实践教程｜PyTorch 并行训练极简 Demo

极市平台 2023-06-11 22:01:29 发表于江苏 手机阅读

以下文章来源于天才程序员周弈帆，作者学深度学习的

天才程序员周弈帆

ACM金牌选手教你编程、算法、深度学习。个人博客：zhouyifan.net

↑ 点击**蓝字** 关注极市平台

作者丨周弈帆

来源丨天才程序员周弈帆

编辑丨极市平台

极市导读

一份非常简单的PyTorch并行训练代码。希望读者能够在接触尽可能少的新知识的前提下学会写并行训练。 >>加入极市CV技术交流群，走在计算机视觉的最前沿

完整代码 `main.py` ：

```
import os
import torch
import torch.distributed as dist
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torch.utils.data.distributed import DistributedSampler
from torch.nn.parallel import DistributedDataParallel
```

```
def setup():
    dist.init_process_group('nccl')


def cleanup():
    dist.destroy_process_group()


class ToyModel(nn.Module):

    def __init__(self) -> None:
        super().__init__()
        self.layer = nn.Linear(1, 1)

    def forward(self, x):
        return self.layer(x)


class MyDataset(Dataset):

    def __init__(self):
        super().__init__()
        self.data = torch.tensor([1, 2, 3, 4], dtype=torch.float32)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        return self.data[index:index + 1]


ckpt_path = 'tmp.pth'


def main():
    setup()
    rank = dist.get_rank()
    pid = os.getpid()
    print(f'current pid: {pid}')
    print(f'Current rank {rank}')
    device_id = rank % torch.cuda.device_count()

    dataset = MyDataset()
    sampler = DistributedSampler(dataset)
    dataloader = DataLoader(dataset, batch_size=2, sampler=sampler)
```

```
    model = ToyModel().to(device_id)
    ddp_model = DistributedDataParallel(model, device_ids=[device_id])
    loss_fn = nn.MSELoss()
    optimizer = optim.SGD(ddp_model.parameters(), lr=0.001)

    if rank == 0:
        torch.save(ddp_model.state_dict(), ckpt_path)

    dist.barrier()

    map_location = {'cuda:0': f'cuda:{device_id}'}
    state_dict = torch.load(ckpt_path, map_location=map_location)
    print(f'rank {rank}: {state_dict}')
    ddp_model.load_state_dict(state_dict)

    for epoch in range(2):
        sampler.set_epoch(epoch)
        for x in dataloader:
            print(f'epoch {epoch}, rank {rank} data: {x}')
            x = x.to(device_id)
            y = ddp_model(x)
            optimizer.zero_grad()
            loss = loss_fn(x, y)
            loss.backward()
            optimizer.step()

    cleanup()


if __name__ == '__main__':
    main()
```

假设有4张卡，使用第三和第四张卡的并行运行命令（torch v1.10 以上）：

```
export CUDA_VISIBLE_DEVICES=2,3
torchrun --nproc_per_node=2 dldemos/PyTorchDistributed/main.py
```

较老版本的PyTorch应使用下面这条命令（这种方法在新版本中也能用，但是会报Warning）：

```
export CUDA_VISIBLE_DEVICES=2,3
python -m torch.distributed.launch --nproc_per_node=2 dldemos/PyTorchDistributed/main.py
```

程序输出：

```
current pid: 3592707
Current rank 1
current pid: 3592706
Current rank 0
rank 0: OrderedDict([('module.layer.weight', tensor([[0.3840]], device='cuda:0')), ('modul
rank 1: OrderedDict([('module.layer.weight', tensor([[0.3840]], device='cuda:1')), ('modul
epoch 0, rank 0 data: tensor([[1.],
        [4.]])
epoch 0, rank 1 data: tensor([[2.],
        [3.]])
epoch 1, rank 0 data: tensor([[2.],
        [3.]])
epoch 1, rank 1 data: tensor([[4.],
        [1.]])
```

下面来稍微讲解一下代码。这份代码演示了一种较为常见的PyTorch并行训练方式：一台机器，多GPU。一个进程管理一个GPU。每个进程共享模型参数，但是使用不同的数据，即batch size扩大了 GPU个数 倍。

为了实现这种并行训练：需要解决以下几个问题：

- 怎么开启多进程？
- 模型怎么同步参数与梯度？
- 数据怎么划分到多个进程中？

带着这三个问题，我们来从头看一遍这份代码。

这份代码要拟合一个恒等映射 y=x 。使用的数据集非常简单，只有 [1, 2, 3, 4] 四个数字。

```
class MyDataset(Dataset):

    def __init__(self):
        super().__init__()
        self.data = torch.tensor([1, 2, 3, 4], dtype=torch.float32)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        return self.data[index:index + 1]
```

模型也只有一个线性函数：

```
class ToyModel(nn.Module):

    def __init__(self) -> None:
        super().__init__()
        self.layer = nn.Linear(1, 1)

    def forward(self, x):
        return self.layer(x)
```

为了并行训练这个模型，我们要开启多进程。PyTorch提供的 `torchrun` 命令以及一些API封装了多进程的实现。我们只要在普通单进程程序前后加入以下的代码：

```
def setup():
    dist.init_process_group('nccl')

def main():
    setup()

    ...

    cleanup()

def cleanup():
    dist.destroy_process_group()
```

再用 `torchrun \--nproc_per_node=GPU_COUNT main.py` 去跑这个脚本，就能用 `GPU_COUNT` 个进程来运行这个程序，每个进程分配一个GPU。我们可以用 `dist.get_rank()` 来查看当前进程的GPU号。同时，我们也可以验证，不同的GPU号对应了不同的进程id。

```
def main():
    setup()
    rank = dist.get_rank()
    pid = os.getpid()
    print(f'current pid: {pid}')
    print(f'Current rank {rank}')
    device_id = rank % torch.cuda.device_count()

Output:
current pid: 3592707
Current rank 1
current pid: 3592706
Current rank 0
```

接下来，我们来解决数据并行的问题。我们要确保一个epoch的数据被分配到了不同的进程上，以实现batch size的扩大。在PyTorch中，只要在生成 `Dataloader` 时把 `DistributedS`

ampler 的实例传入 sampler 参数就行了。 DistributedSampler 会自动对数据采样，并放到不同的进程中。我们稍后可以看到数据的采样结果。

```
dataset = MyDataset()
sampler = DistributedSampler(dataset)
dataloader = DataLoader(dataset, batch_size=2, sampler=sampler)
```

接下来来看模型并行是怎么实现的。在这种并行训练方式下，每个模型使用同一份参数。在训练时，各个进程并行；在梯度下降时，各个进程会同步一次，保证每个进程的模型都更新相同的梯度。PyTorch又帮我们封装好了这些细节。我们只需要在现有模型上套一层 DistributedDataParallel ，就可以让模型在后续 backward 的时候自动同步梯度了。其他的操作都照旧，把新模型 ddp_model 当成旧模型 model 调用就行。

```
model = ToyModel().to(device_id)
ddp_model = DistributedDataParallel(model, device_ids=[device_id])
loss_fn = nn.MSELoss()
optimizer = optim.SGD(ddp_model.parameters(), lr=0.001)
```

准备好了一切后，就可以开始训练了：

```
for epoch in range(2):
    sampler.set_epoch(epoch)
    for x in dataloader:
        print(f'epoch {epoch}, rank {rank} data: {x}')
        x = x.to(device_id)
        y = ddp_model(x)
        optimizer.zero_grad()
        loss = loss_fn(x, y)
        loss.backward()
        optimizer.step()
```

sampler 自动完成了打乱数据集的作用。因此，在定义 DataLoader 时，不用开启 shuffle 选项。而在每个新epoch中，要用 sampler.set_epoch(epoch) 更新 sampler ，重新打乱数据集。通过输出也可以看出，数据集确实被打乱了。

```
Output:
epoch 0, rank 0 data: tensor([[1.],
        [4.]])
epoch 0, rank 1 data: tensor([[2.],
        [3.]])
epoch 1, rank 0 data: tensor([[2.],
        [3.]])
epoch 1, rank 1 data: tensor([[4.],
        [1.]])
```

大家可以去掉这行代码，跑一遍脚本，看看这行代码的作用。如果没有这行代码，每轮的数据分配情况都是一样的。

```
epoch 0, rank 1 data: tensor([[2.],
        [3.]])
epoch 0, rank 0 data: tensor([[1.],
        [4.]])
epoch 1, rank 1 data: tensor([[2.],
        [3.]])
epoch 1, rank 0 data: tensor([[1.],
        [4.]])
```

其他的训练代码和单进程代码一模一样，我们不需要做任何修改。

训练完模型后，应该保存模型。由于每个进程的模型都是一样的，我们只需要让一个进程来保存模型即可。注意，在保存模型时，其他进程不要去修改模型参数。这里最好加上一行 `dist.barrier()` ，它可以用来同步进程的运行状态。只有0号GPU的进程存完了模型，所有模型再进行下一步操作。

```
if rank == 0:
    torch.save(ddp_model.state_dict(), ckpt_path)

dist.barrier()
```

读取时需要注意一下。模型存储参数时会保存参数所在设备。由于我们只用了0号GPU的进程存模型，所有参数的 `device` 都是 `cuda:0` 。而读取模型时，每个设备上的模型都要去读一次模型，参数的位置要做一个调整。

```
map_location = {'cuda:0': f'cuda:{device_id}'}
state_dict = torch.load(ckpt_path, map_location=map_location)
print(f'rank {rank}: {state_dict}')
ddp_model.load_state_dict(state_dict)
```

从输出中可以看出，在不同的进程中，参数字典是不一样的：

```
rank 0: OrderedDict([('module.layer.weight', tensor([[0.3840]], device='cuda:0')), ('modul
rank 1: OrderedDict([('module.layer.weight', tensor([[0.3840]], device='cuda:1')), ('modul
```

这里还有一个重要的细节。使用 `DistributedDataParallel` 把 `model` 封装成 `ddp_model` 后，模型的参数名里多了一个 `module` 。这是因为原来的模型 `model` 被保存到了 `ddp_model.module` 这个成员变量中（ `model == ddp_model.module` ）。在混用单GPU和多GPU的训练代码时，要注意这个参数名不兼容的问题。最好的写法是每次存取 `ddp_model.module` ，这样单GPU和多GPU的checkpoint可以轻松兼容。

到此，我们完成了一个极简的PyTorch并行训练Demo。从代码中能看出，PyTorch的封装非常到位，我们只需要在单进程代码上稍作修改，就能开启并行训练。最后，我再来总结一下单卡训练转换成并行训练的修改处：

1. 程序开始时执行 `dist.init_process_group('nccl')` ，结束时执行 `dist.destroy_process_group()` 。
2. 用 `torchrun --nproc_per_node=GPU_COUNT main.py` 运行脚本。
3. 进程初始化后用 `rank = dist.get_rank()` 获取当前的GPU ID，把模型和数据都放到这个GPU上。
4. 封装一下模型

```
ddp_model = DistributedDataParallel(model, device_ids=[device_id])
```

5. 封装一下 `DataLoader`

```
dataset = MyDataset()
sampler = DistributedSampler(dataset)
dataloader = DataLoader(dataset, batch_size=2, sampler=sampler)
```

6. 训练时打乱数据。 `sampler.set_epoch(epoch)`
7. 保存只在单卡上进行。

```
if rank == 0:
    torch.save(ddp_model.state_dict(), ckpt_path)
dist.barrier()
```

8. 读取数据时注意 `map_location` ，也要注意参数名里的 `module` 。

```
map_location = {'cuda:0': f'cuda:{device_id}'}
state_dict = torch.load(ckpt_path, map_location=map_location)
ddp_model.load_state_dict(state_dict)
```

项目网址：https://github.com/SingleZombie/DL-Demos/tree/master/dldemos/PyTorchDistributed

参考资料：

1. 官方教程：https://pytorch.org/tutorials/intermediate/ddp_tutorial.html
2. 另一个展示简单Demo的文章：https://zhuanlan.zhihu.com/p/350301395

点击阅读原文进入CV社区

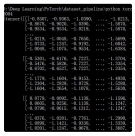
收获更多技术干货

阅读原文

喜欢此内容的人还喜欢

实践教程｜PyTorch数据导入机制与标准化代码模板

极市平台



实践教程｜使用 OpenCV 进行特征提取（颜色、形状和纹理）

极市平台

ICCV23｜将隐式神经表征用于低光增强，北大张健团队提出NeRCo

极市平台