Multi-GPU Training in PyTorch with Code (Part 1): Single GPU Example

Anthony Peng · Follow

Published in Polo Club of Data Science · 5 min read · Jul 7

This tutorial series will cover how to launch your deep learning training on multiple GPUs in PyTorch. We will discuss how to extrapolate a single GPU training example to multiple GPUs via Data Parallel (DP) and Distributed Data Parallel (DDP), compare the performance, analyze details inside DDP distributed sampler, and ensure fault tolerance with torchrun. I spent tons of time going over the official tutorial, API manual, and other blogs to ensure the correctness of this tutorial. Feel free to comment below if anything is unclear to you.

• • •

> ***Part 1. Single GPU Example (this article)*** *— Training ResNet34 on CIFAR10*

Part2. Data Parallel — Training code & issue between DP and NVLink

Part3. Distributed Data Parallel — Training code & Analysis

Part4. Torchrun — Fault tolerance

. . .

In this article, we provide an example of training ResNet34 on CIFAR10 with a single GPU. If any of the below code is unfamiliar to you, please check the official tutorial on PyTorch Basics.

Basics

Necessary packages & hyperparameters. All trained models are saved at "./trained_models", and the CIFAR10 dataset is saved at "./data". These hyperparameters will stay the same during multi-GPU training.

```
from pathlib import Path
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torchvision.models import resnet34
from torchvision.transforms import transforms
from torchvision.datasets import CIFAR10
import torch.optim as optim
from torch import Tensor
from typing import Iterator, Tuple
import torchmetrics

def prepare_const() -> dict:
    """Data and model directory + Training hyperparameters"""
    data_root = Path("data")
    trained_models = Path("trained_models")

    if not data_root.exists():
```

```
        data_root.mkdir()

    if not trained_models.exists():
        trained_models.mkdir()

    const = dict(
        data_root=data_root,
        trained_models=trained_models,
        total_epochs=15,
        batch_size=128,
        lr=0.1,  # learning rate
        momentum=0.9,
        lr_step_size=5,
        save_every=3,
    )

    return const
```

ResNet34 model. Torchvision ResNets are defined for ImageNet, which has a higher resolution than CIFAR10, so we replace the stem stage with a smaller kernel_size (7 to 3) and remove the maxpooling layer.

```
def cifar_model() -> nn.Module:
    model = resnet34(num_classes=10)
    model.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1, bias=Fals
    model.maxpool = nn.Identity()
    return model
```

CIFAR10 dataset.

```
def cifar_dataset(data_root: Path) -> Tuple[Dataset, Dataset]:
    transform = transforms.Compose(
        [
            transforms.ToTensor(),
            transforms.Normalize(
```

```
                mean=(0.49139968, 0.48215827, 0.44653124),
                std=(0.24703233, 0.24348505, 0.26158768),
            ),
        ]
    )

    trainset = CIFAR10(root=data_root, train=True, transform=transform, download
    testset = CIFAR10(root=data_root, train=False, transform=transform, download

    return trainset, testset
```

. . .

Single GPU Training

Dataloader for single gpu

```
def cifar_dataloader_single(
    trainset: Dataset, testset: Dataset, bs: int
) -> Tuple[DataLoader, DataLoader]:
    trainloader = DataLoader(trainset, batch_size=bs, shuffle=True, num_workers=
    testloader = DataLoader(testset, batch_size=bs, shuffle=False, num_workers=8

    return trainloader, testloader
```

Trainer class. We use torchmetrics to compute the classification accuracy since it supports distributed scenarios. We will verify its correctness in the DDP article. Note torchmetrics.Accuracy contain parameters. so it has to be on GPU. The code is pretty straightforward as *_run_batch* takes care of each batch and *_run_epoch* takes care of each epoch. The lr_scheduler decreases the learning rate (lr) by 10 for every 5 epochs.

```
class TrainerSingle:
    def __init__(
        self,
        gpu_id: int,
        model: nn.Module,
        trainloader: DataLoader,
        testloader: DataLoader,
    ):
        self.gpu_id = gpu_id

        self.const = prepare_const()
        self.model = model.to(self.gpu_id)
        self.trainloader = trainloader
        self.testloader = testloader
        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = optim.SGD(
            self.model.parameters(),
            lr=self.const["lr"],
            momentum=self.const["momentum"],
        )
        self.lr_scheduler = optim.lr_scheduler.StepLR(
            self.optimizer, self.const["lr_step_size"]
        )
        self.train_acc = torchmetrics.Accuracy(
            task="multiclass", num_classes=10, average="micro"
        ).to(self.gpu_id)

        self.valid_acc = torchmetrics.Accuracy(
            task="multiclass", num_classes=10, average="micro"
        ).to(self.gpu_id)

    def _run_batch(self, src: Tensor, tgt: Tensor) -> float:
        self.optimizer.zero_grad()

        out = self.model(src)
        loss = self.criterion(out, tgt)
        loss.backward()
        self.optimizer.step()

        self.train_acc.update(out, tgt)
        return loss.item()

    def _run_epoch(self, epoch: int):
        loss = 0.0
        for src, tgt in self.trainloader:
            src = src.to(self.gpu_id)
            tgt = tgt.to(self.gpu_id)
            loss_batch = self._run_batch(src, tgt)
```

```
                loss += loss_batch
            self.lr_scheduler.step()

            print(
                f"{'-' * 90}\n[GPU{self.gpu_id}] Epoch {epoch:2d} | Batchsize: {self
                flush=True,
            )

            self.train_acc.reset()

        def _save_checkpoint(self, epoch: int):
            ckp = self.model.state_dict()
            model_path = self.const["trained_models"] / f"CIFAR10_single_epoch{epoch
            torch.save(ckp, model_path)

        def train(self, max_epochs: int):
            self.model.train()
            for epoch in range(max_epochs):
                self._run_epoch(epoch)
                if epoch % self.const["save_every"] == 0:
                    self._save_checkpoint(epoch)
            # save last epoch
            self._save_checkpoint(max_epochs - 1)

        def test(self, final_model_path: str):
            self.model.load_state_dict(torch.load(final_model_path))
            self.model.eval()
            with torch.no_grad():
                for src, tgt in self.testloader:
                    src = src.to(self.gpu_id)
                    tgt = tgt.to(self.gpu_id)
                    out = self.model(src)
                    self.valid_acc.update(out, tgt)
            print(
                f"[GPU{self.gpu_id}] Test Acc: {100 * self.valid_acc.compute().item(
            )
```

Main function. Finally, we load all hyperparameters, load the dataset and dataloader, and train the model. After training, we test the data on the testset. Note that CIFAR10 has 50,000 training samples and 10,000 testing samples.

```
def main_single(gpu_id: int, final_model_path: str):
    const = prepare_const()
    train_dataset, test_dataset = cifar_dataset(const["data_root"])
    train_dataloader, test_dataloader = cifar_dataloader_single(
        train_dataset, test_dataset, const["batch_size"]
    )
    model = cifar_model()
    trainer = TrainerSingle(
        gpu_id=gpu_id,
        model=model,
        trainloader=train_dataloader,
        testloader=test_dataloader,
    )
    trainer.train(const["total_epochs"])
    trainer.test(final_model_path)
```

. . .

Experiments

It's time to roll! The model is trained for 15 epochs, and the weights are saved after every 3 epochs.

```
if __name__ == "__main__":
    gpu_id = 0
    final_model_path =  Path("./trained_models/CIFAR10_single_epoch14.pt")
    main_single(gpu_id, final_model_path)
```

Output

```
$ CUDA_VISIBLE_DEVICES=0 python main.py
Files already downloaded and verified
Files already downloaded and verified
```

```
-----------------------------------------------------------------------------
[GPU0] Epoch  0 | Batchsize: 128 | Steps: 391 | LR: 0.1000 | Loss: 2.0430 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch  1 | Batchsize: 128 | Steps: 391 | LR: 0.1000 | Loss: 1.3259 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch  2 | Batchsize: 128 | Steps: 391 | LR: 0.1000 | Loss: 1.0207 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch  3 | Batchsize: 128 | Steps: 391 | LR: 0.1000 | Loss: 0.8059 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch  4 | Batchsize: 128 | Steps: 391 | LR: 0.0100 | Loss: 0.6558 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch  5 | Batchsize: 128 | Steps: 391 | LR: 0.0100 | Loss: 0.3658 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch  6 | Batchsize: 128 | Steps: 391 | LR: 0.0100 | Loss: 0.2757 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch  7 | Batchsize: 128 | Steps: 391 | LR: 0.0100 | Loss: 0.2090 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch  8 | Batchsize: 128 | Steps: 391 | LR: 0.0100 | Loss: 0.1468 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch  9 | Batchsize: 128 | Steps: 391 | LR: 0.0010 | Loss: 0.0967 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch 10 | Batchsize: 128 | Steps: 391 | LR: 0.0010 | Loss: 0.0514 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch 11 | Batchsize: 128 | Steps: 391 | LR: 0.0010 | Loss: 0.0407 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch 12 | Batchsize: 128 | Steps: 391 | LR: 0.0010 | Loss: 0.0366 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch 13 | Batchsize: 128 | Steps: 391 | LR: 0.0010 | Loss: 0.0339 | Acc:
-----------------------------------------------------------------------------
[GPU0] Epoch 14 | Batchsize: 128 | Steps: 391 | LR: 0.0001 | Loss: 0.0319 | Acc:
[GPU0] Test Acc: 78.3000%
```

Cool! We have successfully trained a ResNet34 on CIFAR10 with a single GPU. In the next article, we will explore how to leverage multiple GPUs to accelerate our training.

Deep Learning | Gpu | Machine Learning

Written by Anthony Peng

Follow

13 Followers · Editor for Polo Club of Data Science

CS PhD @Georgia Tech; Homepage: shengyun-peng.github.io/

More from Anthony Peng and Polo Club of Data Science

Anthony Peng in Polo Club of Data Science

How to measure inter-GPU connection speed (single node)?

Key GPU Knowledge for ML Researchers Series

8 min read · Oct 12

63

Duen Horng "Polo" Ch... in Polo Club of Data Scien...

How to Import Editable PDF into Figma (Easy and Free!)

Key idea is to convert the PDF to SVG that Figma can import and edit

1 min read · Jul 15

52

Duen Horng "Polo" Ch... in Polo Club of Data Scien...

How to link a file in Notes on a Mac? Easy! Also work for folders.

Easy steps to add a link to a file in Notes on Mac

1 min read · Nov 8

3

Anthony Peng in Polo Club of Data Science

Multi-GPU Training in PyTorch with Code (Part 4): Torchrun

We discussed single-GPU training in Part 1, multi-GPU training with DP in Part 2, and...

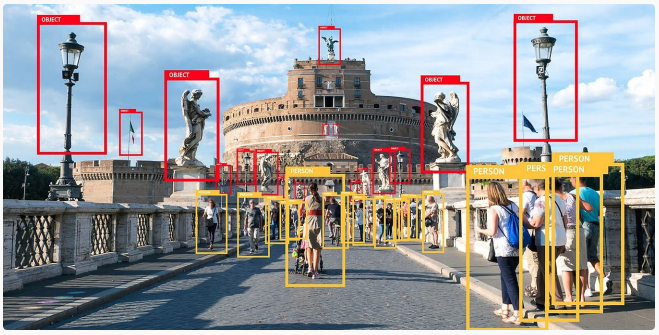
5 min read · Jul 7

67

See all from Anthony Peng

See all from Polo Club of Data Science

Recommended from Medium



Gavin Li in AI Advances

Unbelievable! Run 70B LLM Inference on a Single 4GB GPU wi...

Large language models require huge amounts of GPU memory. Is it possible to ru...

6 min read · Nov 18

1.6K 23

Francesco Franco in GoPenAI

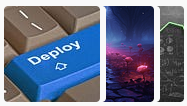
Object Detection with Python and HuggingFace Transformers

YOLO! If you're into machine learning, it's a term that rings a bell. Indeed, You Only Look...

10 min read · Nov 21

129 1

Lists



Predictive Modeling w/ Python

20 stories · 652 saves

Practical Guides to Machine Learning

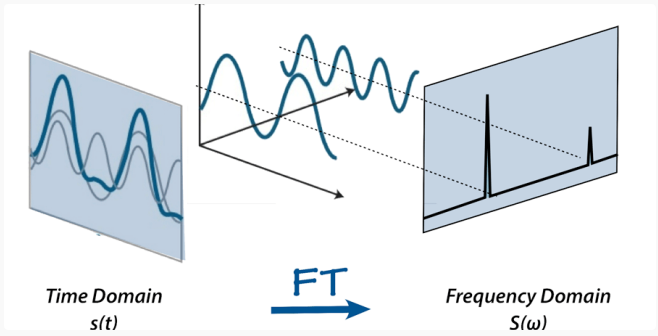
10 stories · 732 saves

Natural Language Processing

920 stories · 436 saves

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 226 saves

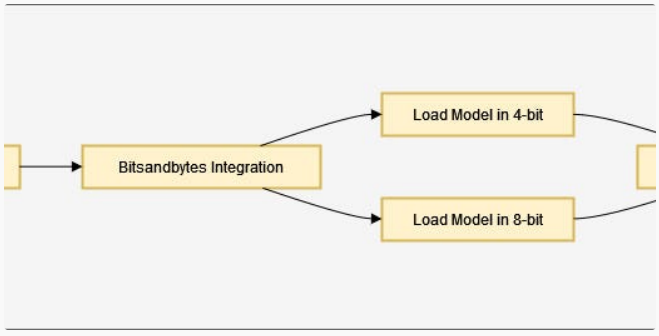


Everton Gomede, PhD

The Fourier Transform and its Application in Machine Learning

Introduction

5 min read · Nov 2



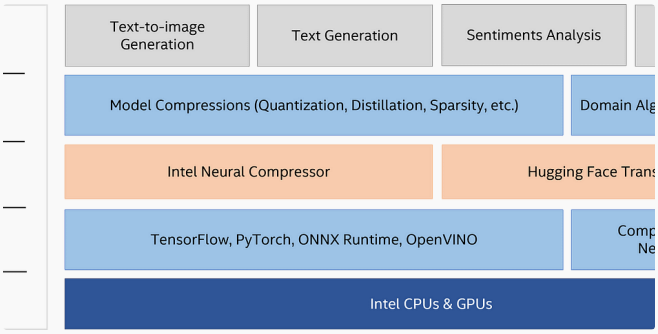
Rakesh Rajpurohit

Model Quantization with 🤗 Hugging Face Transformers and...

Introduction:

4 min read · Aug 20

483 2



Intel(R) Neural Compressor

Intel-Optimized Llama.CPP

Intel® Extension for Transformers is an innovative toolkit to accelerate Transformer-...

2 min read · Oct 20

74

33

Mustafa Mujahid

Pytorch for Mac M1/M2 with GPU acceleration 2023. Jupyter and V...

Introduction

5 min read · Aug 6

31

See more recommendations