Distributed Deep Learning on the HAL System

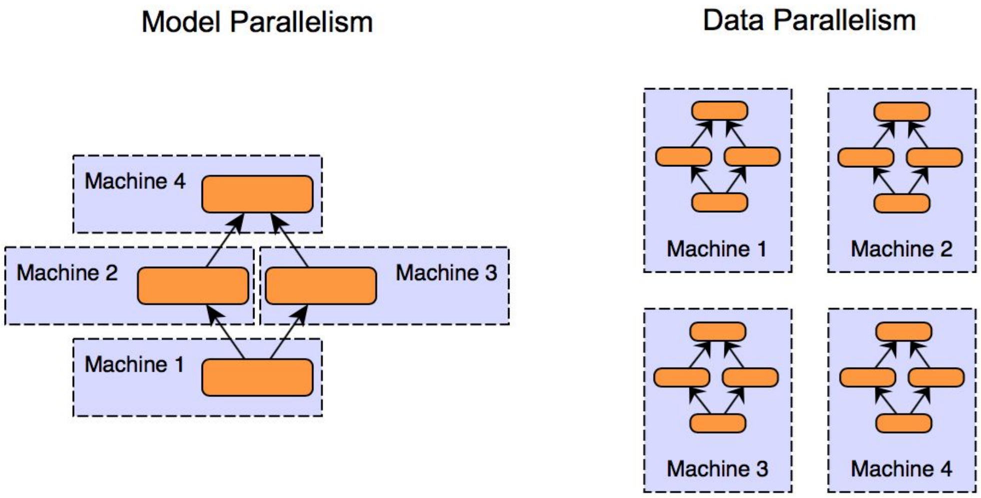
Benjamin Rabe, NCSA

What are the benefits of distributed training?

- Acceleration of the training process
  - Faster training equates to less wasted developer time in the {training, evaluating, hyperparameter tuning} development cycle.
- Making the most of available GPU memory
  - Data parallelism allows for tradeoff between local batch size and number of workers, and model parallelism allows for models to exceed the memory capacity of a single GPU.
- Decreased reliance on hardware capability
  - Scaling out rather than up can allow high performance even on systems without cutting-edge hardware.
- Future-proofing
  - Network architectures and datasets will continue to grow in complexity and scale.

What types of distributed training exist?

- Data vs model parallelism



- Synchronous vs asynchronous training
- Focus of this talk is data parallel, synchronous training. However, resources on other types will be provided.

Image source: https://xiandong79.github.io/Intro-Distributed-Deep-Learning

How / why / when does data parallel training work?

- Gradient descent overview
  - Gradient descent (1)
  - Minibatch SGD (2)
  - Distributed minibatch SGD (3)

$$w_{t+1} = w_t - \eta \nabla L(w_t) \tag{1}$$

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t) \tag{2}$$

$$w_{t+1} = w_t - \eta \frac{1}{kn} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_t) \tag{3}$$

- **Core idea:** network is replicated on each worker, gradients are shared through an all-reduce.
- Train steps of single-worker and distributed training are identical provided global batch size (kn) is constant (e.g. 1x128 vs 4x32) and losses for each example are independent.
  - Important exception: batch normalization layers break independence of example losses. This doesn't necessarily mean performance will be worse, but it makes local batch size a network hyperparameter.
- Weights on each worker are identical during sync training (relevant for weight decay computation).

Will your application benefit from distributed training?

- A training application won't always get linear speedup by adding hardware
  - Bottlenecks may arise in computation other than backward propagation including:
    - File I/O for input pipeline
    - Input preprocessing
    - All-reduce of gradients at the end of each step
- A toy example illustrating scaling with number of devices (num gpus = g)
  - For MNIST, global batch size kept constant at 20. For ImageNet, global batch size kept constant at 128.
  - Table shows rate in img/sec
  - Tracing profile of MNIST with g = 4, pink NcclAllReduce op dominates timing:

MNIST - 1 hidden layer (95 units) FCN				ImageNet - ResNet 50		
g = 1	g = 2	g = 4		g = 1	g = 2	g = 4
12700	10264	7146		365	680	1148

Native framework support for distributed training on HAL

- TensorFlow (tf.Estimator API)
  - Info on data-parallel training: https://www.tensorflow.org/guide/distribute_strategy
    - Current support (1.13): replicated training on single worker (1-4 GPUs on HAL) and parameter server asynchronous training on multiple workers
    - Support for multi-node replicated training is currently experimental (~2.0)
  - Model parallelism only supported through custom device placement or Mesh TensorFlow
    - https://github.com/tensorflow/mesh
- PyTorch
  - Supports distributed synchronous training on multiple workers
    - See `torch.nn.DataParallel` and `torch.nn.parallel.DistributedDataParallel`
  - More sophisticated support for model parallelism, and model parallelism is compatible with `DistributedDataParallel`
    - https://pytorch.org/tutorials/intermediate/model_parallel_tutorial.html

Other distributed training frameworks available on HAL

- Horovod
  - Supports TensorFlow, Keras, PyTorch, and MXNet
  - Pre-dates native TensorFlow support for synchronous replicated training
- IBM WML-CE (PowerAI)
  - ddlrun
    - Supports data parallel training in TensorFlow, PyTorch, and Caffe
  - TensorFlow Large Model Support
    - Allows training of models too large to fit in single GPU memory by swapping tensors back and forth from host
- More information on both can be found on the wiki
  - https://wiki.ncsa.illinois.edu/display/ISL20/Multi-node+distributed+training
  - Information subject to change, as it is tied to IBM's WML-CE release schedule

Resources and examples

- Distributed MNIST example from previous slide
  - https://github.com/bendrabe/ddl_training/blob/master/mnist/mnist.py
- Two more examples of tf.Estimator distributed training applications can be found in above GitHub repo
  - SVHN: straightforward pure CNN + dropout for Street View House Number dataset
    - http://ufldl.stanford.edu/housenumbers/
  - SqueezeNet: TensorFlow port of SqueezeNet 1.0 for ImageNet classification task
    - https://arxiv.org/abs/1602.07360
- Insightful paper that covers subtleties of large-scale distributed training and suggests general guidelines for adaptation of concepts to other DL tasks
  - https://arxiv.org/abs/1706.02677
- TensorFlow debugger
  - Documentation: https://www.tensorflow.org/guide/debugger
  - Demo will be debugging NaN bug in SqueezeNet implementation in repo
    - https://github.com/tensorflow/models/blob/497989e0705abc2d7069b3ffde6a42a11929e500/research/slim/train_image_classifier.py#L187