Implementing Training Using Single and Multiple Processors

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Overview

Understand options to run training using multiple processes, devices and machines

Use the torch.multiprocessing package

Run multiple processes on the same CPU

Parallelize training across multiple GPUs

Distributed Training in PyTorch

Multiprocessing

Data Parallel

Model Parallel

Distributed Data Parallel

Distributed Training in PyTorch

Multiprocessing

Data Parallel

Model Parallel

Distributed Data Parallel

Multiprocessing

torch.multiprocessing

Wrapper around native Python multiprocessing module

All data handling done by user

Prone to memory leaks

Multiprocessing

Tensors moved to shared memory

Accessible by all processes

CPU tensors shared using:

- file_descriptor
- file_system

Distributed Training in PyTorch

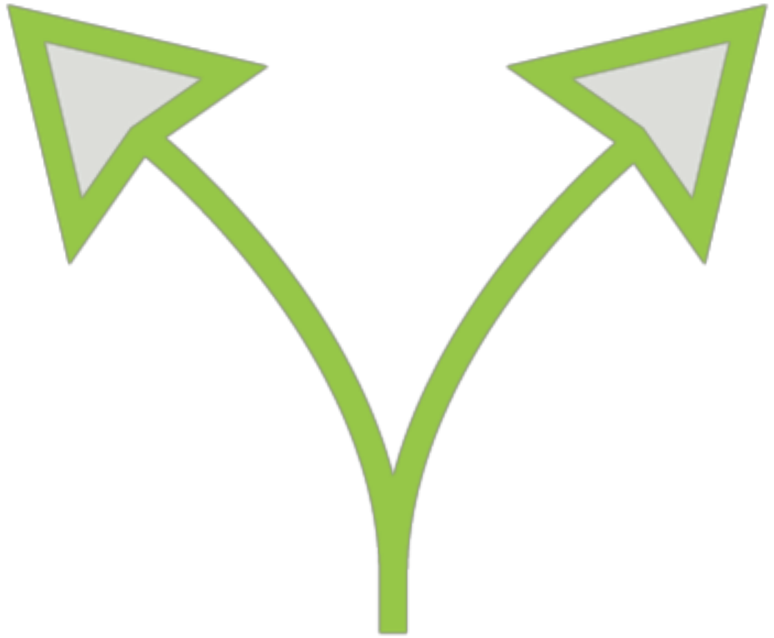
Multiprocessing

Data Parallel

Model Parallel

Distributed Data Parallel

Data Parallel

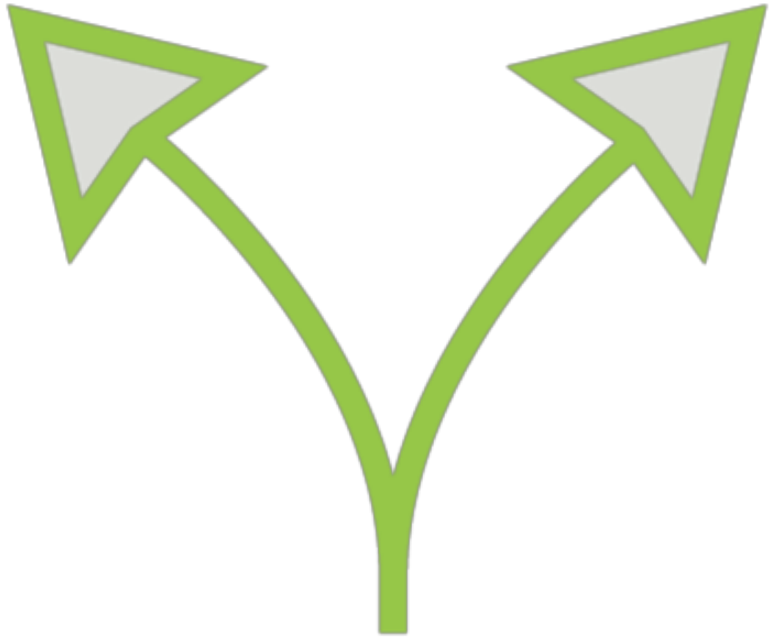


Very easy to place model on GPU

By default PyTorch will use single GPU

To use multiple GPUs, employ nn.DataParallel

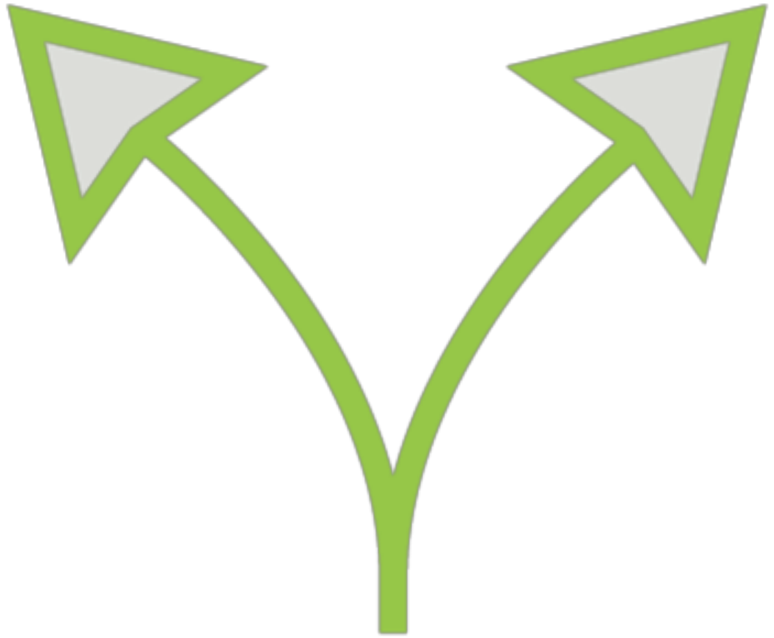
Data Parallel



Replicates same model to all GPUs

Can significantly accelerate training

Data Parallel



Chunks the input along the batch dimension

Each replica of the model handles a subset of data

Mitigates the data handling issues encounter with torch.multiprocessing

Distributed Training in PyTorch

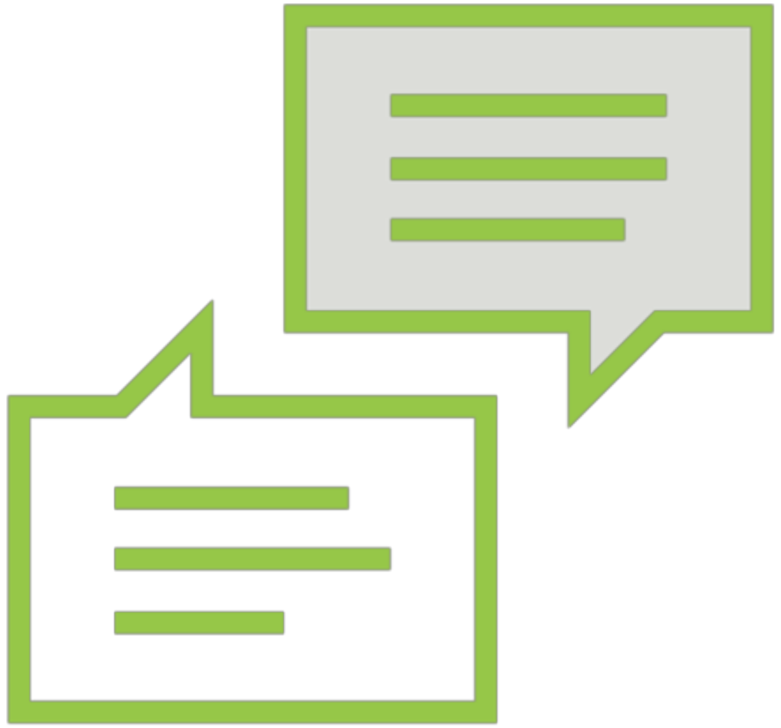
Multiprocessing

Data Parallel

Model Parallel

Distributed Data Parallel

Model Parallel

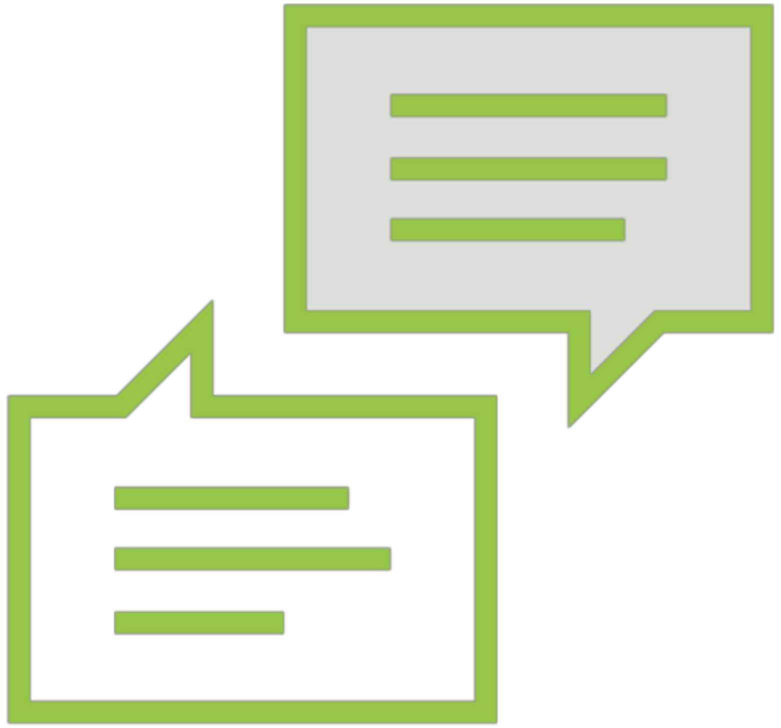


Data Parallel will not work when model too large to fit into single GPU

Use Model Parallel in such cases

Place different sub-networks on different devices

Model Parallel



Only a subset of model operates on an individual device

Many devices collectively used to train a single model

Distributed Training in PyTorch

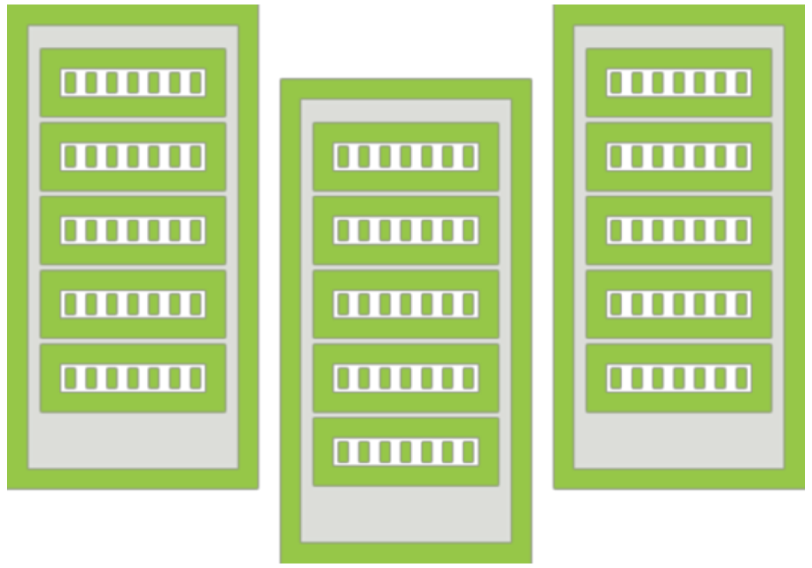
Multiprocessing

Data Parallel

Model Parallel

Distributed Data Parallel

Distributed Data Parallel

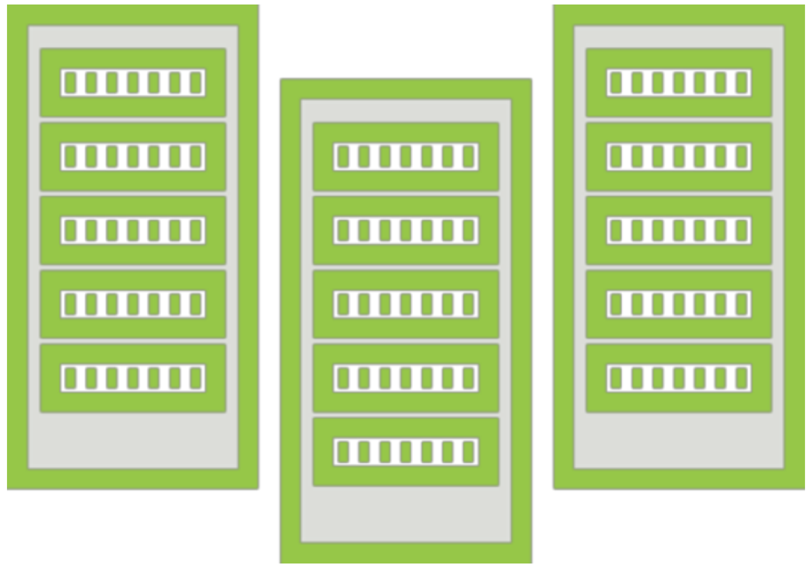


Synchronous distributed training wrapper around PyTorch model

Supports multiple network-connected machines

User must explicitly launch separate copies of training scripts

Distributed Data Parallel



Preferable even for single-machine usage:

- Each process has own optimizer
- No parameter broadcast needed
- Each process has own python interpreter
- Makes training more efficient

Distributed Data Parallel

The *torch.distributed* package provides PyTorch support and communication primitives for multiprocess parallelism across several computation nodes running on one or more machines. The class `torch.nn.parallel.DistributedDataParallel()` builds on this functionality to provide synchronous distributed training as a wrapper around any PyTorch model. This differs from the kinds of parallelism provided by Multiprocessing package - torch.multiprocessing and `torch.nn.DataParallel()` in that it supports multiple network-connected machines and in that the user must explicitly launch a separate copy of the main training script for each process.

In the single-machine synchronous case, *torch.distributed* or the `torch.nn.parallel.DistributedDataParallel()` wrapper may still have advantages over other approaches to data-parallelism, including `torch.nn.DataParallel()`:

- Each process maintains its own optimizer and performs a complete optimization step with each iteration. While this may appear redundant, since the gradients have already been gathered together and averaged across processes and are thus the same for every process, this means that no parameter broadcast step is needed, reducing time spent transferring tensors between nodes.
- Each process contains an independent Python interpreter, eliminating the extra interpreter overhead and “GIL-thrashing” that comes from driving several execution threads, model replicas, or GPUs from a single Python process. This is especially important for models that make heavy use of the Python runtime, including models with recurrent layers or many small components.

Demo

Training using multiple processes with the torch.multiprocessing module

Demo

Running distributed training on multiple GPUs on a virtual machine using torch.nn.DataParallel

Summary

Understand options to run training using multiple processes, devices and machines

Use the torch.multiprocessing package

Run multiple processes on the same CPU

Parallelize training across multiple GPUs