Contents

Dear Lisa,

I am working on a PyTorch application (python) that I want to parallelize over (CPU) several nodes. (On CPU, because it has complicated serial computation that GPU offers no benefit for.) Currenlty I have code that works on one node. I want it to work on mutliple nodes. For example: spawn 32 processes on 2 nodes: 16 on one node and 16 on the other, and communicate as if it were just 32 on one node. Can this be done?

This is a pretty long and maybe complicated/confused question. I've tried to write it out clearly, but I could also come by for a short appointment if that is easier?

Note I'm using PyTorch v0.4.0 that I installed myself using pip: `pip install --user torch torchvision` after `module load Python/3.6.3-foss-2017b`.

0.1 Single-node

Currently, I have code that works on one node. It is completely synchronous. It spawns a number of processes, each of these computes gradients on a different example, and these gradients are then communicated and averaged.

My code looks roughly like this, and is adapted from this this PyTorch tutorial. The code is self-sufficient, I hope, and I've teste it on a single Lisa node.

```
#!/usr/bin/env python
"""run.py:"""
import os

import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.distributed as dist
import torch.nn.functional as F
from torch.multiprocessing import Process


def run(rank, size):
    """ Distributed Synchronous SGD Example """
    torch.manual_seed(1234)
    # Restrict the number of threads that each process uses.
    torch.set_num_threads(1)

    # Random objective.
    data, target = Variable(torch.ones(1, 500).uniform_()), Variable(torch.ones(1, 500).uniform_())
    # Dummy model.
    model = nn.Linear(500, 500)  # y = Wx + b

    optimizer = torch.optim.SGD(model.parameters(),
                          lr=0.01, momentum=0.5)

    for epoch in range(100000):
        output = model(data)
        # MSE loss.
        loss = torch.mean((output - target)**2)

        optimizer.zero_grad()
        loss.backward()
        average_gradients(model)
        optimizer.step()
        if rank == 0 and epoch % 100 == 0:
            print(f'Epoch {epoch:,}: {loss.item():.2e}')


def average_gradients(model):
    """ Gradient averaging. """
    size = float(dist.get_world_size())
    for param in model.parameters():
        dist.all_reduce(param.grad.data, op=dist.reduce_op.SUM)  # This does the communication.
        param.grad.data /= size


def init_processes(rank, size, fn, backend='tcp'):
    """ Initialize the distributed environment. """
    os.environ['MASTER_ADDR'] = '127.0.0.1'
    os.environ['MASTER_PORT'] = '29500'
    dist.init_process_group(backend, rank=rank, world_size=size)
    fn(rank, size)


if __name__ == "__main__":
    size = 2
    processes = []
    for rank in range(size):
        p = Process(target=init_processes, args=(rank, size, run))
        p.start()
        processes.append(p)

    for p in processes:
        p.join()
```

There is on side-question I have about this:

1. I figured I should call only 16 processes, given that there are 16 cores on one CPU node. But I also tested with 32 processes, and this makes it almost twice as fast. How should I reason about this? The examples are of varying sizes (they are sentences), so some processes stop much earlier, and are left idle while waiting for the largest example to finish. Maybe this time is then filled with the other processes? How do I know the maximum number of processes to call? Experimentally?
2. Which `backends` are do you advice me to use on Lisa? I am using `tcp`, but I noticed that `mpi` is also supported when I `module load PyTorch/0.3.0-foss-2017b-Python-3.6.3-CUDA-9.0.176`. However, I prefer to use PyTorch 4, in which case mpi is not available, unless I compile from source with extra flags and options, which I'd like to avoid doing.
3. What are `export MASTER_ADDR=127.0.0.1` and `export MASTER_PORT=29500` used for? What values to choose for this? These numbers are totally out of the blue for me. Which values to use on Lisa?

0.2 Multi-node

I want to make the above work on multiple nodes.

I have a feeling that the key to this is in `init_processes`, and especially `dist.init_process_group`, correct? Unfortunately, I don't understand it at all. I'm reading the PyTorch documentation, but there is too much that I am completely ignorant of, like the IP adresses and Port and more. Pytorch also provides a launch utility. There is an example for multinode training in the link, but I have a hard time understanding it:

1. Again, what IP-address and port to use on Lisa?
2. How do I submit code to each of the different nodes, and make them communicate?

0.3 Sources

The following is a list of relevant PyTorch sources relating to this question: * `torch.distributed` package documentation: https://pytorch.org/docs/master/distributed.html * `torch.distributed.init_process_group()` documentation: https://pytorch.org/docs/master/distributed.html#initialization * `torch.distributed` launch utility documentation: https://pytorch.org/docs/master/distributed.html#launch-utility * Distributed implementations tutorial: https://pytorch.org/tutorials/intermediate/dist_tuto.html

I hope my questions are clear and I hope that you can help me with this!

Best, Daan van Stigt, ILLC username: daanvans