Efficient and Distributed training with TensorFlow on Piz Daint

Synchronous Distributed Training with TensorFlow and Horovod

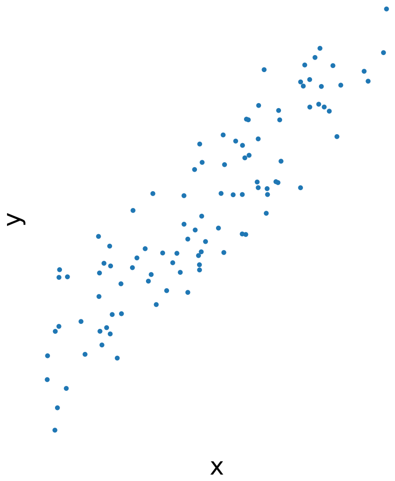
Rafael Sarmiento and Guilherme Peretti-Pezzi
ETHZürich / CSCS
Lugano, 14th-15th March 2019

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
- [lab] CNN models with `tf.keras` + Horovod
- [lab] CNN models with TensorFlow’s Estimator + Horovod

We want to train a model on this data



We choose a model and a cost function

$$y = mx + n$$

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

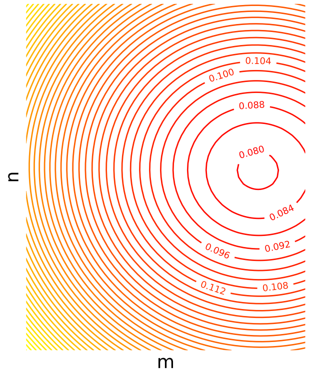
We choose a model and a cost function

$$y = mx + n$$

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

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

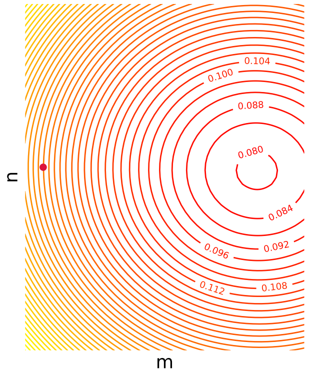
We choose a model and a cost function

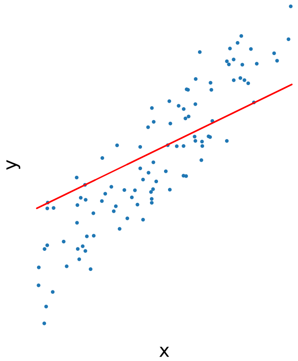


$$y = mx + n$$

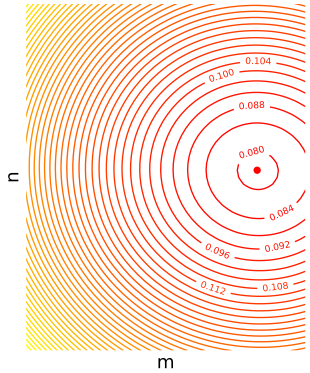
$$L = \frac{1}{N}\sum_{i}^{N}(mx_i + n - y_i)^2$$

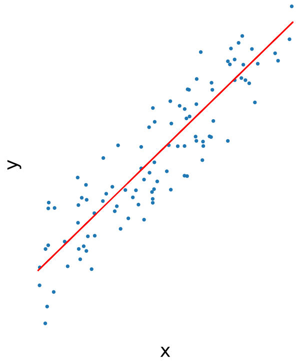
We need to choose an optimizer



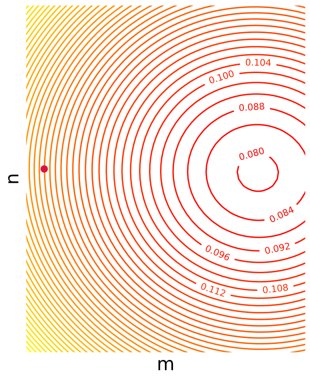


We need to choose an optimizer



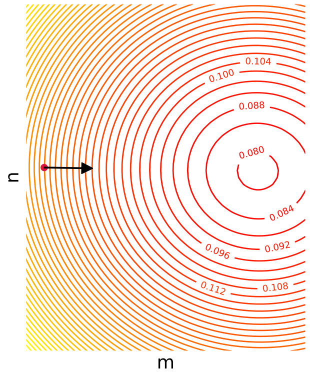


<Stochastic> Gradient Descent



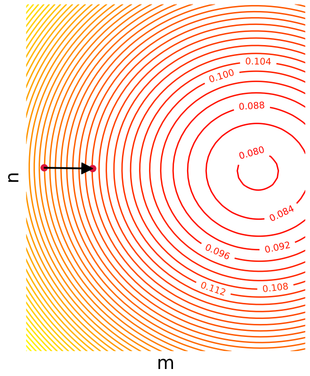
- Evaluate the loss function $L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$ for a batch of N samples $\{x, y\}$ (forward pass)

<Stochastic> Gradient Descent



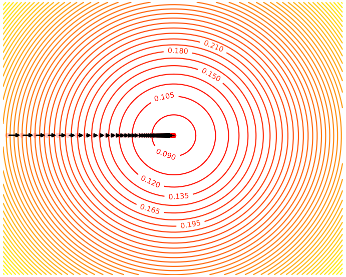
- Evaluate the loss function $L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$ for a batch of N samples $\{x, y\}$ (forward pass)
- Compute the gradients of the loss function with respect to the parameters of the model $\frac{\partial L}{\partial W}\Big|_{\{x,y\}}$ (backpropagation)

<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 $W_t = W_{t-1} - \eta \frac{\partial L}{\partial W}\Big|_{\{x,y\}_{t-1}}$

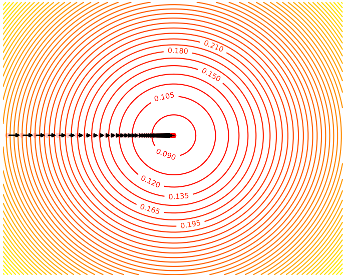
<Stochastic> Gradient Descent



Gradient
Descent

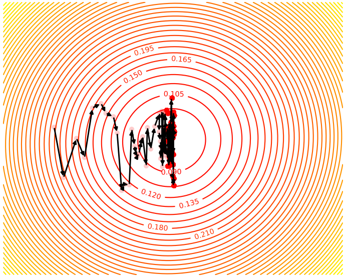
`batch_size = training_set_size`

<Stochastic> Gradient Descent



Gradient
Descent

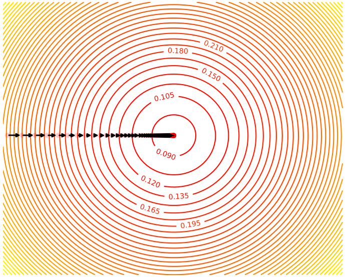
`batch_size = training_set_size`



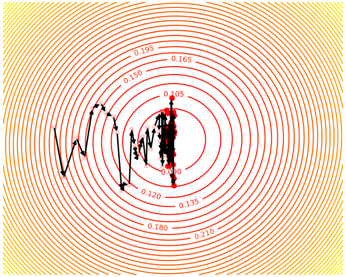
Stochastic Gradient
Descent

`batch_size = 1`

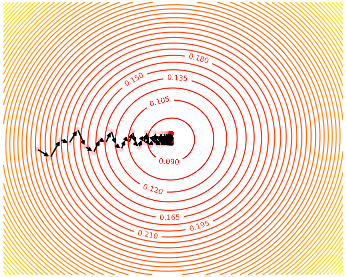
<Stochastic> Gradient Descent



Gradient
Descent
`batch_size = training_set_size`



Stochastic Gradient
Descent
`batch_size = 1`



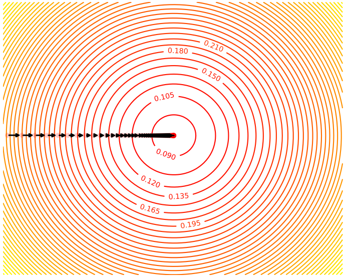
Minibatch Stochastic Gradient
Descent
`1 < batch_size < training_set_size`

[lab] Simple Stochasting Gradient Descent

Let’s run the notebook `SGD/simple_SGD_with_custom_model.ipynb`. There we use an unidimensional linear model to understand the trajectories of the SGD minimization.

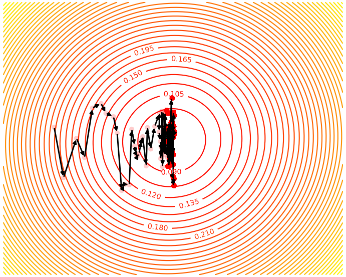
Try different batch sizes and see how the trajectory changes.

<Stochastic> Gradient Descent



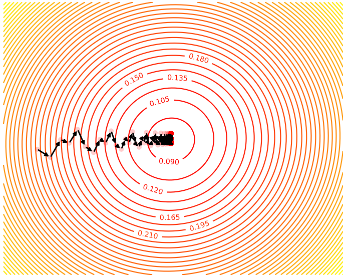
Gradient Descent

`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

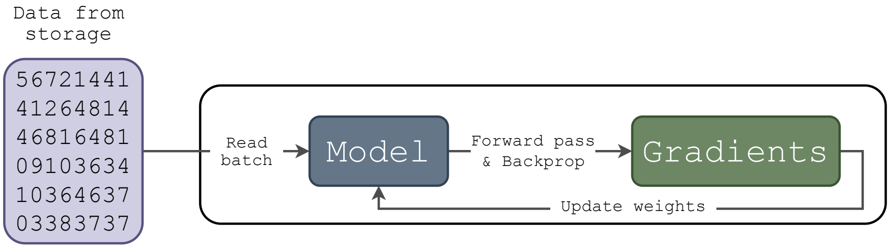
- The batch size is a hyperparameter

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

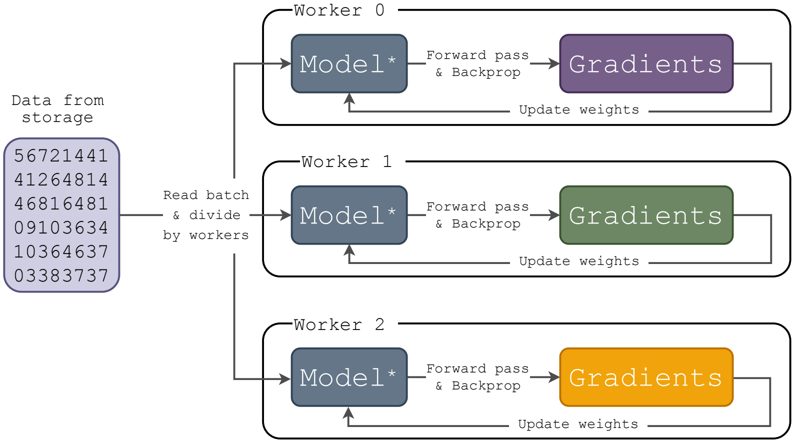
- The batch size is a hyperparameter
- Large batches may not fit on the GPU memory
- Splitting the training into multiple workers enables the use of large batches

- The batch size is a hyperparameter
- Large batches may not fit on the GPU memory
- Splitting the training into multiple workers enables the use of large batches
- A large batch size does not necessarily mean faster convergence

Distributing the training with data parallelism

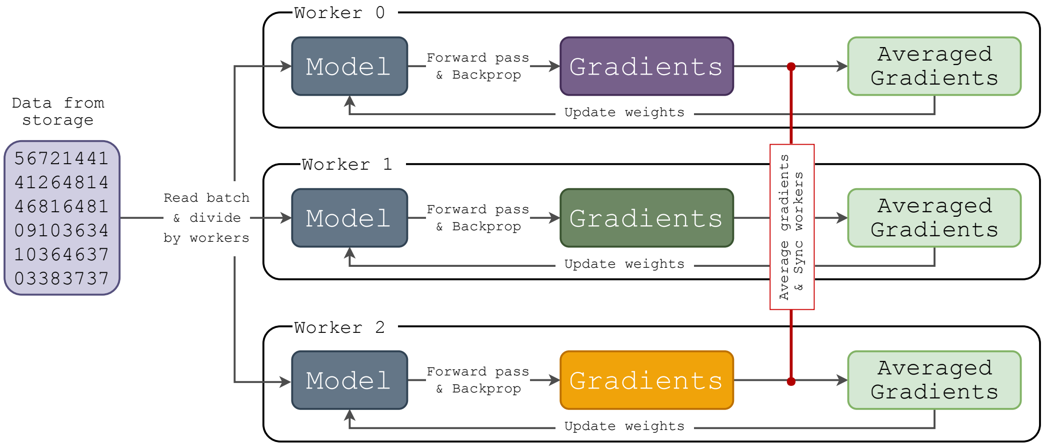


Distributing the training with data parallelism



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

Distributing the training with data parallelism

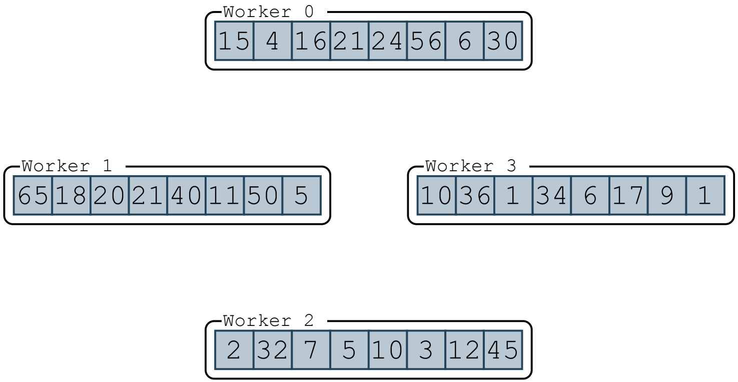


[1] A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

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.

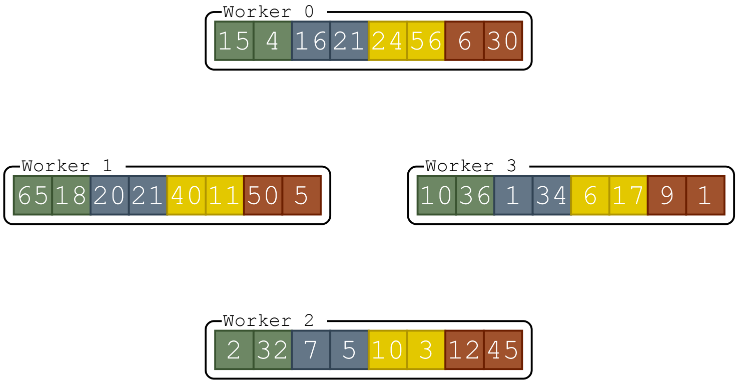
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

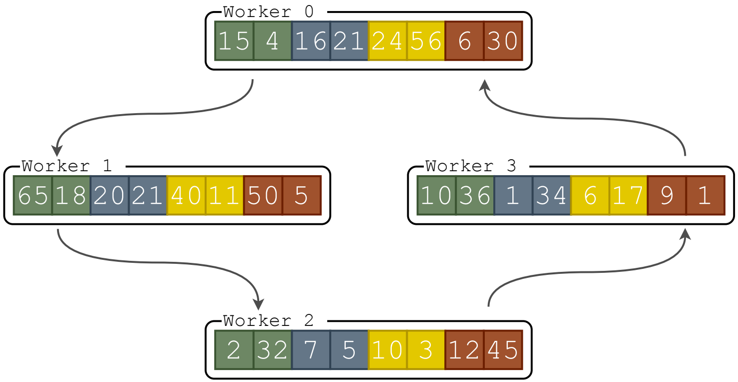
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

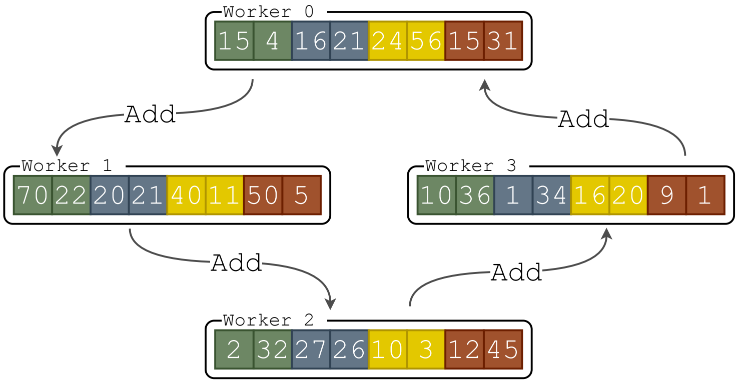
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

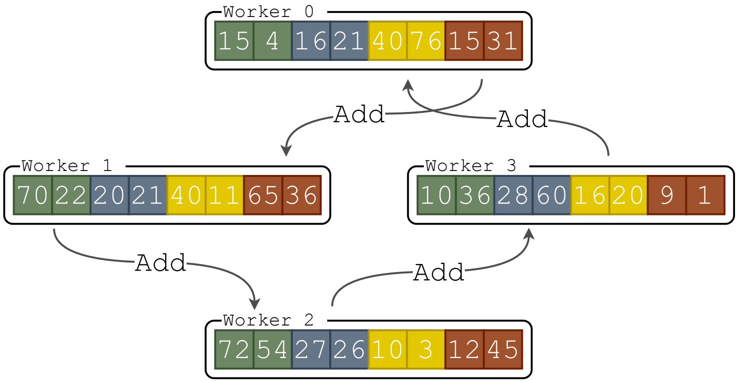
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

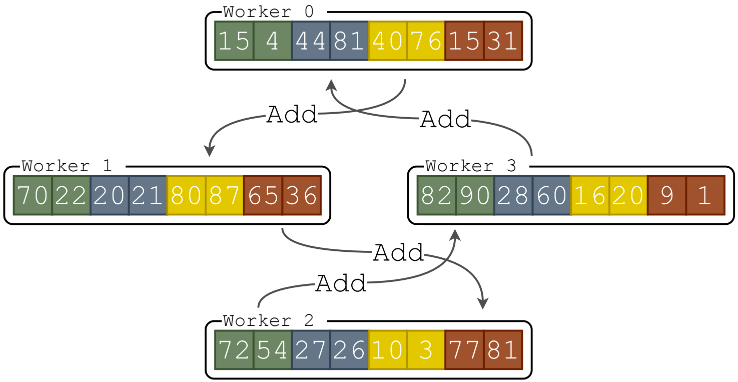
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

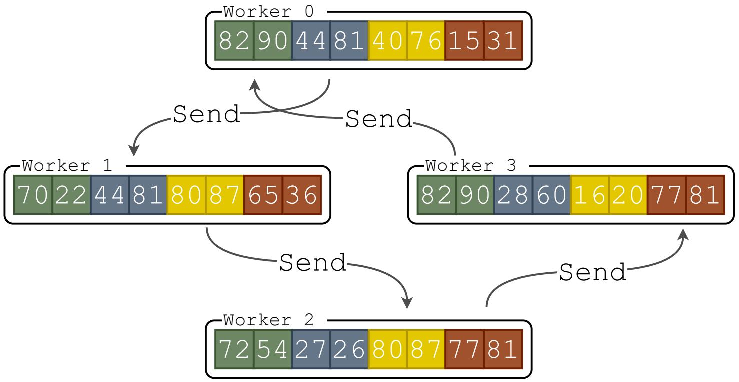
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

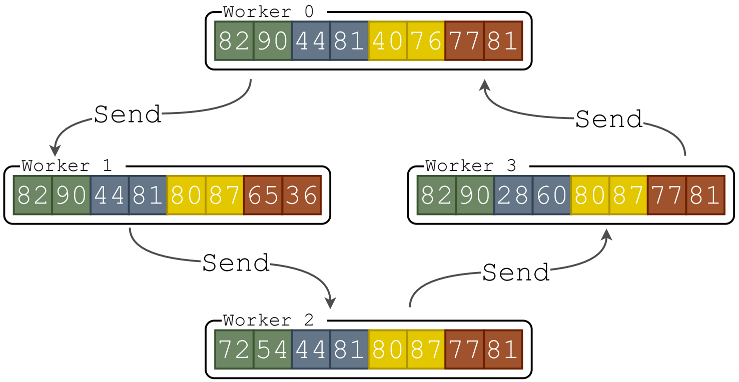
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

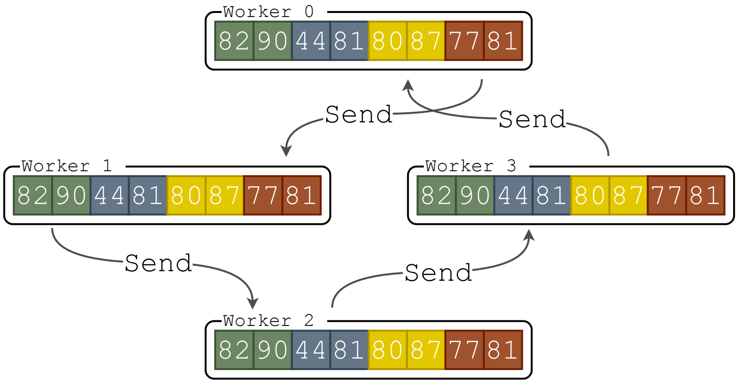
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

Ring Allreduce



[1] Baidu-Allreduce on GitHub

[2] 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 $\left[2(N-1)\frac{\text{ArraySize}}{N}\right]$ is virtually independent of the number of workers .

[1] Baidu-Allreduce on GitHub

[2] 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 `#BATCH --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 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.

Running TensorFlow + Horovod on Piz Daint

```
#!/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 ——constraint=gpu

module load daint—gpu
module load Horovod/0.16.0—CrayGNU—18.08—tf—1.12.0
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
```

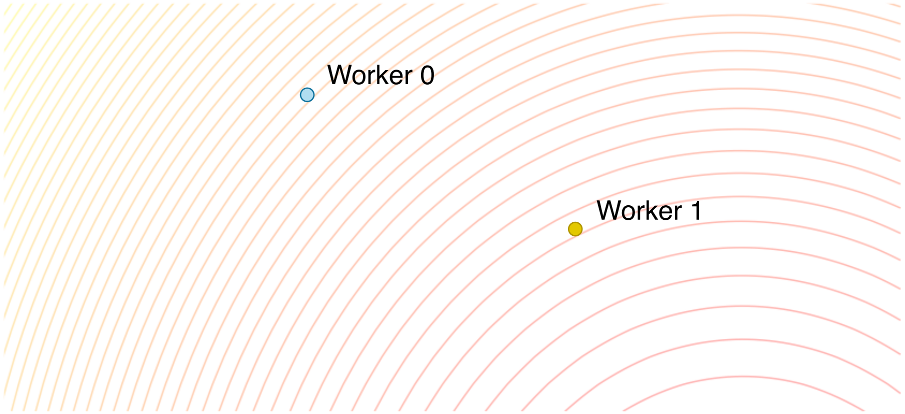
Horovod: 1. Initialize the library (TensorFlow)

```python
import horovod.tensorflow as hvd
hvd.init()
```

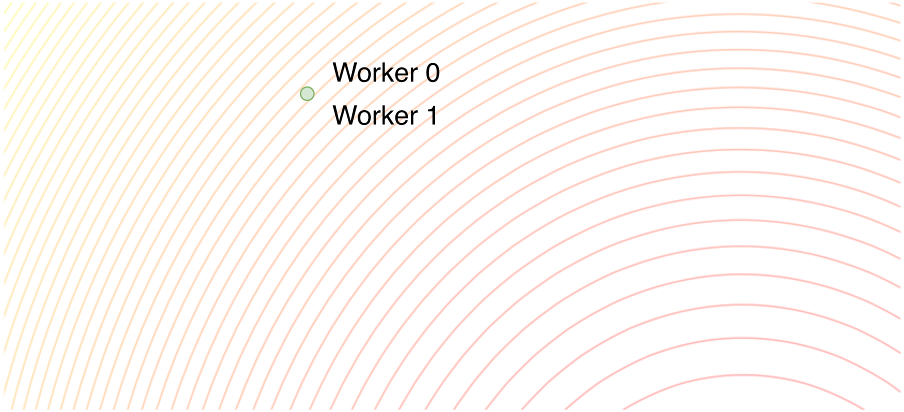
Horovod: 1. Initialize the library (`tf.keras`)

```python
import horovod.tensorflow.keras as hvd
hvd.init()
```

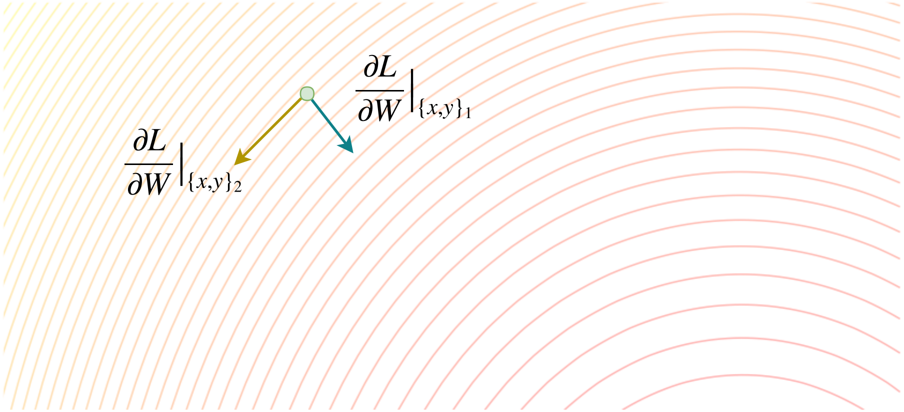
Horovod: 2. Sync initial state among workers



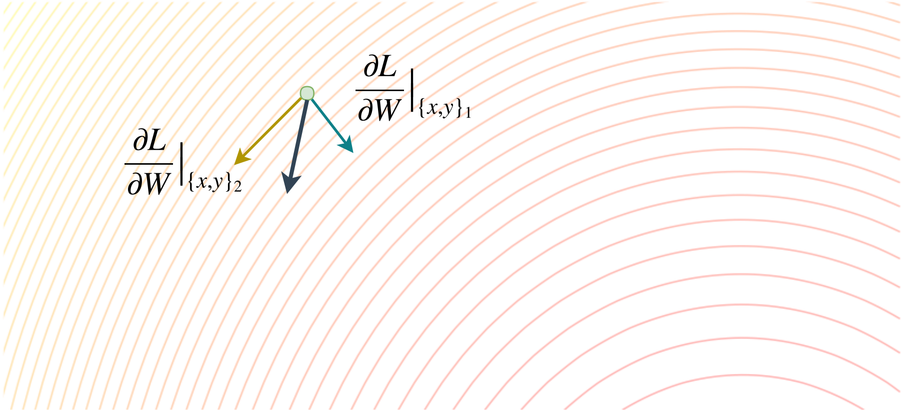
Horovod: 2. Sync initial state among workers



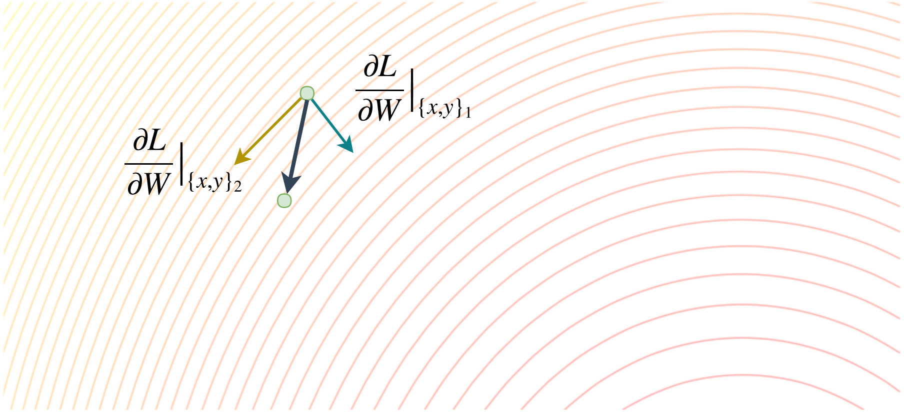
Horovod: 2. Sync initial state among workers


$\frac{\partial L}{\partial W}\Big|_{\{x,y\}_1}$
$\frac{\partial L}{\partial W}\Big|_{\{x,y\}_2}$

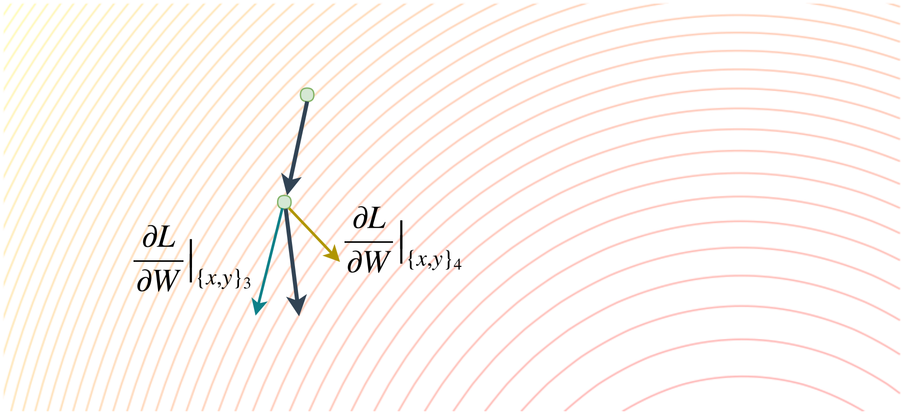
Horovod: 2. Sync initial state among workers


$\frac{\partial L}{\partial W}\Big|_{\{x,y\}_1}$
$\frac{\partial L}{\partial W}\Big|_{\{x,y\}_2}$

Horovod: 2. Sync initial state among workers


$\frac{\partial L}{\partial W}\Big|_{\{x,y\}_1}$
$\frac{\partial L}{\partial W}\Big|_{\{x,y\}_2}$

Horovod: 2. Sync initial state among workers


$\frac{\partial L}{\partial W}\Big|_{\{x,y\}_3}$
$\frac{\partial L}{\partial W}\Big|_{\{x,y\}_4}$

Horovod: 2. Sync initial state among workers (TensorFlow)

```
hooks = [hvd.BroadcastGlobalVariablesHook(0)]
with tf.train.MonitoredTrainingSession(hooks=hooks, ...) as sess:
    ...
```

Horovod: 2. Sync initial state among workers (TensorFlow - Estimator API)

```
estimator = tf.estimator.Estimator(...)

hooks = [hvd.BroadcastGlobalVariablesHook(0)]
estimator.train(input_fn=train_input_fn,
                steps=NUM_STEPS,
                hooks=hooks)
```

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

```
callbacks = [hvd.callbacks.BroadcastGlobalVariablesCallback(0)]
model.fit(dataset, ..., callbacks=callbacks)
```

Horovod: 3. Checkpoints

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

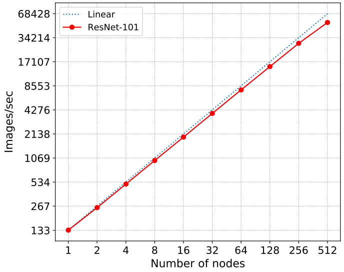
Horovod: 4. Wrap optimizer with Horovod's distributed one (TensorFlow)

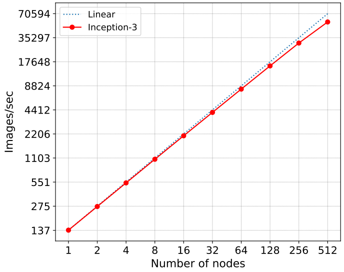
```
opt = tf.train.GradientDescentOptimizer(learning_rate)
opt = hvd.DistributedOptimizer(opt)
```

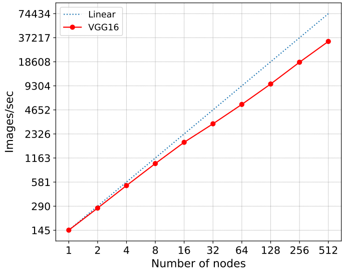
Horovod: 4. Wrap optimizer with Horovod's distributed one (`tf.keras`)

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

Benchmarks results on Piz Daint (CNNs on Imagenet)







[1] TensorFlow benchmarks

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

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

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

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

Intuition on scaling the learning rate

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

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

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

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

Intuition on scaling the learning rate

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

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

Intuition on scaling the learning rate

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

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

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

Intuition on scaling the learning rate

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

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

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

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

TensorFlow's distribution strategy

- TensorFlow-1.12.0 includes support for synchronous distributed training using Ring Allreduce with `CollectiveAllReduceStrategy` (already available, but still under development)
- Currently it involves the definition of an environment variable which is different for each worker

```
TF_CONFIG='{
    "cluster": {"worker": ["IP_NODE1:PORT", "IP_NODE2:PORT"]},
    "task": {"type": "worker", "index": WORKER_RANK}
}'
```

- It looks forward to involve as little code modification as possible
- An example is included with the lab material

[lab] Simple Stochastic Gradient Descent with Horovod

The script `hvd_simple_SGD_with_custom_model_exercise.py` uses the same model that we saw before on the notebook. We will addapt it to Horovod and we will run it with 2 workers.

```
# login with the -X option to open plot windows
ssh -X studxx@ela.cscs.ch
ssh -X studxx@daint.cscs.ch

alloc -C gpu -N 2 --res tensor2
module load daint-gpu
module load Horovod/0.16.0-CrayGNU-18.08-tf-1.12.0
cd tensorflow-training-cscs/SGD
srun python hvd_simple_SGD_with_custom_model_exercise.py
```

The script will save the trajectories of the two workers that can be visualized simply by running

```
python plot_hvd_simple_SGD_with_custom_model.py
```

Visualize the trajectories before and after adding each Horovod modification.

[lab] CNN models with `tf.keras` + Horovod

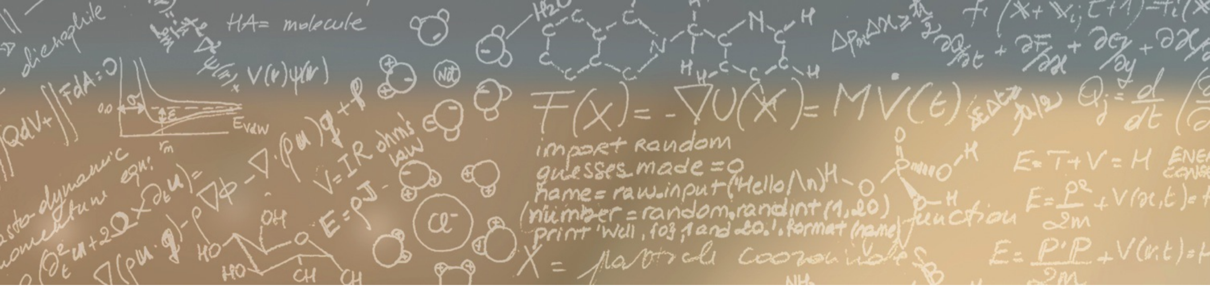
On `models_from_keras_applications` there are scripts to do simple training of popular Convolutional Neural Networks models on ImageNet. The scripts starting with `hvd_` contain Horovod code, while the other ones contain the equivalent single-node code. Addapt them to Horovod and run them in two nodes.

```
# login with SSH
ssh studxx@ela.cscs.ch
ssh studxx@daint.cscs.ch

cd models_from_keras_applications

alloc —C gpu —N 2 ——res tensor2
module load daint—gpu
module load Horovod/0.16.0—CrayGNU—18.08—tf—1.12.0
srun python keras_<model>_imagenet.py
```

[lab] CNN models with TensorFlow's Estimator + Horovod

On `custom_estimators_from_benchmaks_models` there are scripts to do simple training of popular Convolutional Neural Networks models on ImageNet. The scripts starting with `hvd_` contain Horovod code, while the other ones contain the equivalent single-node code. Addapt them to Horovod and run them in two nodes.

```
cd custom_estimators_from_benchmaks_models
# download the models from https://github.com/tensorflow/benchmarks
bash get_models_from_tfbenchmarks.sh
cd models_from_benchmark

alloc —C gpu —N 2 ——res tensor2
module load daint—gpu
module load Horovod/0.16.0—CrayGNU—18.08—tf—1.12.0
cp ../estimator_<model>_imagenet.py .
srun python estimator_<model>_imagenet.py
```



Thank you for your attention!