Multi-GPU training with TensorFlow on Piz Daint

Synchronous Distributed Training with TensorFlow and Horovod

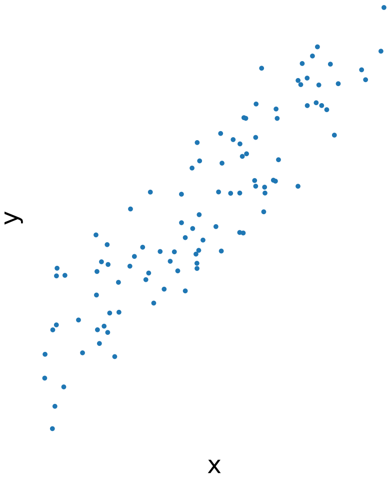
Rafael Sarmiento and Henrique Mendonça
ETHZürich / CSCS
7th-8th September 2020

ETH zürich
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Outline

- Stochastic Gradient Descent
- [lab] Simple Stochastic Gradient Descent
- Synchronous Distributed Stochastic Gradient Descent
- Ring Allreduce
- Horovod
- [lab] Simple Stochastic Gradient Descent with Horovod

We want to train a model on this data



We choose a model and a cost function

$$y = mx + n$$

$$L = \frac{1}{N}\sum_{i}^{N}(\hat{y}_i - y_i)^2$$

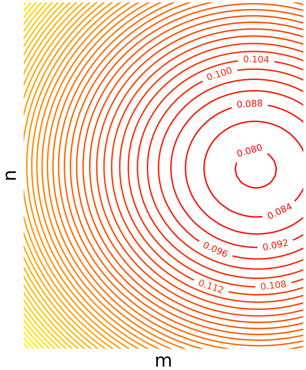
We choose a model and a cost function

$$y = mx + n$$

$$L = \frac{1}{N}\sum_{i}^{N}(\hat{y}_i - y_i)^2$$

$$L = \frac{1}{N}\sum_{i}^{N}(mx_i + n - y_i)^2$$

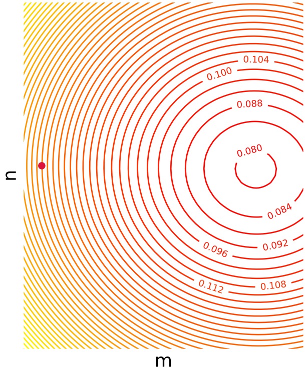
We choose a model and a cost function

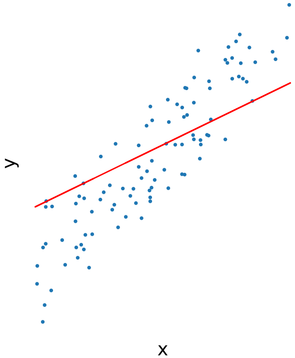


$$y = mx + n$$

$$L = \frac{1}{N}\sum_{i}^{N}(mx_i + n - y_i)^2$$

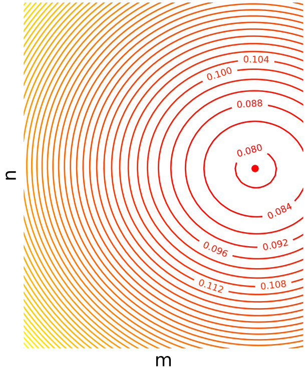
We need to choose an optimizer

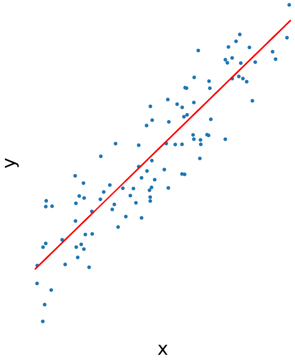




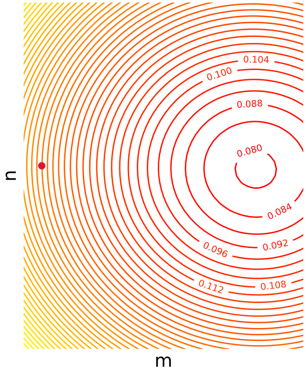
ETH zürich
cscs
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

We need to choose an optimizer



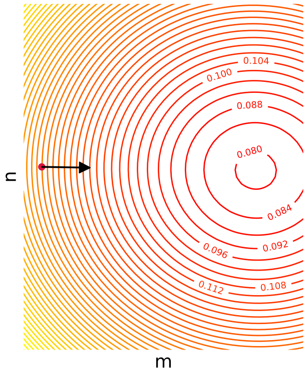


<Stochastic> Gradient Descent



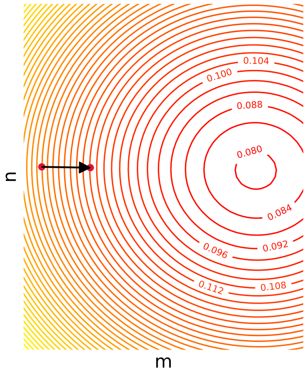
- Evaluate the loss function $L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$ for a batch of N samples $\{x, y\}$ (forward pass)

<Stochastic> Gradient Descent



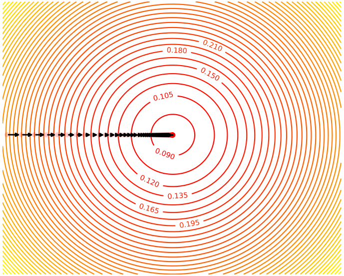
- Evaluate the loss function $L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$ for a batch of N samples $\{x, y\}$ (forward pass)
- Compute the gradients of the loss function with respect to the parameters of the model $\left.\frac{\partial L}{\partial W}\right|_{\{x,y\}}$ (backpropagation)

<Stochastic> Gradient Descent



- Evaluate the loss function $L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$ for a batch of N samples $\{x, y\}$ (forward pass)
- Compute the gradients of the loss function with respect to the parameters of the model $\frac{\partial L}{\partial W}\Big|_{\{x,y\}}$ (backpropagation)
- Update the parameters $W_t = W_{t-1} - \eta \frac{\partial L}{\partial W}\Big|_{\{x,y\}_{t-1}}$

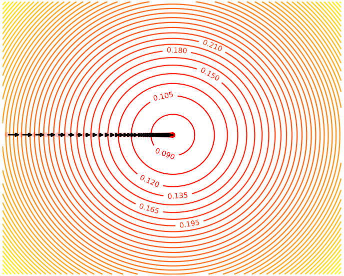
<Stochastic> Gradient Descent



Gradient Descent

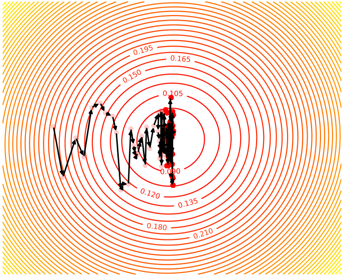
`batch_size = training_set_size`

<Stochastic> Gradient Descent



Gradient
Descent

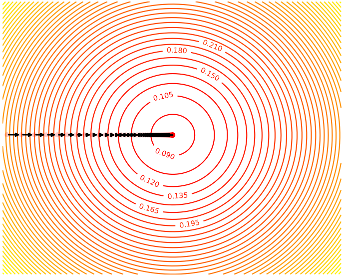
`batch_size = training_set_size`



Stochastic Gradient
Descent

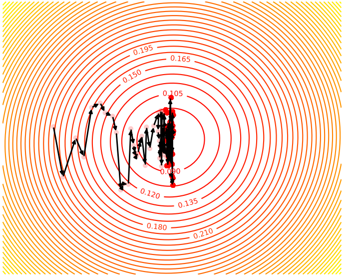
`batch_size = 1`

<Stochastic> Gradient Descent



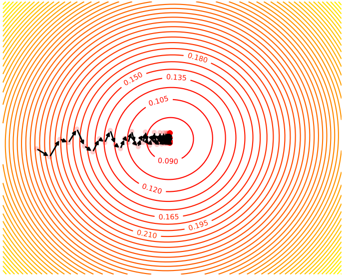
Gradient
Descent

`batch_size = training_set_size`



Stochastic Gradient
Descent

`batch_size = 1`



Minibatch Stochastic Gradient
Descent

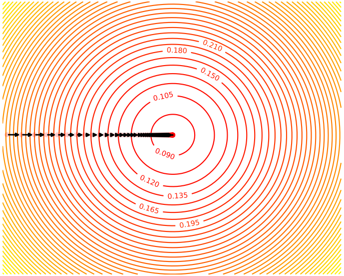
`1 < batch_size < training_set_size`

[lab] Simple Stochastic Gradient Descent

Let's run the notebook `SGD/1-linear_regression_SGD_TF2-simple.ipynb`. There we use an unidimensional linear model to understand the trajectories of the SGD minimization.

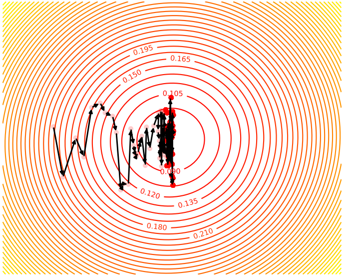
Try different batch sizes and see how the trajectory changes.

<Stochastic> Gradient Descent



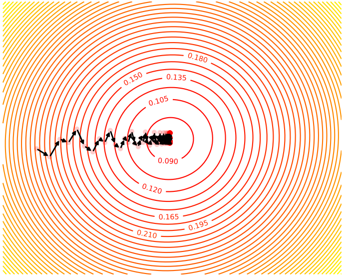
Gradient
Descent

`batch_size = training_set_size`



Stochastic Gradient
Descent

`batch_size = 1`



Minibatch Stochastic Gradient
Descent

`1 < batch_size < training_set_size`

ETH zürich

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Why training with multiple GPUs?

- The batch size is a hyperparameter

Why training with multiple GPUs?

- The batch size is a hyperparameter
- Large batches may not fit on the GPU memory

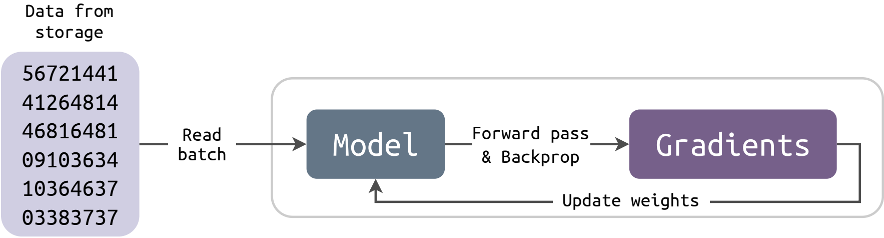
Why training with multiple GPUs?

- The batch size is a hyperparameter
- Large batches may not fit on the GPU memory
- Splitting the training into multiple nodes/GPUs enables the use of large batches

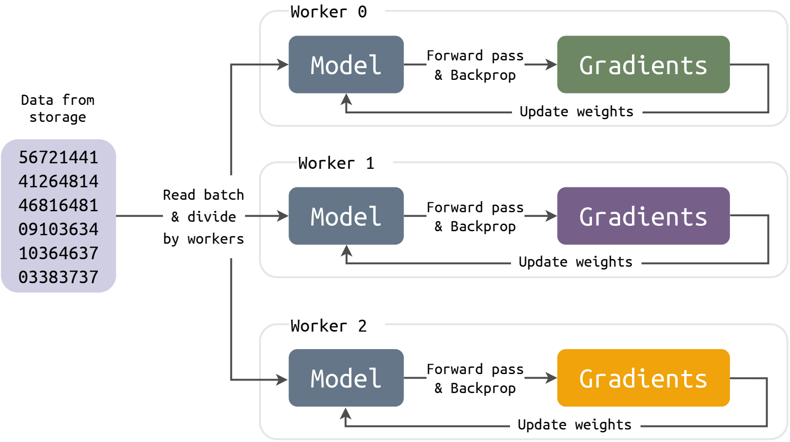
Why training with multiple GPUs?

- The batch size is a hyperparameter
- Large batches may not fit on the GPU memory
- Splitting the training into multiple nodes/GPUs enables the use of large batches
- Multiple nodes/GPUs does not necessarily mean more throughput or faster convergence

Distributing the training with data parallelism

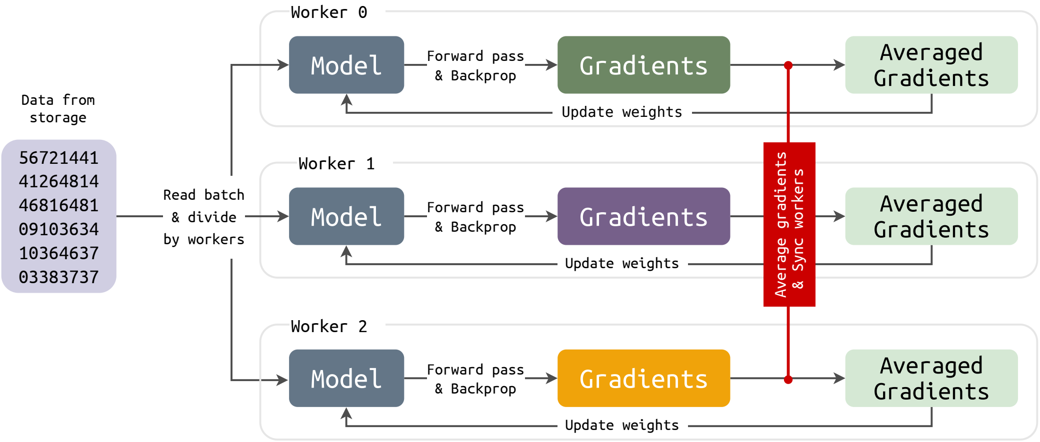


Distributing the training with data parallelism



A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

Distributing the training with data parallelism

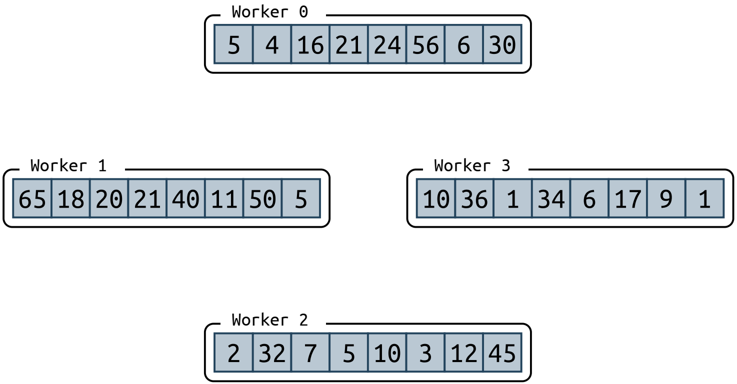


A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

The Allreduce operation

- The Allreduce name comes from the MPI standard.
- MPI defines the function `MPI_Allreduce` to reduce values from all ranks and broadcast the result of the reduction such that all processes have a copy of it at the end of the operation.
- Allreduce can be implemented in different ways depending on the problem.

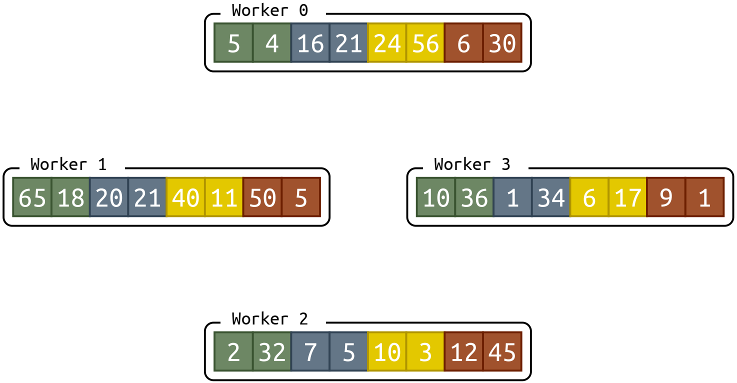
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

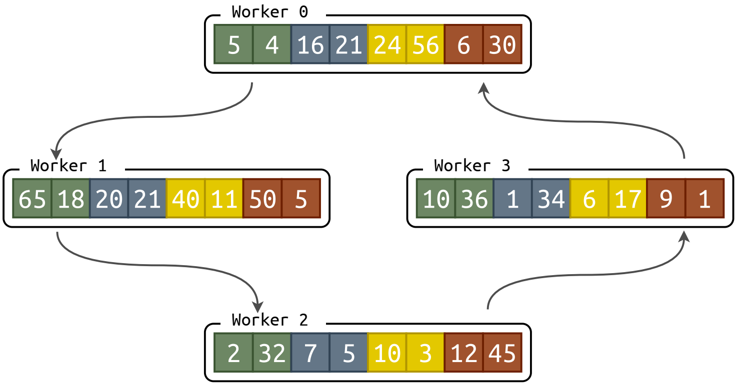
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

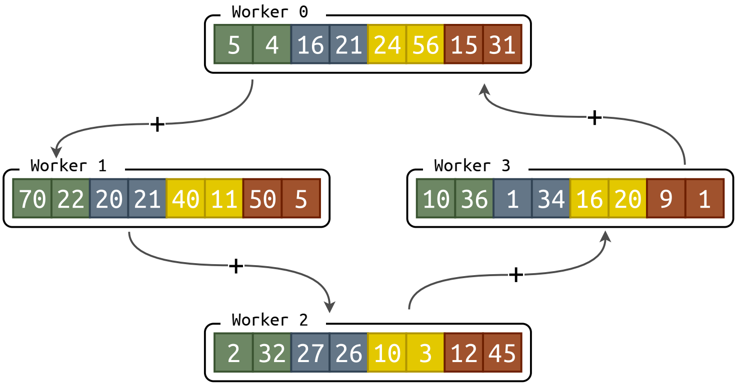
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

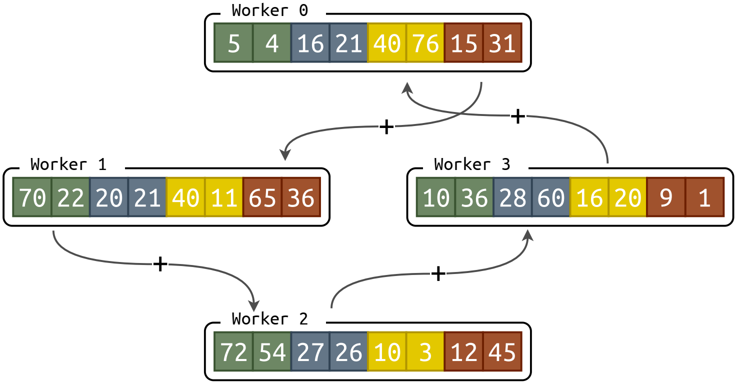
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

Ring Allreduce

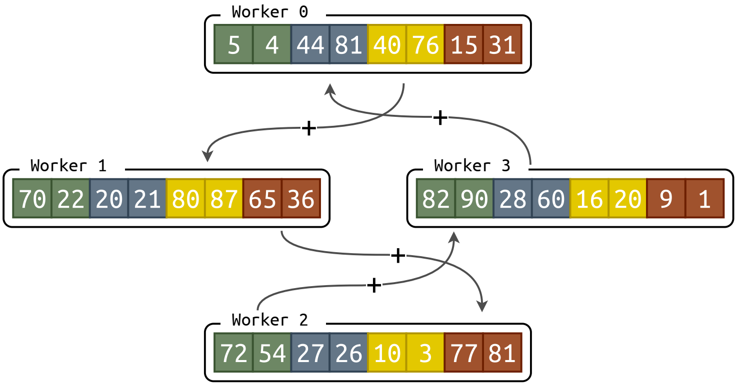


Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

ETH zürich CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

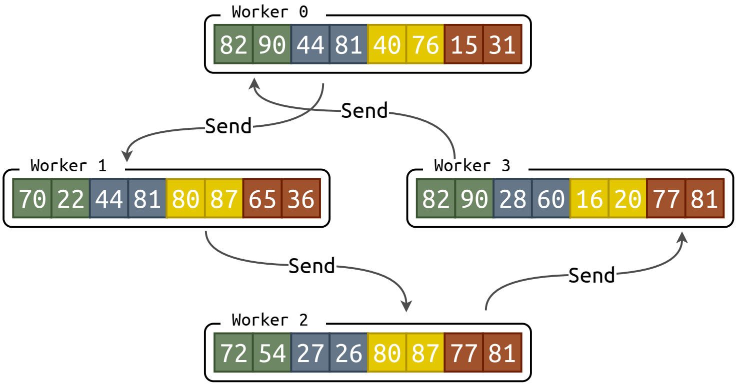
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

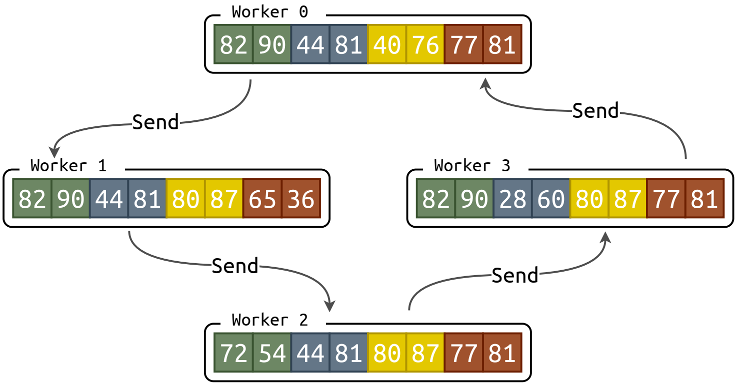
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

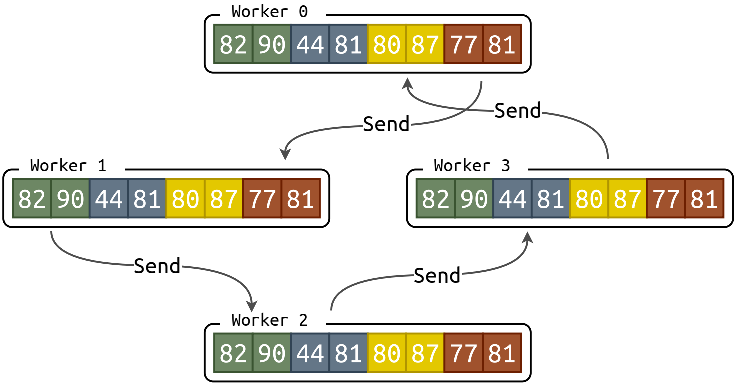
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

Ring Allreduce

- Each of the N workers communicates only with other two workers $2(N-1)$ times.
- The values of the reduction are obtained with the first $N-1$ communications.
- The second $N-1$ communications are performed to update the reduced values on all workers.
- The total amount of data sent by each worker $\left[2(N-1)\frac{\texttt{array\_size}}{N}\right]$ is virtually independent of the number of workers.

Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

Communication between Cray XC50 Nodes on Piz Daint

- Aries interconnect with the Dragonfly topology
- Direct communications between nodes on the same electrical group (2 cabinets / 384 nodes)
- Communications between nodes on different electrical groups passes by switches (submit with option `#SBATCH --switches=1` to make your job wait for a single-group allocation)
- More info on CSCS user portal

Horovod

Horovod is an open-source distributed training framework for TensorFlow, Keras, PyTorch, and MXNet developed by Uber. The goal of Horovod is to make distributed Deep Learning fast and easy to use.

Horovod

- Minimal code modification required
- Uses bandwidth-optimal communication protocols
- Seamless integration with Cray-MPICH and use of the NVidia Collective Communications Library (NCCL-2)
- Actively developed
- Growing community

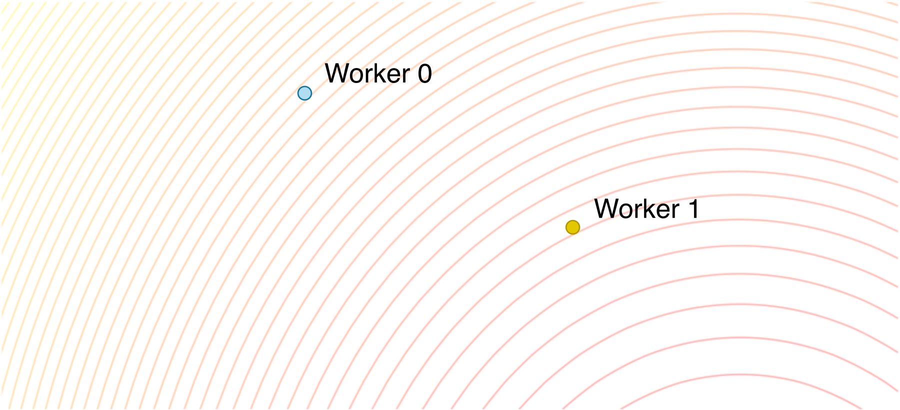
NVIDIA Collective Communications Library (NCCL)

NCCL implements multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs. NCCL provides routines such as Allgather, Allreduce and Broadcast, optimized to achieve high bandwidth over PCIe and NVLink high-speed interconnect.

Horovod: 1. Import and initialize the library (`tf.keras`)

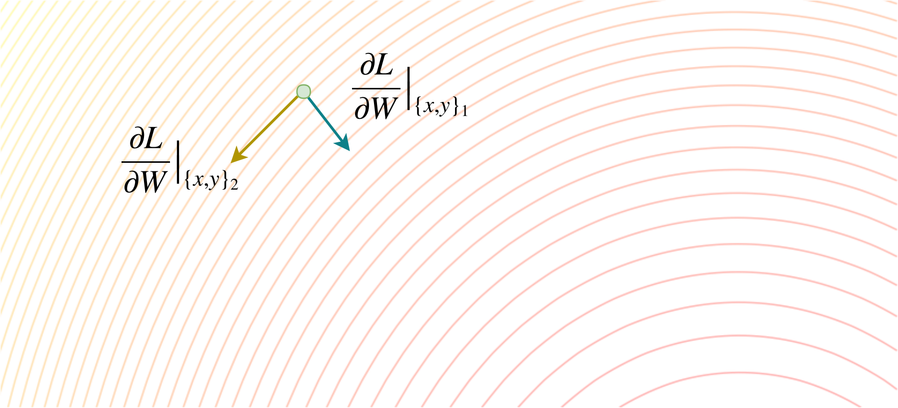
```
import horovod.tensorflow.keras as hvd

hvd.init()
```

Horovod: 2 and 3. Sync initial state and optimization

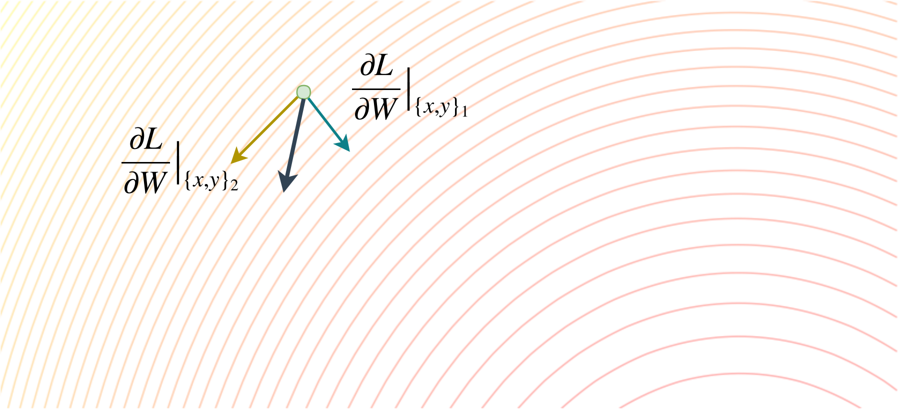


Horovod: 2 and 3. Sync initial state and optimization

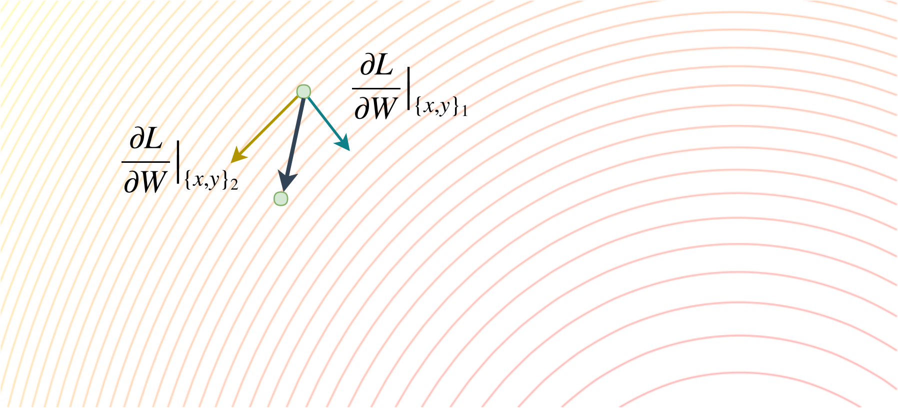
Horovod: 2 and 3. Sync initial state and optimization



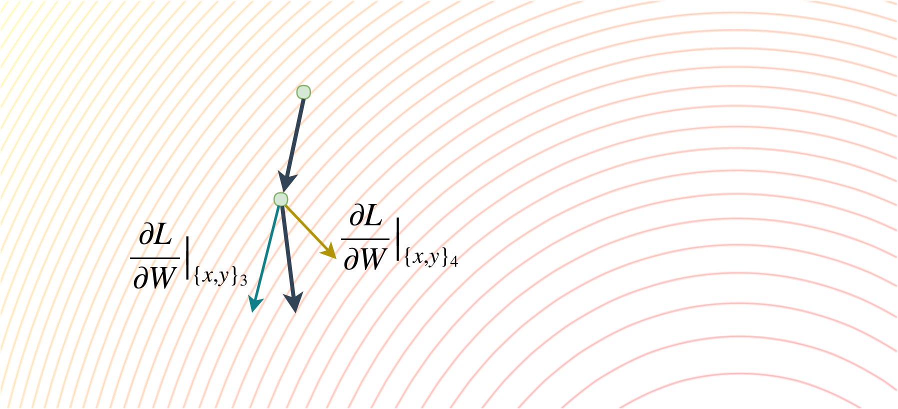
Horovod: 2 and 3. Sync initial state and optimization



Horovod: 2 and 3. Sync initial state and optimization



Horovod: 2 and 3. Sync initial state and optimization



Horovod: 2. Sync the initial state of the workers (`tf.keras`)

```
initial_sync = hvd.callbacks.BroadcastGlobalVariablesCallback(0)

model.fit(dataset, ..., callbacks=[initial_sync])
```

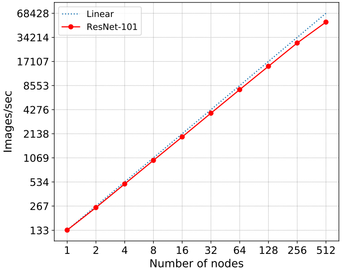
Horovod: 3. Wrap the optimizer with Horovod's one (`tf.keras`)

```
opt = tf.keras.optimizers.SGD(learning_rate)
opt = hvd.DistributedOptimizer(opt)
```

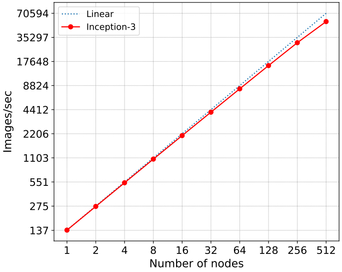
Horovod: 4. Checkpoints

```python
# Save checkpoints for the worker of rank 0.
# This will prevent all workers from corrupting a
# single checkpoint file.
if hvd.rank() == 0:
    ...
```

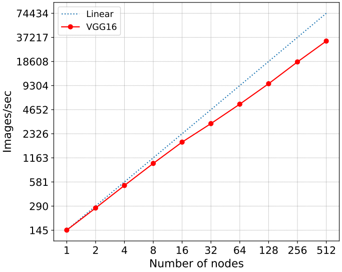
Benchmarks results on Piz Daint (CNNs on Imagenet)




```
num layers : 347
num weights: 44,601,832
```




```
num layers : 313
num weights: 23,817,352
```




```
num layers : 23
num weights: 138,357,544
```

https://github.com/tensorflow/benchmarks

Running TensorFlow + Horovod on Piz Daint

```bash
#!/bin/bash -l
#SBATCH --job-name=tf_hvd
#SBATCH --time=00:15:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-core=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=12
#SBATCH --hint=nomultithread
#SBATCH --constraint=gpu

module load daint-gpu
module load Horovod/0.19.1-CrayGNU-19.10-tf-2.2.0
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

export NCCL_DEBUG=INFO
export NCCL_IB_HCA=ipogif0
export NCCL_IB_CUDA_SUPPORT=1

srun python my_script.py
```

Some additional considerations

- Data must be split equally by workers to avoid load imbalance.
- If applicable, data can be split such that each worker does not need to read all files.
- Dataset splits resulting in non-homogeneous datasets may harm the convergence.
- Consider scaling the learning rate (`learning_rate * hvd.size()`)

Intuition on scaling the learning rate

$$L = \frac{1}{N}\sum_i^N l(\hat{y}_i, y_i)$$

$$W_{t+1}^{\text{SGD}} = W_t^{\text{SGD}} - \eta \frac{\partial L}{\partial W}\Big|_{\{x,y\}_t}$$

P. Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Intuition on scaling the learning rate

$$L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$$

$$W_{t+1}^{\text{SGD}} = W_t^{\text{SGD}} - \eta \frac{\partial L}{\partial W}\Big|_{\{x,y\}_t}$$

$$W_{t+1}^{\text{SGD}} = W_t^{\text{SGD}} - \frac{\eta}{N}\sum_{i\in t}^{N} \frac{\partial l}{\partial W}\Big|_{\{x,y\}_i}$$

P. Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Intuition on scaling the learning rate

$$W_{t+k}^{\text{SGD}} = W_t^{\text{SGD}} - \frac{\eta}{N} \sum_{j}^{k} \sum_{i \in t_j}^{N} \frac{\partial l}{\partial W}\Big|_{\{x,y\}_i}$$

P. Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Intuition on scaling the learning rate

$$W_{t+k}^{\text{SGD}} = W_t^{\text{SGD}} - \frac{\eta}{N} \sum_{j}^{k} \sum_{i \in t_j}^{N} \frac{\partial l}{\partial W}\Big|_{\{x,y\}_i}$$

$$W_{t+1}^{\text{distrSGD}} = W_t^{\text{distrSGD}} - \frac{\eta}{kN} \sum_{i \in t}^{kN} \frac{\partial l}{\partial W}\Big|_{\{x,y\}_i}$$

P. Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Intuition on scaling the learning rate

$$W_{t+k}^{\text{SGD}} = W_t^{\text{SGD}} - \frac{\eta}{N} \sum_{j}^{k} \sum_{i \in t_j}^{N} \frac{\partial l}{\partial W}\Big|_{\{x,y\}_i}$$

$$W_{t+1}^{\text{distrSGD}} = W_t^{\text{distrSGD}} - \frac{\eta}{kN} \sum_{i \in t}^{kN} \frac{\partial l}{\partial W}\Big|_{\{x,y\}_i}$$

$$W_{t+1}^{\text{distrSGD}} = W_t^{\text{distrSGD}} - \frac{k\eta}{kN} \sum_{i \in t}^{kN} \frac{\partial l}{\partial W}\Big|_{\{x,y\}_i}$$

P. Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

TensorFlow's distribution strategy

- TensorFlow includes support for synchronous distributed training using Ring Allreduce through `tf.distribute`.
- `tf.distribute` support training over multiple nodes with Slurm through `MultiWorkerMirroredStrategy` and `SlurmClusterResolver`.

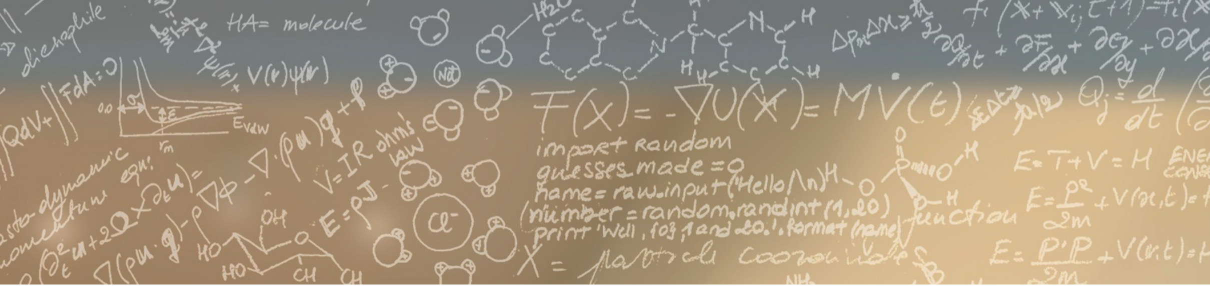
[lab] Simple Stochastic Gradient Descent with Horovod

The notebook `SGD/2-exercise-linear_regression_SGD_TF2-horovod.ipynb` uses the same model that we saw before. We will addapt it to Horovod and we will run it with 2 workers.

Visualize the trajectories before and after adding each Horovod modification. Try to understand why each line of Horovod is needed.

[lab] CNN on MNIST: Horovod and `tf.distribute`

Let's edit `SGD/mnist/01-mnist.ipynb` to run the training in 2 nodes.
Let's start first with the Horovod implementation. After that we will introduce TensorFlow's solution.



Thank you for your attention!