Synchronous Distributed Training with TensorFlow and Horovod

Rafael Sarmiento
ETHZürich / CSCS
CSCS-USI Summer School 2019

TensorFlow is an open source software library for numerical computation using data flow graphs. It serves as an end-to-end platform for machine learning with a comprehensive, flexible ecosystem of tools, libraries and community resources.

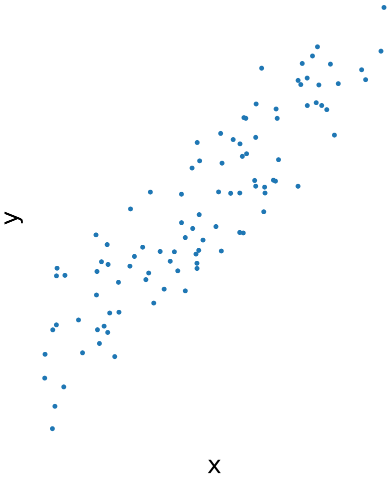
- Within TensorFlow’s data flow graphs, nodes represent mathematical operations, while the edges represent multidimensional data arrays (tensors) that flow between them.
- TensorFlow provides APIs for Python, C, C++, Go, Java, JavaScript, and Swift.

- TensorFlow can function in two ways: **Graph mode**, where the graph is built by the user and is executed later in the code, and **Eager Mode** where the construction of the graph is hidden from the user and every line of code is executed in-place.
- TensorFlow-2.0 (beta) is being released and now it coexists with TensorFlow-1.x.
- TensorFlow-2.0 works in Eager Mode.

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] A CNN model with `tf.keras` + Horovod

We want to train a model on this data



We choose a model and a cost function

$$y = mx + n$$

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

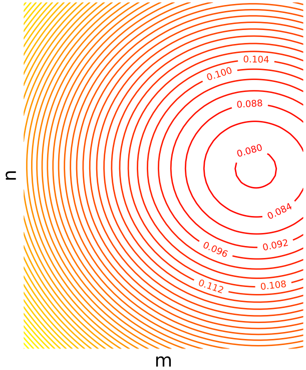
We choose a model and a cost function

$$y = mx + n$$

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

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

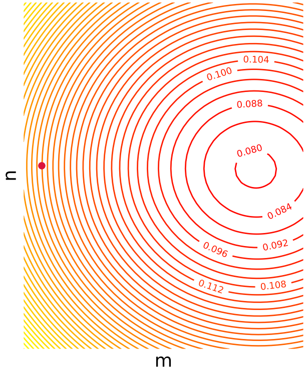
We choose a model and a cost function

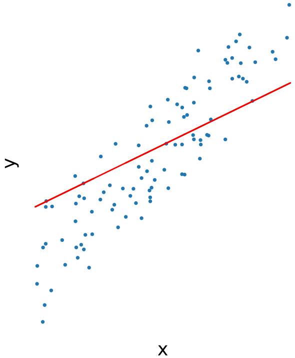


$$y = mx + n$$

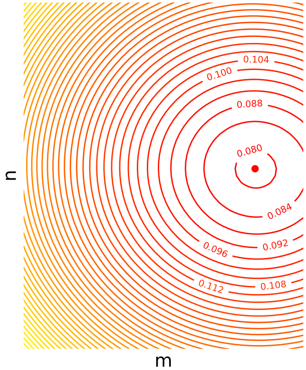
$$L = \frac{1}{N}\sum_{i}^{N}(mx_i + n - y_i)^2$$

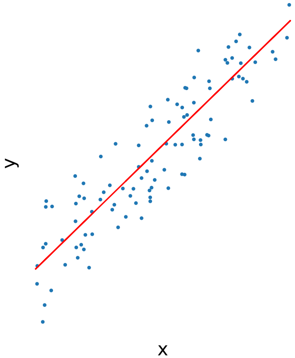
We need to choose an optimizer



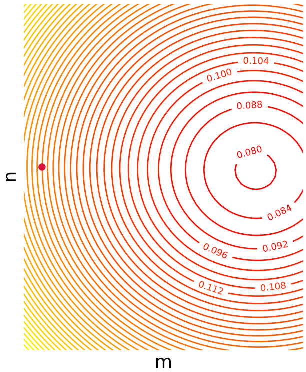


We need to choose an optimizer



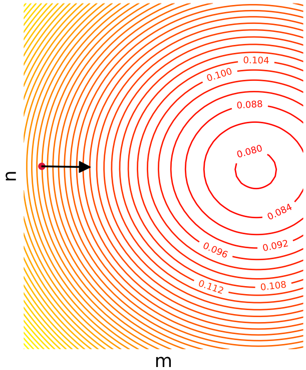


<Stochastic> Gradient Descent



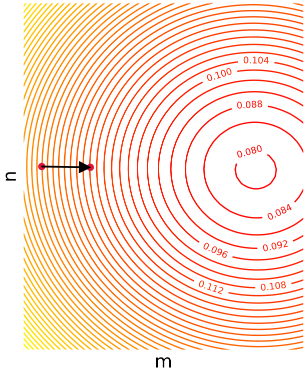
- Evaluate the loss function $L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$ for a batch of N samples $\{x, y\}$ (forward pass)

<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



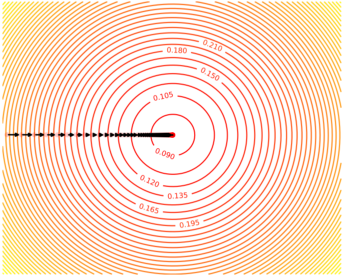
- Evaluate the loss function $L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$ for a batch of N samples $\{x, y\}$ (forward pass)

- Compute the gradients of the loss function with respect to the parameters of the model $\left.\frac{\partial L}{\partial W}\right|_{\{x,y\}}$ (backpropagation)

- Update the parameters $W_t = W_{t-1} - \eta \left.\frac{\partial L}{\partial W}\right|_{\{x,y\}_{t-1}}$

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

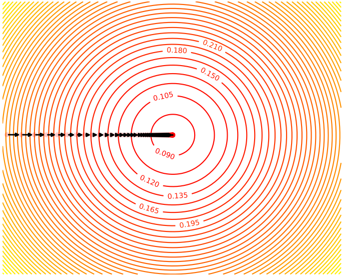
<Stochastic> Gradient Descent



Gradient
Descent

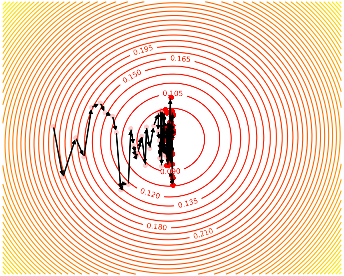
`batch_size = training_set_size`

<Stochastic> Gradient Descent



Gradient
Descent

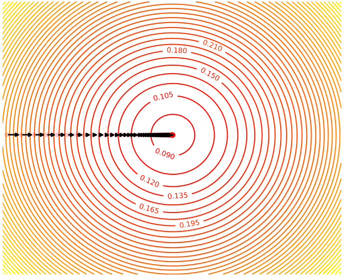
`batch_size = training_set_size`



Stochastic Gradient
Descent

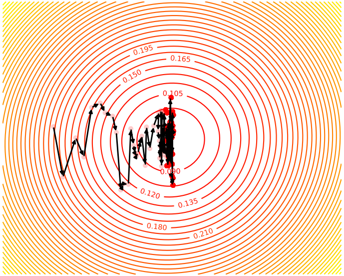
`batch_size = 1`

<Stochastic> Gradient Descent



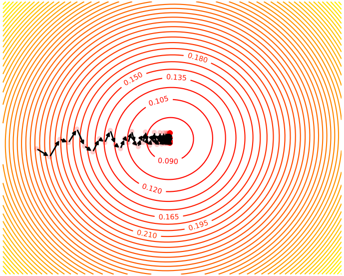
Gradient
Descent

`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 see together the following notebooks

- `linear_regression_SGD_TF-1.x-session.ipynb` - Traditional TensorFlow
- `linear_regression_SGD_TF-1.x-eager.ipynb` - TensorFlow-1.x in Eager mode
- `linear_regression_SGD_TF-2.0-keras.ipynb` - TensorFlow-2.0
- `linear_regression_SGD_TF-2.0-keras-advanced.ipynb` - TensorFlow-2.0

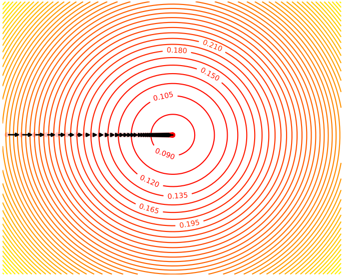
Please, create the file `$HOME/.jupyterhub.env` and add the following:

```
module use /apps/daint/UES/6.0.UP04/sandboxes/tensorflow—sumsch/modules/all
module load TensorFlow/2.0.0—beta1—CrayGNU—18.08—cuda—9.2—python3
```

There we use an unidimensional linear model to understand the trajectories of the SGD minimization.

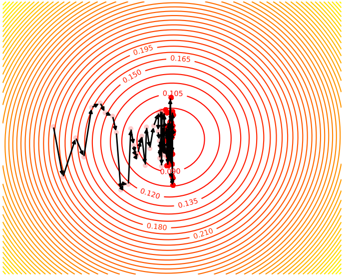
Try different batch sizes and see how the trajectory changes.

<Stochastic> Gradient Descent



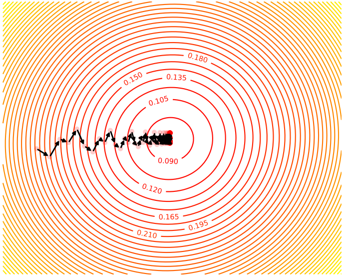
Gradient
Descent

`batch_size = training_set_size`



Stochastic Gradient
Descent

`batch_size = 1`



Minibatch Stochastic Gradient
Descent

`1 < batch_size < training_set_size`

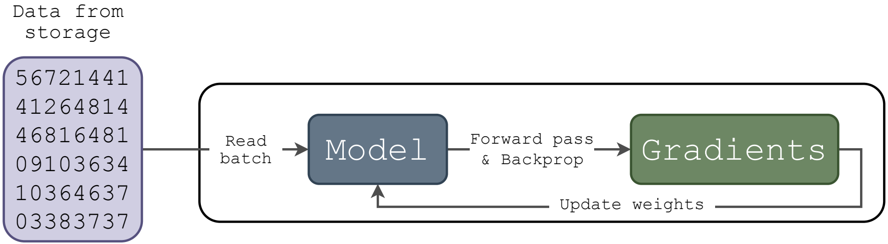
- The batch size is a hyperparameter

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

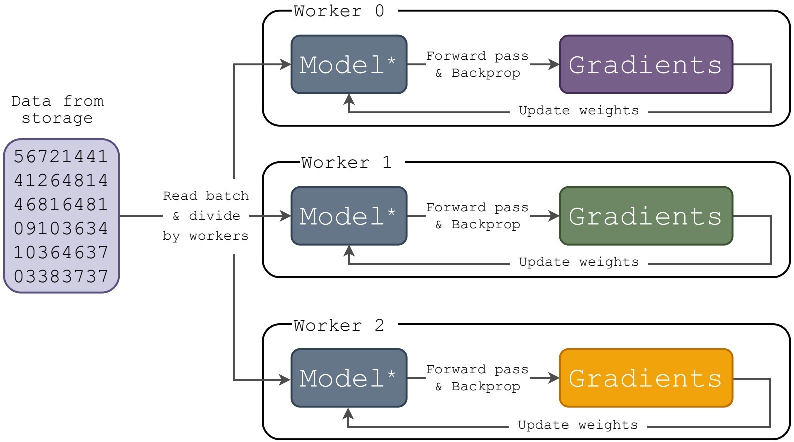
- The batch size is a hyperparameter
- Large batches may not fit on the GPU memory
- Splitting the training into multiple compute nodes enables the use of large batches

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

Distributing the training with data parallelism

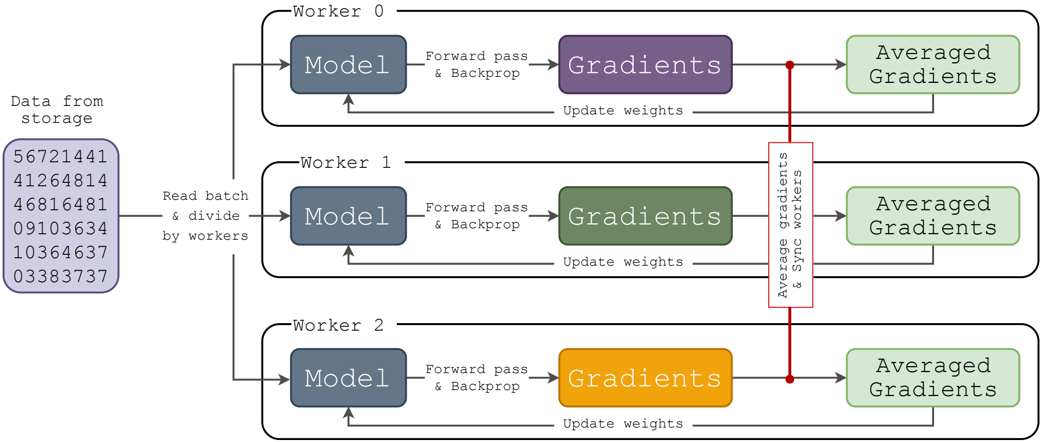


Distributing the training with data parallelism



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

Distributing the training with data parallelism

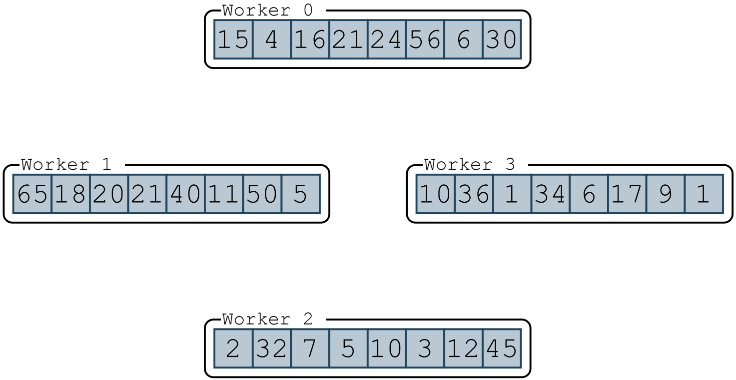


[1] A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

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.

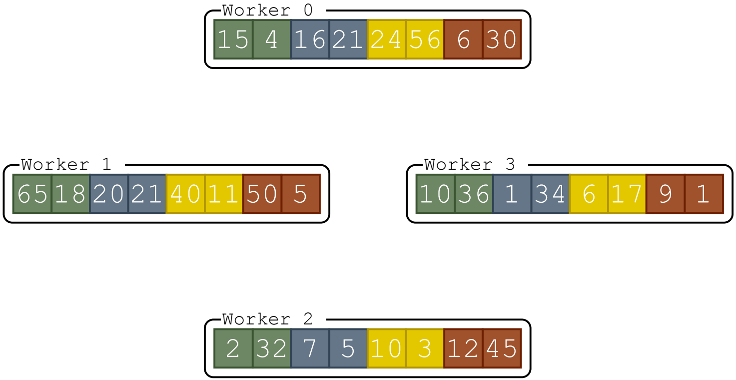
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

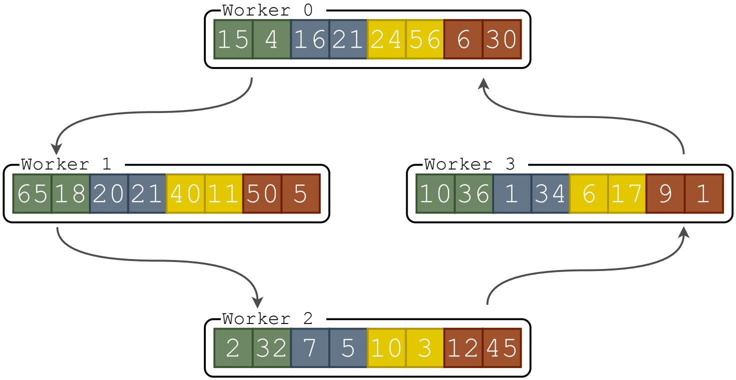
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

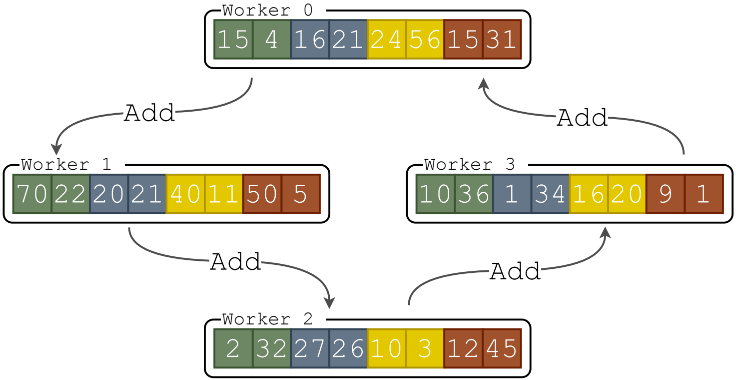
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

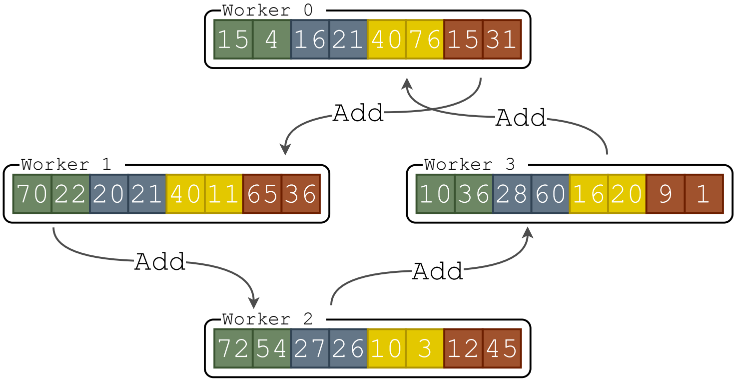
Ring Allreduce



1 Baidu-Allreduce on GitHub

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

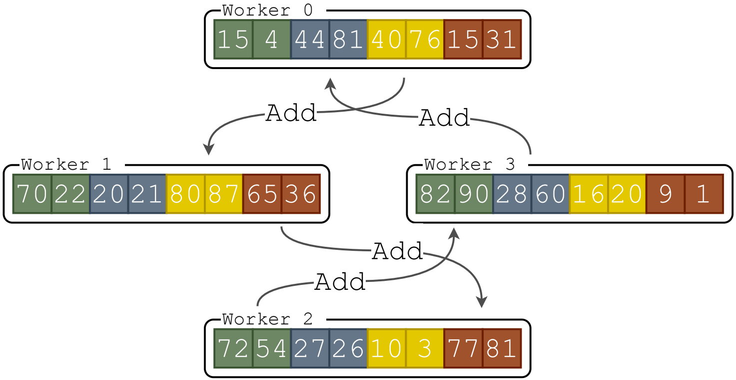
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

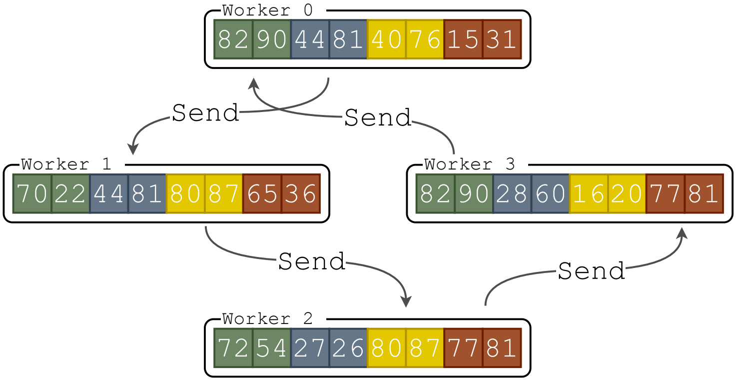
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

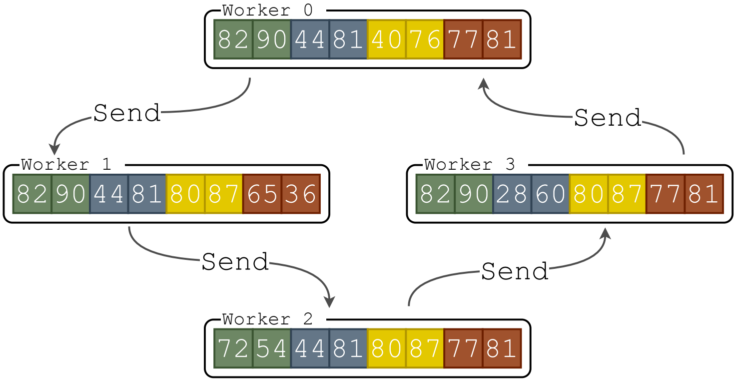
Ring Allreduce



[1] Baidu-Allreduce on GitHub

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

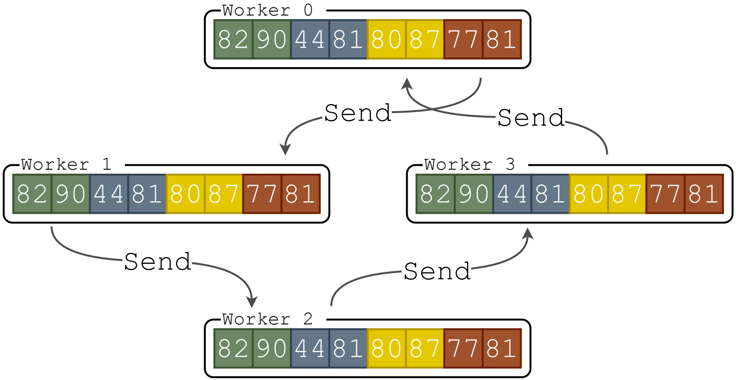
Ring Allreduce



[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

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
```

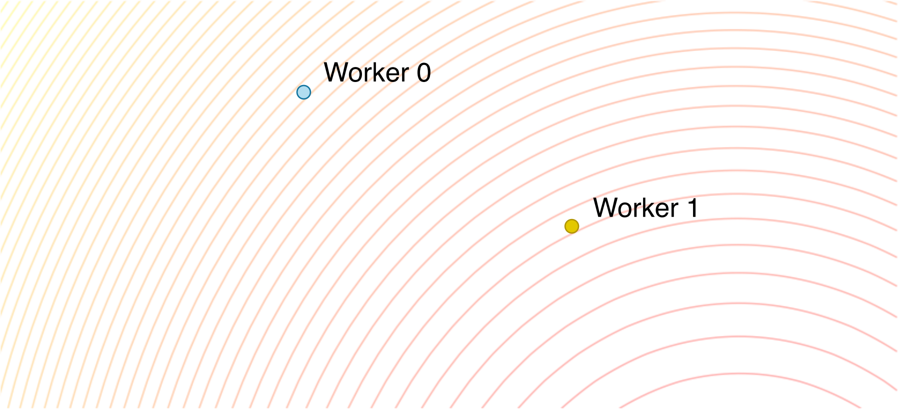
Horovod: 1. Initialize the library (TensorFlow)

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

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

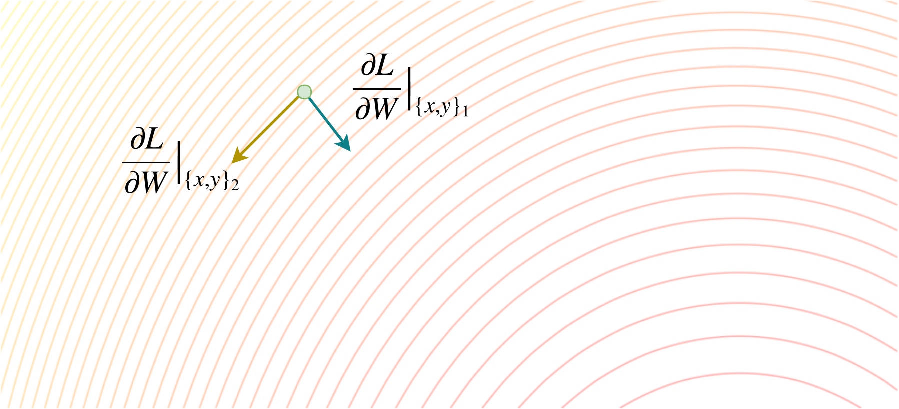
```python
import horovod.tensorflow.keras as hvd
hvd.init()
```

Horovod: 2. Sync initial state among workers

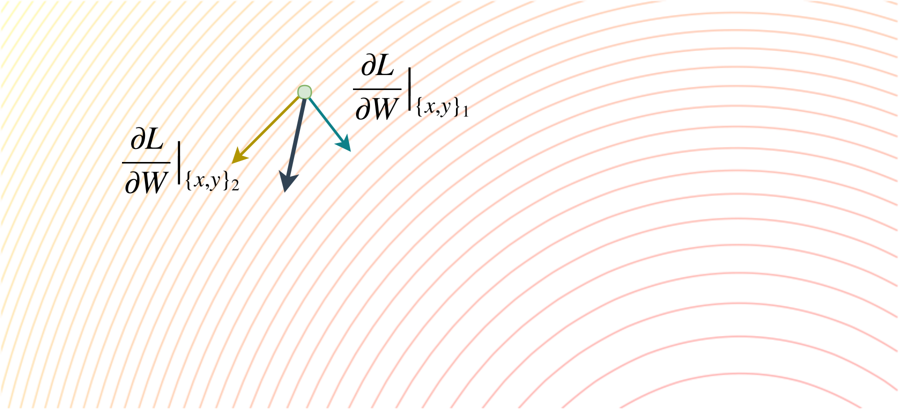


Horovod: 2. Sync initial state among workers

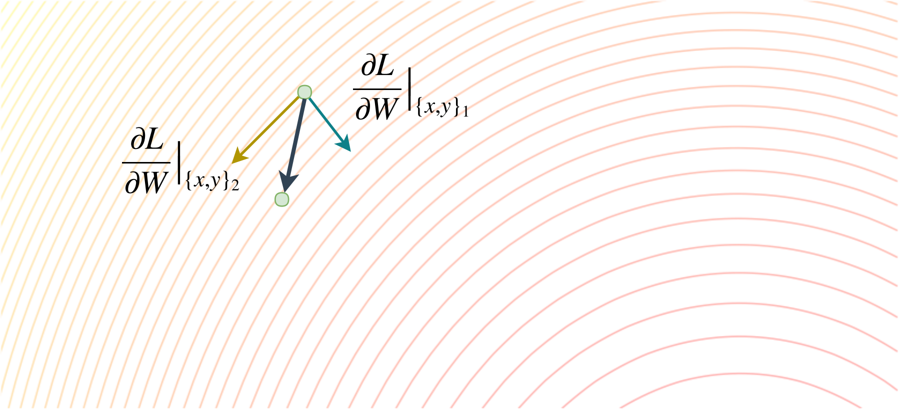
Horovod: 2. Sync initial state among workers



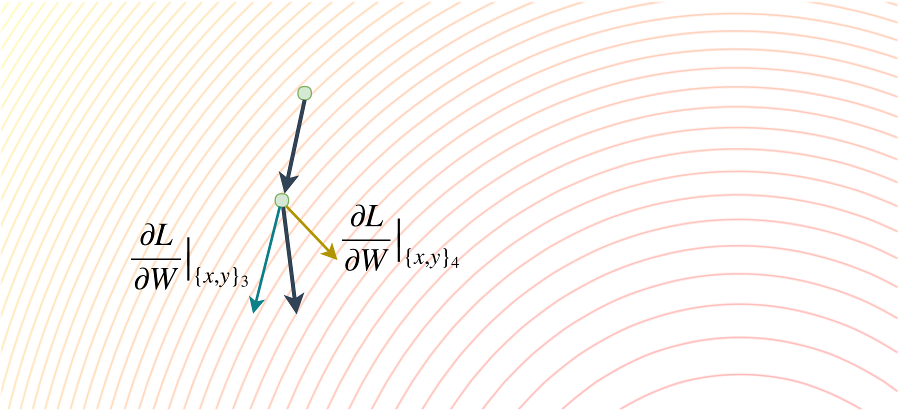
Horovod: 2. Sync initial state among workers



Horovod: 2. Sync initial state among workers



Horovod: 2. Sync initial state among workers



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

```
hvd.broadcast_variables(list_of_model_variables, root_rank=0)
# ex. hvd.broadcast_variables([slope, offset], root_rank=0)
```

Horovod: 3. 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:
    ...
```

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`)

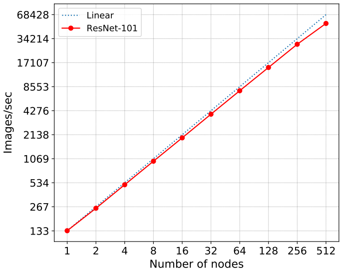
```
opt = tf.keras.optimizers.SGD(learning_rate)
opt = hvd.DistributedOptimizer(opt)
```

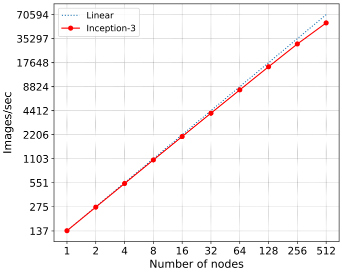
Horovod: 4. Wrap the gradient tape with Horovod's distributed one (TensorFlow Eager)

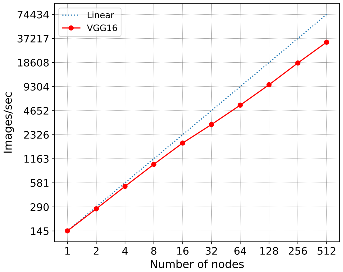
```
with tf.GradientTape() as tape:
    ...

tape = hvd.DistributedGradientTape(tape)
```

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 in https://github.com/eth-cscs/tensorflow-training

[lab] Simple Stochastic Gradient Descent with Horovod

Here we will addapt the model that we saw before to Horovod and 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

salloc -C gpu -N 2 --res summer_school
module load daint-gpu
module load Horovod/0.16.0-CrayGNU-18.08-tf-1.12.0

srun python hvd_linear_regression_SGD_TF-1.x-<eager/session>_exercise.py
```

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

```
python plot_hvd_SGD.py
```

This will save a figure in png format that you can copy to your computer. Alternatively you can edit the script to plot it directly.
Visualize the trajectories before and after adding each Horovod modification.

[lab] A CNN model with `tf.keras` + Horovod

On `keras_applications` there's a script to do simple training of ResNet50 Convolutional Neural Networks model 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.

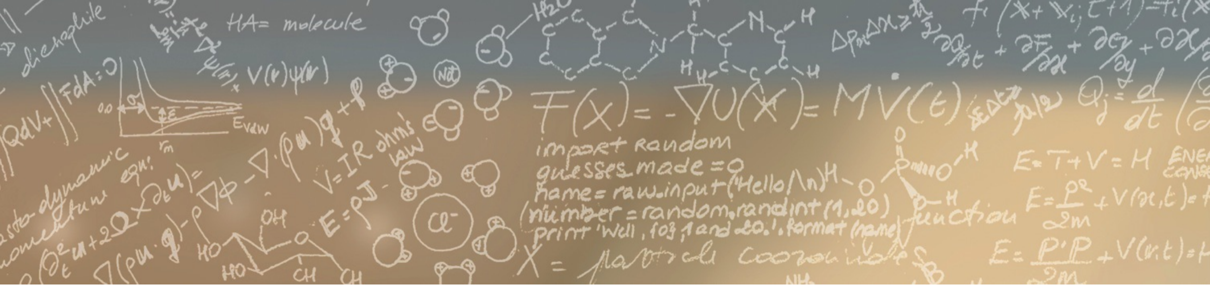
```
# login with SSH
ssh studxx@ela.cscs.ch
ssh studxx@daint.cscs.ch

cd models_from_keras_applications

salloc -C gpu -N 2 --res summer_school
module load daint-gpu
module load Horovod/0.16.0-CrayGNU-18.08-tf-1.12.0
srun python keras_resnet50_imagenet.py
```

More examples

https://github.com/eth-cscs/tensorflow-training
https://github.com/horovod/horovod/tree/master/examples



Thank you for your attention!