Distributed Training

02457 Machine Learning Operations

Nicki Skafte Detlefsen,

Postdoc

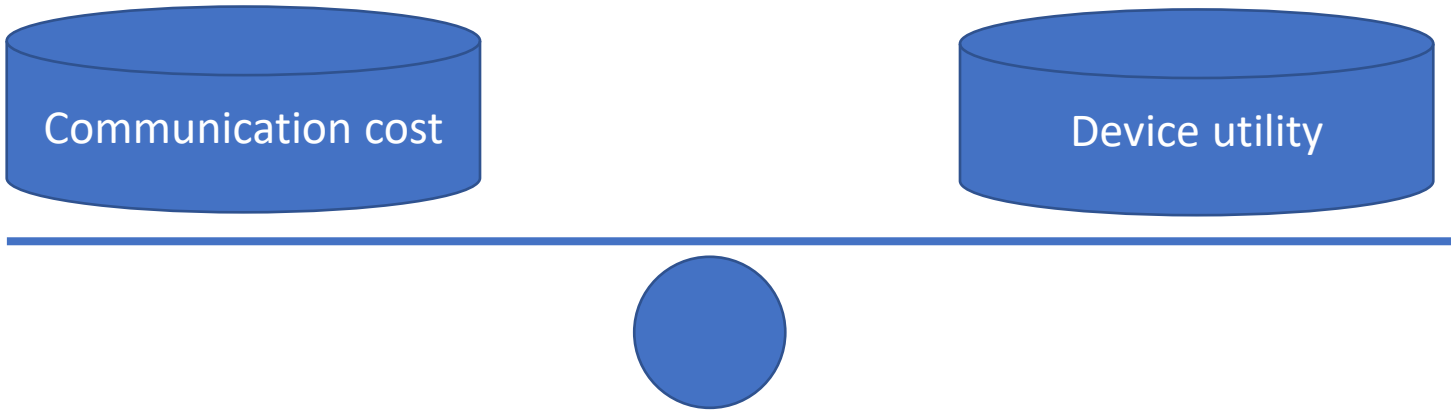
DTU Compute

What is distributed computations?

Computing on multiple threads/devices/nodes in parallel

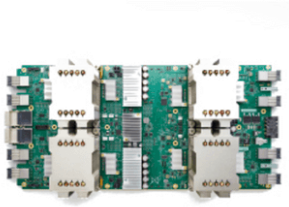
We focus on training as it is the most computationally expensive part but doing testing or inference can also be done in distributed manner

Distributed computing is not always beneficial:

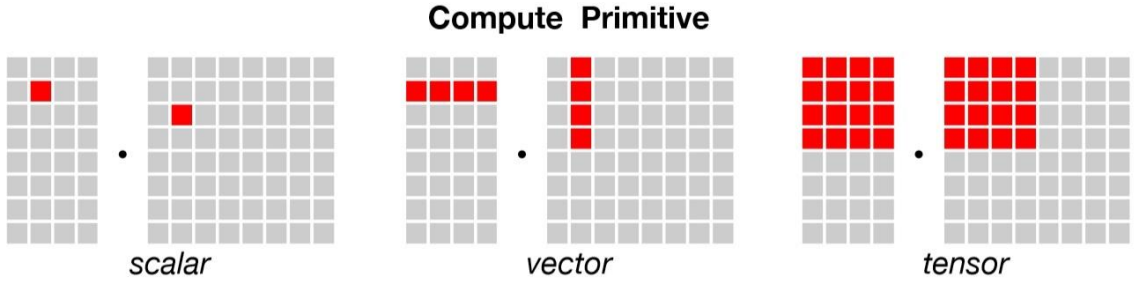


Devices

- Three types of devices
  - CPU
    - General compute unit
    - 2-128 threads
  - GPU
    - Rendering unit
    - 1000-10000 threads
  - TPU
    - Specialized unit
    - 8-2048 threads



CPU GPU TPU



Note that we are comparing apples to bananas!

Memory

Equally important to what device you are using, is the amount of memory that you have available

With more memory

- Faster data transfer
- Higher data modality
- Larger models

	CPU	GPU	TPU
Standard	32-64 GiB	12 GiB	64 GiB
Maximum	256 GiB	24 GiB	32 TiB

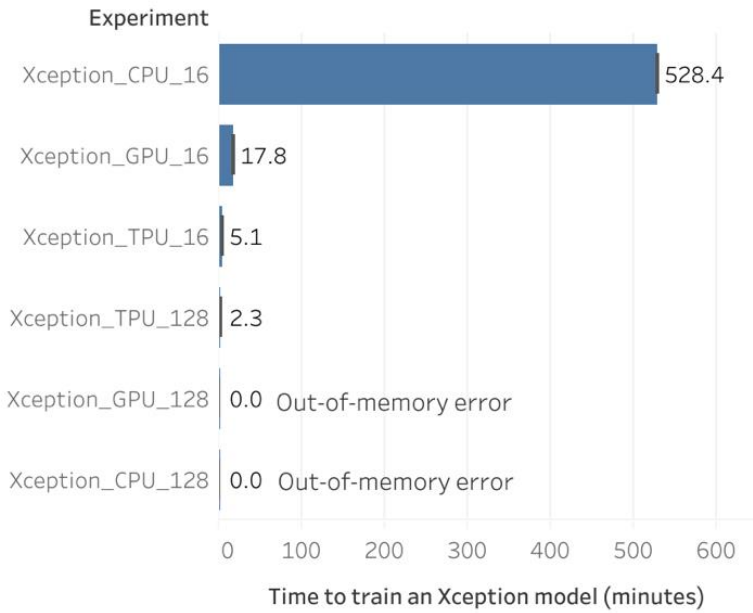
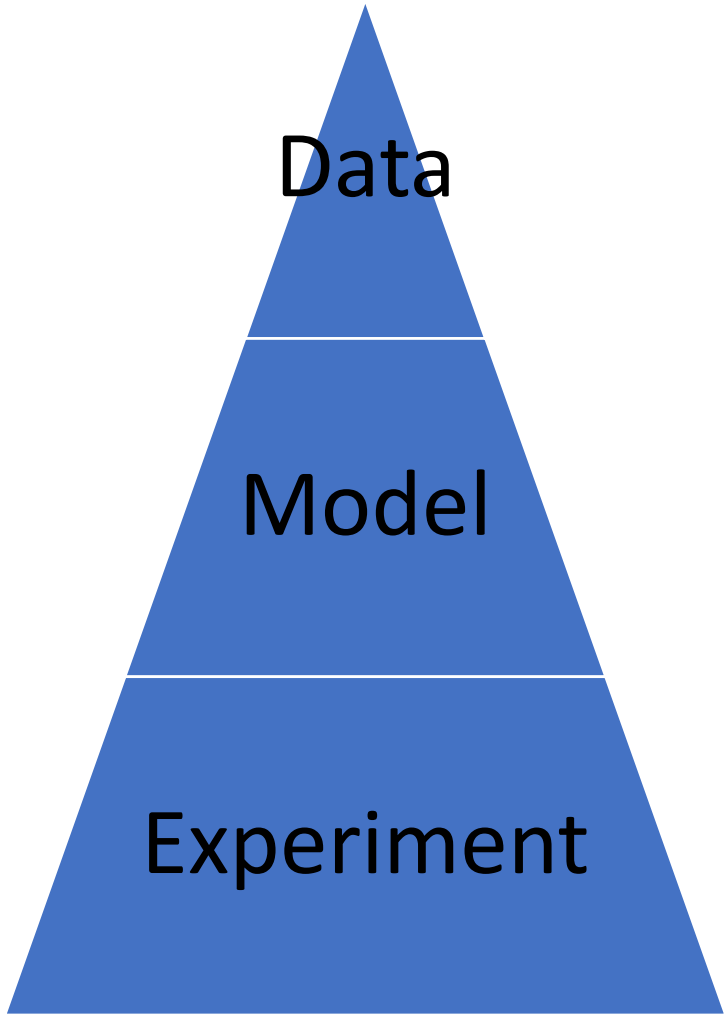


Figure 3: CPUs vs GPUs vs TPUs for training an Xception model for 12 epochs. Y-Axis labels indicate the choice of model, hardware, and batch size for each experiment. Increasing the batch size to 128 for TPUs resulted in an additional ~2x speedup.

https://towardsdatascience.com/when-to-use-cpus-vs-gpus-vs-tpus-in-a-kaggle-competition-9af708a8c3eb

Many layers of distributed computations



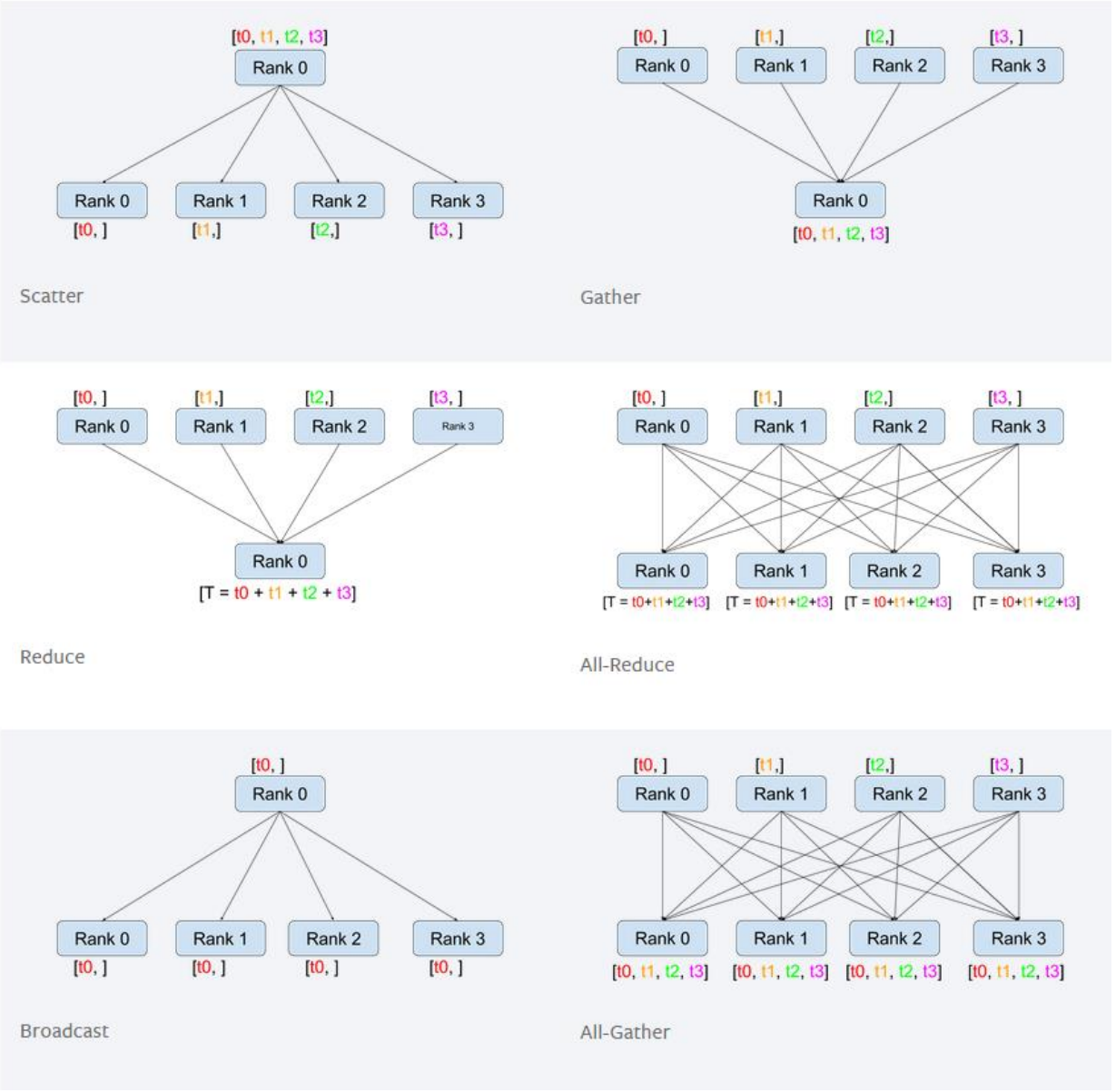
Load data in paralle
Inference on data in parallel

Stack models in parallel

Launch multiple experiments at once

You may use multiple of these at the same time

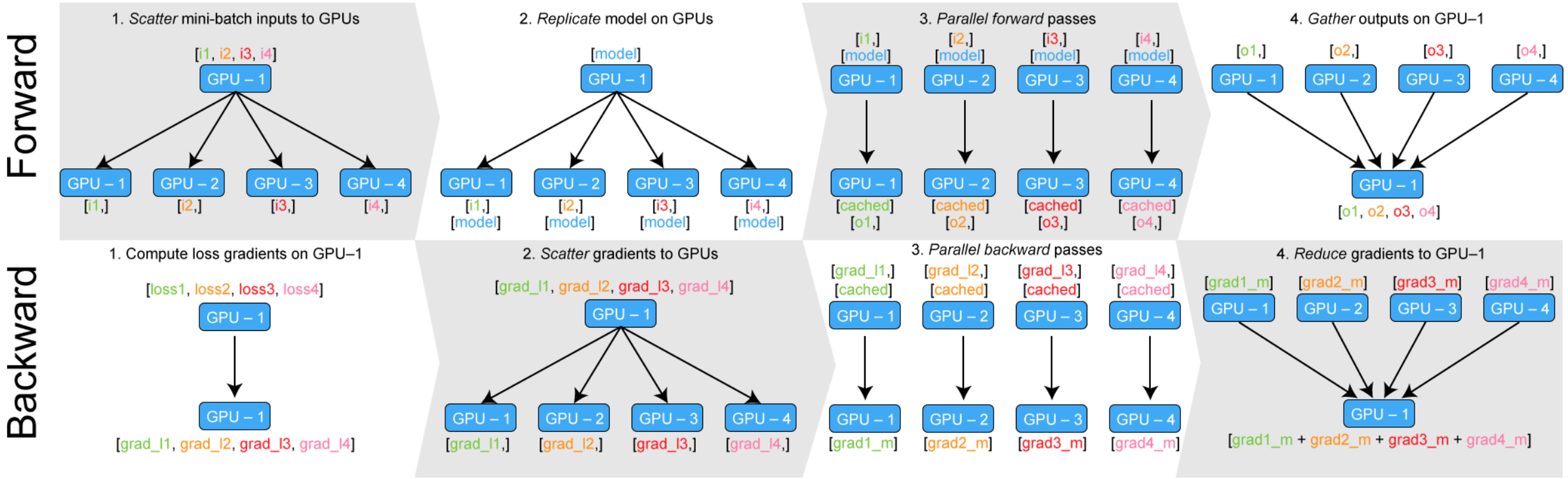
The six imporatnt communication types



Data Parallel: one process controls all

MLOps

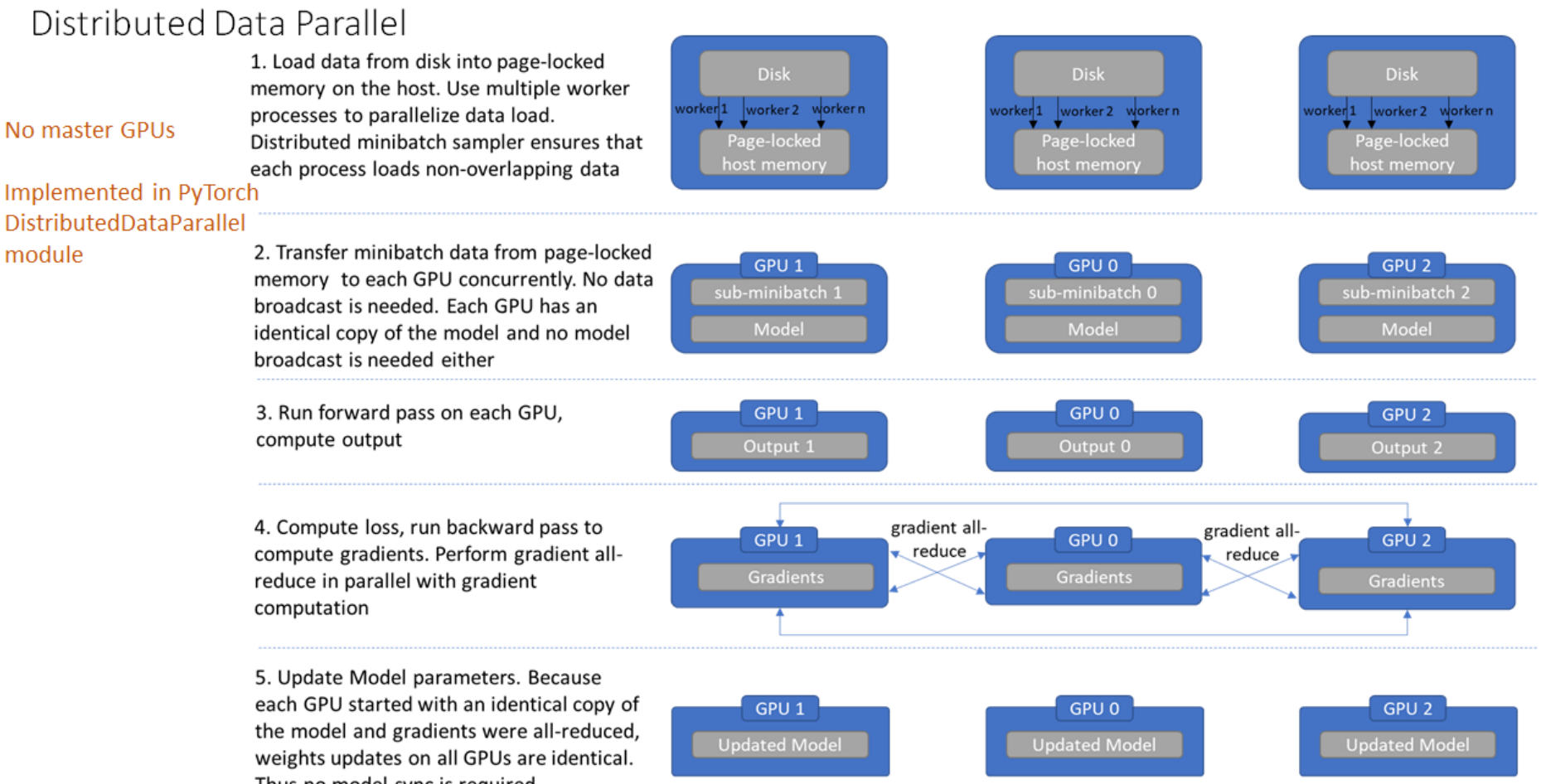
Simple as parallel_model = torch.nn.DataParallel(model)



Note that only GPU-1 parameters are updated, the replicas are destroyed after backward

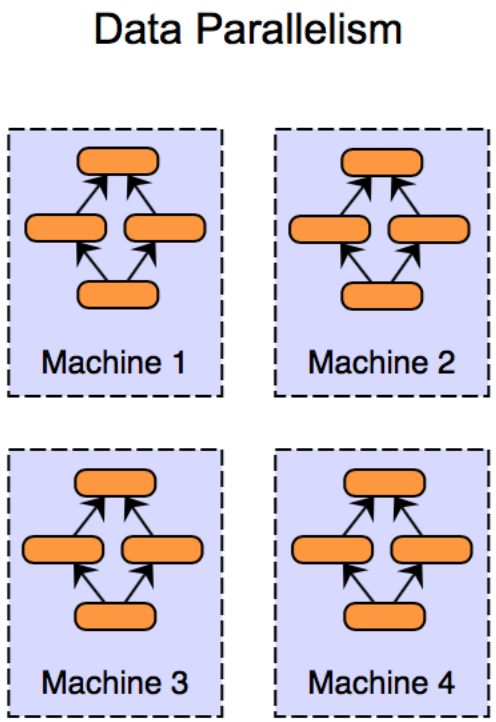
Distributed Data Parallel

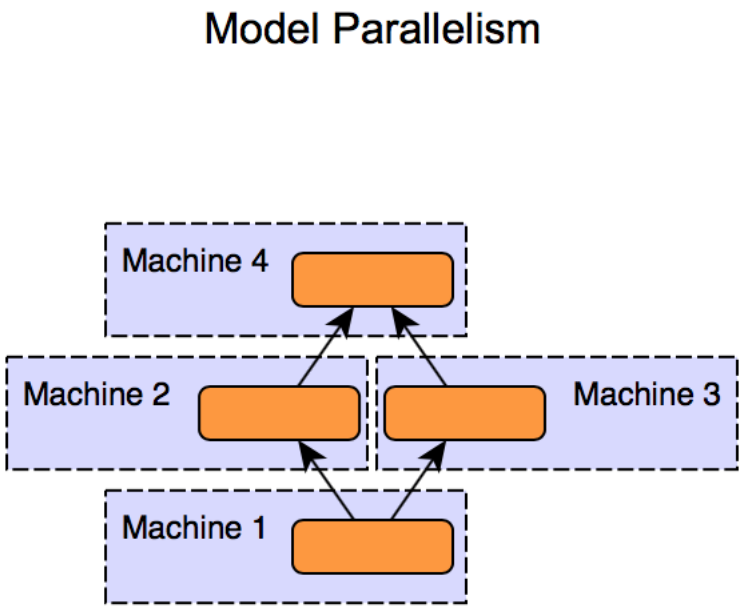
- In Distributed Data Parallel (DDP) all processes gets equal workload



Model paralliseme

- When your model is too big for one device





How to do this in practise

- DataParallel
  - parallel_model = torch.nn.DataParallel(model)

- Distributed Data Parallel
  - Set a enviroment MASTER_ADDR and MASTER_PORT
  - Init a process group
  - parallel_model = nn.parallel.DistributedDataParallel(model, device_ids=[gpu]
  - Use mp.spawn to spawn multiple processes
  - ...

- Model parallizeme
  - A shit ton of tensor.to('cuda:x') calls

Seperating engineering and research code

Getting code to run in parallel has somewhat become a research task!

However maybe it should not be like that?

Research Code

```
l1 = nn.Linear(...)
l2 = nn.Linear(...)
decoder = Decoder()

x1 = l1(x)
x2 = l2(x2)
out = decoder(features, x)

loss = perceptual_loss(x1, x2, x) + CE(out, x)
```

Engineering code

```
model.cuda(0)
x = x.cuda(0)

distributed = DistributedParallel(model)

with gpu_zero:
    download_data()

dist.barrier()
```

Spend time on research code and not engineering code!

Why using a training framework

Spend time on research code and not engineering code

-> Why training frameworks exist!

- Reduce boilerplate = increase turn-around time
- Focus on what is important
- Reproduceability
- Shareability
- Consistency
- Scaleability

Training Fremworks

Many fremeworks exist for reducing boilerplate

Many frameworks for accelerating training