8.4 多GPU计算

> **注：相对于本章的前面几节，我们实际中更可能遇到本节所讨论的情况：多GPU计算。原书将MXNet的多GPU计算分成了8.4和8.5两节，但我们将关于PyTorch的多GPU计算统一放在本节讨论。 需要注意的是，这里我们谈论的是单主机多GPU计算而不是分布式计算。如果对分布式计算感兴趣可以参考PyTorch官方文档。**

本节中我们将展示如何使用多块GPU计算，例如，使用多块GPU训练同一个模型。正如所期望的那样，运行本节中的程序需要至少2块GPU。事实上，一台机器上安装多块GPU很常见，这是因为主板上通常会有多个PCIe插槽。如果正确安装了NVIDIA驱动，我们可以通过在命令行输入 `nvidia-smi` 命令来查看当前计算机上的全部GPU（或者在jupyter notebook中运行 `!nvidia-smi` ）。

```
nvidia-smi
```

输出：

```
Wed May 15 23:12:38 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 390.48                 Driver Version: 390.48                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  TITAN X (Pascal)    Off  | 00000000:02:00.0 Off |                  N/A |
| 46%   76C    P2    87W / 250W |  10995MiB / 12196MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  TITAN X (Pascal)    Off  | 00000000:04:00.0 Off |                  N/A |
| 53%   84C    P2   143W / 250W |  11671MiB / 12196MiB |      4%      Default |
+-------------------------------+----------------------+----------------------+
|   2  TITAN X (Pascal)    Off  | 00000000:83:00.0 Off |                  N/A |
| 62%   87C    P2   190W / 250W |  12096MiB / 12196MiB |    100%      Default |
+-------------------------------+----------------------+----------------------+
|   3  TITAN X (Pascal)    Off  | 00000000:84:00.0 Off |                  N/A |
| 51%   83C    P2   255W / 250W |   8144MiB / 12196MiB |     58%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
```

```
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     44683      C   python                                    3289MiB |
|    0    155760      C   python                                    4345MiB |
|    0    158310      C   python                                    2297MiB |
|    0    172338      C   /home/yzs/anaconda3/bin/python            1031MiB |
|    1    139985      C   python                                   11653MiB |
|    2     38630      C   python                                    5547MiB |
|    2     43127      C   python                                    5791MiB |
|    2    156710      C   python3                                    725MiB |
|    3     14444      C   python3                                   1891MiB |
|    3     43407      C   python                                    5841MiB |
|    3     88478      C   /home/tangss/.conda/envs/py36/bin/python     379MiB |
+-----------------------------------------------------------------------------+
```

从上面的输出可以看到一共有四块TITAN X GPU，每一块总共有约12个G的显存，此时每块的显存都占得差不多了.....此外还可以看到GPU利用率、运行的所有程序等信息。

Pytorch在0.4.0及以后的版本中已经提供了多GPU训练的方式，本文用一个简单的例子讲解下使用Pytorch多GPU训练的方式以及一些注意的地方。

8.4.1 多GPU计算

先定义一个模型：

```
import torch
net = torch.nn.Linear(10, 1).cuda()
net
```

输出：

```
Linear(in_features=10, out_features=1, bias=True)
```

要想使用PyTorch进行多GPU计算，最简单的方法是直接用 `torch.nn.DataParallel` 将模型wrap一下即可：

```
net = torch.nn.DataParallel(net)
net
```

输出：

```
DataParallel(
  (module): Linear(in_features=10, out_features=1, bias=True)
)
```

这时，默认所有存在的GPU都会被使用。

如果我们机子中有很多GPU(例如上面显示我们有4张显卡，但是只有第0、3块还剩下一点点显存)，但我们只想使用0、3号显卡，那么我们可以用参数 `device_ids` 指定即可：`torch.nn.DataParallel(net, device_ids=[0, 3])` 。

8.4.2 多GPU模型的保存与加载

我们现在来尝试一下按照4.5节（读取和存储）推荐的方式进行一下模型的保存与加载。 保存模型:

```
torch.save(net.state_dict(), "./8.4_model.pt")
```

加载模型前我们一般要先进行一下模型定义，此时的 `new_net` 并没有使用多GPU:

```
new_net = torch.nn.Linear(10, 1)
new_net.load_state_dict(torch.load("./8.4_model.pt"))
```

然后我们发现报错了:

```
RuntimeError: Error(s) in loading state_dict for Linear:
    Missing key(s) in state_dict: "weight", "bias".
    Unexpected key(s) in state_dict: "module.weight", "module.bias".
```

事实上 `DataParallel` 也是一个 `nn.Module` ，只是这个类其中有一个module就是传入的实际模型。因此当我们调用 `DataParallel` 后，模型结构变了（在外面加了一层而已，从8.4.1节两个输出可以对比看出来）。所以直接加载肯定会报错的，因为模型结构对不上。

所以正确的方法是保存的时候只保存 `net.module` ：

```
torch.save(net.module.state_dict(), "./8.4_model.pt")
new_net.load_state_dict(torch.load("./8.4_model.pt")) # 加载成功
```

或者先将 `new_net` 用 `DataParallel` 包括以下再用上面报错的方法进行模型加载:

Copy to clipboard

```
torch.save(net.state_dict(), "./8.4_model.pt")
new_net = torch.nn.Linear(10, 1)
new_net = torch.nn.DataParallel(new_net)
new_net.load_state_dict(torch.load("./8.4_model.pt")) # 加载成功
```

注意这两种方法的区别，推荐用第一种方法，因为可以按照普通的加载方法进行正确加载。