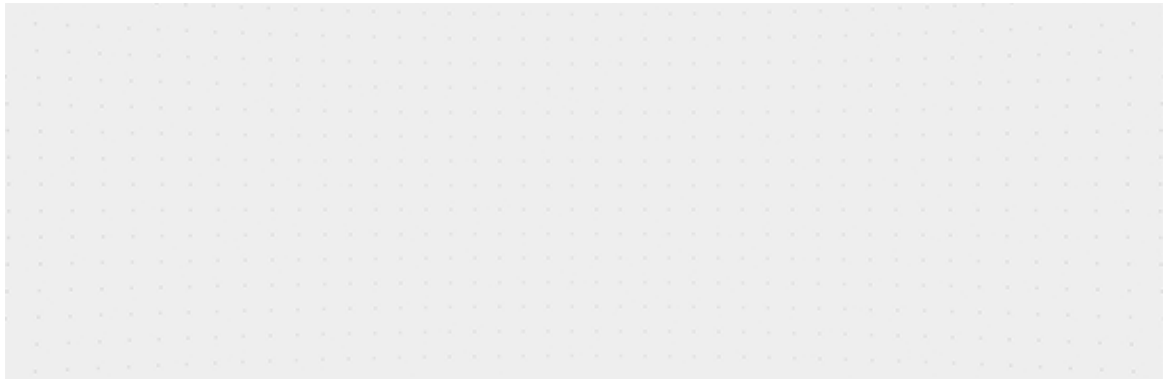


Pytorch翻车记录：单卡改多卡踩坑记！

CV开发者都爱看的 极市平台 2022-12-20 22:00:22 发表于广东 手机阅读 𠄎

↑ 点击蓝字 关注极市平台



作者 | 哟林小平

来源 | 夕小瑶的卖萌屋

编辑 | 极市平台

极市导读

本文记录了作者尝试单卡改多卡加速的过程中出现的bug记录：一是继承DistributedSampler的漏洞百出，二是master进程无法正常结束，作者详细的阐述了出错的细节以及给出了修改的方法。>>加入极市CV技术交流群，走在计算机视觉的最前沿

先说明一下背景，目前正在魔改以下这篇论文的代码：

<https://github.com/QipengGuo/GraphWriter-DGLgithub.com>

由于每次完成实验需要5个小时（baseline），自己的模型需要更久（2倍），非常不利于调参和发现问题，所以开始尝试使用多卡加速。

`torch.nn.DataParallel` ==> 简称 DP

`torch.nn.parallel.DistributedDataParallel` ==> 简称DDP

一开始采用dp试图加速，结果因为dgl的实现（每个batch的点都会打包进一个batch，从而不可分割），而`torch.nn.DataParallel`的实现是把一个batch切分成更小，再加上他的加速性能也不如ddp，所以我开始尝试魔改成ddp。

另外，作者在实现Sampler的时候是继承了`torch.utils.data.Sampler`这个类的，目的在于a

genda数据集的文本长度严重不均衡，如下：

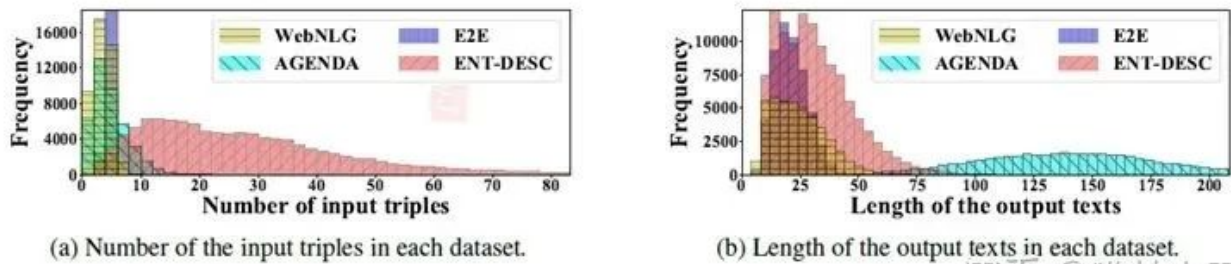


Figure 2: Dataset comparison among WebNLG, AGENDA, E2E and our ENT-DESC.

为了让模型更快train完，把长度相近的文本打包成一个batch（温馨提醒，torchtext也有相关的类 `bucketiterator`[1]，大概形式如下：

```
class BucketSampler(torch.utils.data.Sampler):
    def __init__(self, data_source, batch_size=32):
        self.data_source = data_source
        self.batch_size = batch_size

    def __iter__(self):
        idxs, lens, batch, middle_batch_size, long_batch_size = basesampler
        for idx in idxs:
            batch.append(idx)
            mlen = max([0]+[lens[x] for x in batch])
            #if (mlen<100 and len(batch) == 32) or (mlen>100 and mlen<220
            if (mlen<100 and len(batch) == self.batch_size) or (mlen>100 a
                yield batch
                batch = []
        if len(batch) > 0:
            yield batch

    def __len__(self):
        return (len(self.data_source)+self.batch_size-1)//self.batch_size
```

这是背景。

写bug第一步：继承DistributedSampler的漏洞百出

我一开始理想当然的把作者的sampler源码ctrl-cv下来，唯独只改动了这里：

```
class DDPBaseBucketSampler(torch.utils.data.distributed.DistributedSampler):
```

随后就发现了几个问题：

- dataloader不会发包；

- dataloader给每个进程发的是完整的数据，按武德来说，应该是1/n的数据，n为你设置的gpu数量；

然后我就开始看起了源码[2]，很快啊：

```
def __iter__(self) -> Iterator[T_co]:
    if self.shuffle:
        # deterministically shuffle based on epoch and seed
        g = torch.Generator()
        g.manual_seed(self.seed + self.epoch)
        indices = torch.randperm(len(self.dataset), generator=g).tolist()
    else:
        indices = list(range(len(self.dataset))) # type: ignore

    if not self.drop_last:
        # add extra samples to make it evenly divisible
        padding_size = self.total_size - len(indices)
        if padding_size <= len(indices):
            indices += indices[:padding_size]
        else:
            indices += (indices * math.ceil(padding_size / len(indices)
    else:
        # remove tail of data to make it evenly divisible.
        indices = indices[:self.total_size]
    assert len(indices) == self.total_size

    # subsample
    indices = indices[self.rank:self.total_size:self.num_replicas] # 这
    assert len(indices) == self.num_samples

    return iter(indices)
```

这里最关键的问题是是什么呢？首先在torch.utils.data.distributed.DistributedSampler里面，数据集的变量叫self.dataset而不是data_source；其次和torch.utils.data.Sampler要求你_重写__iter__函数不同：

```
def __iter__(self) -> Iterator[T_co]:
    raise NotImplementedError
```

DistributedSampler这个父类里有部分实现，如果你没有考虑到这部分，就自然会出现每个进程拿到的数据都是all的情况。

于是我重写了我的DDPBaseBucketSampler类：

```
def basesampler(lens, indices, batch_size):
    # the magic number comes from the author's code
```

```

t1 = []
t2 = []
t3 = []
for i, l in enumerate(lens):
    if (l<100):
        t1.append(indices[i])
    elif (l>100 and l<220):
        t2.append(indices[i])
    else:
        t3.append(indices[i])
datas = [t1,t2,t3]
random.shuffle(datas)
idxs = sum(datas, [])
batch = []

#为了保证不爆卡，我们给不同长度的数据上保护锁
middle_batch_size = min(int(batch_size * 0.75) , 32)
long_batch_size = min(int(batch_size * 0.5) , 24)

return idxs, batch, middle_batch_size, long_batch_size

class DDPBaseBucketSampler(torch.utils.data.distributed.DistributedSampler):
    '''
    这里要注意和单GPU的sampler类同步
    '''
    def __init__(self, dataset, num_replicas, rank, shuffle=True, batch_size=batch_size):
        super(DDPBaseBucketSampler, self).__init__(dataset, num_replicas, rank, shuffle=shuffle, batch_size=batch_size)

    def __iter__(self):
        # deterministically shuffle based on epoch
        g = torch.Generator()
        g.manual_seed(self.epoch)
        #print('here is pytorch code and you can delete it in the /home/lz')
        if self.shuffle:
            indices = torch.randperm(len(self.dataset), generator=g).tolist()
        else:
            indices = list(range(len(self.dataset)))
        # add extra samples to make it evenly divisible
        indices += indices[::(self.total_size - len(indices))]
        assert len(indices) == self.total_size

        indices = indices[self.rank:self.total_size:self.num_replicas]
        assert len(indices) == self.num_samples

        # 然后我也要拿到每个数据的长度（每个rank不同）
        lens = torch.Tensor([len(x) for x in self.dataset])

        idxs, batch, middle_batch_size, long_batch_size = basesampler(lens[
        for idx in idxs:
            batch.append(idx)
            mlen = max([0]+[lens[x] for x in batch])
            #if (mlen<100 and len(batch) == 32) or (mlen>100 and mlen<220

```

```

        if (mlen<100 and len(batch) == self.batch_size) or (mlen>100 a
            yield batch
            batch = []
        # print('应该出现2次如果是2个进程的话')
        if len(batch) > 0:
            yield batch

def __len__(self):
    return (len(self.dataset)+self.batch_size-1)//self.batch_size

```

后面每个进程终于可以跑属于自己的数据了（1/n，n=进程数量=GPU数量，单机）

紧接着问题又来了，我发现训练过程正常结束后，主进程无法退出mp.spawn()函数。

写bug第二步，master进程无法正常结束

number workers ddp pytorch下无法正常结束。具体表现为，mp.spawn传递的函数参数可以顺利运行完，但是master进程一直占着卡，不退出。一开始我怀疑是sampler函数的分发batch的机制导致的，什么意思呢？就是由于每个进程拿到的数据不一样，各自进程执行sampler类的时候，由于我规定了长度接近的文本打包在一起，所以可能master进程有一百个iter，slave只有80个，然后我马上试了一下，很快啊：

```

debug_count_iter = 0

for idx in idxs:
    batch.append(idx)
    mlen = max([0]+[lens[x] for x in batch])
    #if (mlen<100 and len(batch) == 32) or (mlen>100 and mlen<2
    if (mlen<100 and len(batch) == self.batch_size) or (mlen>100
        yield batch
        batch = []
        debug_count_iter += 1
print('应该出现2次如果是2个进程的话：', debug_count_iter)
if len(batch) > 0:
    yield batch

```

知乎 @哟林小平

▲DDPBucketSampler(torch.utils.data.distributed.DistributedSampler)类迭代函数__iter__

```

iter: 33 rank: 0
iter: 34 rank: 0
iter: 34 rank: 1
iter: 35 rank: 1
iter: 35 rank: 0
iter: 36 rank: 1
iter: 36 rank: 0
应该出现2次如果是2个进程的话 : 77
iter: 37 rank: 0
iter: 37 rank: 1
应该出现2次如果是2个进程的话 : 78
iter: 38 rank: 0
iter: 38 rank: 1
iter: 39 rank: 0
iter: 39 rank: 1
iter: 40 rank: 0
iter: 40 rank: 1
iter: 41 rank: 0
iter: 41 rank: 1
iter: 42 rank: 1

```

知乎 @哟林小平

▲都能够正常打印，证明__iter__函数没有问题

发现只有细微的差别，并且，程序最后都越过了这些print，应该不会是batch数量不一致导致的问题。（顺便指的一提的是，sampler在很早的时候就把batch打包好了）

加了摧毁进程，也于事无补

```

if args.is_ddp:
    dist.destroy_process_group()
    print('rank destroy_process_group: ', rank)

```

然后只能点击强制退出

```

File "train.py", line 322, in <module>
    main(args.gpu, args)
File "/home/lzk/anaconda3/lib/python3.7/site-packages/torch/multiprocessing
    while not spawn_context.join():
File "/home/lzk/anaconda3/lib/python3.7/site-packages/torch/multiprocessing
    timeout=timeout,
File "/home/lzk/anaconda3/lib/python3.7/multiprocessing/connection.py", li
    ready = selector.select(timeout)

```

```

File "/home/lzk/anaconda3/lib/python3.7/selectors.py", line 415, in select
    fd_event_list = self._selector.poll(timeout)
TypeError: keyboard_interrupt_handler() takes 1 positional argument but 2 were
^CError in atexit._run_exitfuncs:
Traceback (most recent call last):
  File "/home/lzk/anaconda3/lib/python3.7/multiprocessing/popen_fork.py", line
    pid, sts = os.waitpid(self.pid, flag)
TypeError: keyboard_interrupt_handler() takes 1 positional argument but 2 were

```

代码参考：基于Python初探Linux下的僵尸进程和孤儿进程(三)[3]、 Multiprocessing in python blocked[4]

很显然是pytorch master进程产生死锁了，变成了僵尸进程。

再探究，发现当我把dataloader的number workers设为0的时候，程序可以正常结束。经过我的注释大法后我发现，哪怕我把for _i, batch in enumerate(dataloader)内的代码全部注释改为pass，程序还是会出现master无法正常结束的情况。所以问题锁定在dataloader身上。参考：nero：PyTorch DataLoader初探[5]

另外一种想法是，**mp.spawn**出现了问题。使用此方式启动的进程，只会执行和 target 参数或者 run() 方法相关的代码。Windows 平台只能使用此方法，事实上该平台默认使用的也是该启动方式。相比其他两种方式，此方式启动进程的效率最低。参考：Python设置进程启动的3种方式[6]

现在试一下，绕开mp.spawn函数，用shell脚本实现ddp，能不能不报错：

```
python -m torch.distributed.launch --nproc_per_node=2 --nnodes=1 --node_rank
```

参数解释：

- nnodes：因为是单机多卡，所以设为1，显然node_rank 只能是0了
- local_rank:进程在运行的时候，会利用args插入local_rank这个参数标识进程序号

一番改动后，发现问题有所好转，最直观的感受是速度快了非常多！！现在我没有父进程的问题了，但还是在运行完所有的程序后，无法正常结束：

Processes:					GPU Memory Usage
GPU	PID	Type	Process name		
0	5772	C	python		9971MiB
1	36170	C	python		9971MiB
2	32332	C	python		9971MiB
3	34253	C	python		9971MiB
5	30175	C	python		9969MiB
6	30561	C	/home/lzk/anaconda3/bin/python3.7		1445MiB

知乎 @哟林小平

此时我的代码运行到：

```

if args.local_rank == 0 :
    valid_loss = eval_it(model, valid_dataloader, args, epoch, writer)
    #writer.add_scalars('epoch/loss', {'train': train_loss, 'valid': valid_loss}, epoch)
    print('主进程完成记录啦')
    writer.close()
    ddp_barrier(args, 'epoch')
if args.is_ddp:
    dist.destroy_process_group()
    print('args.local_rank destroy_process_group: ', args.local_rank)

if __name__ == '__main__':
    total_time = time.time()
    args = get_args()

    try :
        main(args)
        sendmail_to_my_dear_lzk(args, '恭喜恭喜, 项目总计时:' + str((time.time() - total_time)/60/60), '坤哥, 我
        print('坤哥, 我终于跑完啦, 恭喜恭喜, 项目总计时', (time.time() - total_time)/60/60)
    except Exception as e:
        exc_type, exc_value, exc_traceback = sys.exc_info()
        error = ''.join(traceback.format_exception(exc_type, exc_value, exc_traceback))
        sendmail_to_my_dear_lzk(args, error, '坤哥, 你代码有bug。' + args.save_model)
        print(error)

```

知乎 @哟林小平

上面的代码是main函数，2个进程（master，salve）都可以越过barrier，其中slave顺利结束，但是master却迟迟不见踪影：

```

generative model # xed generative model # xed generative model
generative model
Saving best model Ep 0 loss 8.031548652648926
主进程完成记录啦
rank epoch pre barrier : 0
rank epoch after barrier : 0
rank epoch after barrier : 1
args.local_rank destroy_process_group: 1
坤哥, 我终于跑完啦, 恭喜恭喜, 项目总计时 0.003266143335236443

```

知乎 @哟林小平

这个时候ctrl+c终止，发现：


```

^CTraceback (most recent call last):
  File "/home/lzk/anaconda3/lib/python3.7/runpy.py", line 193, in _run_module_as_main
    "__main__", mod_spec)
  File "/home/lzk/anaconda3/lib/python3.7/runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "/home/lzk/anaconda3/lib/python3.7/site-packages/torch/distributed/launch.py", line 2
46, in <module>
    main()
  File "/home/lzk/anaconda3/lib/python3.7/site-packages/torch/distributed/launch.py", line 2
39, in main
    process.wait()
  File "/home/lzk/anaconda3/lib/python3.7/subprocess.py", line 971, in wait
    return self._wait(timeout=timeout)
  File "/home/lzk/anaconda3/lib/python3.7/subprocess.py", line 1601, in _wait
    (pid, sts) = self._try_wait(0)
  File "/home/lzk/anaconda3/lib/python3.7/subprocess.py", line 1559, in _try_wait
    (pid, sts) = os.waitpid(self.pid, wait_flags)
KeyboardInterrupt
SHELL END

```

知乎 @哟林小平

顺着报错路径去torch/distributed/launch.py, line 239找代码：

```

def main():
    args = parse_args()

    # world size in terms of number of processes
    dist_world_size = args.nproc_per_node * args.nnodes

    # set PyTorch distributed related environmental variables
    current_env = os.environ.copy()
    current_env["MASTER_ADDR"] = args.master_addr
    current_env["MASTER_PORT"] = str(args.master_port)
    current_env["WORLD_SIZE"] = str(dist_world_size)

    processes = []

    if 'OMP_NUM_THREADS' not in os.environ and args.nproc_per_node > 1:
        current_env["OMP_NUM_THREADS"] = str(1)
        print("*****\n"
              "Setting OMP_NUM_THREADS environment variable for each proce
              "to be {} in default, to avoid your system being overloaded,
              "please further tune the variable for optimal performance in
              "your application as needed. \n"
              "*****".format(current_er

    for local_rank in range(0, args.nproc_per_node):
        # each process's rank
        dist_rank = args.nproc_per_node * args.node_rank + local_rank
        current_env["RANK"] = str(dist_rank)
        current_env["LOCAL_RANK"] = str(local_rank)

        # spawn the processes
        if args.use_env:
            cmd = [sys.executable, "-u",
                  args.training_script] + args.training_script_args
        else:
            cmd = [sys.executable,

```

```

        "-u",
        args.training_script,
        "--local_rank={}".format(local_rank)] + args.training_s

    process = subprocess.Popen(cmd, env=current_env)
    processes.append(process)

for process in processes:
    process.wait() # 等待运行结束
    if process.returncode != 0:
        raise subprocess.CalledProcessError(returncode=process.returncode,
                                              cmd=cmd)

```

可恶，master和dataloader到底有什么关系哇。。

这个问题终于在昨天（2020/12/22）被解决了，说来也好笑，左手是graphwriter的ddp实现，无法正常退出，右手是minst的ddp最小例程，可以正常退出，于是我开始了删减大法。替换了数据集，model，然后让dataloader空转，都没有发现问题，最后一步步逼近，知道我把自己的代码这一行注释掉以后，终于可以正常结束了：

```

def main(args):
    #####
    print('local_rank : ' , args.local_rank )
    if args.is_ddp:
        dist.init_process_group(
            backend='nccl',
            init_method='env://',
            world_size=args.world_size,
            rank=args.local_rank
        )
    #####
    # torch.multiprocessing.set_sharing_strategy('file_system') 万恶之源

    os.environ["CUDA_VISIBLE_DEVICES"] = os.environ["CUDA_VISIBLE_DEVICES"].
    args.device = torch.device(0)
    ...

```

为什么我当时会加上这句话呢？因为当时在调试number worker的时候（当时年轻，以为越大越好，所以设置成了number workers = cpu.count()），发现系统报错，说超出了打开文件的最大数量限制。在torch.multiprocessing的设定里，共享策略（参考pytorch中文文档[7]）默认是File descriptor，此策略将使用文件描述符作为共享内存句柄。当存储被移动到共享内存中，一个由 `shm_open` 获得的文件描述符被缓存。当时，文档还提到：

如果你的系统对打开的文件描述符数量有限制，并且无法提高，你应该使用 `file_system` 策略。

所以我换成了`torch.multiprocessing.set_sharing_strategy('file_system')`，但是却忽略文档里的共享内存泄露警告。显然，或许这不是严重的问题，文档里提到：

为了记录共享内存文件泄露数量，`torch.multiprocessing` 将产生一个守护进程叫做 `torch_shm_manager`

将自己与当前进程组隔离，并且将跟踪所有共享内存分配。一旦连接到它的所有进程退出，

它将等待一会儿，以确保不会有新的连接，并且将遍历该组分配的所有共享内存文件。@哟林小平

也有可能我所说的master进程就是这个`torch_shm_manager`，因为destory进程组始终无法结束0号进程：

```
generative model # xed generative model # xed generative model
generative model
Saving best model Ep 0 loss 8.031548652648926
主进程完成记录啦
rank epoch pre barrier : 0
rank epoch after barrier : 0
rank epoch after barrier : 1
args.local_rank destroy_process_group: 1
坤哥，我终于跑完啦，恭喜恭喜，项目总计时 0.003266143335236443

```

这个BUG结束了，真开心，期待下一个BUG快快到来。



极市平台

12月21日 20:00 直播

已结束

NeurIPS 2022-张博航：如何从模型层面获得对抗鲁棒性保证？

视频号

极市干货

技术干货：数据可视化必须注意的30个小技巧总结 | 如何高效实现矩阵乘？万文长字带你从CUDA初学者的角度入门

实操教程：Nvidia Jetson TX2使用TensorRT部署yolov5s模型 | 基于YOLOV5的数据集标注 & 训练，Windows/Linux/Jetson Nano多平台部署全流程



极市原创作者激励计划

极市平台深耕CV开发者领域近5年，拥有一大批优质CV开发者受众，覆盖微信、知乎、B站、微博等多个渠道。通过极市平台，您的文章的观点和看法能分享至更多CV开发者，既能体现文章的价值，又能让文章在视觉圈内得到更大程度上的推广，并且极市还将给予优质的作者可观的稿酬！

我们欢迎领域内的各位来进行投稿或者是宣传自己/团队的工作，让知识成为最为流通的干货！

对于优质内容开发者，极市可推荐至国内优秀出版社合作出书，同时为开发者引荐行业大牛，组织个人分享交流会，推荐名企就业机会等。

投稿须知：

- 1.作者保证投稿作品为自己的原创作品。
- 2.极市平台尊重原作者署名权，并支付相应稿费。文章发布后，版权仍属于原作者。
- 3.原作者可以将文章发在其他平台的个人账号，但需要在文章顶部标明首发于极市平台

投稿方式：

添加小编微信Fengcall（微信号：fengcall19），备注：**姓名-投稿**

△长按添加极市平台小编

[点击阅读原文进入CV社区](#)

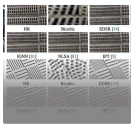
[获取更多技术干货](#)

[阅读原文](#)

喜欢此内容的人还喜欢

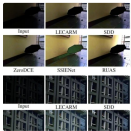
ICCV 2023 | 南开程明明团队提出适用于SR任务的新颖注意力机制（已开源）

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



YOLOv5帮助母猪产仔？南京农业大学研发母猪产仔检测模型并部署到Jetson Nano开发板

极市平台

