

## 多进程最佳实践

`torch.multiprocessing` 是Python `multiprocessing` 的替代品。它支持完全相同的操作，但扩展了它以便通过 `multiprocessing.Queue` 发送的所有张量将其数据移动到共享内存中，并且只会向其他进程发送一个句柄。

### Note

当 `Variable` 发送到另一个进程时，`Variable.data` 和 `Variable.grad.data` 都将被共享。

这允许实现各种训练方法，如Hogwild，A3C或需要异步操作的任何其他方法。

## 共享CUDA张量

仅在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。Python 2中的 `multiprocessing` 只能使用 `fork` 创建子进程，并且不被CUDA运行时所支持。

### Warning


CUDA API要求导出到其他进程的分配，只要它们被使用就要一直保持有效。您应该小心，确保您共享的CUDA张量只要有必要就不要超出范围。这不是共享模型参数的问题，但传递其他类型的数据应该小心。注意，此限制不适用于共享CPU内存。

参考：使用 `nn.DataParallel` 替代 `multiprocessing`

## 最佳实践和提示

### 避免和抵制死锁

当一个新进程被产生时，有很多事情可能会出错，最常见的死锁原因是后台线程。如果有任何线程持有锁或导入模块，并且 `fork` 被调用，则子进程很可能处于损坏的状态，并以不同的方式死锁或失败。注意，即使您没有，Python内置的库也可能会这样做——不需要看得比 `multiprocessing` 更远。`multiprocessing.Queue` 实际上是一个非常复杂的类，它产生用于序列化，发送和接收对象的多个线程，它们也可能引起上述问题。如果您发现自己处于这种情况，请尝试使用 `multiprocessing.queue.SimpleQueue`，这不会使用任何其他线程。

我们正在竭尽全力把它设计得更简单，并确保这些死锁不会发生，但有些事情无法控制  [v: latest](#) ▼  
如果有任何问题您无法一时无法解决，请尝试在论坛上提出，我们将看看是否可以解决问题。

## 重用经过队列的缓冲区

记住每次将 `Tensor` 放入 `multiprocessing.Queue` 时，必须将其移动到共享内存中。如果它已经被共享，它是一个无效的操作，否则会产生一个额外的内存副本，这会减缓整个进程。即使你有一个进程池来发送数据到一个进程，使它返回缓冲区——这几乎是免费的，并且允许你在发送下一个batch时避免产生副本。

## 异步多进程训练（例如Hogwild）

使用 `torch.multiprocessing`，可以异步地训练模型，参数可以一直共享，也可以定期同步。在第一种情况下，我们建议发送整个模型对象，而在后者中，我们建议只发送 `state_dict()`。

我们建议使用 `multiprocessing.Queue` 来在进程之间传递各种PyTorch对象。例如，当使用fork启动方法时，可能会继承共享内存中的张量和存储器，但这是非常容易出错的，应谨慎使用，而且只能由高级用户使用。队列虽然有时是一个较不优雅的解决方案，但基本上能在所有情况下正常工作。

**Warning** 你应该注意有关全局语句，它们没有被 `if __name__ == '__main__':` 保护。如果使用与 `fork` 不同的启动方法，则它们将在所有子进程中执行。

## Hogwild

在 [examples repository](#) 中可以找到具体的Hogwild实现，可以展示代码的整体结构。下面也有一个小例子：

```
import torch.multiprocessing as mp
from model import MyModel

def train(model):
    # Construct data_loader, optimizer, etc.
    for data, labels in data_loader:
        optimizer.zero_grad()
        loss_fn(model(data), labels).backward()
        optimizer.step() # This will update the shared parameters

if __name__ == '__main__':
    num_processes = 4
    model = MyModel()
    # NOTE: this is required for the ``fork`` method to work
    model.share_memory()
    processes = []
    for rank in range(num_processes):
        p = mp.Process(target=train, args=(model,))
        p.start()
        processes.append(p)
    for p in processes:
        p.join()
```