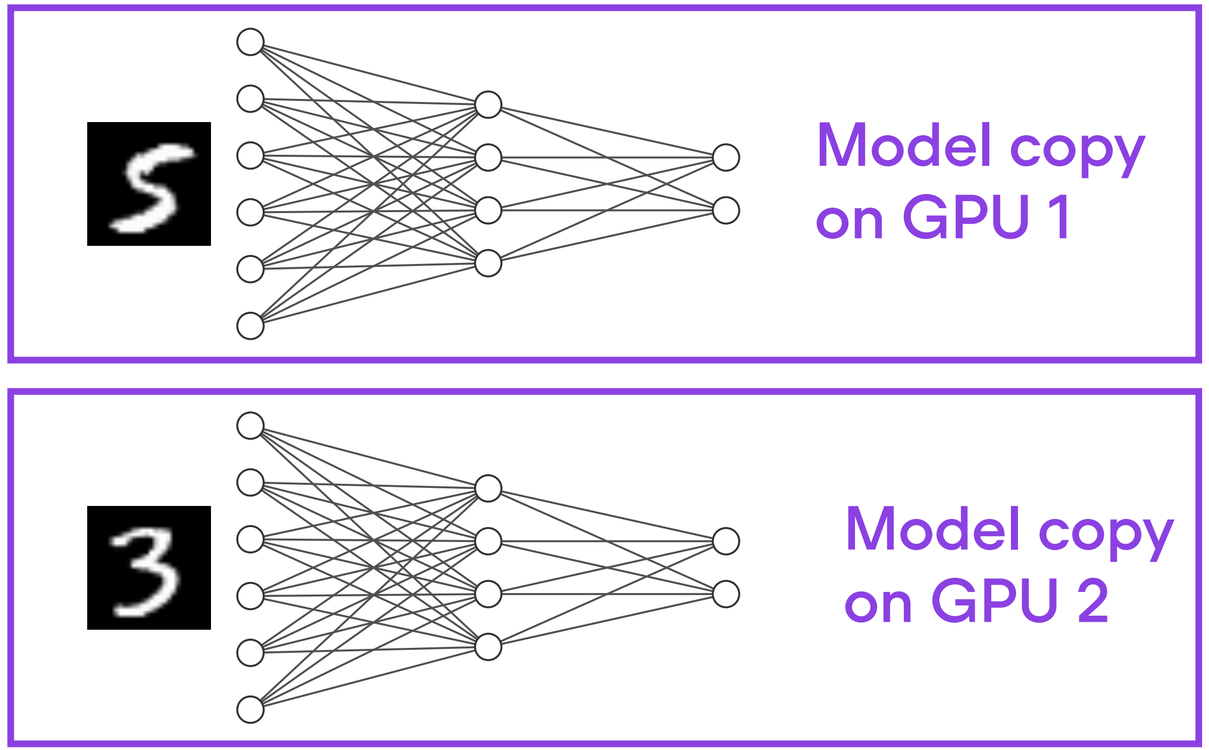
Sequence parallelism

Broad Categories of Parallelism for Multi-GPU Training



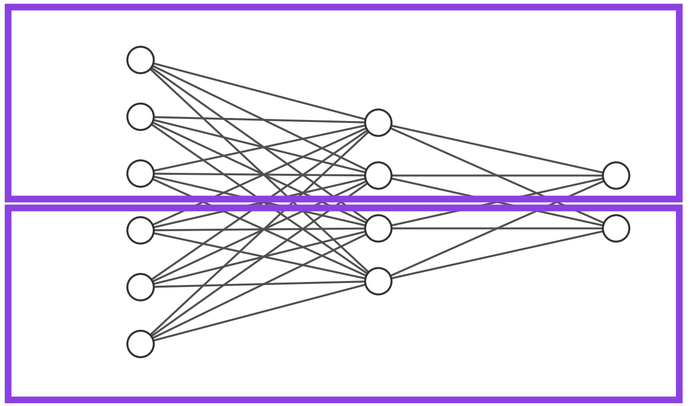
Data Parallelism

Split batch to train model(s) on more data in parallel



Tensor Parallelism

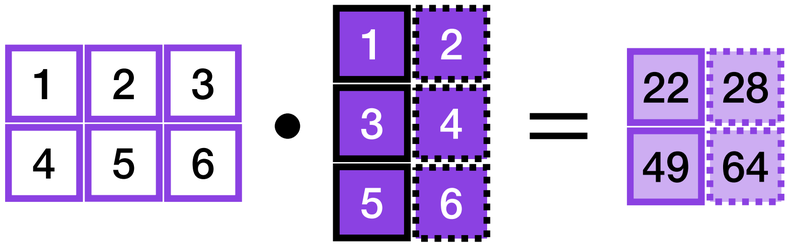
Related to model parallelism, but split horizontally instead of vertically



(Depending on the implementation, you may split weight matrices, optimizer states, gradients)

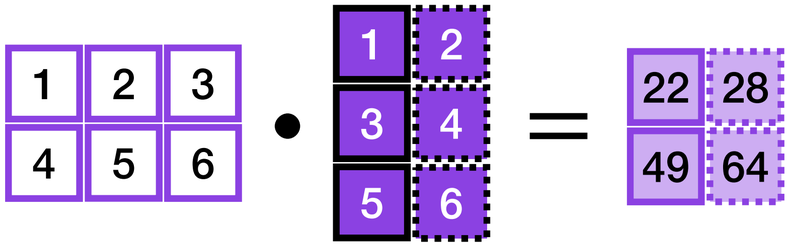
Tensor Parallelism

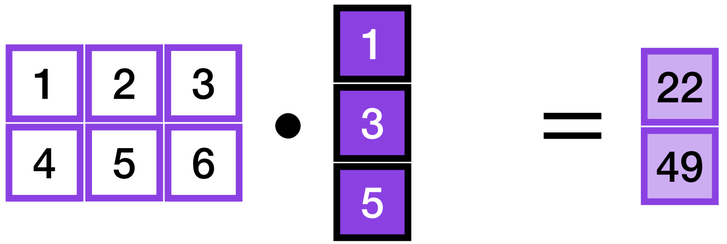
For example, split the matrix multiplication by column

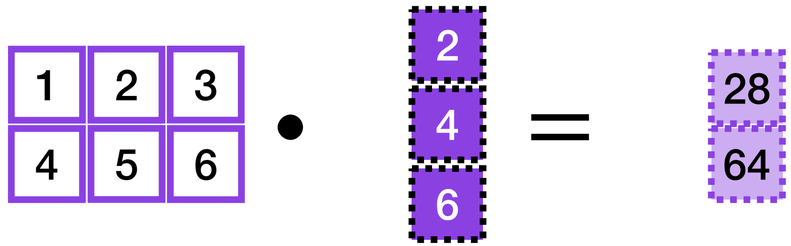


Tensor Parallelism

For example, split the matrix multiplication by column

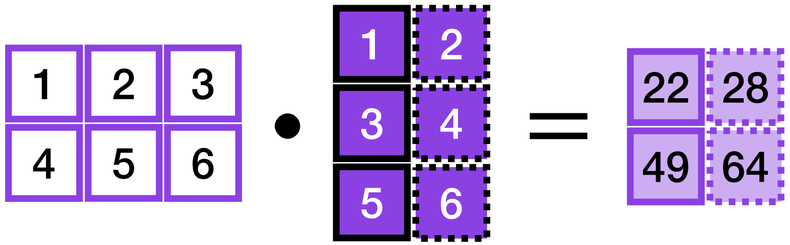


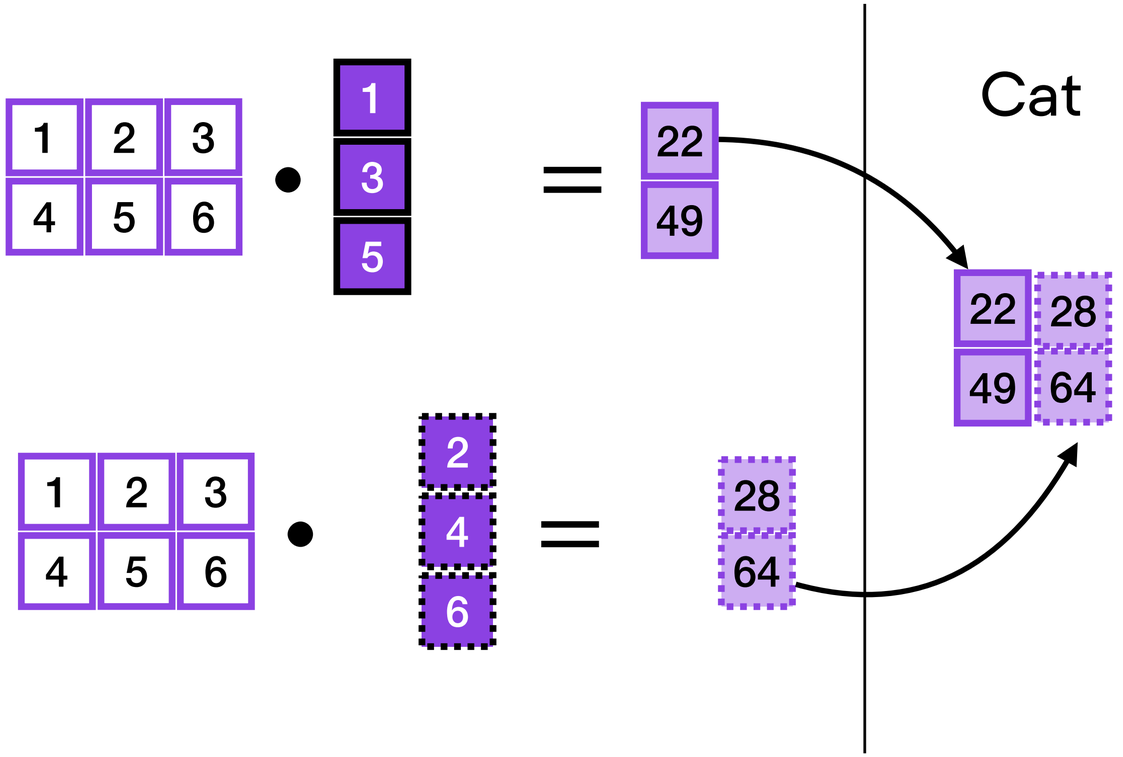




Tensor Parallelism

For example, split the matrix multiplication by column

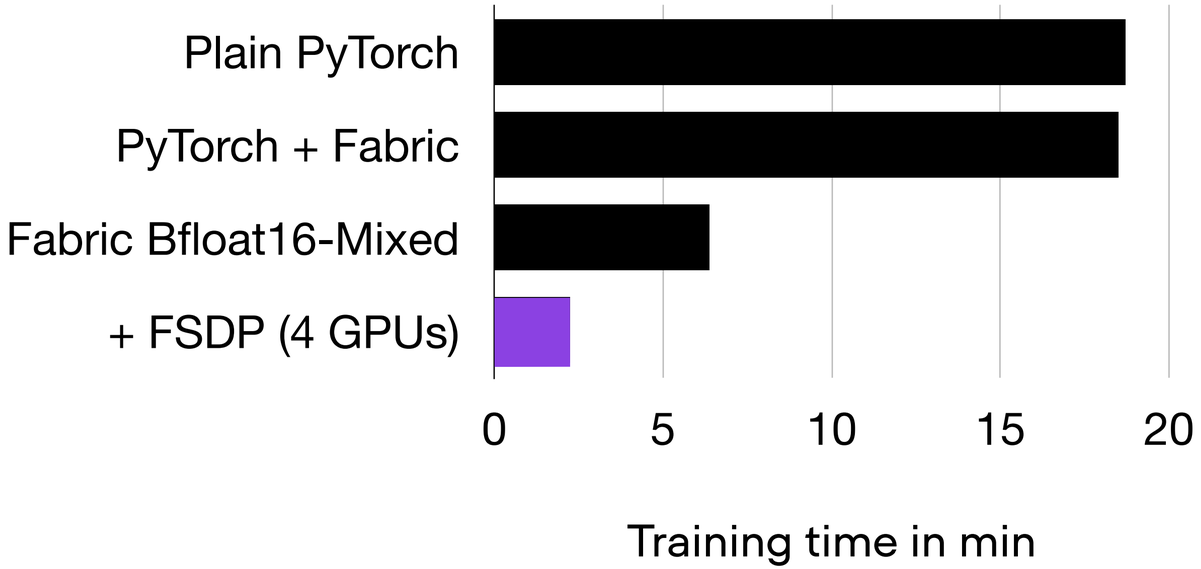


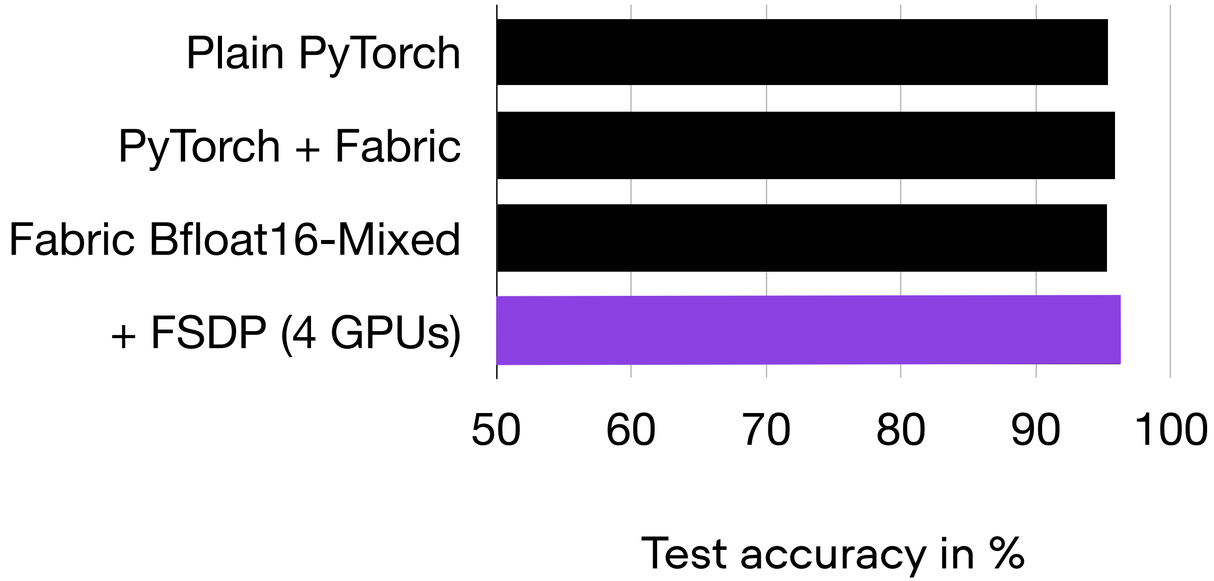


Required Change: 1/2 Lines of Code

```
fabric = Fabric(
    accelerator="cuda", precision="bf16-mixed",
    devices=4, strategy="FSDP"
)
```

Multi-GPU Training with
Fully Sharded Data Parallelism





Contact

@rasbt

sebastian@lightning.ai

https://sebastianraschka.com

https://lightning.ai

Code & slides

https://github.com/rasbt/cvpr2023