Data Parallelism

March 6, 2018

1 Data Parallelism

Ih this tutorial, we will learn how to use multiple GPUs using `DataParallel`. To put the model on GPU:

`model.gpu()`

Copy all the tensors to the GPU:

`mytensor = my_tensor.gpu()`

Please note that just calling `mytensor.gpu()` won't copy the tensor to the GPU. You need to assign it to a **new tensor 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)`

```
In [3]: import torch
        import torch.nn as nn
        from torch.autograd import Variable
        from torch.utils.data import Dataset, DataLoader
```

Define parameters

```
In [4]: input_size = 5
        output_size = 2

        batch_size = 30
        data_size = 100
```

Make a random dataset. You just need to implement the `getitem`.

```
In [5]: 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)
```

1.1 Simple Model

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.

```
In [6]: 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("  In Model: input size", input.size(),
                      "output size", output.size())

                return output
```

1.2 Create Model and DataParallel

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.gpu()`.

```
In [9]: model = Model(input_size, output_size)
        print(torch.cuda.device_count())
        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)

        if torch.cuda.is_available():
           model.cuda()
```

```
0
```

1.3 Run the Model

```
In [8]: for data in rand_loader:
            if torch.cuda.is_available():
                input_var = Variable(data.cuda())
```

```
        else:
            input_var = Variable(data)

        output = model(input_var)
        print("Outside: input size", input_var.size(),
              "output_size", output.size())
```

```
  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])
```

```
In [10]: torch.cuda.is_available()
```

```
Out[10]: False
```