Optional: Data Parallelism

Authors: Sung Kim and Jenny Kang

In this tutorial, we will learn how to use multiple GPUs using `DataParallel`.

It's very easy to use GPUs with PyTorch. You can put the model on a GPU:

```
device = torch.device("cuda:0")
model.to(device)
```

Then, you can copy all your tensors to the GPU:

```
mytensor = my_tensor.to(device)
```

Please note that just calling `mytensor.to(device)` returns a new copy of `mytensor` on GPU instead of rewriting `mytensor`. You need to assign it to a new variable and use that tensor on the GPU.

It's natural to execute your forward, backward propagations on multiple GPUs. However, Pytorch will only use one GPU by default. You can easily run your operations on multiple GPUs by making your model run parallelly using `DataParallel`:

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

That's the core behind this tutorial. We will explore it in more detail below.

Imports and parameters

Import PyTorch modules and define parameters.

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

# Parameters and DataLoaders
input_size = 5
output_size = 2

batch_size = 30
data_size = 100
```

Device

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Dummy DataSet

Make a dummy (random) dataset. You just need to implement the getitem

```
class RandomDataset(Dataset):

    def __init__(self, size, length):
        self.len = length
        self.data = torch.randn(length, size)

    def __getitem__(self, index):
        return self.data[index]

    def __len__(self):
        return self.len

rand_loader = DataLoader(dataset=RandomDataset(input_size, 100),
                         batch_size=batch_size, shuffle=True)
```

Simple Model

For the demo, our model just gets an input, performs a linear operation, and gives an output. However, you can use `DataParallel` on any model (CNN, RNN, Capsule Net etc.)

We've placed a print statement inside the model to monitor the size of input and output tensors. Please pay attention to what is printed at batch rank 0.

```
class Model(nn.Module):
    # Our model

    def __init__(self, input_size, output_size):
        super(Model, self).__init__()
        self.fc = nn.Linear(input_size, output_size)

    def forward(self, input):
        output = self.fc(input)
        print("\tIn Model: input size", input.size(),
              "output size", output.size())

        return output
```

Create Model and DataParallel

This is the core part of the tutorial. First, we need to make a model instance and check if we have multiple GPUs. If we have multiple GPUs, we can wrap our model using `nn.DataParallel`. Then we can put our model on GPUs by `model.to(device)`

```
model = Model(input_size, output_size)
if torch.cuda.device_count() > 1:
  print("Let's use", torch.cuda.device_count(), "GPUs!")
  # dim = 0 [30, xxx] -> [10, ...], [10, ...], [10, ...] on 3 GPUs
  model = nn.DataParallel(model)

model.to(device)
```

Run the Model

Now we can see the sizes of input and output tensors.

```
for data in rand_loader:
    input = data.to(device)
    output = model(input)
    print("Outside: input size", input.size(),
          "output_size", output.size())
```

Out:

```
In Model: input size torch.Size([30, 5]) output size torch.Size([30, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
        In Model: input size torch.Size([30, 5]) output size torch.Size([30, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
        In Model: input size torch.Size([30, 5]) output size torch.Size([30, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
        In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
Outside: input size torch.Size([10, 5]) output_size torch.Size([10, 2])
```

Results

When we batch 30 inputs and 30 outputs, the model gets 30 and outputs 30 as expected. But if you have GPUs, then you can get results like this.

2 GPUs

If you have 2, you will see:

```
# on 2 GPUs
Let's use 2 GPUs!
    In Model: input size torch.Size([15, 5]) output size torch.Size([15, 2])
    In Model: input size torch.Size([15, 5]) output size torch.Size([15, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
    In Model: input size torch.Size([15, 5]) output size torch.Size([15, 2])
    In Model: input size torch.Size([15, 5]) output size torch.Size([15, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
    In Model: input size torch.Size([15, 5]) output size torch.Size([15, 2])
    In Model: input size torch.Size([15, 5]) output size torch.Size([15, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
    In Model: input size torch.Size([5, 5]) output size torch.Size([5, 2])
    In Model: input size torch.Size([5, 5]) output size torch.Size([5, 2])
Outside: input size torch.Size([10, 5]) output_size torch.Size([10, 2])
```

3 GPUs

If you have 3 GPUs, you will see:

```
Let's use 3 GPUs!
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([2, 5]) output size torch.Size([2, 2])
Outside: input size torch.Size([10, 5]) output_size torch.Size([10, 2])
```

8 GPUs

If you have 8, you will see:

```
Let's use 8 GPUs!
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([2, 5]) output size torch.Size([2, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([2, 5]) output size torch.Size([2, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([2, 5]) output size torch.Size([2, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
    In Model: input size torch.Size([2, 5]) output size torch.Size([2, 2])
    In Model: input size torch.Size([2, 5]) output size torch.Size([2, 2])
    In Model: input size torch.Size([2, 5]) output size torch.Size([2, 2])
    In Model: input size torch.Size([2, 5]) output size torch.Size([2, 2])
    In Model: input size torch.Size([2, 5]) output size torch.Size([2, 2])
Outside: input size torch.Size([10, 5]) output_size torch.Size([10, 2])
```

Summary

DataParallel splits your data automatically and sends job orders to multiple models on several GPUs. After each model finishes their job, DataParallel collects and merges the results before returning it to you.

For more information, please check out http://pytorch.org/tutorials/beginner/former_torchies/parallelism_tutorial.html.

Total running time of the script: (0 minutes 0.008 seconds)

Download Python source code: data_parallel_tutorial.py

Download Jupyter notebook: data_parallel_tutorial.ipynb

Gallery generated by Sphinx-Gallery