Neural Network Distributed Training and Optimization

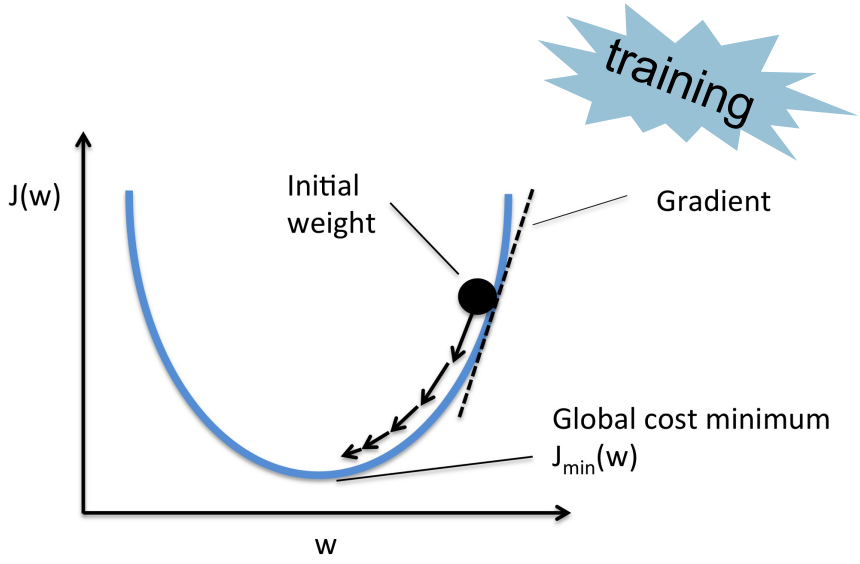
4th ATLAS Machine Learning Workshop Nov 11-15, 2019

Jean-Roch Vlimant, with many others

Outline

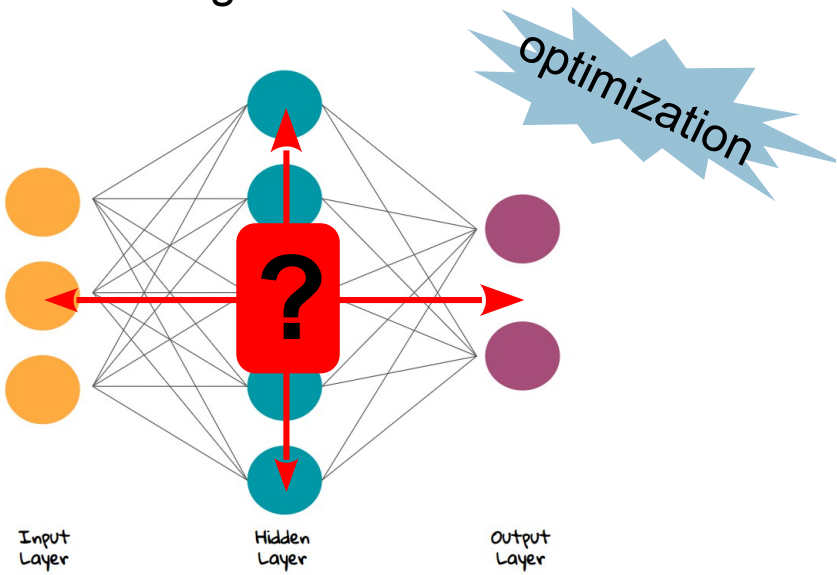
- Motivations
- Training-workload parallelism
- Hyper-parameters optimization
- Interface overview
- Discussion on Performance
- Summary and Outlooks

Training & Optimization



Parameters of the models (weights, bias, ...) are learning with respect to a loss that is optimized with (stochastic) gradient descent.
Only tractable if gradients can be computed analytically.

Models also contains parameters that are best optimized otherwise (stride, number of neurons, number of layers, …).
Commonly optimized by grid scan, or with bayesian optimization using gaussian processes (Or by grad-student descent ...)
Other evolutionary methods with transfer learning out there.



Motivations

- Large models on large dataset can take days-week to converge on single GPU.
- Simpler models can take as long to converge, on CPU-only hosts.
- Prototyping with model architecture is like testing a new idea for analysis, you want to have the answer “fast”
- Dismissing large model, large dataset because of train time ?

Community Interest

- E4/Nvidia/Openlab project
  - M. Girone, M. Pierini, V. Loncar, …
  - https://indico.cern.ch/event/784202/
  - Access to Flatiron/SDSC cluster
- Exa.TrkX DOE: scaling tracking GNN training at NERSC
  - S. Farrell, P. Calafiura, J. Kowalkowski, ...
  - Allocations on Cory, Cory GPU, Summit
- HEPGan NESAP: scaling/developing calorimeter GAN
  - B. Nachman, W. Bhimji, S. Vallecorsa, ...
  - Allocation on Cory, Cory GPU
  - Pending hiring a postdoc at NERSC https://inspirehep.net/record/1733162
- BNL: study scaling of various training frameworks
  - A. Malik, ...
  - Allocation on summit
- IRIS-HEP:
  - FastML workshop https://indico.cern.ch/event/822126
  - “Industry tools vs in-house development” dilemna
- ATLAS & CMS ML groups:
  - Interest in training as a service, integration of training in experiment workflow management

Project History

- Started with RNN acceleration: easgd, downpour.
  https://arxiv.org/abs/1712.05878
  https://github.com/vlimant/mpi_learn
- Included Horovod with multiple rings
  https://github.com/horovod/horovod/pull/394
- Extended to hyper optimization
  https://github.com/vlimant/mpi_opt
- Incorporated torch backend
- Integrated GEM
  from https://arxiv.org/abs/1805.08469
- Interface for GAN
  https://doi.org/10.1051/epjconf/201921406025
- ANN python model interface
- Repository consolidation
  https://github.com/vlimant/NNLO
- Complete checkpointing for short HPC queues
- ...

Goal

- Aiming for a “plug and play” software for model optimization
  - Training as a service
  - HPC as a service
  - Workflow management system integration
  - Deploy on system with mpi

Package Features

- Provide model in Torch or Keras (TF upcoming)
- Data caching (also used with ec3) , pre-loading
- Data adaptor (from stored tensors to model input)
- Distributed training engine : easgd, downpour, gem
- Early stopping mechanism
- Model parallelism : limited, only with keras
- Hyper-optimization engine : GP-opt, evolutionary algo
- Cross-validated hyper-optimzation
- Tracking CPU/GPU utilization
- Profiling with tracing function calls
- Checkpointing (save/restore) for long optimization

- Some limitations (# of gpu per process, node/rank association, process crash on gpu memory, ...)

Distributed Training

Parallelism Overview

- Data distribution

  Compute the gradients on several batches independently and update the model synchronously or not. **Applicable to large dataset**

- Gradient distribution

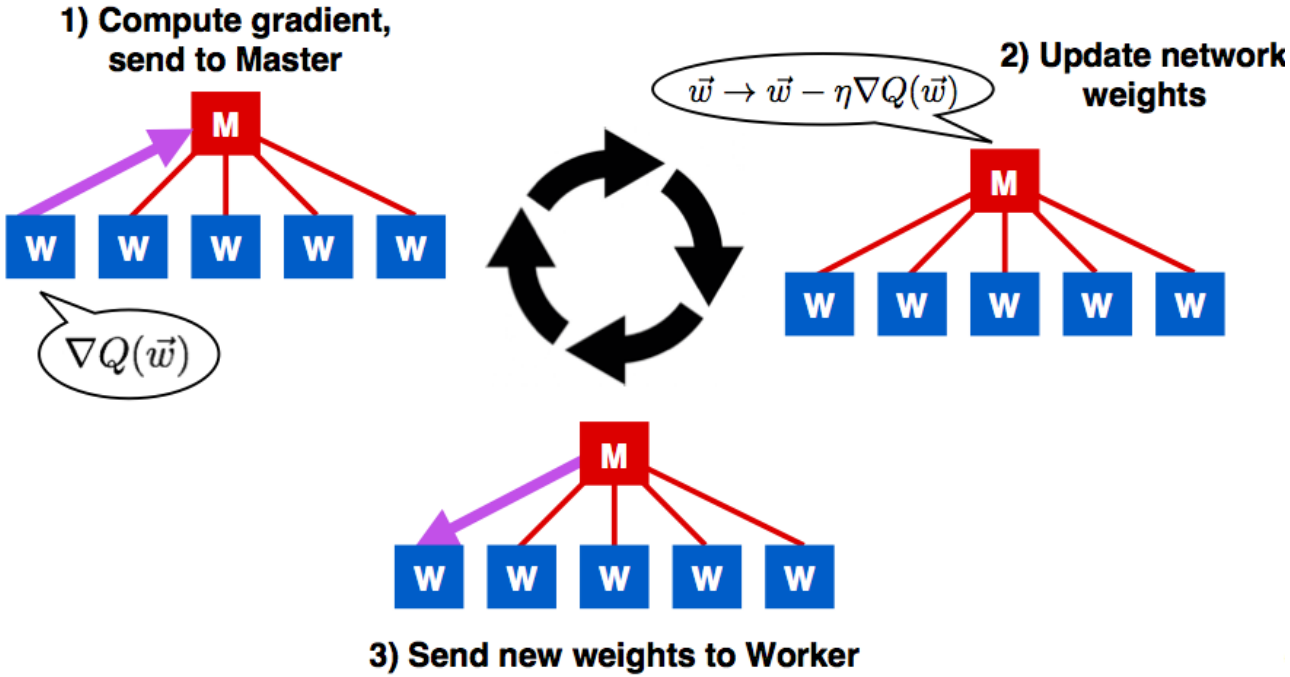
  Compute the gradient of one batch in parallel and update the model with the aggregated gradient. **Applicable to large sample ≡ large event**

- Model distribution

  Compute the gradient and updates of part of the model separately in chain. **Applicable to large model**

Data Distribution

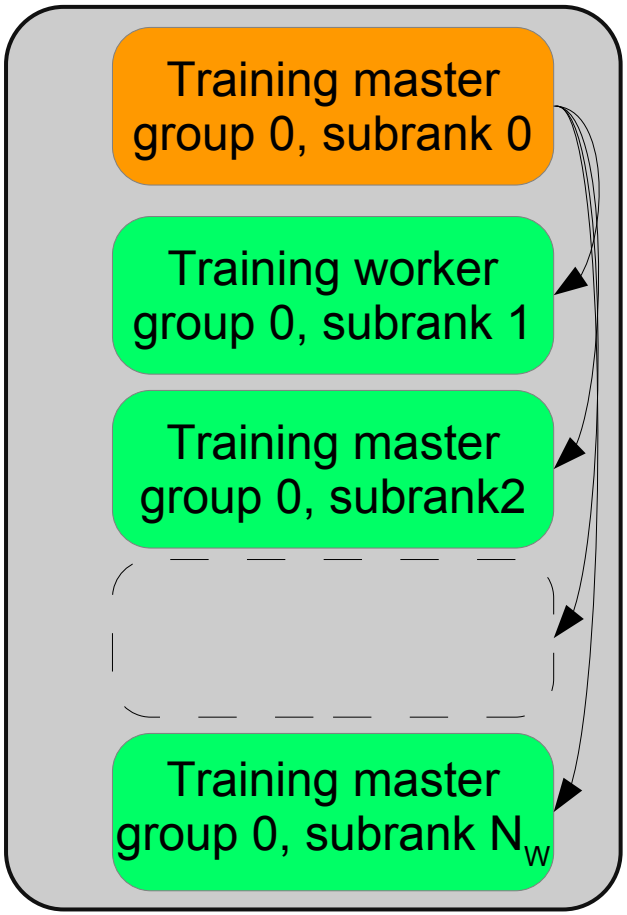
Data Distribution



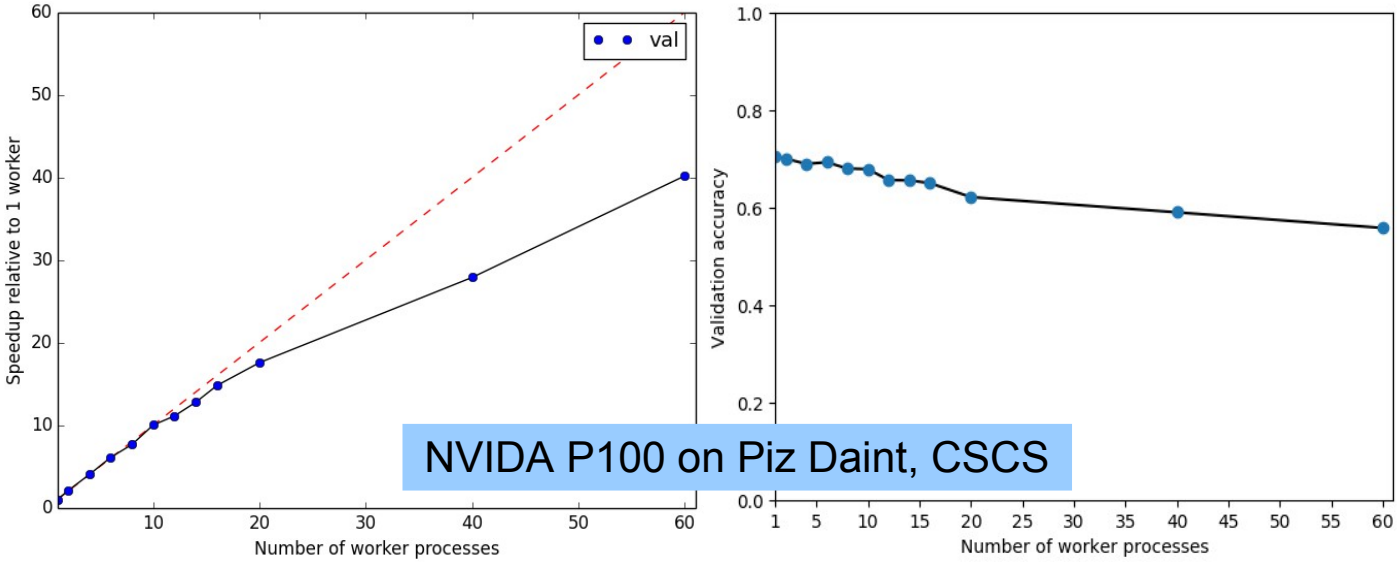
https://arxiv.org/abs/1712.05878

- Master node operates as parameter server
- Work nodes compute gradients
- Master handles gradients to update the central model
  - downpour sgd https://tinyurl.com/ycfpwec5
  - Elastic averaging sgd https://arxiv.org/abs/1412.6651
  - Gradient energy matching https://arxiv.org/abs/1805.08469

Basic Layout



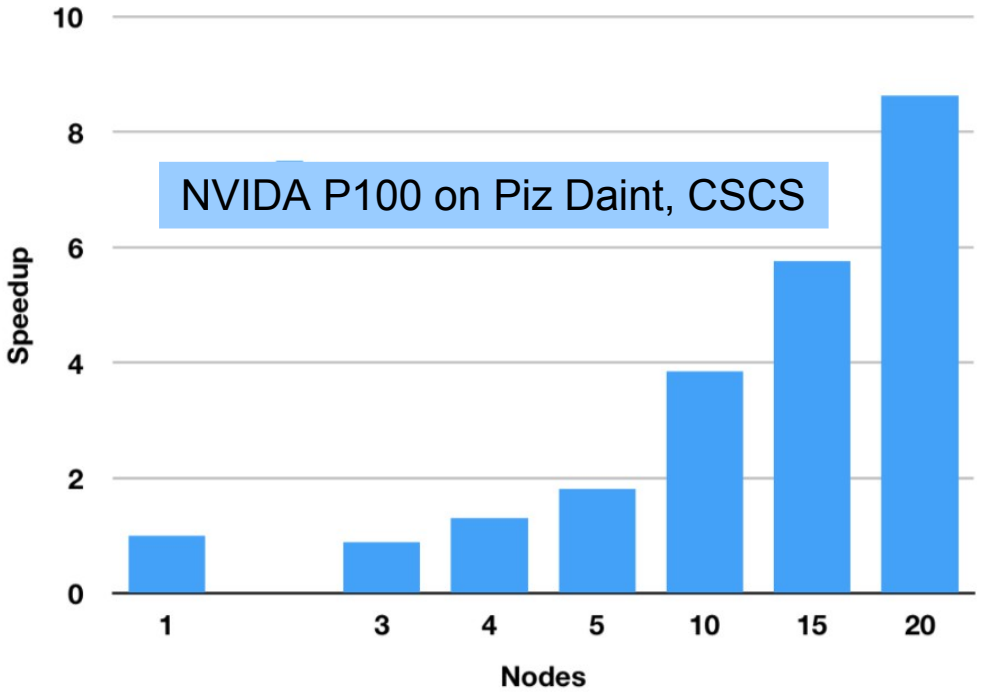
Performance with ANN

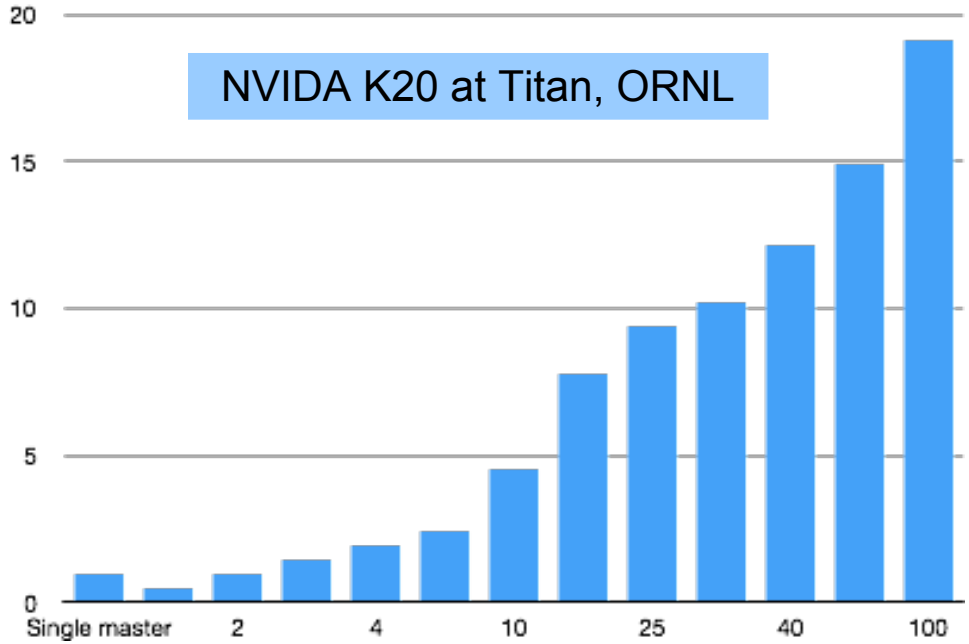


https://arxiv.org/abs/1712.05878

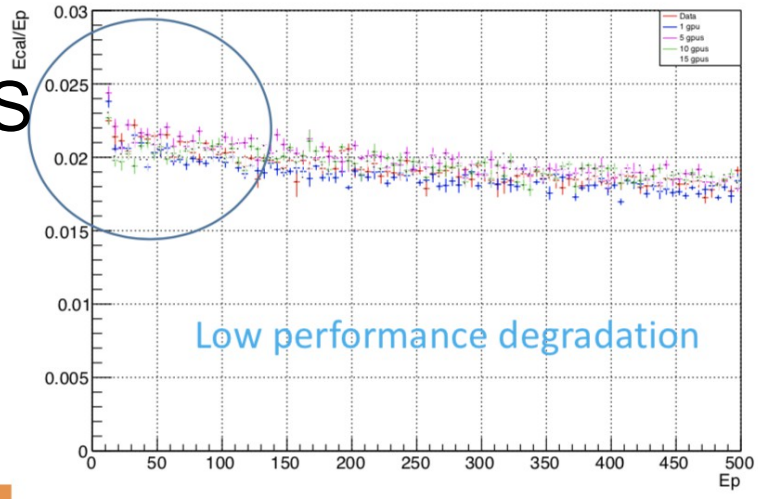
- Speed up in training recurrent neural networks on Piz Daint CSCS supercomputer
  - ➔ Linear speed up with up to ~20 nodes.
  - ➔ Needs to compensate for staleness of gradients (see GEM https://arxiv.org/abs/1805.08469)
- ➔ Linear scaling on servers with 8 GPUs

Performance with GAN





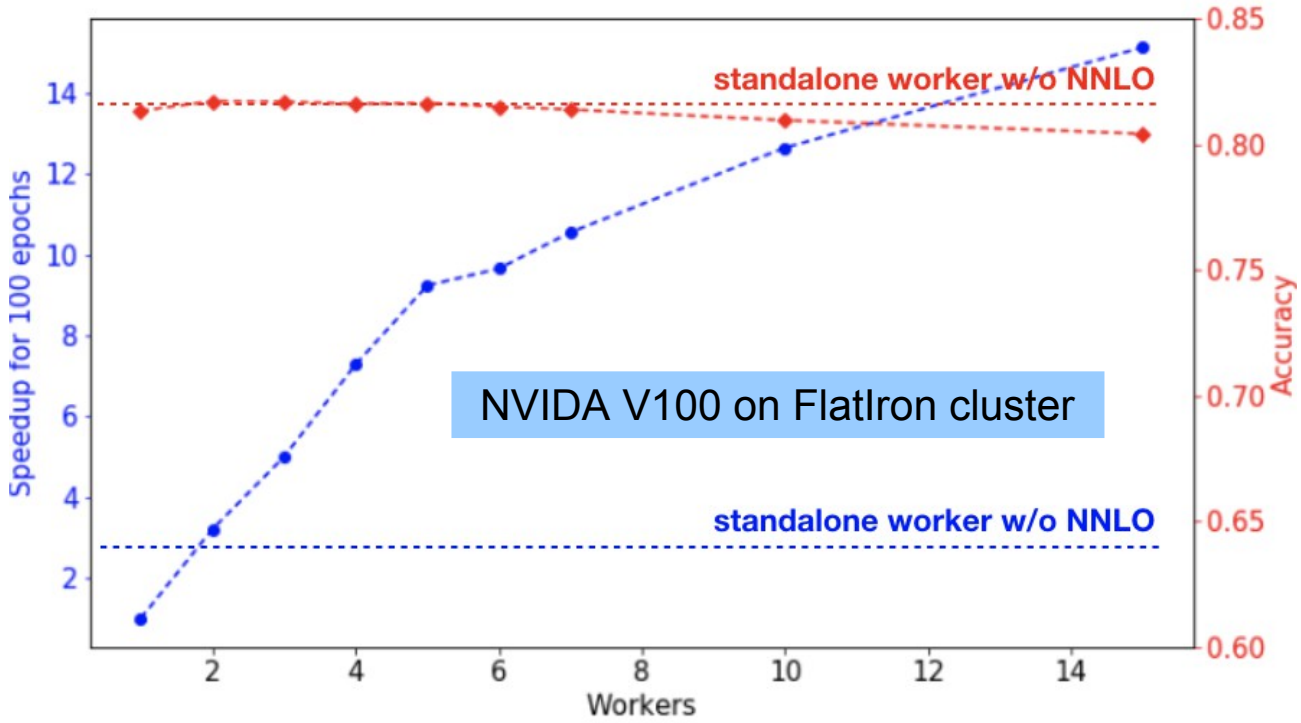
- Speed up in training generative adversarial networks on Piz Daint CSCS and Titan ORNL supercomputers
  - ➔ Using easgd algorithm with rmsprop
  - ➔ Speed up is not fully efficient. Bottlenecks to be identified



Performance with GNN

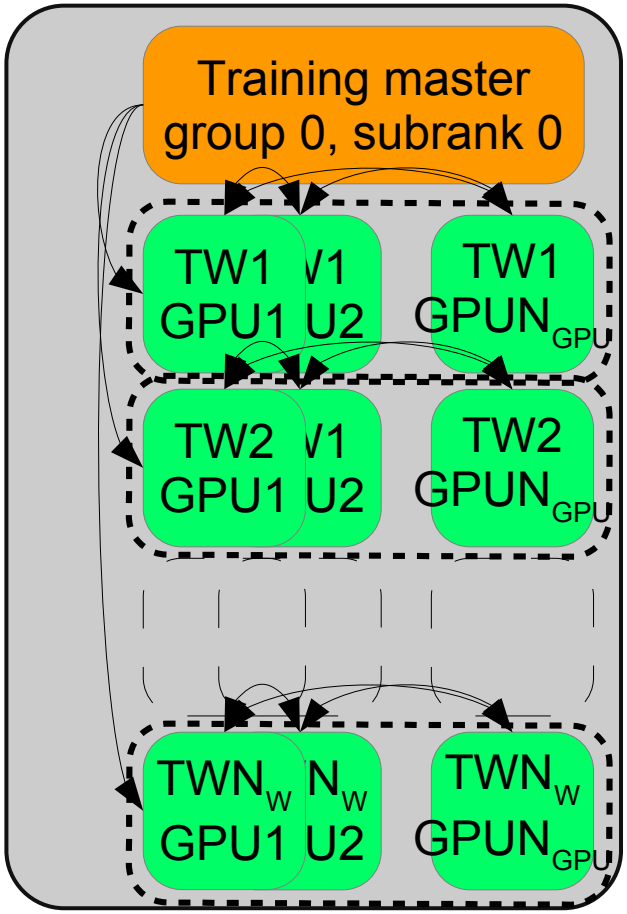
JEDI-net Graph network for jet identification
https://arxiv.org/abs/1908.05318

- 33625 parameters over three ANN
- 116M FLOP per forward pass
- Speedup shown with respect to using 1master+1worker
- Standalone training reference
- EASG (master) + adam (worker)



Gradient Distribution

Horovod Layout



- A logical worker is spawn over multiple processes
- Communicator passed to **horovod** https://github.com/uber/horovod
- Nvidia NCCL enabled for fast GPU-GPU communication

Intel MKL-DNN

Use keras 2.13 /Tensorflow 1.9 (Intel optimised)

- AVX512 –FMA-XLA support
- Intel® MKL-DNN (with 3D convolution support)

Optimised multicore utilisation

- inter_op_paralellism_threads/intra_op_paralellism threads

Horovod 0.13.4

- Synchronous SGD approach
- MPI_AllReduce

Run on TACC Stampede2 cluster:

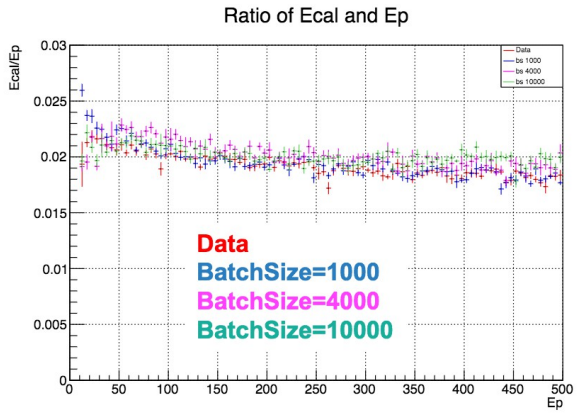
- Dual socket Intel Xeon 8160
- 2x 24 cores per node, 192 GB RAM
- Intel® Omni-Path Architecture

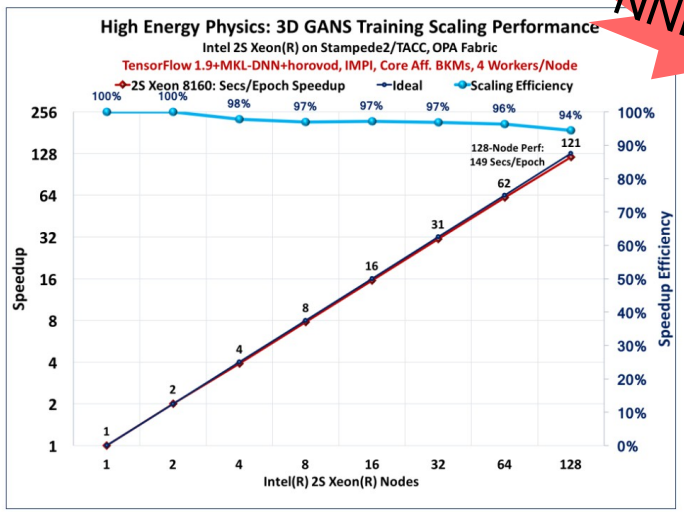
Test several MPI scheduling configurations

- 2,4, 8 processes per nodes.
- Best machine efficiency with 4 processes/node

Not with the NNLO package

Some performance degradation
Mostly in the low energy regions for large batchsize



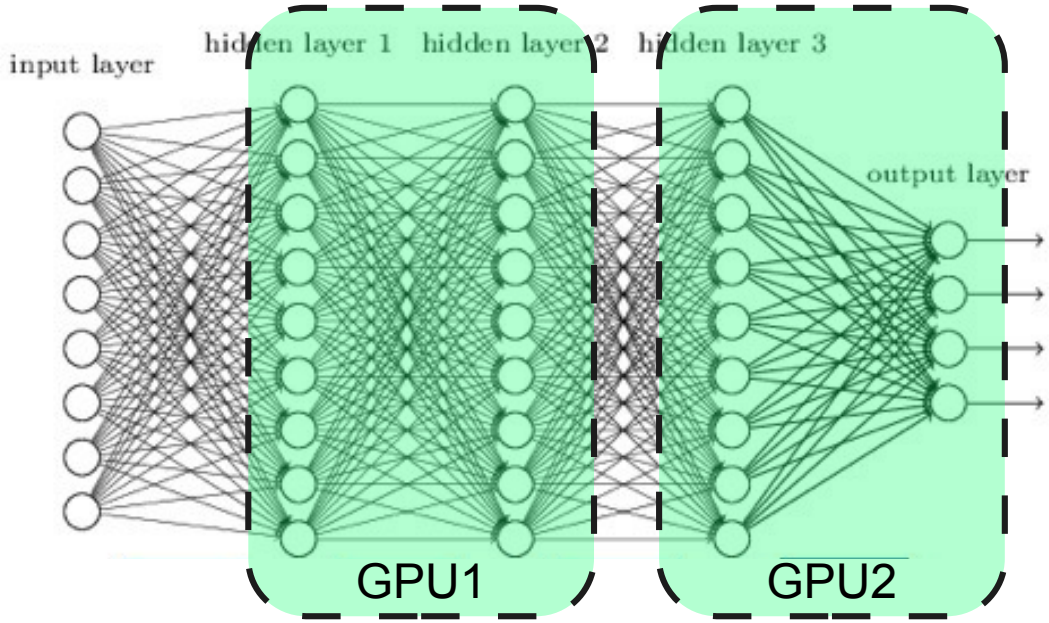


Slide S. Vallecorsa

https://sites.google.com/nvidia.com/ai-hpc

Model Distribution

Intra-Node Model Parallelism



- Perform part of the forward and backward pass on different devices
- Require good device to device communication
- Utilize native tensorflow multi-device manager
- Aiming for machines with multi-gpu per node topology (e.g summit)

Hyper-Parameters Optimization

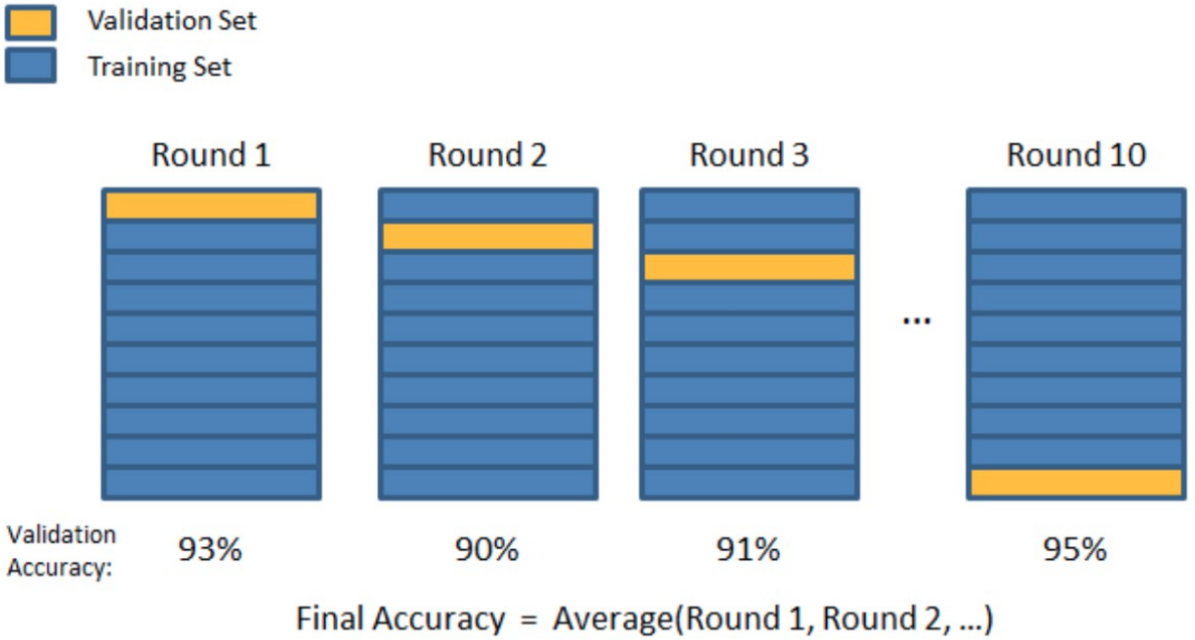
Hyper-Parameters

- Various parameters of the model cannot be learned by gradient descent
  - Learning rate, batch size, number of layers, size of kernels, …
- Tuning to the right architecture is an “art”. Can easily spend a lot of time scanning many directions
- Full parameter scan is resource/time consuming.

➔ Hence looking for a way to reach the **optimum hyper-parameter** set for a provided **figure of merit** (the loss by default, but any other *fom* may work)

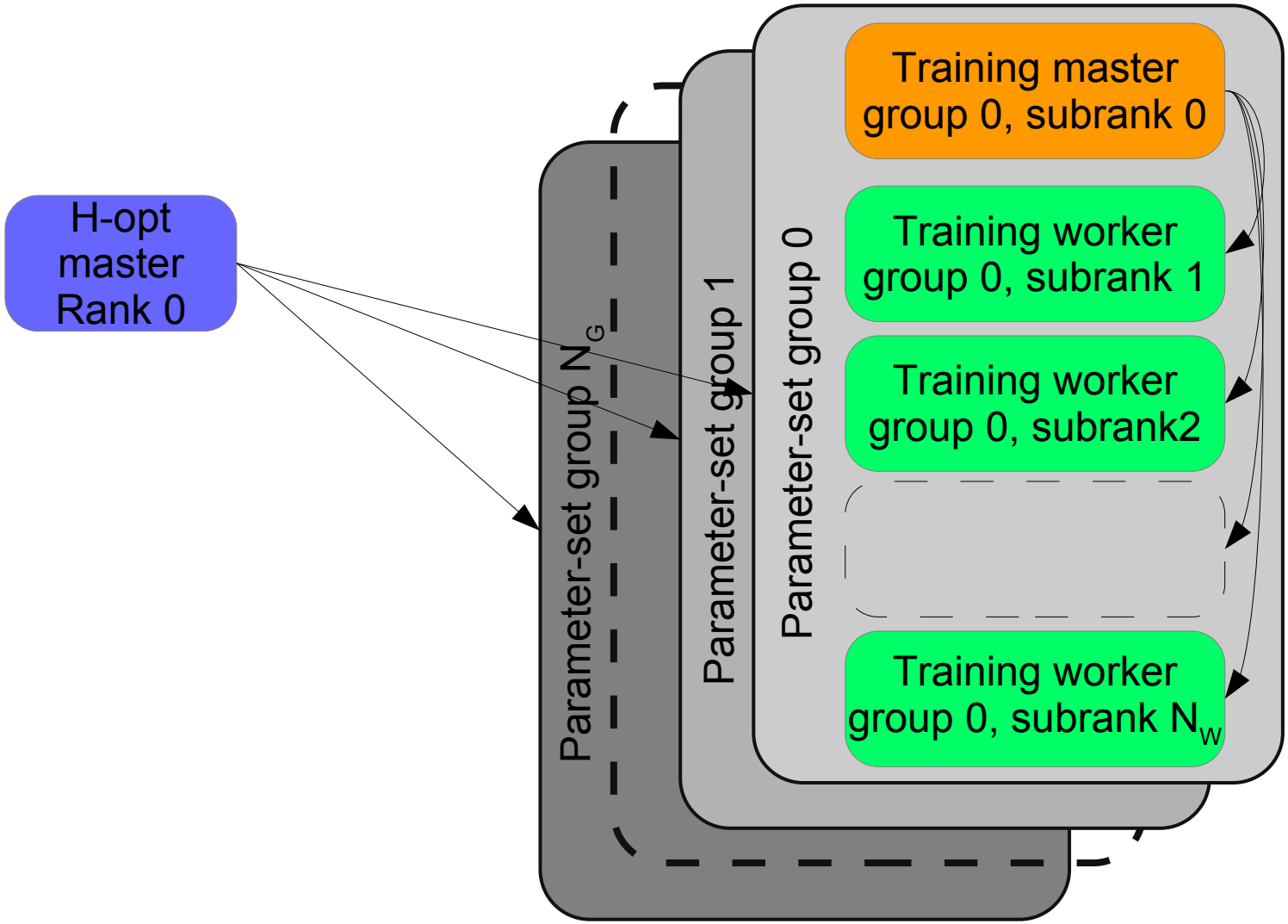
➔ Possible optimization engine
  - Bayesian optimization with gaussian processes prior
  - Evolutionary algorithm

K-Folding Cross Validation



- Estimate the performance of multiple model training over different validation part of the training dataset
- Allows to take into account variance from multiple source (choice of validation set, choice of random initialization, ...)
- Crucial when comparing models performance
- Training on folds can proceed in parallel

K-Folding Layout



Putting all Features Together

$$N_{nodes} = 1 + N_G \times N_F \times (N_M \times N_W \times N_P)$$

N_G : # of concurrent hyper-parameter set

N_F : # of folds

N_M : # of masters

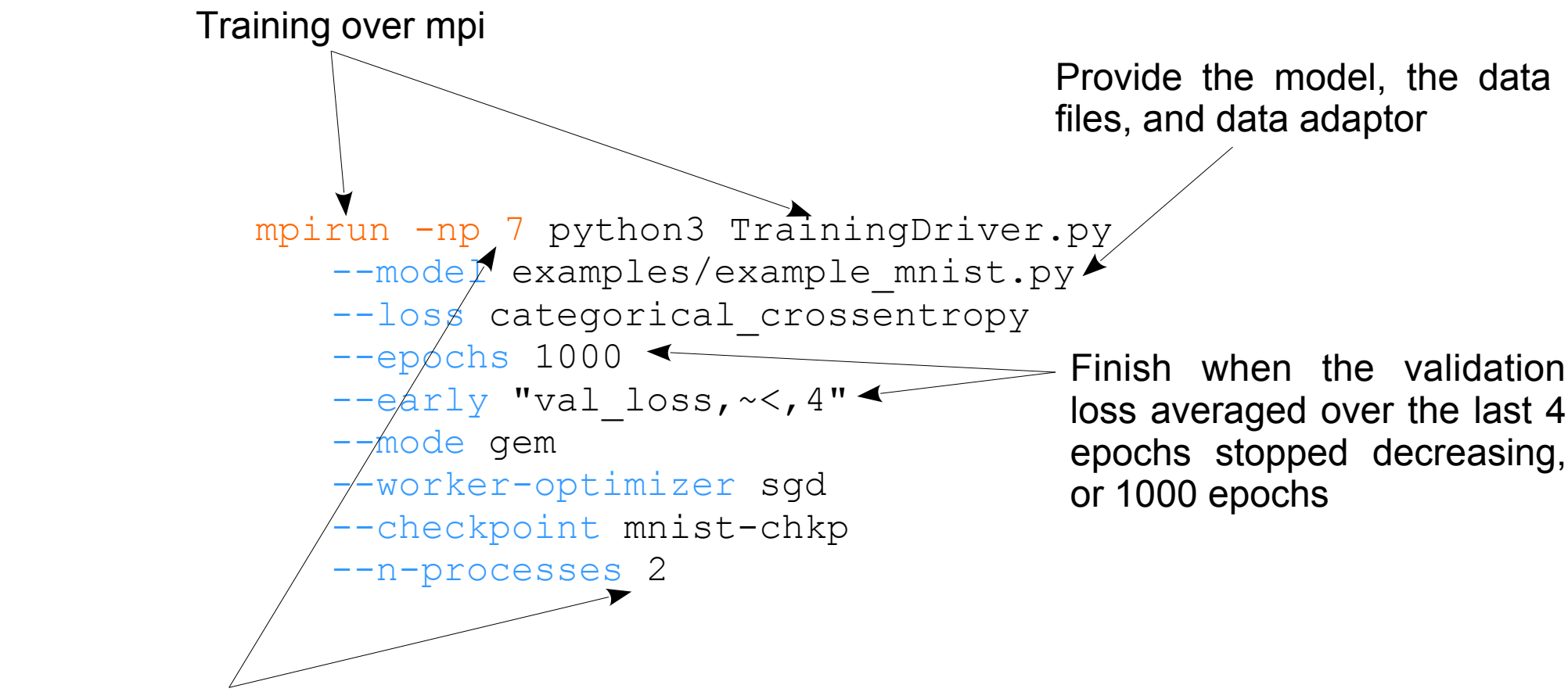
N_W : # of workers per master

N_P : # of process per worker

Reaches hundreds/thousands
of processes quickly

Interfaces

Training Command



Model Interface

The input for training can be provided in a single python file implementing the following (can also be defined on command line)

- `get_model`
  takes hyper-parameters as wild arguments, returns a Keras Model, or Torch nn.Module
  - `get_model.parameter_range`
    defines the list of hyper-parameters to be optimized
- `get_train`
  returns a list of files for training
- `get_val`
  returns a list of files for validation
- `get_features`
  the name of the input dataset/group in the files, with a function to adapt to the model input
- `get_labels`
  the name of the target dataset/group in the files, with a function to adapt to the model output

https://github.com/vlimant/NNLO/blob/master/examples/example_jedi_torch.py

Optimization Command

Optimization over mpi

```
mpirun -np 141 python3 OptimizationDriver.py
    --model examples/example_mnist.py
    --loss categorical_crossentropy
    --epochs 1000
    --early "val_loss,~<,4"
    --mode gem
    --worker-optimizer sgd
    --checkpoint mnist-chkp
    --n-processes 2
    --hyper-opt bayesian
    --n-fold 5
    --block-size 35
```

Same as for training

Implicit number (4) of hyper-parameter set tested in parallel : 140 = **4**x35
Plus 1 process for the optimization coordinator

Number of mpi processes per hyper-parameter set : 5x(1+(3x2))

Discussion on performance

Data Distribution

- Parameter Server setup has its limitations
- Gradient staleness with large number of workers : GEM
  issue with convergence of GEM/JEDI-net to be understood
- Any speedup has a cut off because of synchronization with the master weights
- Burn-in inefficiency to desynchronize workers
- The maximum number of workers with efficient scaling should be related to
  (time per worker batch) / (time per master update)

Gradient Distribution

- Using *Horovod*, with no further tweak
- Where to expect speed-up
  - If one batch does not fit in memory, spreading on multiple process will help (∞ speed up)
  - Training on CPU: Parallelizing computation. Linear speed up expected up ~ to
    (***# of processes) = (batch size)***
    as long as processes are not interfering in memory.
  - Training on GPU: Linear speed up expected with
    (batch size) α (# of processes)
    as long as GPU-GPU communication is good.

Scaling Bottleneck

- Possible ways to idle processes and loose speedup factors
  - Not enough time spend per batch
  - Slow update on the master
  - Large number of weights to be communicated
  - Slow communication between processes
  - Processes sharing RAM
  - Processes sharing a GPU
  - ...

Summary & Outlook

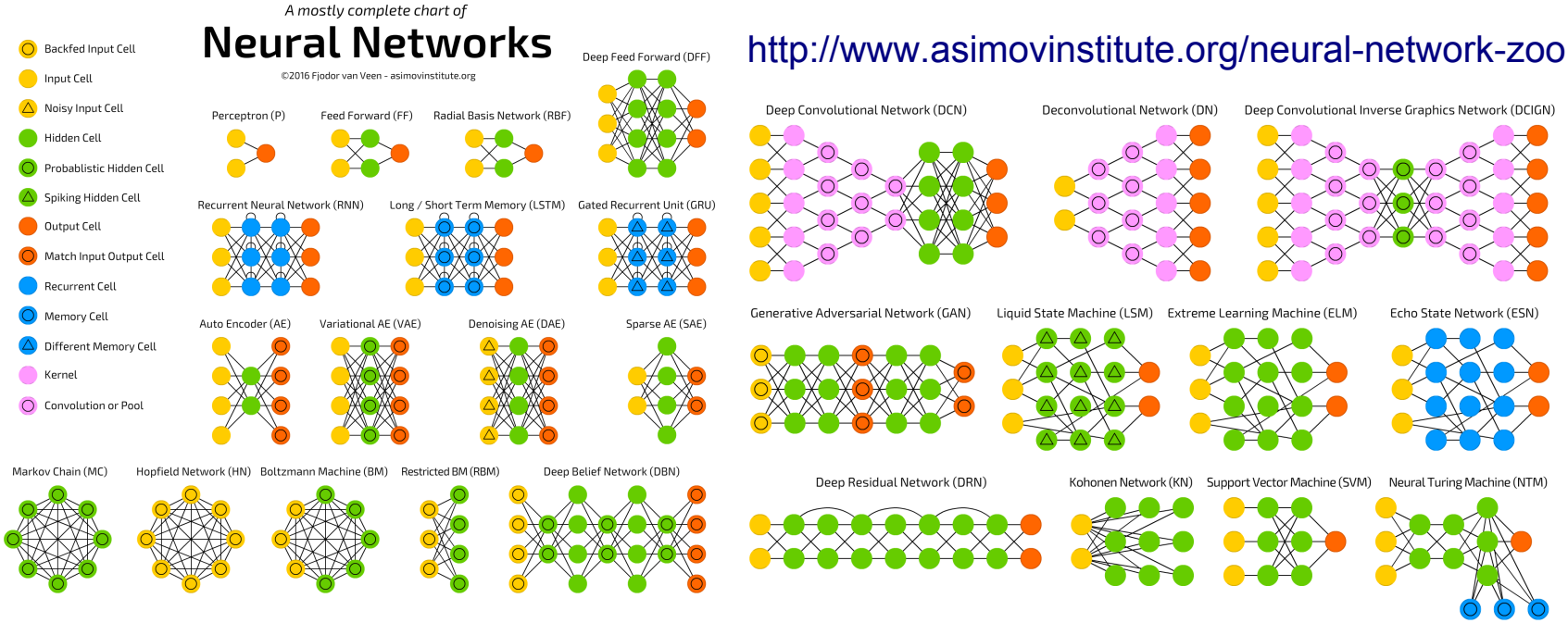
- Distributed training is not always necessary (short training time?)
- Several aspects to distributed training to consider
- Several x-factor speedup for ANN, efficient at low number of nodes. Bottleneck on master/node load balance.
- Several inefficient x-factors to be gained for GAN training
- Distributed training over CPU facilities is efficient (but not necessarily cost effective)
- Cross validation is a must and can be done in parallel
- Hyper-parameter optimization is almost mandatory, but not fully parallelizable

- Interest in the community to have a such (common?) software
- “in-house development”, or use “industry provided software” ?
- speed-up is a moving target, hard to get a one-for-all solution

→ You have access to an HPC (or any cluster with mpi), and a slow training problem ; get in touch ! jvlimant@caltech.edu

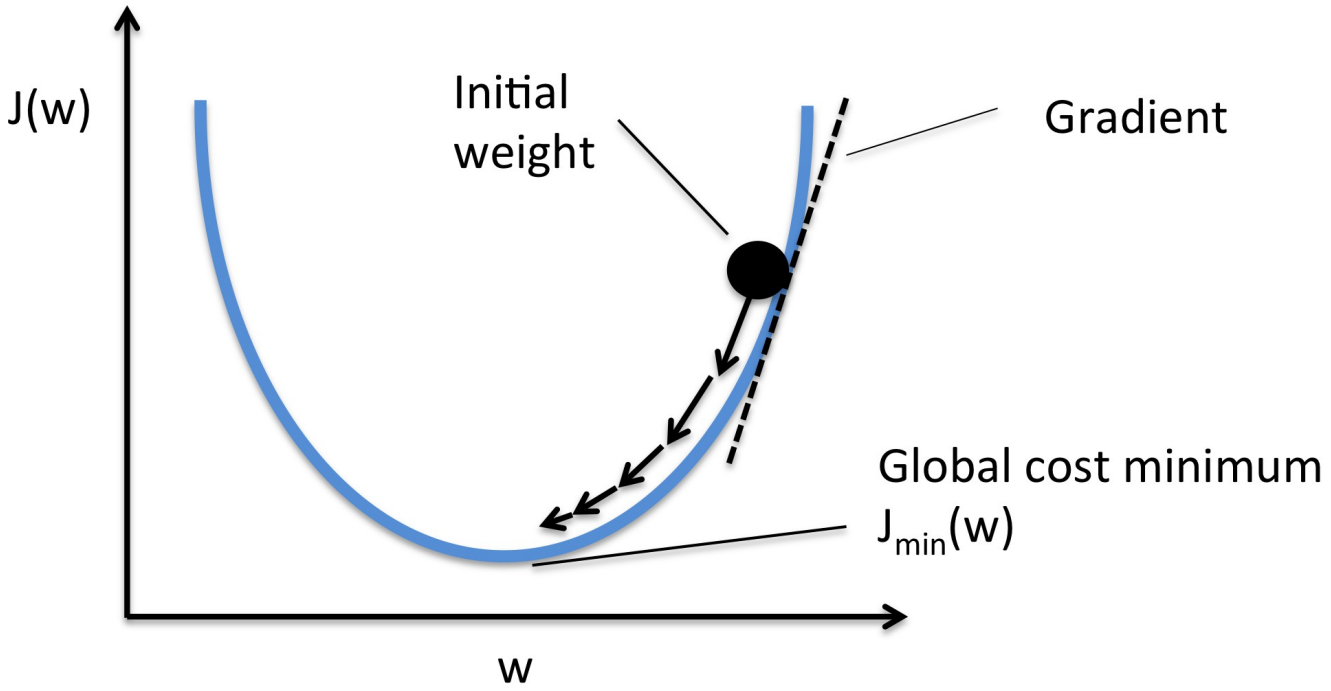
Extra Slides

Artificial Neural Network



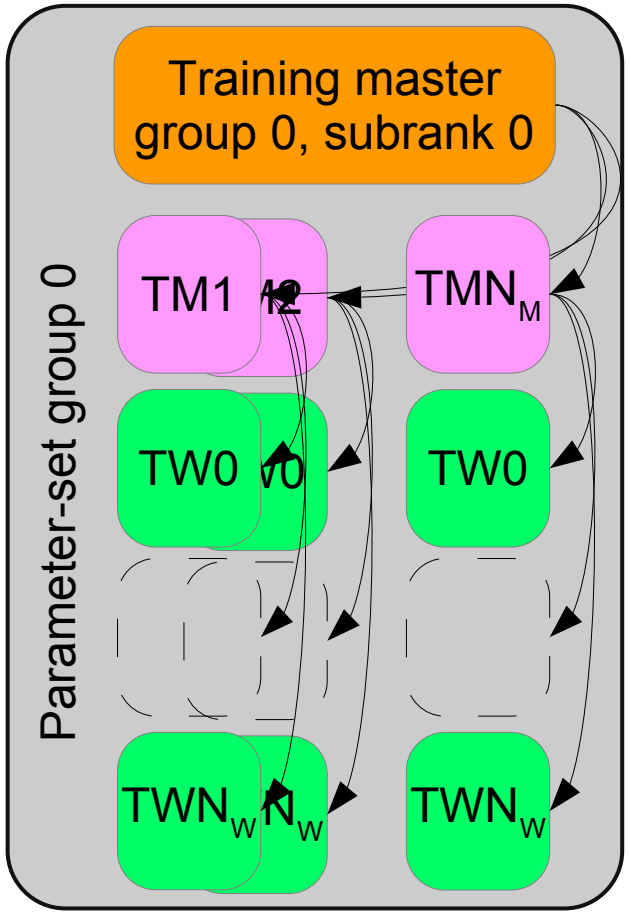
- Large number of parameters
- Efficiently adjusted with stochastic gradient descent
- The more parameters, the more data required
- Training to convergence can take minutes to several days, ...

Training Artificial Neural Networks



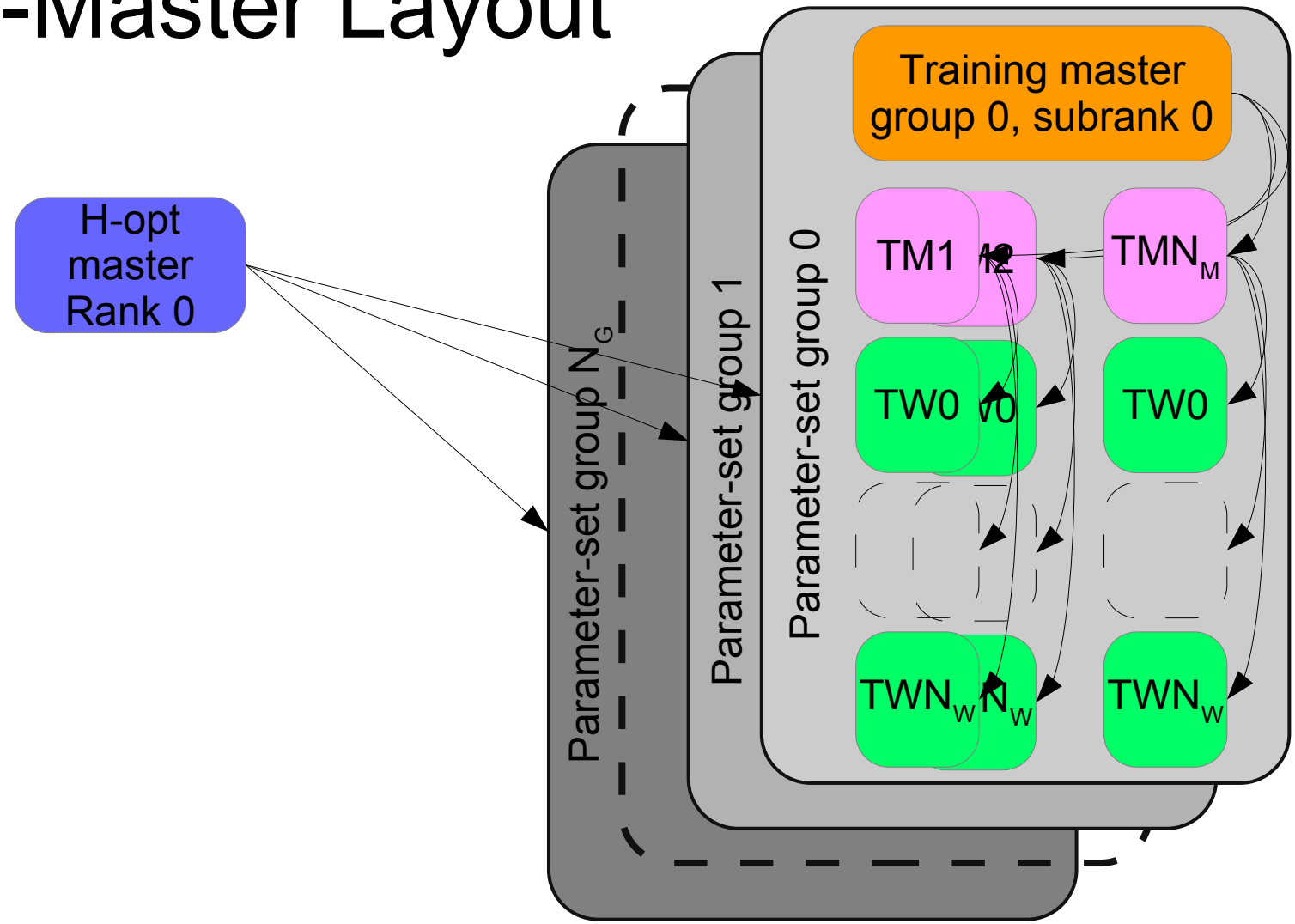
- ANN and associated loss function have fully analytical formulation and are differentiable with respect to model parameters
- Gradient evaluated over batch of data
  - Too small : very noisy and scattering
  - Too large : information dilution and slow convergence

Sub-master Layout



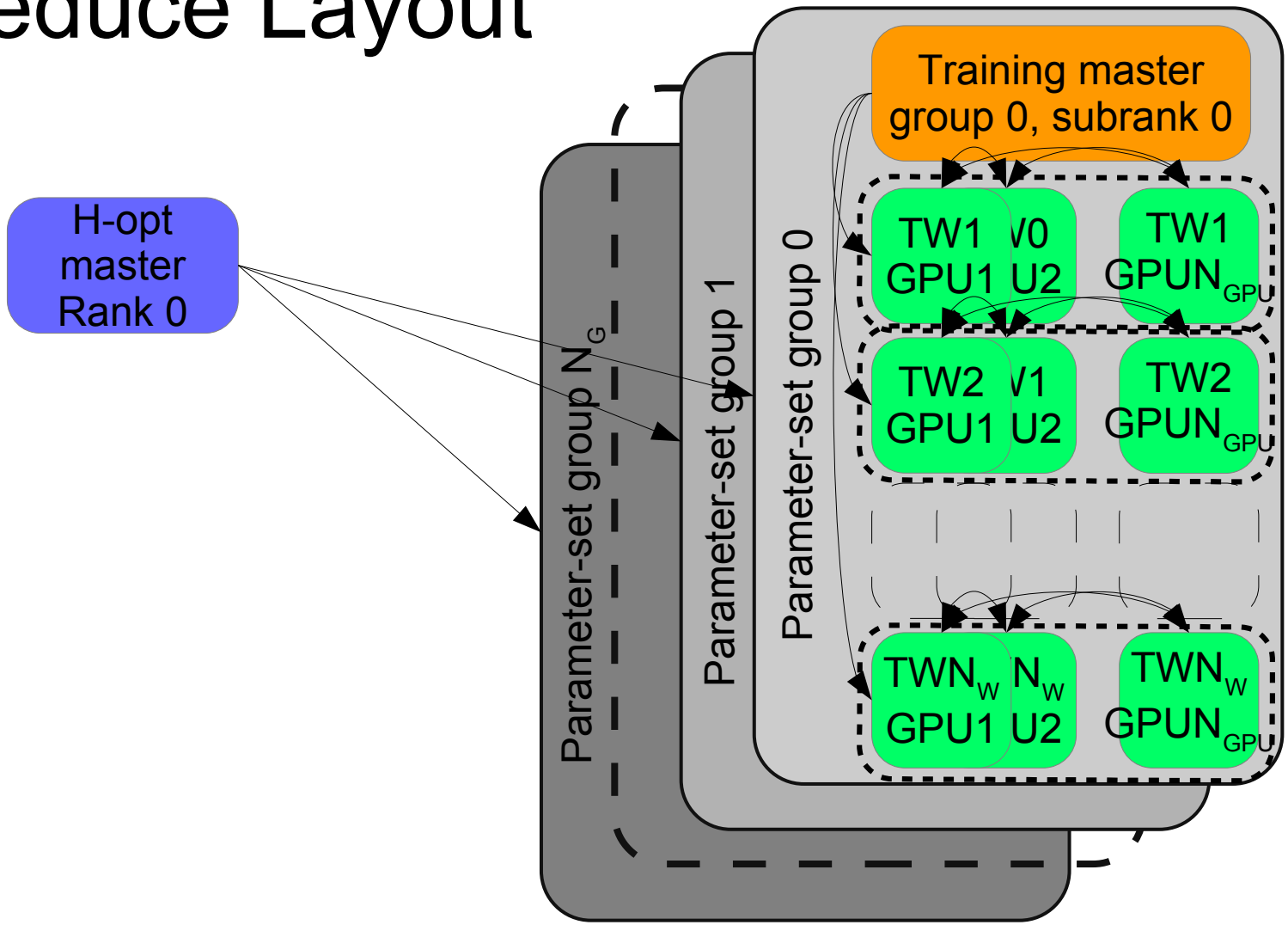
- Putting workers in several groups
- Aim at spreading communication to the main master
- Need to strike a balance between staleness and update frequency

Sub-Master Layout



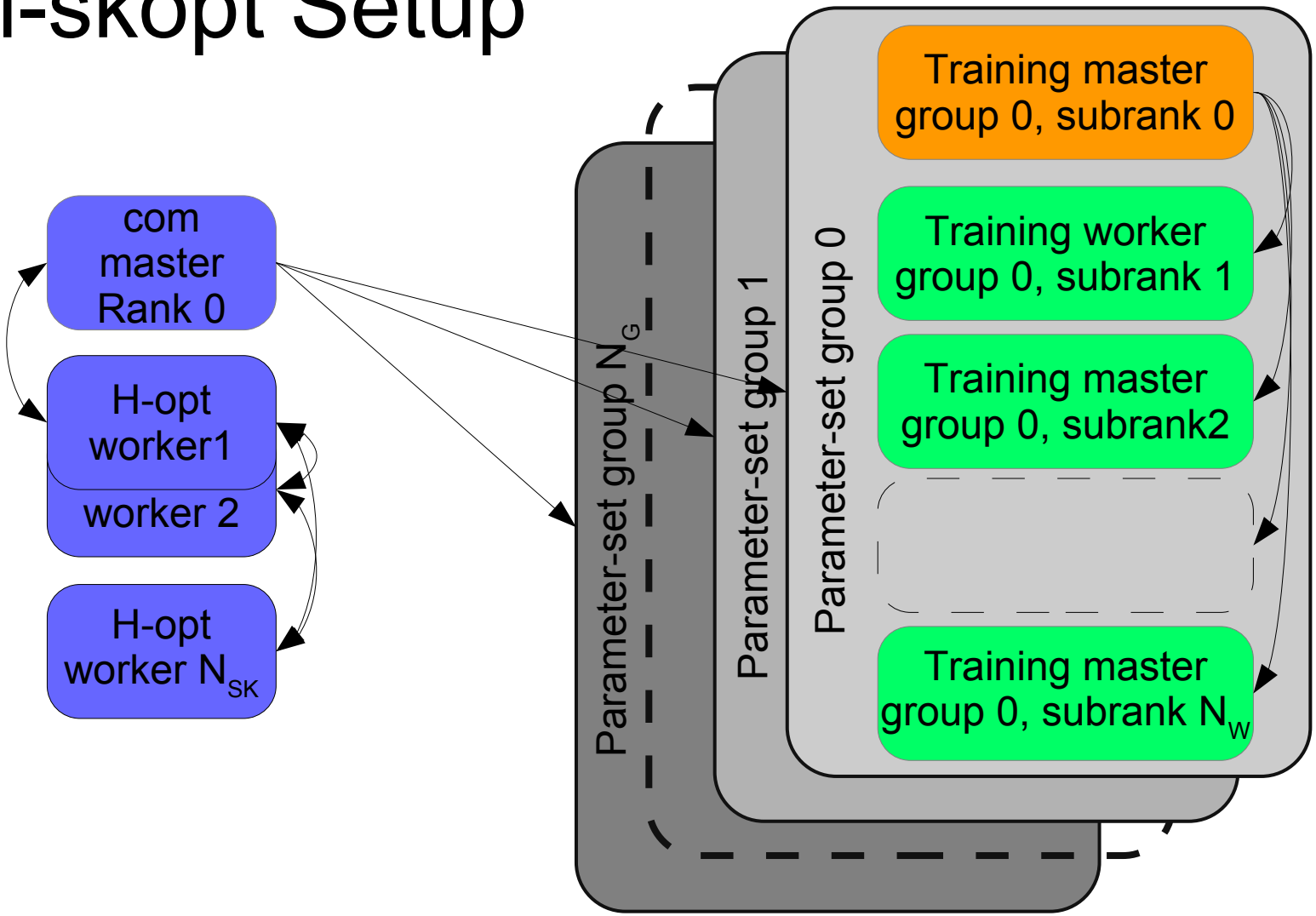
- One master running the bayesian optimization
- N_G groups of nodes training on a parameter-set on simultaneously
  - One training master
    - N_M training sub-masters
      - N_W training workers

all-reduce Layout



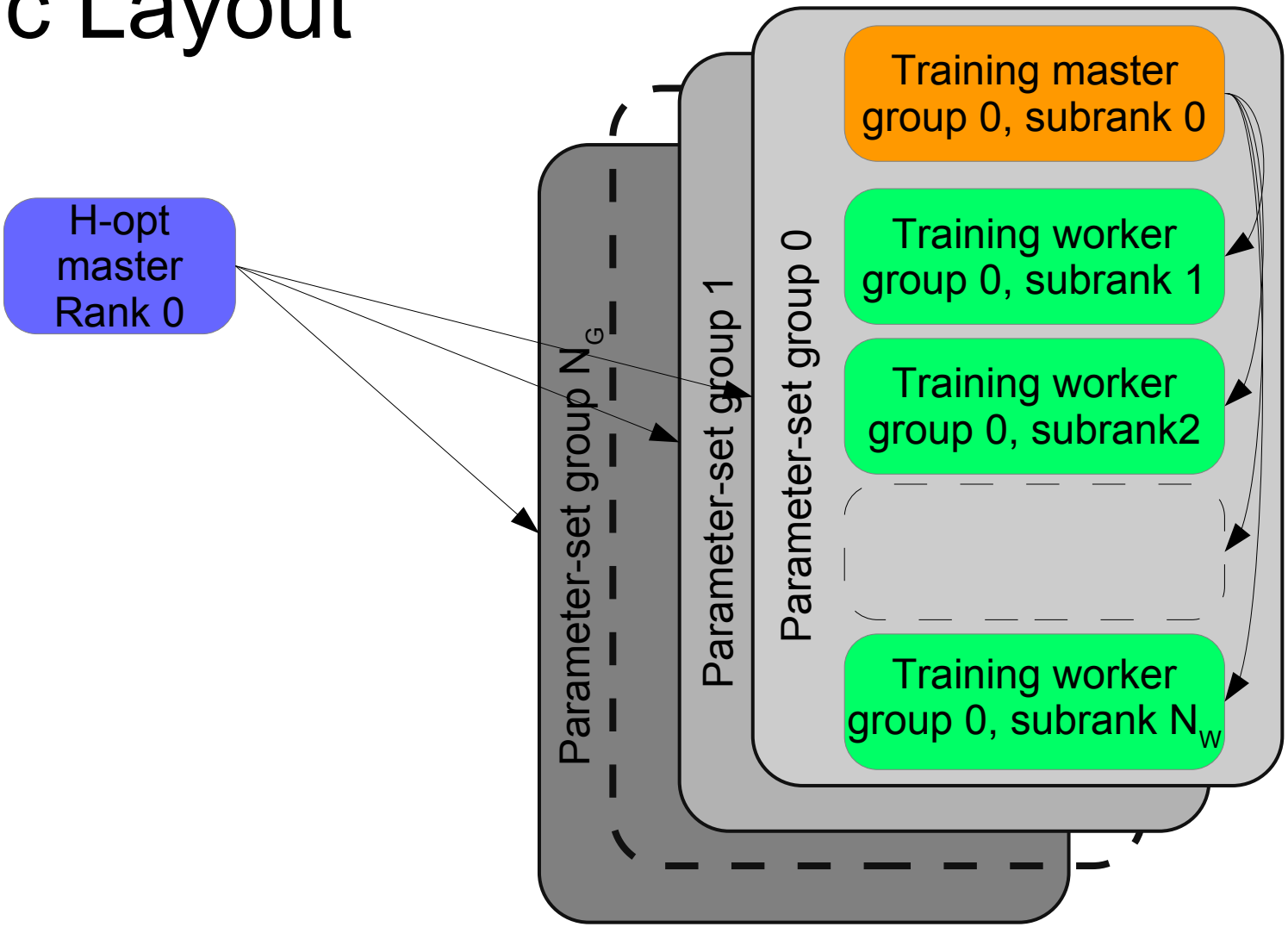
- One master running the bayesian optimization
- N_G groups of nodes training on a parameter-set on simultaneously
  - One training master
  - N_W training worker groups
    - N_{GPU} used for each worker group (either nodes or gpu)

mpi-skopt Setup



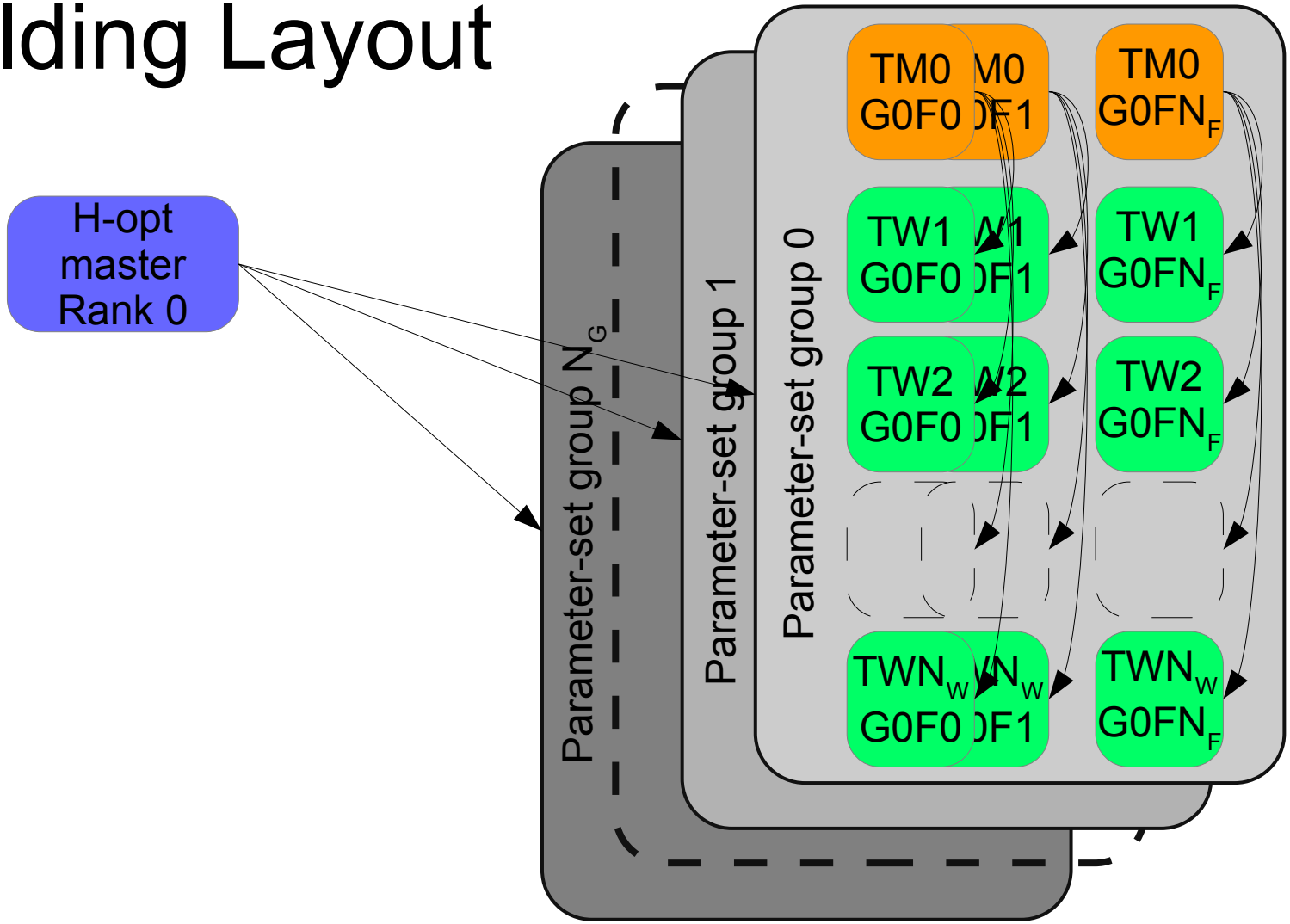
- One master running communication of parameter set
- N_{SK} workers running the bayesian optimization
- N_G groups of nodes training on a parameter-set on simultaneously
  - One training master
  - N_W training workers

Basic Layout



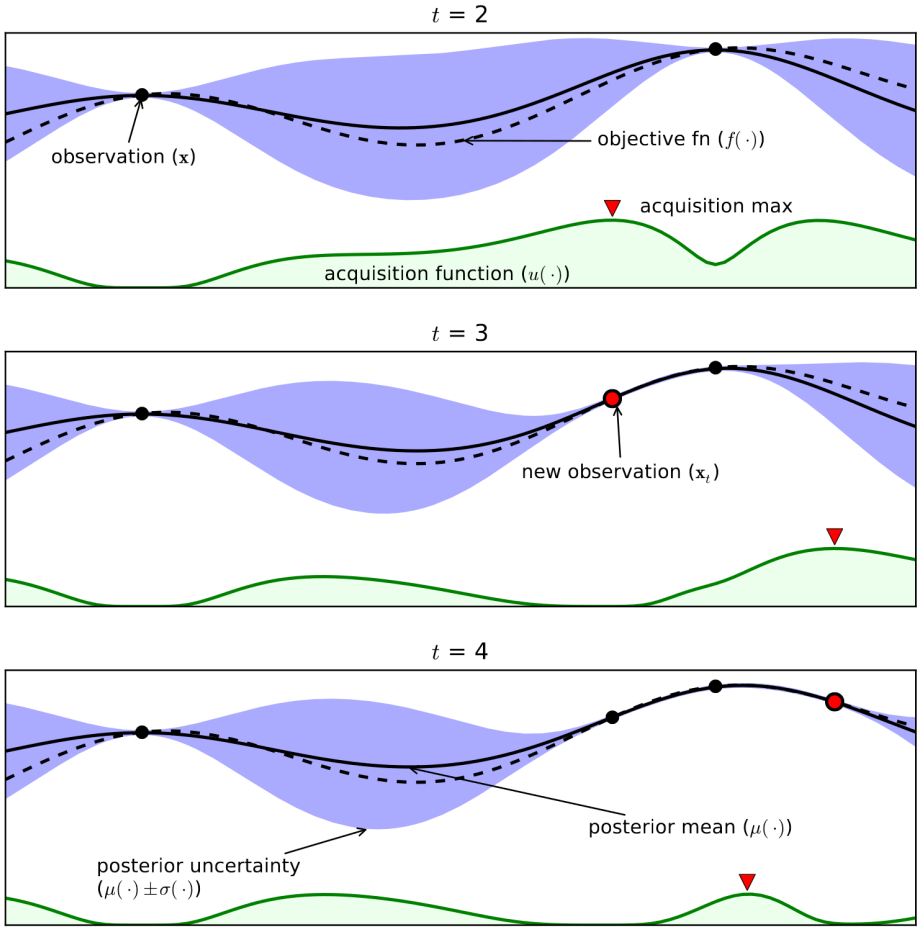
- One master process drives the hyper-parameter optimization
- N_G groups of nodes training on a parameter-set on simultaneously
  - One training master
  - N_W training workers

K-folding Layout



- One master running the optimization. Receiving the average figure of merit over N_F folds of the data
  - N_G groups of nodes training on a parameter-set on simultaneously
    - N_F groups of nodes running one fold each

Bayesian Optimization



https://tinyurl.com/yc2phuaj

- Objective function is approximated as a multivariate gaussian
- Measurements provided one by one to improve knowledge of the objective function
- Next best parameter to test is determined from the acquisition function
- Using the python implementation from https://scikit-optimize.github.io

Evolutionary Algorithm

- Chromosomes are represented by the hyper-parameters
- Initial population taken at random in the parameter space
- Population is stepped through generations
  - Select the 20% fittest solutions
  - Parents of offspring selected by binary tournament based on fitness function
  - Crossover and mutate to breed offspring
- Alternative to bayesian opt. Indications that it works better for large number of parameters and non-smooth objective function

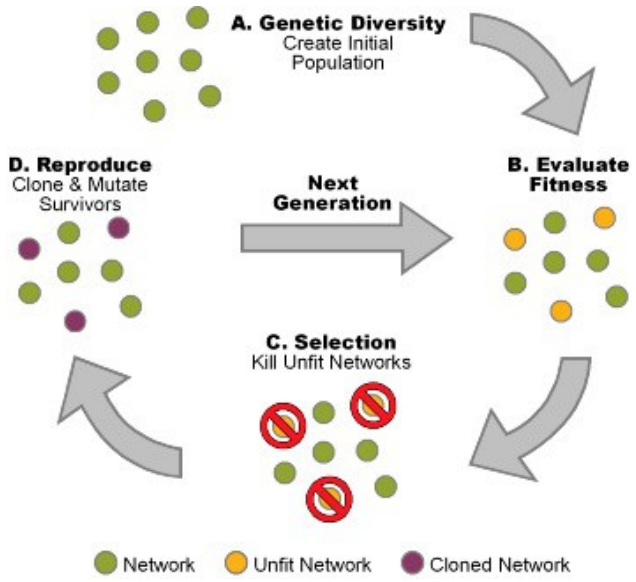
- Chromosome crossover:
  - Let Parent A be more fit than Parent B
  - For each parameter p, generate a random number r in (0, 1) to find p_{child}

$$p_{child} = (r)(p_{Parent\ A} - p_{Parent\ B}) + p_{Parent\ A}$$

- Non-uniform mutation (Michalewicz):
  - In generation g out of a total G generations, for each parameter p in a child, generate random numbers $r_1, r_2 \in (0, 1)$ to define a mutation m:

$$m = \left(1 - r_1^{\left(1-\frac{g}{G}\right)^3}\right) * \begin{cases} (p_{MAX} - p_{child}) & IF \quad r_2 > 0.5 \\ (p_{LOW} - p_{child}) & IF \quad r_2 \leq 0.5 \end{cases}$$

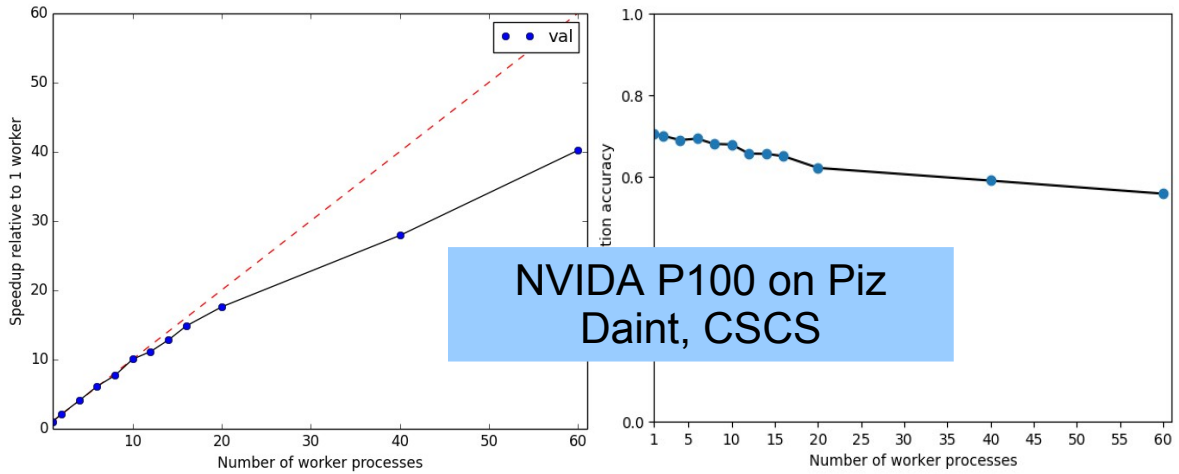
$$p_{child} = p_{child} + m$$



Code Status

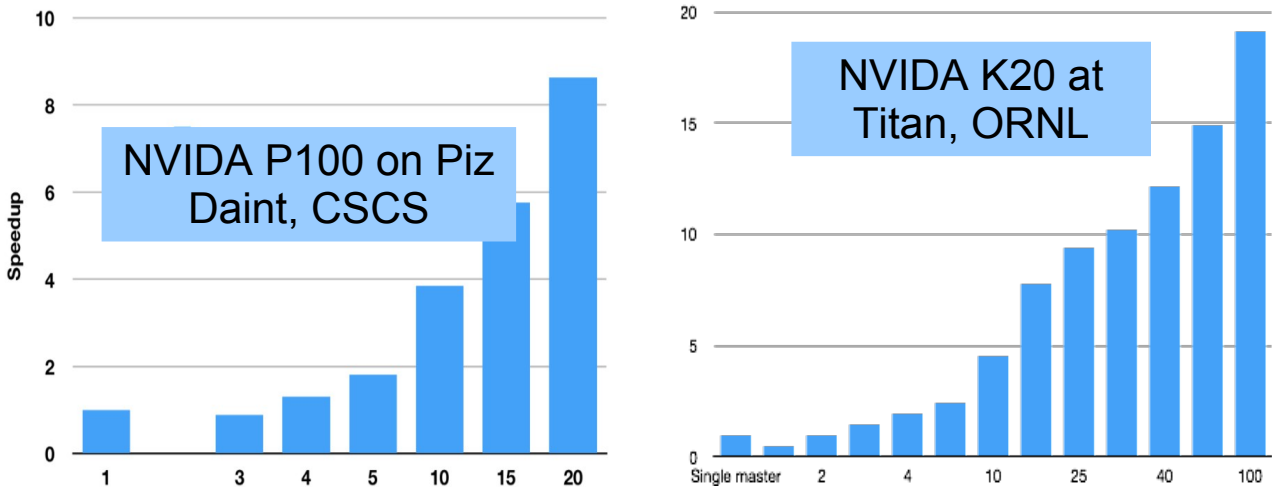
- Neural Network Learning and Optimization: NNLO
  https://github.com/vlimant/NNLO
- Lots of development done over the last few months
  - ✔ GEM https://arxiv.org/abs/1805.08469
  - ✔ Full checkpointing of training and optimization
  - ✔ Model interfacing with python script
  - ✔ Streamlined repository
  - ✔ Consolidation of options
  - ✔ Full logging
  - ✔ Better documentation
  - ✔ Graph network example (torch)
  - ✔ Data adaptor
- Upcoming
  - ➔ Option restriction (high prio)
  - ➔ Catching worker failure (medium/high prio)
  - ➔ TF model adaptor (medium prio)
  - ➔ GAN Interface (medium prio)
  - ➔ BatchNorm support (low prio)
  - ➔ ROOT data format adaptor (low prio)

(Past) Performance



ANN/RNN
Parameter server setup
Linear scaling
Downpour SGD
Worker SGD

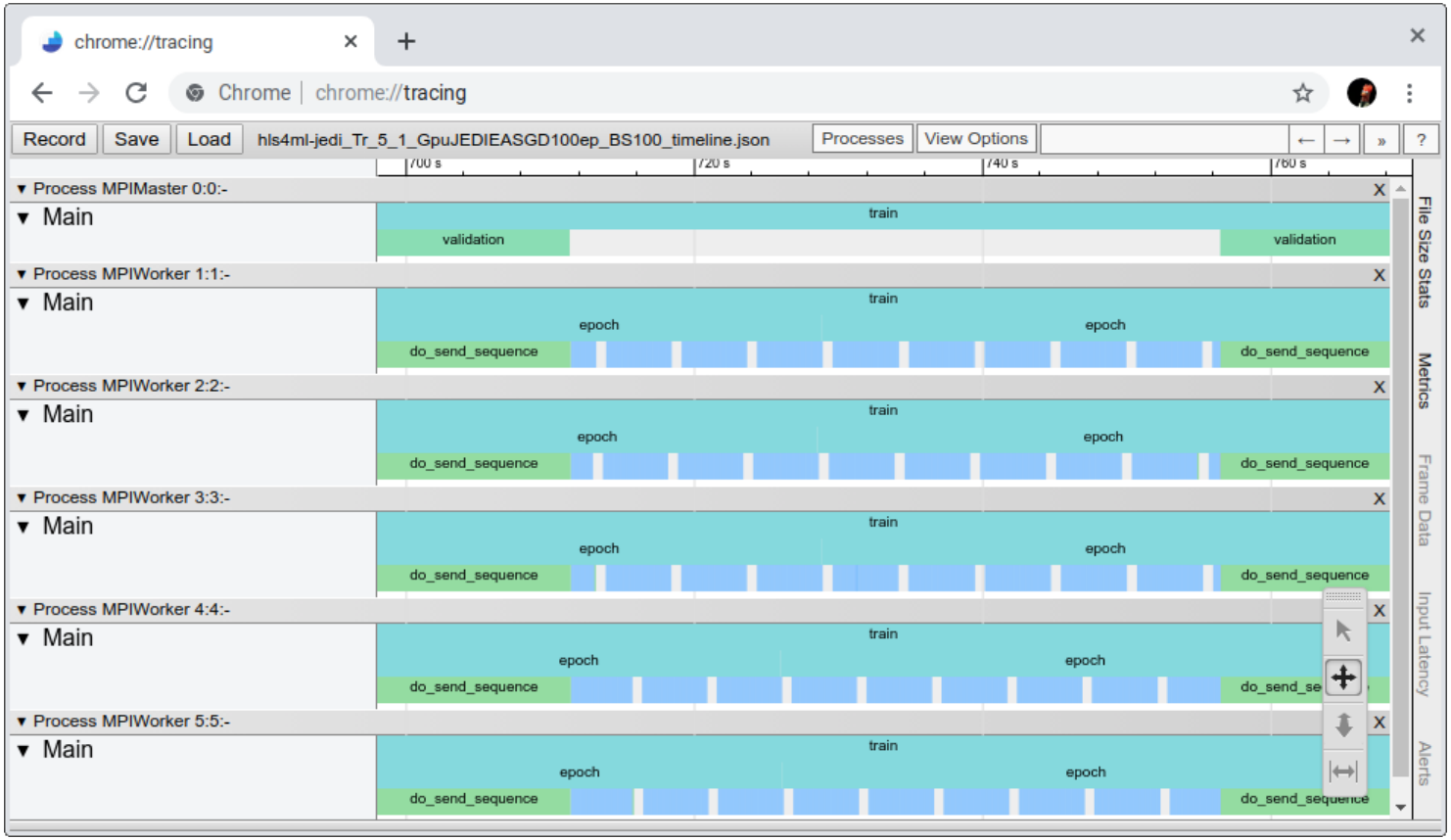
https://arxiv.org/abs/1712.05878



GAN
Parameter server setup
Linear scaling
Elastic averaging SGD
Worker rmsprop

https://doi.org/10.1051/epjconf/201921406025

Profiling


chrome://tracing
Chrome | chrome://tracing
Record
Save
Load
hls4ml-jedi_Tr_5_1_GpuJEDIEASGD100ep_BS100_timeline.json
Processes
View Options
700 s
720 s
740 s
760 s
Process MPIMaster 0:0:-
Main
train
validation
Process MPIWorker 1:1:-
epoch
do_send_sequence
Process MPIWorker 2:2:-
Process MPIWorker 3:3:-
Process MPIWorker 4:4:-
Process MPIWorker 5:5:-
File Size Stats
Metrics
Frame Data
Input Latency
Alerts