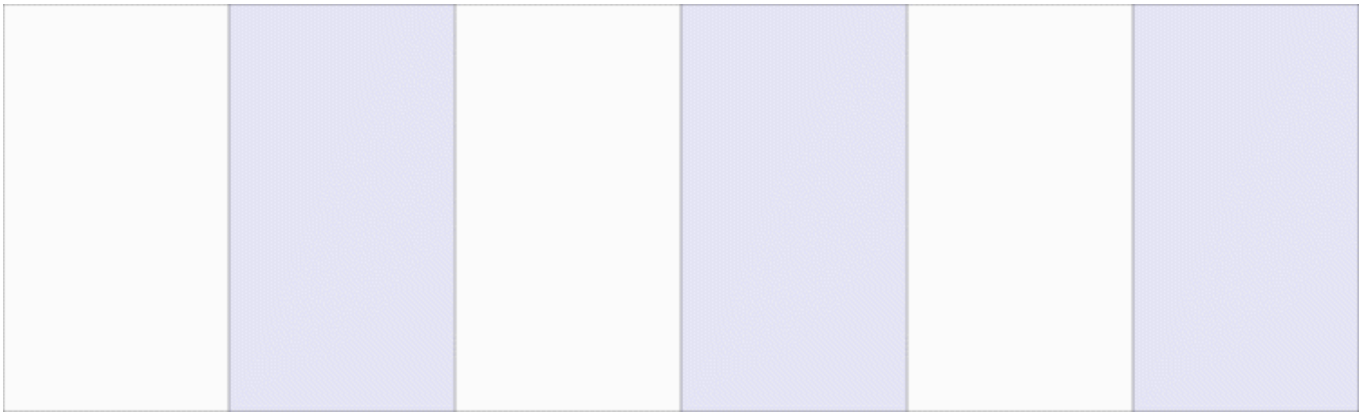
实践教程｜PyTorch训练加速技巧

CV开发者都爱看的 极市平台 2023-01-08 22:00:40 发表于广东 手机阅读

收录于合集

#pytorch 2 #实践教程 11



作者｜用什么名字没那么重要@知乎（已授权）

来源｜https://zhuanlan.zhihu.com/p/360697168

编辑｜极市平台

极市导读

本篇讲述了如何应用torch实现混合精度运算、数据并行和分布式运算。 >>加入极市CV技术交流群，走在计算机视觉的最前沿

由于最近的程序对速度要求比较高，想要快速出结果，因此特地学习了一下混合精度运算和并行化操作，由于已经有很多的文章介绍相关的原理，因此本篇只讲述如何应用**torch**实现混合精度运算、数据并行和分布式运算，不具体介绍原理。

混合精度

自动混合精度训练（auto Mixed Precision，AMP）可以大幅度降低训练的成本并提高训练的速度。在此之前，自动混合精度运算是使用NVIDIA开发的Apex工具。从PyTorch1.6.0开始，**PyTorch**已经自带了AMP模块，因此接下来主要对**PyTorch**自带的amp模块进行简单的使用介绍。

```
## 导入amp工具包
from torch.cuda.amp import autocast, GradScaler

model.train()

## 对梯度进行scale来加快模型收敛，
## 因为float16梯度容易出现underflow（梯度过小）
scaler = GradScaler()

batch_size = train_loader.batch_size
num_batches = len(train_loader)
end = time.time()
for i, (images, target) in tqdm.tqdm(
    enumerate(train_loader), ascii=True, total=len(train_loader)
):
    # measure data loading time
    data_time.update(time.time() - end)
    optimizer.zero_grad()
    if args.gpu is not None:
        images = images.cuda(args.gpu, non_blocking=True)

    target = target.cuda(args.gpu, non_blocking=True)
    # 自动为GPU op选择精度来提升训练性能而不降低模型准确度
    with autocast():
    # compute output
        output = model(images)

        loss = criterion(output, target)

    scaler.scale(loss).backward()
    # optimizer.step()
    scaler.step(optimizer)
    scaler.update()
```

数据并行

当服务器有单机有多卡的时候，为了实现模型的加速（可能由于一张GPU不够），可以采用单机多卡对模型进行训练。为了实现这个目的，我们必须想办法让一个模型可以分布在多个GPU上进行训练。

PyTorch中，**nn.DataParallel**为我提供了一个简单的接口，可以很简单的实现对模型的并行化，我们只需要用**nn.DataParallel**对模型进行包装，在设置一些参数，就可以很容易的实现模型的多卡并行。

```
# multigpu表示显卡的号码
multigpu = [0,1,2,3,4,5,6,7]
# 设置主GPU,用来汇总模型的损失函数并且求导，对梯度进行更新
torch.cuda.set_device(args.multigpu[0])
# 模型的梯度全部汇总到gpu[0]上来
model = torch.nn.DataParallel(model, device_ids=args.multigpu).cuda(
        args.multigpu[0]
        )
```

nn.DataParallel使用混合精度运算

nn.DataParallel对模型进行混合精度运算需要进行一些特殊的配置，不然模型是无法实现数据并行化的。autocast 设计为 “thread local” 的，所以只在 main thread 上设 autocast 区域是不 work 的。借鉴自（https://zhuanlan.zhihu.com/p/348554267） 这里先给出错误的操作：

```
model = MyModel()
dp_model = nn.DataParallel(model)

with autocast():     # dp_model's internal threads won't autocast.
     #The main thread's autocast state has no effect.
     output = dp_model(input)     # loss_fn still autocasts, but it's too late...
     loss = loss_fn(output)
```

解决的方法有两种，下面分别介绍：**1. 在模型模块的forward函数中加入装饰函数**

```
MyModel(nn.Module):
    ...
    @autocast()
    def forward(self, input):
       ...
```

2. 另一个正确姿势是在 forward 的里面设 autocast 区域： `python MyModel(nn.Module): ... def forward(self, input): with autocast(): ...` **在对forward函数进行操作后，再在main thread中使用autocast** ```python model = MyModel() dp_model = nn.DataParallel(model)

with autocast(): output = dp_model(input) loss = loss_fn(output) ```

nn.DataParallel缺点

在每个训练的batch中，nn.DataParallel模块会把所有的loss全部反传到gpu[0]上，几个G的数据传输，loss的计算都需要在一张显卡上完成，这样子很容易造成显卡的负载不均匀，经常可以看到gpu[0]的负载会明显比其他的gpu高很多。此外，显卡的数据传输速度会对模型的训练速度造成很大的瓶颈，这显然是不合理的。因此接下来我们将介绍，具体原理可以参考单机多卡操作(分布式DataParallel，混合精度，Horovod)（https://zhuanlan.zhihu.com/p/158375055）

分布式运算

nn.DistributedDataParallel：多进程控制多 GPU，一起训练模型。

优点

每个进程控制一块GPU，可以保证模型的运算可以不受到显卡之间通信的影响，并且可以使得每张显卡的负载相对比较均匀。但是相对于单机单卡或者单机多卡(nn.DataParallel）来说，就有几个问题

1. 同步不同GPU上的模型参数，特别是BatchNormalization 2. 告诉每个进程自己的位置，使用哪块GPU，用args.local_rank参数指定 3. 每个进程在取数据的时候要确保拿到的是不同的数据（DistributedSampler）

使用方式介绍

启动程序 由于博主目前也只是实践了单机多卡操作，因此主要对单机多卡进行介绍。区别于平时简单的运行python程序，我们需要使用PyTorch自带的启动器 torch.distributed.launch 来启动程序。

```
# 其中CUDA_VISIBLE_DEVICES指定机器上显卡的数量
# nproc_per_node程序进程的数量
CUDA_VISIBLE_DEVICES=0,1,2,3 python -m torch.distributed.launch --nproc_per_node=4 main.py
```

配置主程序

```
parser.add_argument('--local_rank', type=int, default=0, help='node rank for distributed t
# 配置local_rank参数，告诉每个进程自己的位置，要使用哪张GPU
```

初始化显卡通信和参数获取的方式

```
# 为这个进程指定GPU
torch.cuda.set_device(args.local_rank)
# 初始化GPU通信方式NCLL和参数的获取方式，其中env表示环境变量
# PyTorch实现分布式运算是通过NCLL进行显卡通信的
torch.distributed.init_process_group(
    backend='nccl',
    rank=args.local_rank
)
```

重新配置DataLoader

```
kwargs = {"num_workers": args.workers, "pin_memory": True} if use_cuda else {}

train_sampler = DistributedSampler(train_dataset)
self.train_loader = torch.utils.data.DataLoader(
            train_dataset,
            batch_size=args.batch_size,
            sampler=train_sampler,
            **kwargs
        )

# 注意，由于使用了Sampler方法，dataloader中就不能加shuffle、drop_last等参数了
'''
PyTorch dataloader.py 192-197 代码
        if batch_sampler is not None:
            # auto_collation with custom batch_sampler
            if batch_size != 1 or shuffle or sampler is not None or drop_last:
                raise ValueError('batch_sampler option is mutually exclusive '
                                 'with batch_size, shuffle, sampler, and '
                                 'drop_last')'''
```

pin_memory就是锁页内存，创建DataLoader时，设置pin_memory=True，则意味着生成的Tensor数据最开始是属于内存中的锁页内存，这样将内存的Tensor转义到GPU的显存就会更快一些。

模型的初始化

```
torch.cuda.set_device(args.local_rank)
device = torch.device('cuda', args.local_rank)
```

```
model.to(device)
model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model)
model = torch.nn.parallel.DistributedDataParallel(
        model,
        device_ids=[args.local_rank],
        output_device=args.local_rank,
        find_unused_parameters=True,
        )
torch.backends.cudnn.benchmark=True
# 将会让程序在开始时花费一点额外时间，为整个网络的每个卷积层搜索最适合它的卷积实现算法，进而实现网络的加速
# DistributedDataParallel可以将不同GPU上求得的梯度进行汇总，实现对模型GPU的更新
```

DistributedDataParallel可以将不同GPU上求得的梯度进行汇总，实现对模型GPU的更新

同步BatchNormalization层

对于比较消耗显存的训练任务时，往往单卡上的相对批量过小，影响模型的收敛效果。跨卡同步 Batch Normalization 可以使用全局的样本进行归一化，这样相当于‘增大‘了批量大小，这样训练效果不再受到使用 GPU 数量的影响。参考自单机多卡操作(分布式DataParallel，混合精度，Horovod) 幸运的是，在近期的Pytorch版本中，PyTorch已经开始原生支持BatchNormalization层的同步。

- **torch.nn.SyncBatchNorm**
- **torch.nn.SyncBatchNorm.convert_sync_batchnorm**：将BatchNorm-alization层自动转化为**torch.nn.SyncBatchNorm**实现不同GPU上的BatchNormalization层的同步

具体实现请参考模型的初始化部分代码 `python model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model)`

同步模型初始化的随机种子

目前还没有尝试过不同进程上使用不同随机种子的状况。为了保险起见，建议确保每个模型初始化的随机种子相同，保证每个GPU进程上的模型是同步的。

总结

站在巨人的肩膀上，对前段时间自学模型加速，踩了许多坑，最后游行都添上了，最后对一些具体的代码进行了一些总结，其中也参考了许多其他的博客。希望能对大家有一些帮助。

引用（不分前后）：

1. PyTorch 21.单机多卡操作(分布式DataParallel，混合精度，Horovod)
2. PyTorch 源码解读之 torch.cuda.amp: 自动混合精度详解
3. PyTorch的自动混合精度（AMP）
4. 训练提速60%！只需5行代码，PyTorch 1.6即将原生支持自动混合精度训练
5. torch.backends.cudnn.benchmark ?!
6. 恶补了 Python 装饰器的八种写法，你随便问~

公众号后台回复“CNN综述”获取67页综述深度卷积神经网络架构

极市平台

为计算机视觉开发者提供全流程算法开发训练平台，以及大咖技术分享、社区交流、竞...

848篇原创内容

公众号

极市干货

技术干货：损失函数技术总结及Pytorch使用示例｜深度学习有哪些trick？｜目标检测正负样本区分策略和平衡策略总结

实操教程：GPU多卡并行训练总结（以pytorch为例）｜CUDA WarpReduce 学习笔记｜卷积神经网络压缩方法总结

收录于合集 #pytorch 2

上一篇 · PyTorch 深度剖析：如何使用模型并行技术 （Model Parallel）

阅读原文

9个数据科学中常见距离度量总结以及优缺点概述

极市平台