# 图解分布式训练（四） —— torch.multiprocessing 详细解析

来自： AiGC面试宝典

宁静致远

2023年09月29日 11:27



**一、torch.multiprocessing 函数介绍一下?**

torch.multiprocessing是本机multiprocessing模块的封装。封装了multiprocessing模块。它注册自定义的reducer，它使用共享内存为不同进程中的相同数据提供视图共享。一旦张量/存储被移动到shared_memory（参见sharememory()），就可以将其发送到其他进程而不进行其它任何操作。

这个API与原始模块100％兼容，将import multiprocessing改为import torch.multiprocessing就可以将所有张量通过队列发送或通过其他机制共享转移到共享内存中

由于API的相似性，我们没有记录这个软件包的大部分内容，我们建议您参考Python multiprocessing原始模块的文档。

警告： 如果主进程突然退出（例如因为传入的信号），Python multiprocessing有时无法清理其子进程。这是一个已知的警告，所以如果您在中断解释器后看到任何资源泄漏，这可能意味着这刚刚发生在您身上。

**二、torch.multiprocessing 函数如何使用?**

```
torch.multiprocessing.get_all_sharing_strategies()
```

返回一组当前系统支持的共享策略。

```
torch.multiprocessing.get_sharing_strategy()
```

返回共享CPU张量的当前策略

```
torch.multiprocessing.set_sharing_strategy(new_strategy)
```

设置共享CPU张量的策略

参数: new_strategy(str)- 所选策略的名称。应当是上面get_all_sharing_strategies()中系统支持的共享策略之一。

**三、介绍一下 共享CUDA张量?**

只支持在Python 3中使用spawn或forkserver启动方法才支持在进程之间共享CUDA张量。

Python2中的multiprocessing只能使用fork创建子进程，并且不支持CUDA运行时。

警告： CUDA API要求导出到其他进程的分配一直保持有效，只要它们被使用。您应该小心，确保您共享的CUDA张力不要超出范围，只要有必要。这不应该是共享模型参数的问题，但传递其他类型的数据应该小心。请注意，此限制不适用于共享CPU内存。

**四、介绍一下 共享策略?**

本节简要概述了不同的共享策略如何工作。请注意，它只适用于CPU张量 - CUDA张量将始终使用CUDA API，因为它们是唯一的共享方式。

**4.1 介绍一下文件描述符 -file_descripor?**

注意： 这是默认策略（除了不支持的MacOS和OS X）。

此策略将使用文件描述符作为共享内存句柄。当存储器被移动到共享存储器时，每当存储器被移动到共享内存中时，从shm_open对象获取的文件描述符被缓存，并且当它被发送到其他进程时，文件描述符也将被传送（例如通过UNIX套接字）。接收器还将缓存文件描述符并且mmap它，以获得对存储数据的共享视图。

请注意，如果要共享很多张量，则此策略将保留大量文件描述符需要很多时间才能打开。如果您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。

**4.2 文件系统 -file_system**

该策略将使用给定的文件名shm_open来标识共享内存区域。这具有不需要实现缓存从其获得的文件描述符的优点，但是同时容易发生共享内存泄漏。该文件创建后不能被删除，因为其他进程需要访问它以打开其视图。如果进程死机或死机，并且不调用存储析构函数，则文件将保留在系统中。这是非常严重的，因为它们在系统重新启动之前不断使用内存，或者手动释放它们。

为了解决共享内存文件泄漏的问题，torch.multiprocessing将产生一个守护程序torch_shm_manager，它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。如果发现它们中的任何一个仍然存在，就释放掉它。我们已经测试了这种方法，并且证明对于各种故障是稳健的。不过，如果您的系统具有足够的限制，并且支持file_descriptor策略，我们不建议切换到该策略。

**五、torch.multiprocessing 函数使用**

下面展示一个采用多进程训练模型的例子：

```
# Training a model using multiple processes:
import torch.multiprocessing as mp
def train(model):
    for data, labels in data_loader:
        optimizer.zero_grad()
        loss_fn(model(data), labels).backward()
        optimizer.step()  # This will update the shared parameters

model = nn.Sequential(
    nn.Linear(n_in, n_h1),
    nn.ReLU(),
    nn.Linear(n_h1, n_out)
)
model.share_memory() # Required for 'fork' method to work
processes = []
for i in range(4): # No. of processes
    p = mp.Process(target=train, args=(model,))
    p.start()
    processes.append(p)
for p in processes:
    p.join()
```