Data Parallel Deep Learning

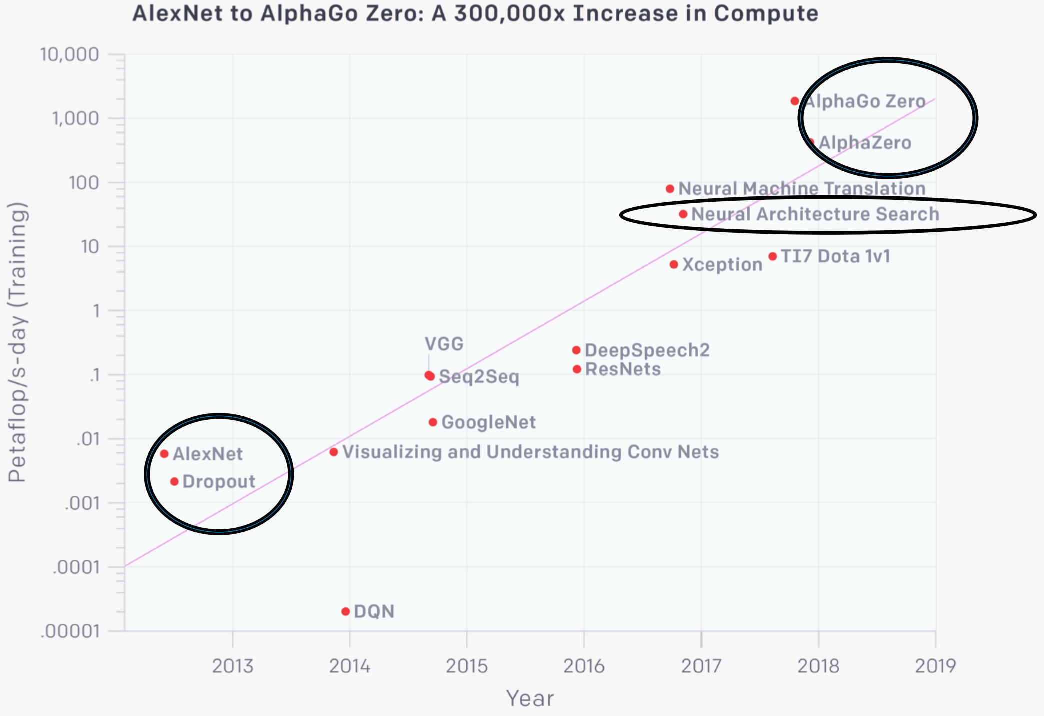
Huihuo Zheng

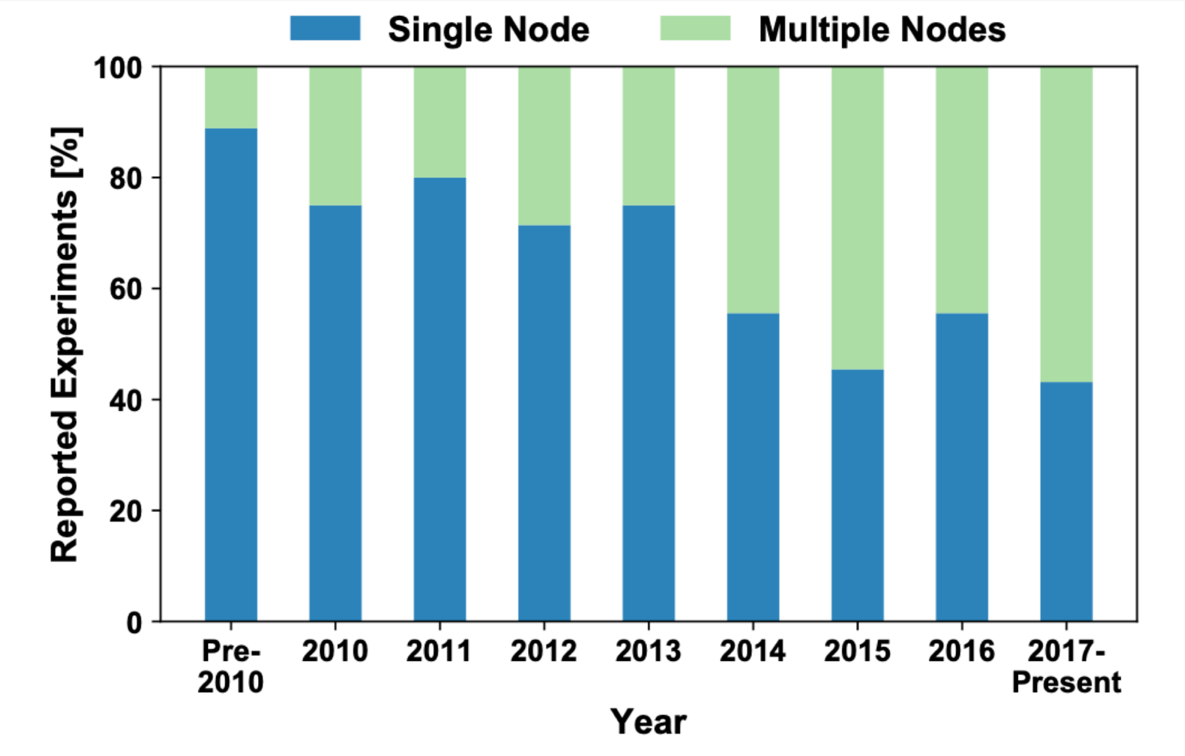
Argonne Leadership Computing Facility
August 7, 2020

huihuo.zheng@anl.gov

The need for distributed training on HPC

"Since 2012, the amount of compute used in the largest AI training runs has been increasing exponentially with a 3.5 month doubling time (by comparison, Moore's Law had an 18 month doubling period)."





https://openai.com/blog/ai-and-compute/

Tal Ben-Nun and Torsten Hoefler, arXiv:1802.09941

Distributed deep learning for ResNet-50

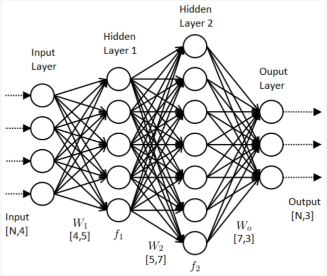
TRAINING TIME AND TOP-1 VALIDATION ACCURACY WITH RESNET-50 ON IMAGENET

		Batch Size	Processor	DL Library	Time	Accuracy
He et al. [1]	2016	256	Tesla P100 × 8	Caffe	29 hours	75.3 %
Goyal et al. [2]		8,192	Tesla P100 × 256	Caffe2	1 hour	76.3 %
Smith et al. [3]		8,192 → 16,384	full TPU Pod	TensorFlow	30 mins	76.1 %
Akiba et al. [4]		32,768	Tesla P100 × 1,024	Chainer	15 mins	74.9 %
Jia et al. [5]		65,536	Tesla P40 × 2,048	TensorFlow	6.6 mins	75.8 %
Ying et al. [6]		65,536	TPU v3 × 1,024	TensorFlow	1.8 mins	75.2 %
Mikami et al. [7]		55,296	Tesla V100 × 3,456	NNL	2.0 mins	75.29 %
Yamazaki et al	2019	**81,920**	**Tesla V100 × 2,048**	**MXNet**	**1.2 mins**	**75.08%**

Quoted from Masafumi Yamazaki, arXiv:1903.12650

The need for distributed training on HPC

- Increase of model complexity leads to dramatic increase of the amount of computation;
- Increase of the size of dataset makes sequentially scanning the whole dataset increasingly impossible;
- The increase in computational power has been mostly coming (and will continue to come) from parallel computing;
- Coupling of deep learning to traditional HPC simulations might require distributed training and inference.



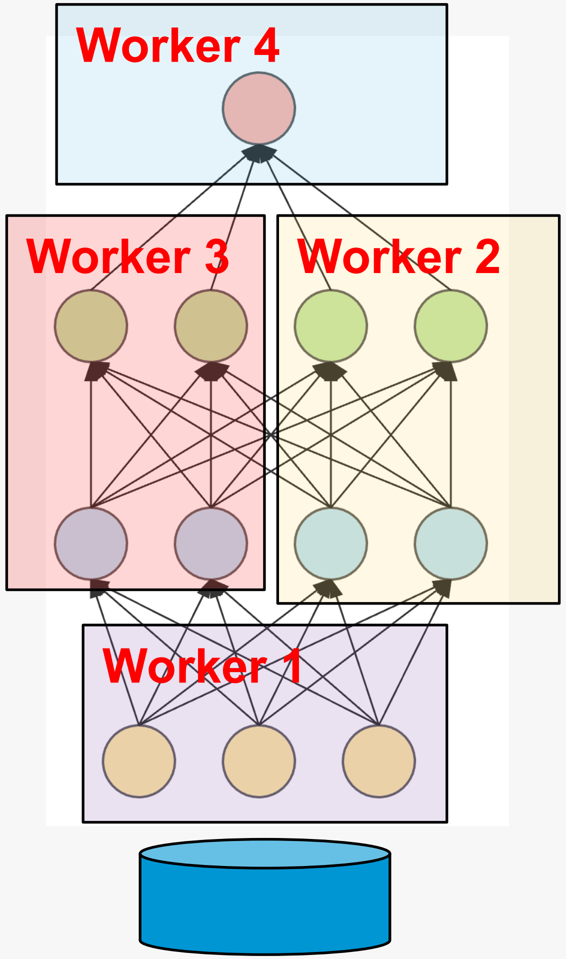
Examples of scientific large scale deep learning

- Thorsten Kurth, Exascale Deep Learning for Climate Analytics, arXiv:1810.01993 (Gordon Bell Prize)
- R. M. Patton, Exascale Deep Learning to Accelerate Cancer Research, arXiv:1909.1229
- N. Laanait, Exascale Deep Learning for Scientific Inverse Problems, arXiv:1909.11150
- W. Dong et al, Scaling Distributed Training of Flood-Filling Networks on HPC Infrastructure for Brain Mapping, arXiv:1905.06236
- A Khan, Deep learning at scale for the construction of galaxy catalogs in the Dark Energy SurveyPhysics Letters B 795, 248-258

Outline

- Different parallelisms for distributed training
- Mini-batch stochastic gradient descent
- Data parallel training using Horovod
- Hands-on examples
  - https://github.com/argonne-lcf/ATPESC_MachineLearning

Parallelization schemes for distributed deep learning



Model parallelism

```python
import torch
import torch.nn as nn
import torch.optim as optim


class ToyModel(nn.Module):
    def __init__(self):
        super(ToyModel, self).__init__()
        self.net1 = torch.nn.Linear(10, 10).to('cuda:0')
        self.relu = torch.nn.ReLU()
        self.net2 = torch.nn.Linear(10, 5).to('cuda:1')

    def forward(self, x):
        x = self.relu(self.net1(x.to('cuda:0')))
        return self.net2(x.to('cuda:1'))
```

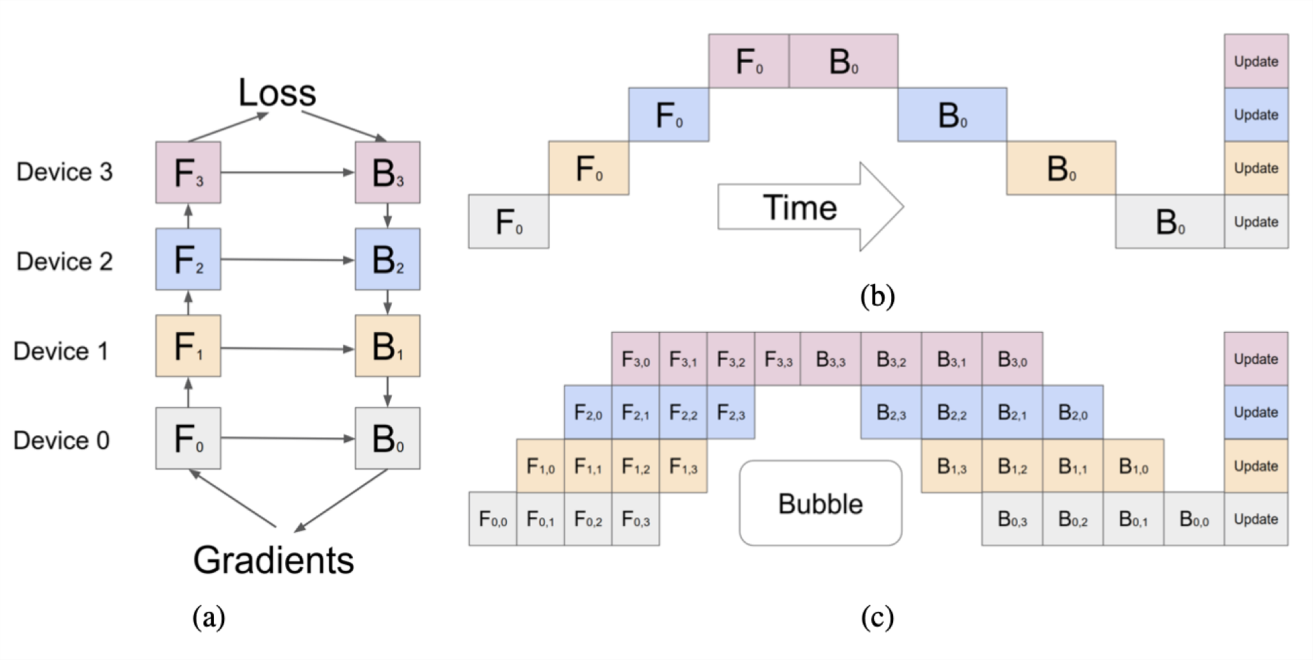
```python
model = ToyModel()
loss_fn = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.001)

optimizer.zero_grad()
outputs = model(torch.randn(20, 10))
labels = torch.randn(20, 5).to('cuda:1')
loss_fn(outputs, labels).backward()
optimizer.step()
```

PyTorch multiple GPU model parallelism within a node

https://pytorch.org/tutorials/intermediate/model_parallel_tutorial.html

Parallelization schemes for distributed deep learning



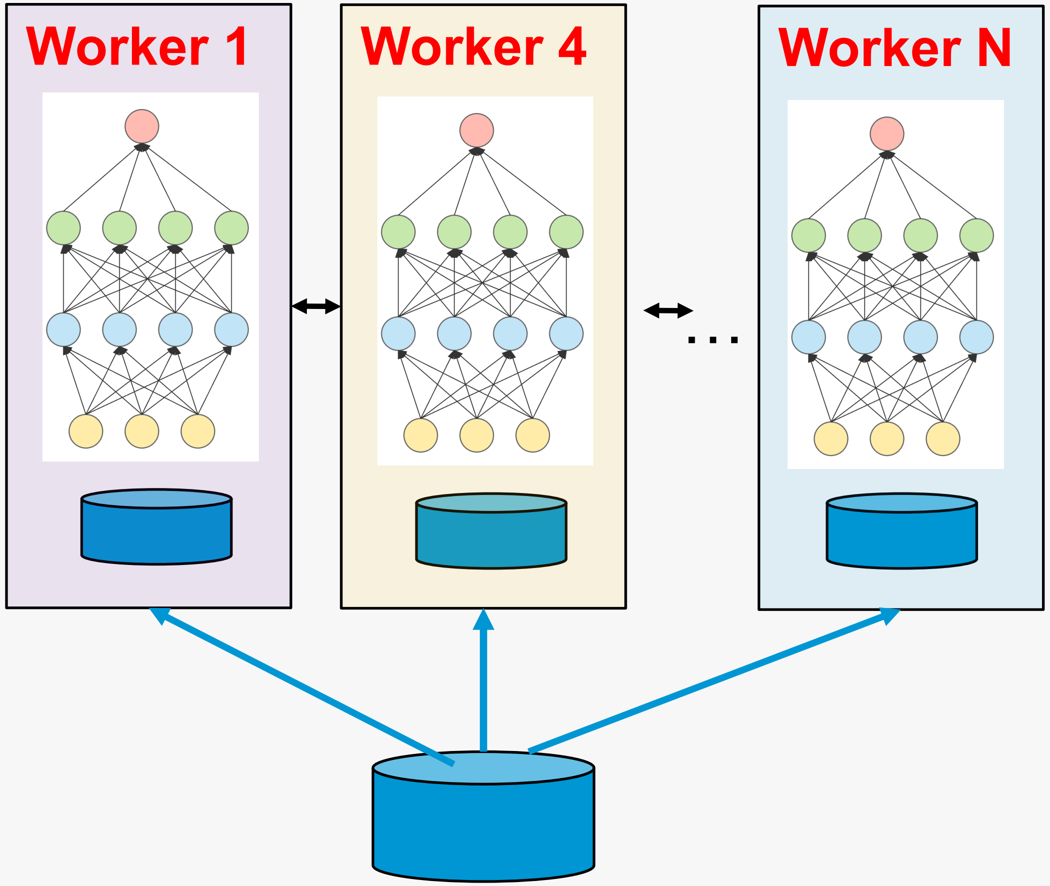
Pipeline libraries:

- GPipe: arXiv:1811.06965
- Pipe-torch:
  DOI: 10.1109/CBD.2019.00020
- PipeDream: arXiv:1806.03377
- HetPipe: arXiv:2005.14038
- DAPPLE: arXiv:2007.01045
- PyTorch Distributed RPC Frameworks: https://pytorch.org/tutorials/intermediate/dist_pipeline_parallel_tutorial.html

Pipeline parallelization

- Partition model layers into multiple groups (stages) and place them on a set of inter-connected devices.
- Each input batch is further divided into multiple micro-batches, which are scheduled to run over multiple devices in a pipelined manner.

Parallelization schemes for distributed deep learning



Data parallelism

- TensorFlow Distributed Training using tf.distribute.Strategy (*MirroredStrategy, MultiWorkerMirroredStrategy, ParameterServerStrategy)*
  https://keras.io/guides/distributed_training/
- PyTorch Distributed Training (torch.nn.parallel.DistributedDataParallel
  https://leimao.github.io/blog/PyTorch-Distributed-Training/
- Horovod – Distributed training framework for TensorFlow, Keras, PyTorch, and Apache MxNet

Mini-batch stochastic gradient descent

Minimizing the loss: $$L(w) = \frac{1}{|X|} \sum_{x \in X} l(x, w).$$

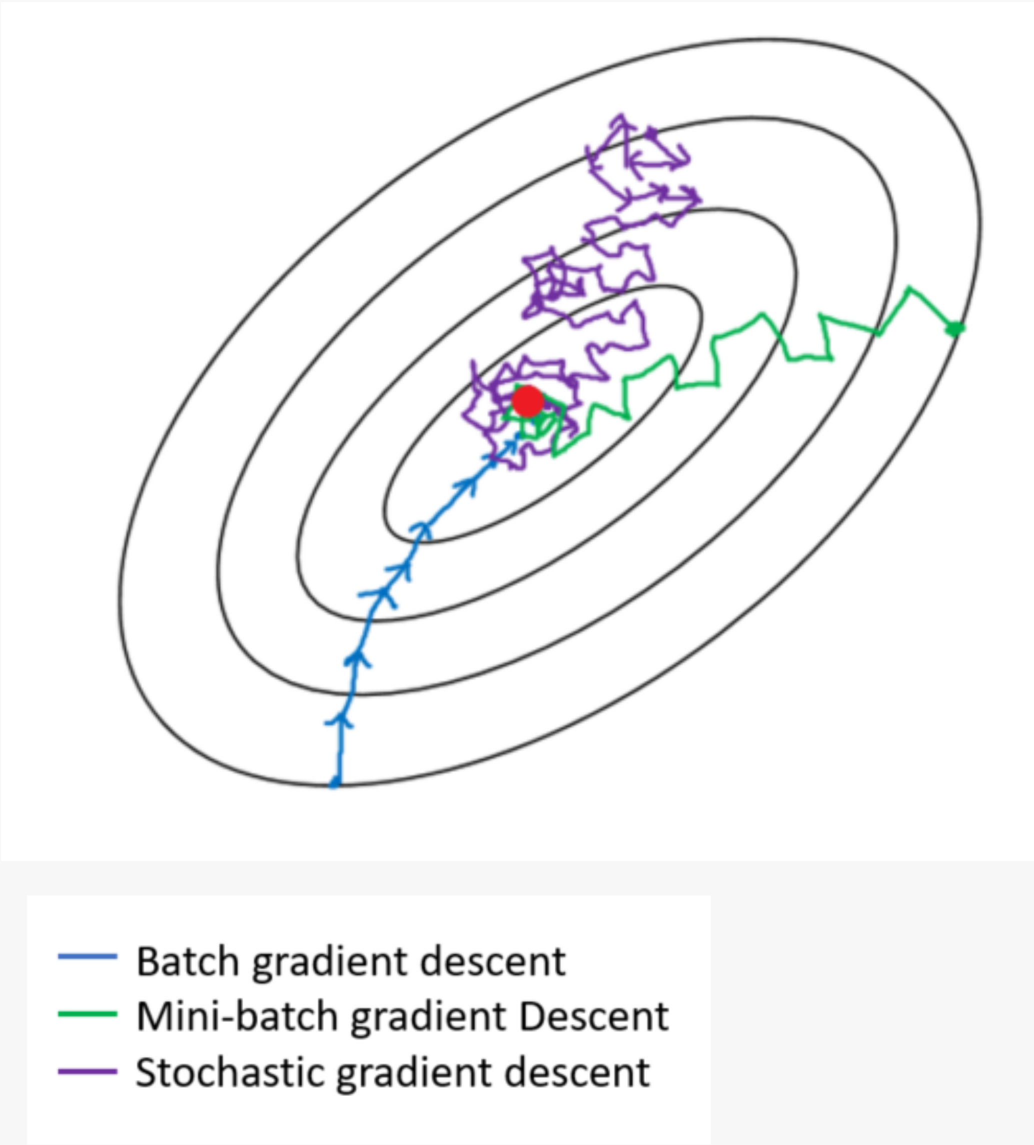
Stochastic Gradient Descent

for $t = 0$ **to** T **do**
  $z \leftarrow$ Random element from S
  $g \leftarrow \nabla \ell(w^{(t)}, z)$
  $w^{(t+1)} \leftarrow w^{(t)} + u(g, w^{(0,\ldots,t)}, t)$
end for

Mini-batch Gradient Descent

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

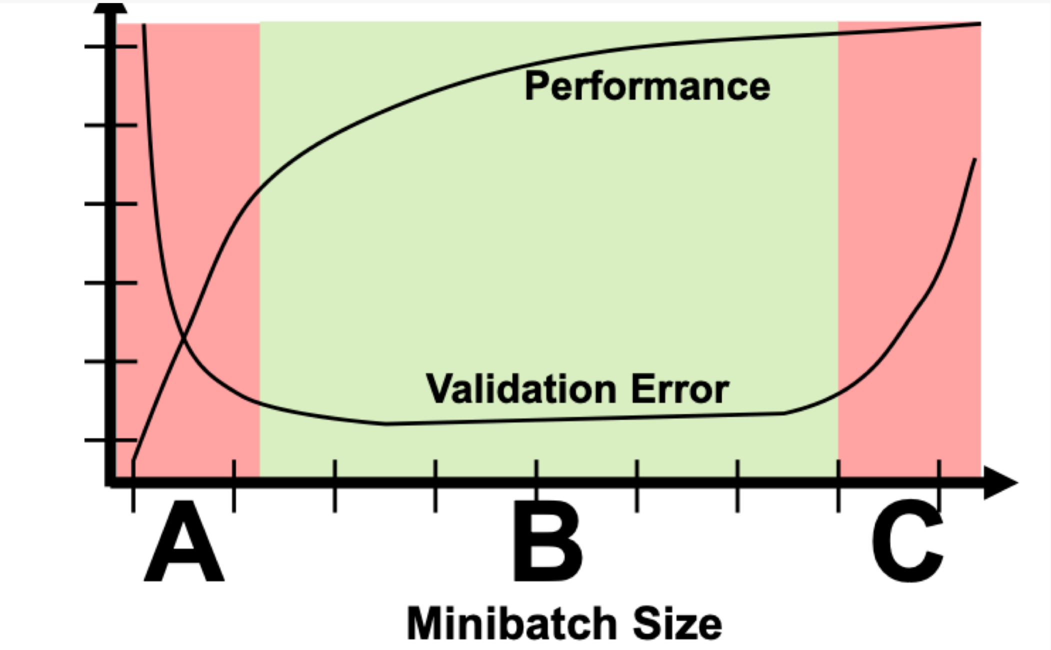
Learning rate   Mini-batch



Minibatch stochastic gradient descent

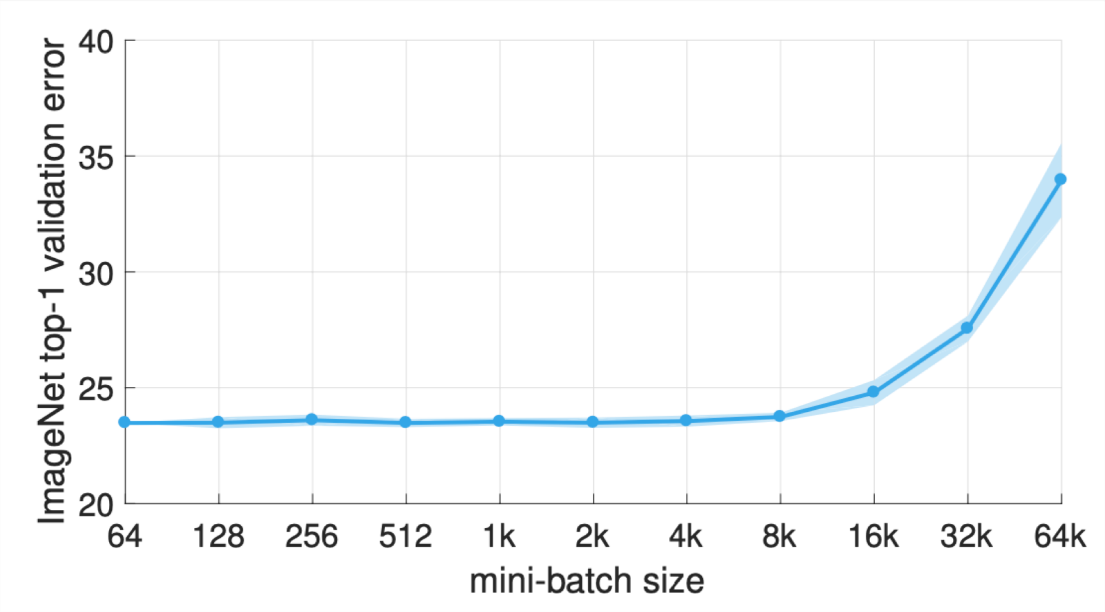
$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

How to choose the minibatch size n?



Minibatch Size Effect on Accuracy and Performance

Tal Ben-Nun and Torsten Hoefler, arXiv:1802.09941



Validation error for different mini-batch size for Resnet50

P. Goyal et al,arXiv: 1706.02677

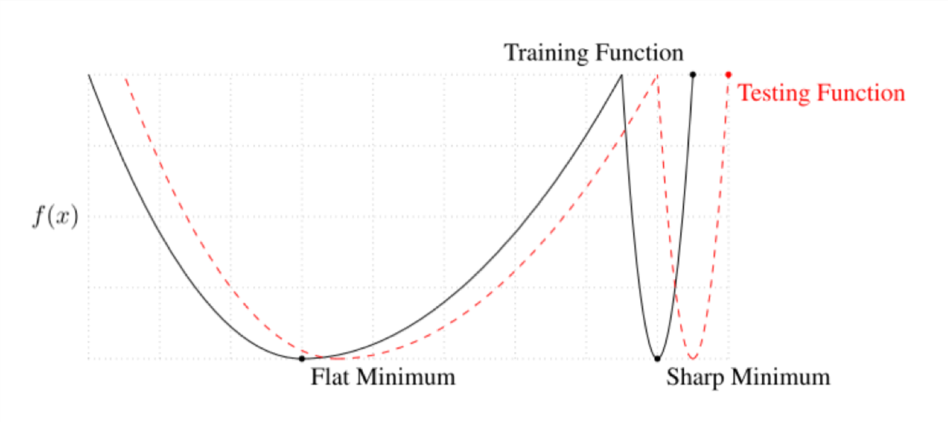
Generalization gap for large mini-batch size

Large mini-batch size training tends to be trapped at local minimum with lower testing accuracy (generalize worse).

Name	Training Accuracy SB	Training Accuracy LB
F_1	$99.66\% \pm 0.05\%$	$99.92\% \pm 0.01\%$
F_2	$99.99\% \pm 0.03\%$	$98.35\% \pm 2.08\%$
C_1	$99.89\% \pm 0.02\%$	$99.66\% \pm 0.2\%$
C_2	$99.99\% \pm 0.04\%$	$99.99\% \pm 0.01\%$
C_3	$99.56\% \pm 0.44\%$	$99.88\% \pm 0.30\%$
C_4	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$

Name	Testing Accuracy SB	Testing Accuracy LB
F_1	$98.03\% \pm 0.07\%$	$97.81\% \pm 0.07\%$
F_2	$64.02\% \pm 0.2\%$	$59.45\% \pm 1.05\%$
C_1	$80.04\% \pm 0.12\%$	$77.26\% \pm 0.42\%$
C_2	$89.24\% \pm 0.12\%$	$87.26\% \pm 0.07\%$
C_3	$49.58\% \pm 0.39\%$	$46.45\% \pm 0.43\%$
C_4	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$

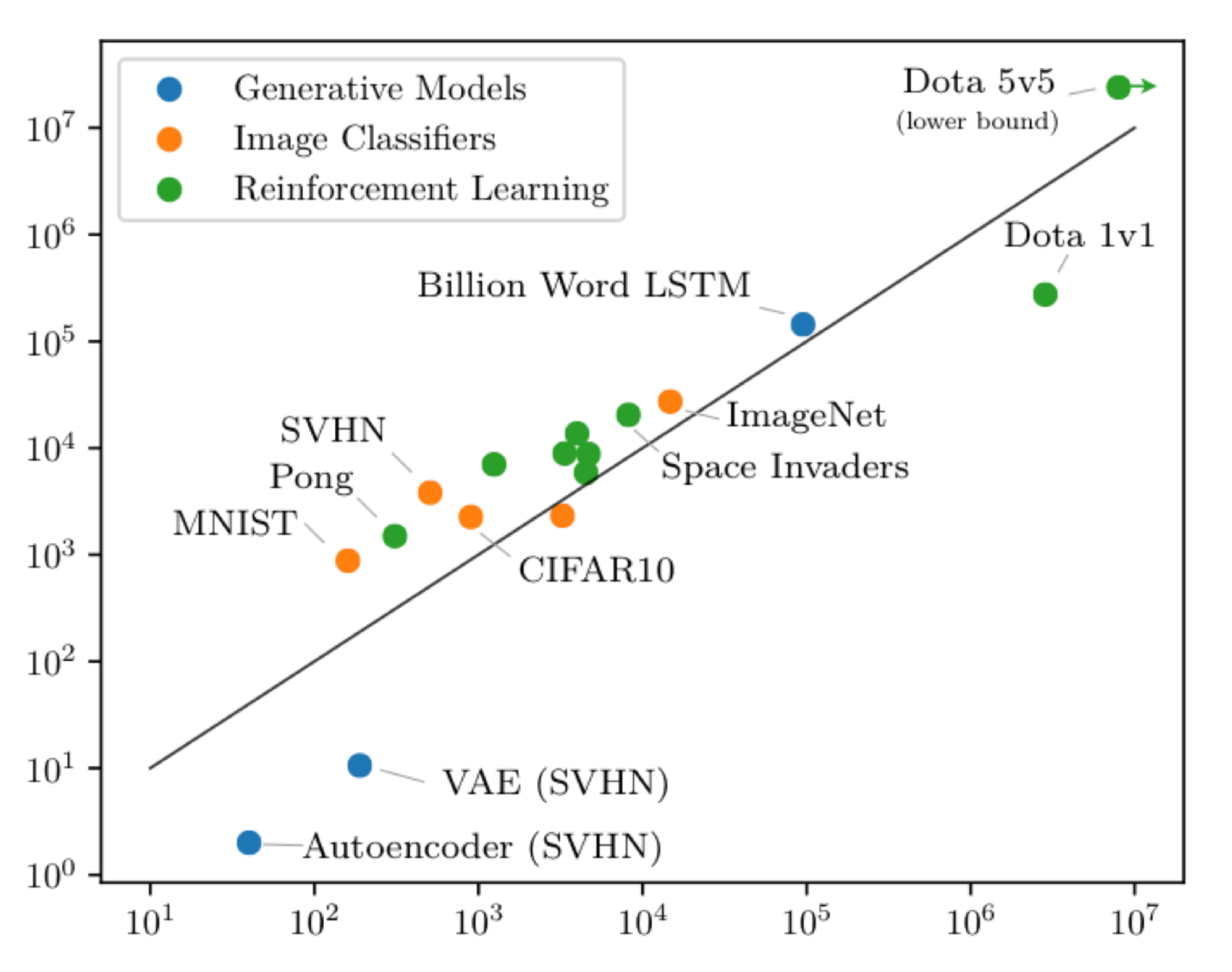
Performance of small-batch (SB) and large-batch (LB) variants of ADAM on the 6 networks



"... large-batch ... converge to sharp minimizers of the training function ... In contrast, small-batch methods converge to flat minimizers"

Keskar et al, arXiv:1609.04836

Challenges with large mini-batch training

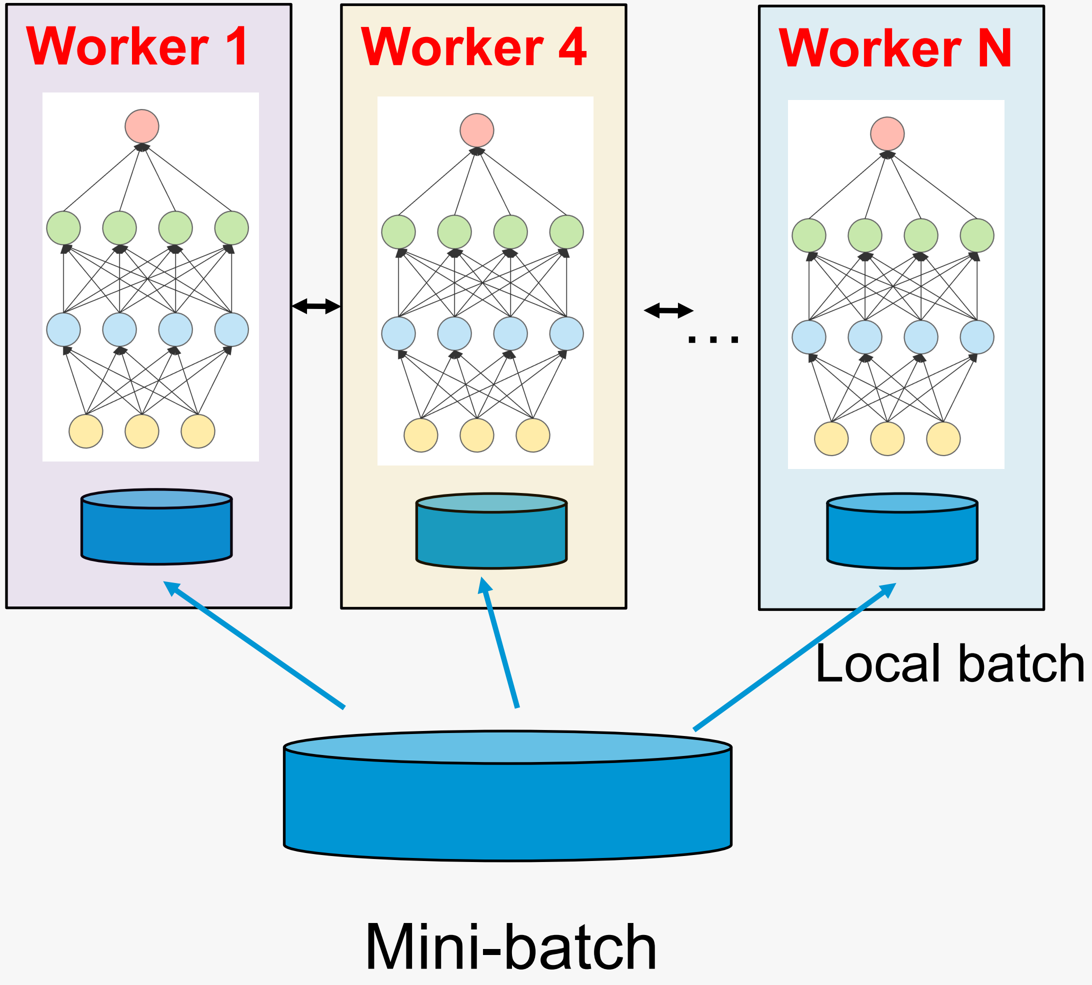


Predicted critical maximum batch size beyond which the model does not perform well.

S. McCandlish, J. Kaplan, D. Amodei, arXiv:1812.06162

Argonne NATIONAL LABORATORY

Data parallel training



Single worker --> N worker

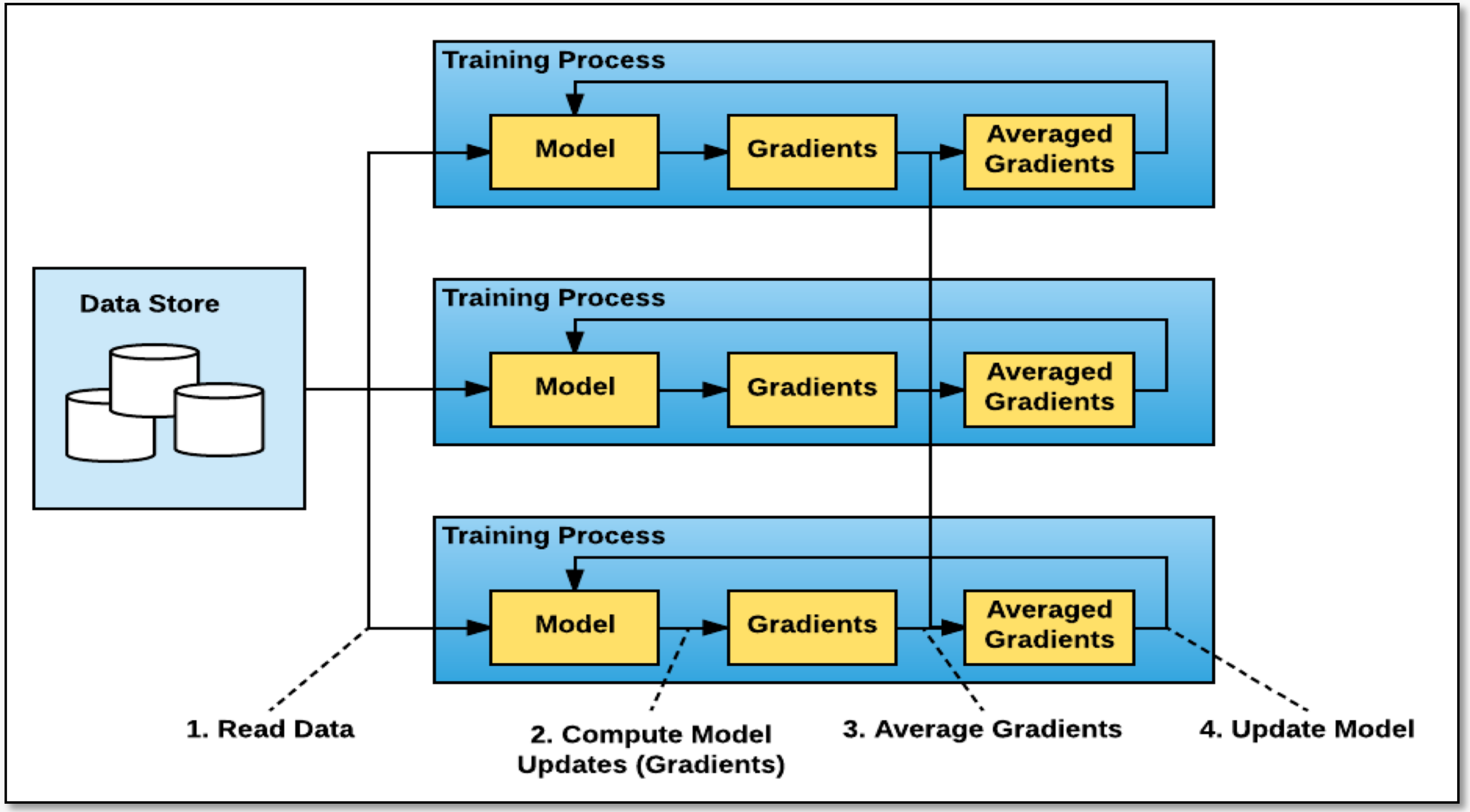
- Mini-batch size increases by N times so that aggregate throughput increases linearly.
- Learning rate should increase proportionally (warmup steps with smaller learning rate might be needed)

Time-to-solution decreases as number of steps reduces.

- Gradients are aggregated over all the workers through MPI_Allreduce

S. McCandlish, J. Kaplan, D. Amodei, arXiv:1812.06162

Data parallel training with Horovod



https://eng.uber.com/horovod/

Data parallel training with Horovod

How to change a series code into a data parallel code:

- Import Horovod modules and initialize horovod
- Wrap optimizer in hvd.DistributedOptimizer & scale the learning rate by number of workers
- Broadcast the weights from worker 0 to all the workers
- Worker 0 saves the check point files
- Data loading:
  - Option 1. All the workers scan through the whole dataset in a random way, and decrease the number of steps per epoch by N.
  - Option 2. Divide the dataset and each worker only scans through a subset of dataset.

https://eng.uber.com/horovod/

TensorFlow V1 with Horovod

```
import tensorflow as tf
import horovod.tensorflow as hvd
layers = tf.contrib.layers
learn = tf.contrib.learn
def main():
   # Horovod: initialize Horovod.
   hvd.init()
   # Download and load MNIST dataset.
   mnist = learn.datasets.mnist.read_data_sets('MNIST-data-%d' % hvd.rank())
   # Horovod: adjust learning rate based on number of GPUs.
   opt = tf.train.RMSPropOptimizer(0.001 * hvd.size())
   # Horovod: add Horovod Distributed Optimizer
   opt = hvd.DistributedOptimizer(opt)
   hooks = [
        hvd.BroadcastGlobalVariablesHook(0),
        tf.train.StopAtStepHook(last_step=20000 // hvd.size()),
        tf.train.LoggingTensorHook(tensors={'step': global_step, 'loss': loss},
                                   every_n_iter=10),
    ]
   checkpoint_dir = './checkpoints' if hvd.rank() == 0 else None
   with tf.train.MonitoredTrainingSession(checkpoint_dir=checkpoint_dir,
                                          hooks=hooks,
                                          config=config) as mon_sess
```

More examples can be found in https://github.com/uber/horovod/blob/master/examples/

TensorFlow V2 with Horovod

```
import tensorflow as tf
import horovod.tensorflow as hvd
hvd.init()
# Horovod: adjust learning rate based on number of GPUs.
opt = tf.optimizers.Adam(0.001* hvd.size())
def  training_step(images, labels, first_batch):
    with tf.GradientTape() as tape:
         probs = mnist_model(images, training=True)
          loss_value = loss(labels, probs)
# Horovod: add Horovod Distributed GradientTape.
     tape = hvd.DistributedGradientTape(tape)
     grads = tape.gradient(loss_value, mnist_model.trainable_variables)
     opt.apply_gradients(zip(grads, mnist_model.trainable_variables))

     if first_batch:
             hvd.broadcast_variables(mnist_model.variables, root_rank=0)
             hvd.broadcast_variables(opt.variables(), root_rank=0)
     return loss_value

for batch, (images, labels) in enumerate(dataset.take(10000 // hvd.size())):
    loss_value = training_step(images, labels, batch == 0)
    if hvd.rank() == 0 and batch % 10 == 0:
        checkpoint.save(checkpoint_dir)
```

PyTorch with Horovod

```
#…
import torch.nn as nn
import horovod.torch as hvd
hvd.init()
train_dataset = datasets.MNIST('datasets', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ]))
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)
# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
# Horovod: scale learning rate by the number of GPUs.
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),
                      momentum=args.momentum)
# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
    optimizer, named_parameters=model.named_parameters())
```

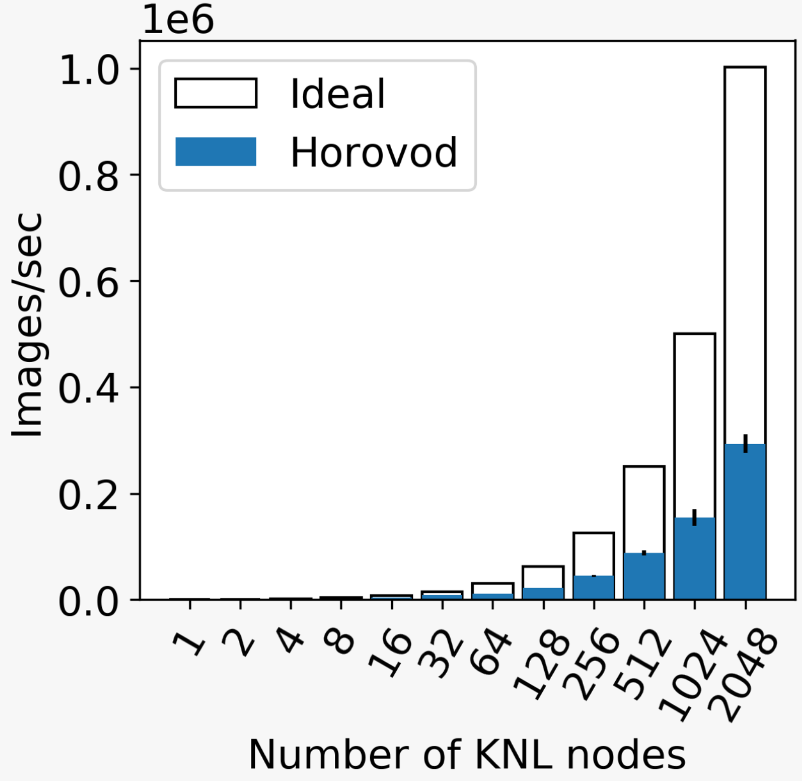
More examples can be found in https://github.com/uber/horovod/blob/master/examples/

Keras with Horovod

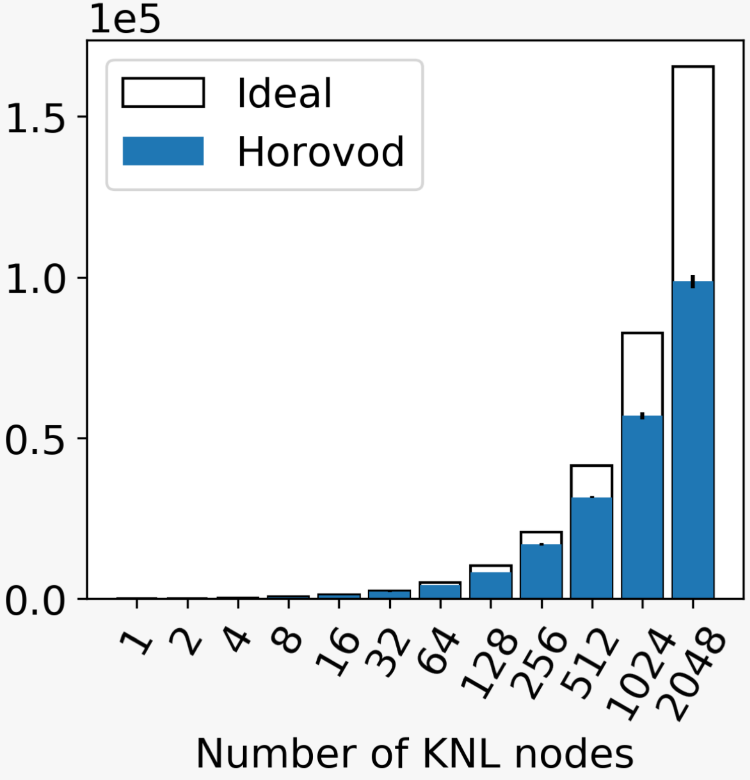
```
import keras
import tensorflow as tf
import horovod.keras as hvd
# Horovod: initialize Horovod.
hvd.init()
# Horovod: adjust learning rate based on number of GPUs.
opt = keras.optimizers.Adadelta(1.0 * hvd.size())
# Horovod: add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=opt,
              metrics=['accuracy'])
callbacks = [
    # Horovod: broadcast initial variable states from rank 0 to all other processes.
    hvd.callbacks.BroadcastGlobalVariablesCallback(0),
]
# Horovod: save checkpoints only on worker 0 to prevent other workers from corrupting them.
if hvd.rank() == 0:
    callbacks.append(keras.callbacks.ModelCheckpoint('./checkpoint-{epoch}.h5'))
model.fit(x_train, y_train, batch_size=batch_size,
          callbacks=callbacks,
          epochs=epochs, steps_per_epochs=num_samples//batch_size//hvd.size(),
          verbose=1, validation_data=(x_test, y_test))
```

More examples can be found in https://github.com/uber/horovod/blob/master/examples/

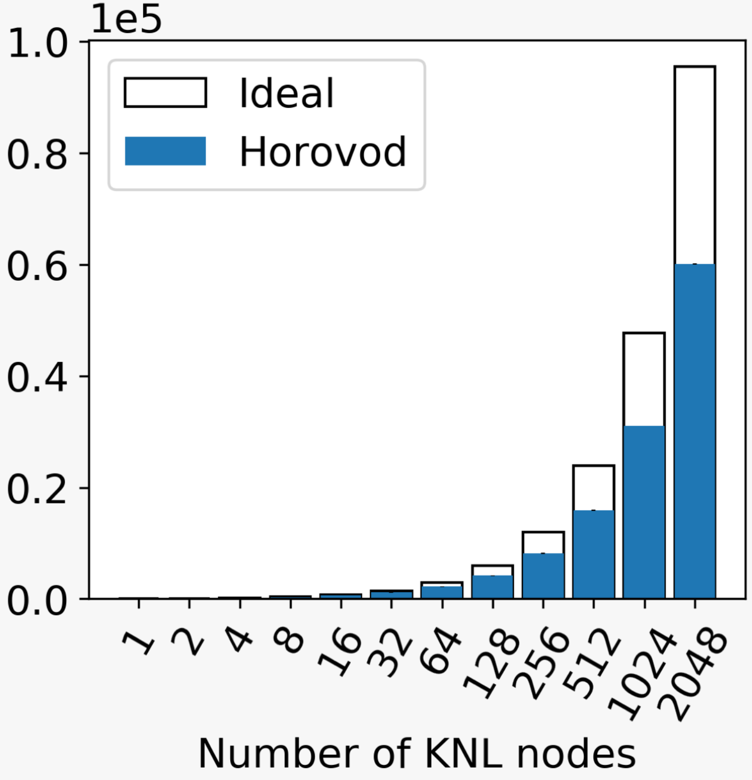
Scaling TensorFlow using Data parallelism on Theta @ ALCF: fixing local batch size = 512



AlexNet



ResNet-50



Inception V3

I/O and data management in distributed deep learning

Streaming I/O provided by frameworks

- TensorFlow Data Pipeline
- PyTorch Data Loader
- Keras DataGenerator

Some suggestions for large scale training

- Organize your dataset in a reasonable way (file per sample shall be avoided if the file is too small; share file performs poorly in some file system, e.g., Lustre)
- Parallel IO might be needed at large scale
- Shuffling in the memory instead of in I/O
- Taking advantage of the node-local storage on a system, for example, SSD @ Theta, Burst buffer @ Summit

*We have developed I/O profiling library, **VaniDL** for analyzing DL I/O on HPC. Contact us if you want to know more.*

Hands on session

I. **Running on Google's Colaboratory Platform**
   https://github.com/argonne-lcf/ATPESC_MachineLearning/

   DataParallelDeepLearning/google_collab.ipynb

II. **Running on Theta**
   https://github.com/argonne-lcf/ATPESC_MachineLearning/blob/master/DataParallelDeepLearning/handson.md

Thank you!

huihuo.zheng@anl.gov