Training neural networks with low precision floats

Alexey Svyatkovskiy (Microsoft/Princeton U/PPPL),
Julian Kates-Harbeck (Harvard U/PPPL),
William Tang (Princeton U/PPPL)

#DL2SAIS

Outline

- Training neural networks with reduced floating-point precision
  - Motivation and challenges
  - Half-precision and mixed precision training approaches
- Enabling FP16 training on Pascal and Volta GPUs
  - Compression of weights and gradients
  - Master copy of weights, loss scaling
  - TensorCores
- Fusion Recurrent Neural Net (FRNN) analysis framework
  - The physics problem we are trying to solve
  - Training LSTMs on long sequences of variable length
  - Distributed training at the largest scale
  - Results on TSUBAME 3.0 and OLCF Summit supercomputers
- Conclusions

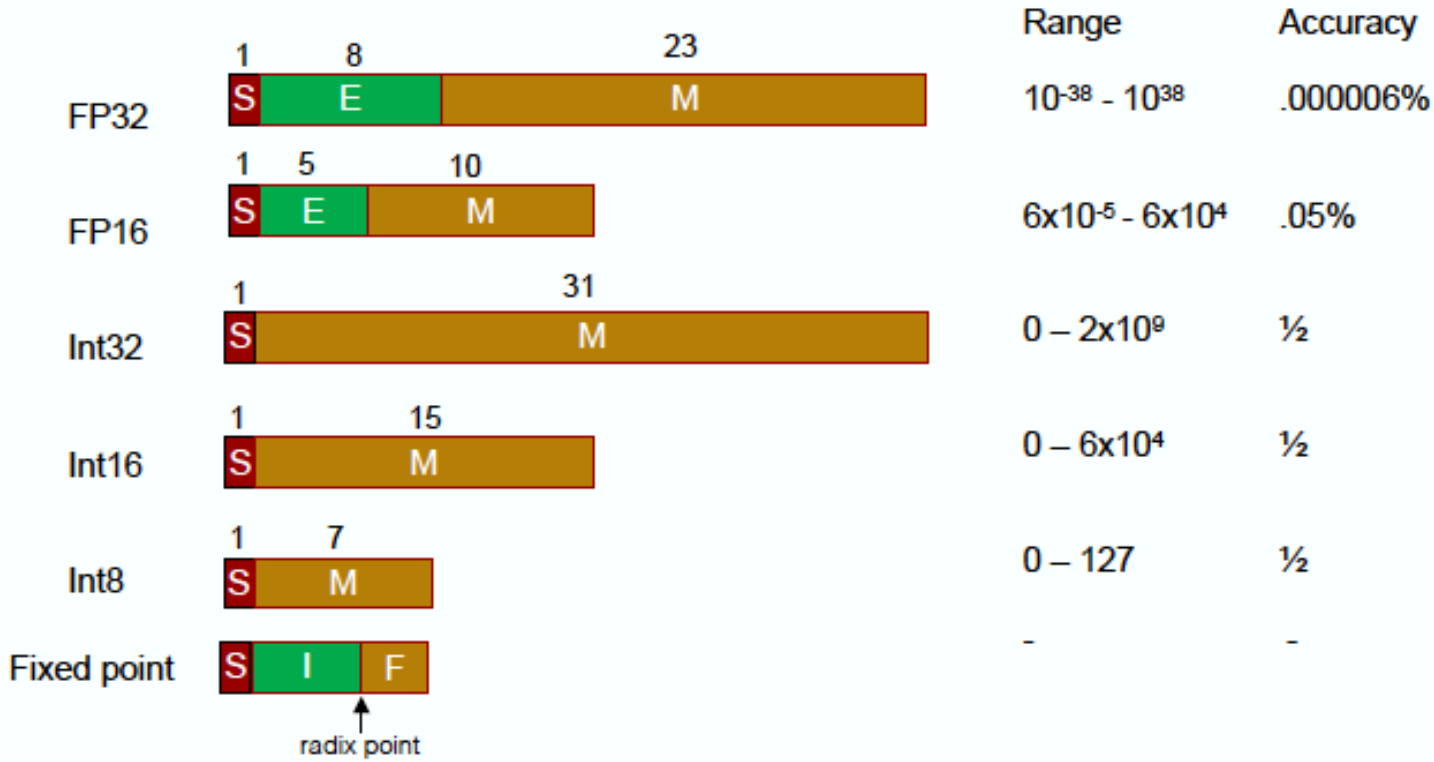
Training with reduced floating-point precision: motivation

- Models are getting larger
  - Optimize memory and network bandwidths

NN architecture	Num. layers	Num. trainable parameters	GFLOPS
AlexNet	8	60 million	1.4
GoogLeNet	22	4 million	4
Inception V3	48	24 million	9
ResNet50	152	26 million	23
Deep Speech 2	11	38 millions	465

- Need faster training
  - Improve maximum computational throughput, use larger mini-batch size
  - Minimize communication overhead during distributed training

FP16 training: motivation and challenges

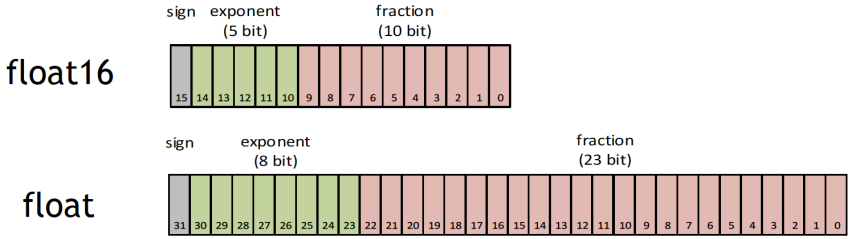


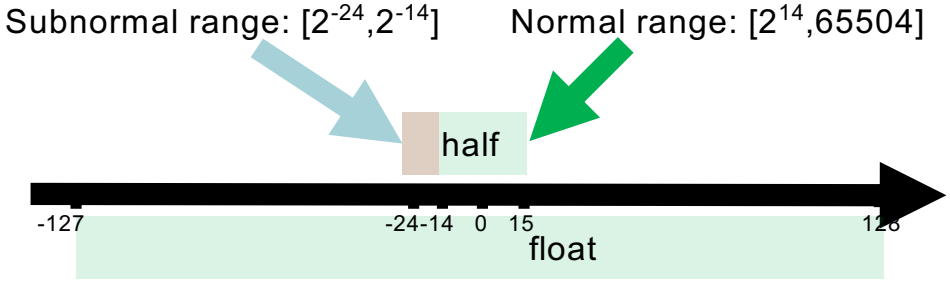
Dally, High Performance Hardware for Machine Learning, NIPS'2015

FP16 training: motivation and challenges

Challenges

- IEEE 754 FP16 has a narrow numerical range
  - Overflow problem: Inf/NaN cause irreversible damage to training (*quite rare*)
  - "Vanishing gradient" (*quite common*)
    - Small valued gradients
    - Large ratio of weight to weight gradient





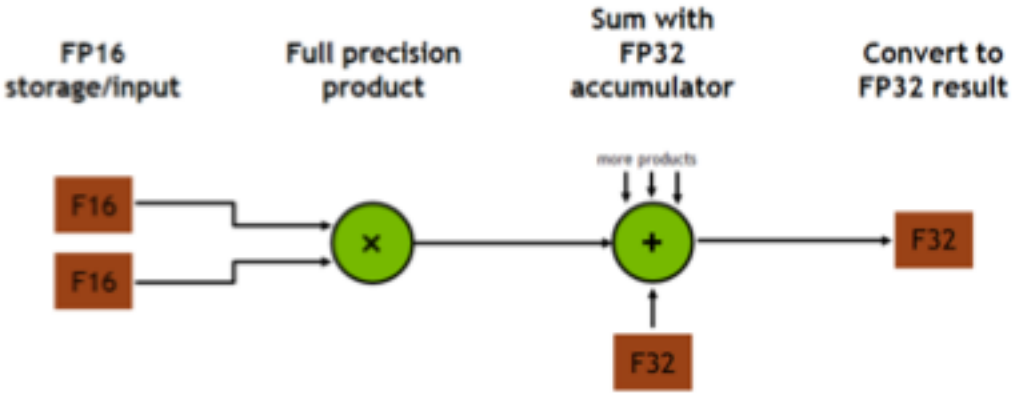
Enabling FP16 training (I)

Mixed precision with master copy of weights

- Store weights, activations, gradients in FP16; perform weight update in FP32
  - Solves “vanishing gradient” problem
  - Disadvantage: conversion between FP32 and FP16 may be slow; extra memory to keep an extra copy of weights in FP32

Mixed arithmetic precision

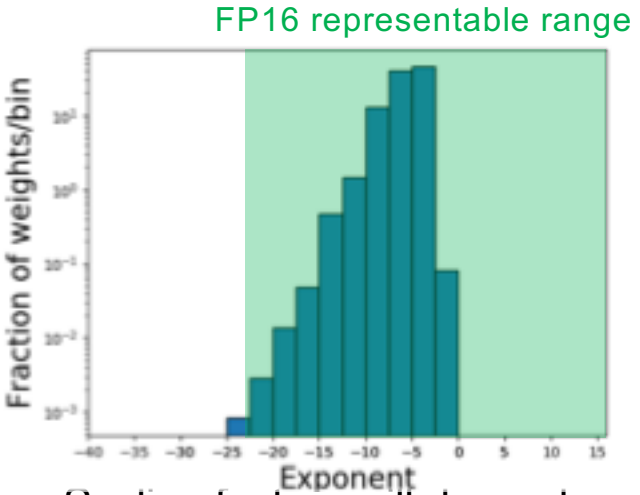
- FP16 matrix multiply, FP32 accumulate, either FP32/FP16 for activations
- Volta TensorCores enable this on the hardware level
  - Solves “vanishing gradient” problem
  - Programmable matrix-multiply-and-accumulate units. 8 TensorCores per streaming multiprocessor

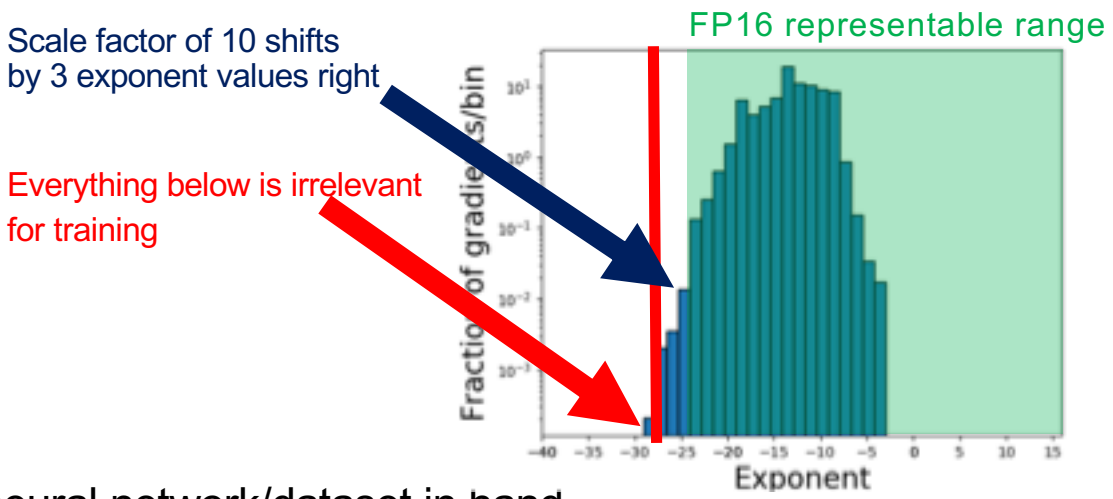


Enabling FP16 training (III)

Loss scaling: shift gradients to occupy higher bins of FP16 representable range, which are not used

- Apply a scalar factor to the loss function before backpropagation step
- Unscale weight gradients before the update step to maintain the magnitude of updates the same as for FP32; alternatively re-tune the hyperparameters

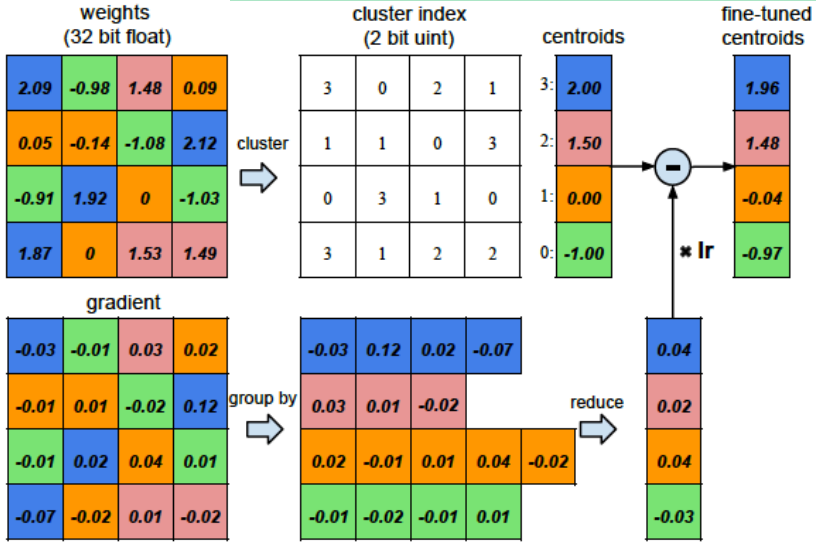




- Scaling factors will depend on the neural network/dataset in hand

Enabling FP16 training (II)

S. Han et. al. “Deep compression”



Weight and gradient sharing

- Quantize weight and gradient matrices into N bins via clustering
  - Weights in the same bin share the same value
  - Store index of shared weights in integer matrix

Dynamic fixed-point representation

- Train in full precision (float)
- Quantize weight and activation
  - Gather statistics about weight and activation (min, max for each layer)
  - Convert each float value to fixed point format (Google TPU uses int8 equivalent representation and arithmetic for that)

Single GPU results

Result from: P. Micikevicius et. al. Mixed Precision Training, https://arxiv.org/abs/1710.03740

- Volta, no loss-scaling required for accurate convergence

Model	FP32 baseline	Mixed precision
AlexNet	56.77 %	56.93 %
VGG-D	65.40 %	65.43 %
GoogLeNet	68.33 %	68.43 %
Inception V3	73.85 %	74.13 %
Resnet50	75.92 %	76.04 %

- Volta, loss-scaling required for accurate convergence

Model	FP32 baseline	MP no loss scaling	MP loss sclaing
Faster R-CNN	69.1 %	68.6 %	69.7 %
Multibox SSD	76.9 %	diverges	77.1 %

OLCF Summit: hardware and setup

- Summit is an IBM system located at the Oak Ridge Leadership Computing Facility. Having theoretical peak-performance of 200 PF, it is one of the “smartest” computers in the world for deep learning applications with mixed-precision capability in excess of 3 EF
  - To be ranked as #1 supercomputing system worldwide this summer
  - 4600 compute nodes, two IBM POWER9, PPC64LE processors and
    6 Volta V100 accelerators per node, NVlink
  - Deploy using Singularity containers
  - LSF scheduler, jsrun to execute jobs
  - AlpineTDS IBM Spectrumscale filesystem
  - Globus DTN, HPSS

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

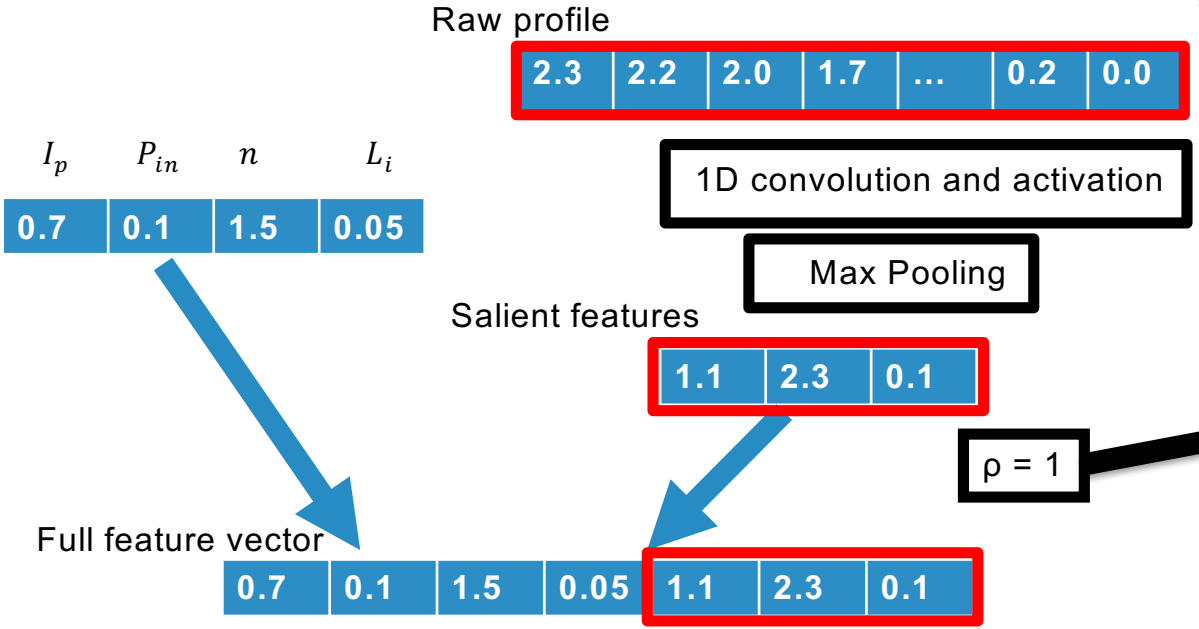
Physics problem we are trying to solve

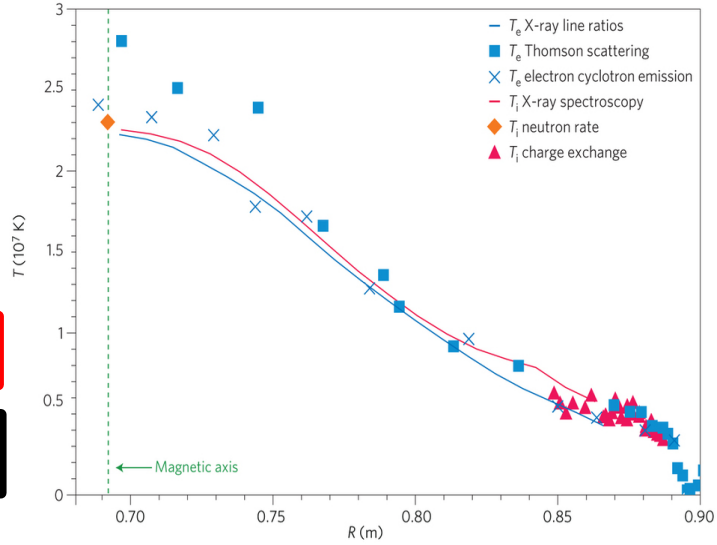
- Fusion energy: a clean source of energy that can be produced on Earth in a tokamak-style fusion reactor
- Tokamak: a type of design for a nuclear fusion device. It is a torus-shaped vacuum chamber surrounded by magnetic coils
  - Existing tokamak-style fusion reactors: JET, DIII-D, NSTX
- Plasma disruptions are large macroscopic instabilities resulting in:
  - Loss of confinement – ends fusion reaction
  - Intense radiation – damaging concentration in small areas
  - Current quench – high magnetic forces

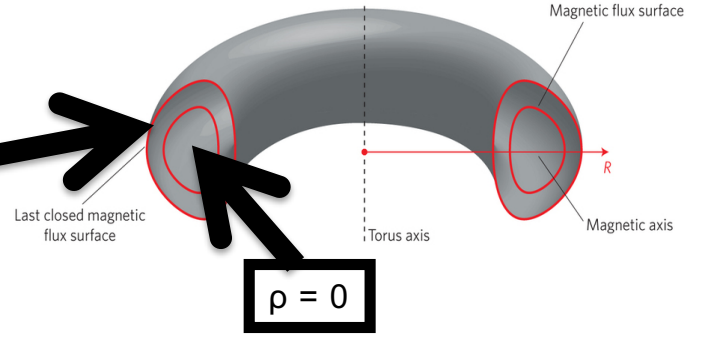
- Prediction and avoidance of disruptions is a crucial aspect of tokamak operation, especially in high-performance regime

Data: scalar and vector time series

- Scalar time series: plasma current, lock-mode amplitude
- Vector time series
  - All as a function of ρ (normalized flux surface)
  - 1D current, electron temperature, radiation profiles







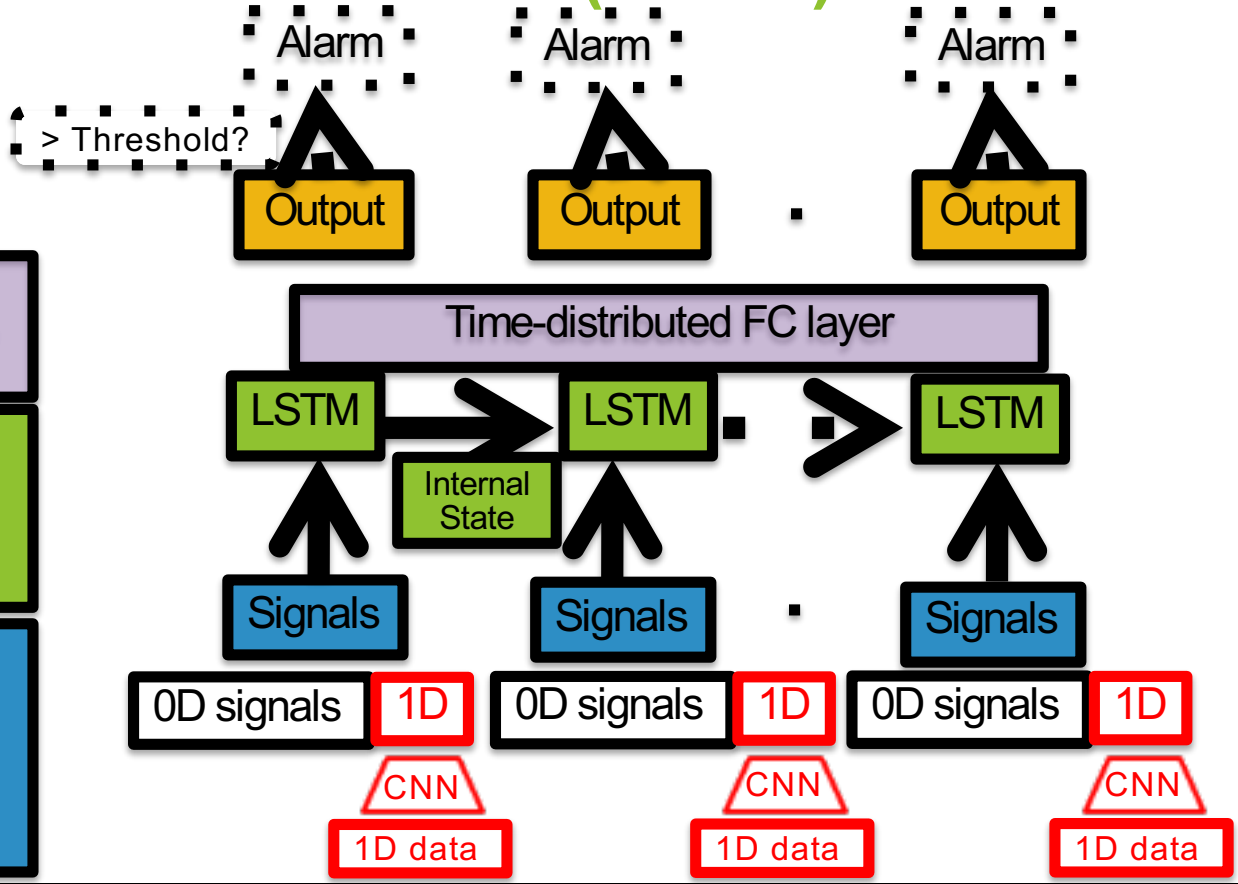
Fusion Recurrent Neural Net (FRNN)

Output: Disruption coming?

Time-distributed FC layer
- apply to every temporal slice on LSTM output

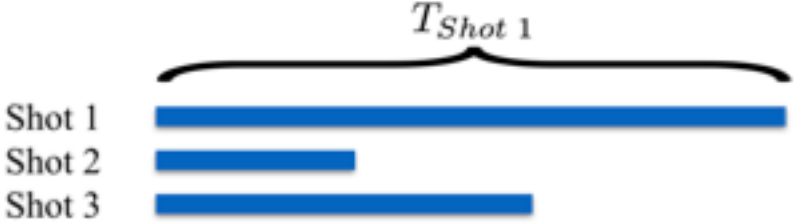
RNN Architecture:
- LSTM, 3 layers
- 300 hidden units per cell
- Stateful, returns sequences

CNN architecture:
- Number of conv. filters: 10
- Size of conv. filters: 3
- Number of conv. layers: 2
- Pool size: 2



Training LSTM on sequences of variable lengths (I)

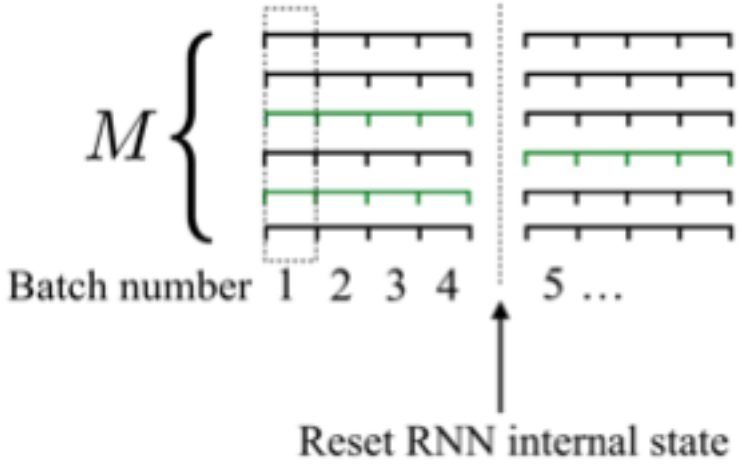
- Time series lengths in our problem vary by orders of magnitude: challenge for sequence-based learning tasks



- We approximate gradient computation by truncated backprop through time (BPTT)
  - Gradients are computed over a subsequence of time series, the internal states are saved, and then the next subsequence is fed to the RNN while using the last internal states from the previous subsequence as the initial internal states of the current subsequence
- Traditional approach of bucketing would not work as the sequence length is strongly correlated with whether shots are disruptive or non-disruptive and thus individual batches would be biased
- We implement custom approach resetting internal state of individual examples within mini batch
  - Since there is a persistent internal state between successive subsequences in time, it is not possible to use more than one subsequence from a given shot in a given mini-batch

Training LSTM on sequences of variable lengths (II)

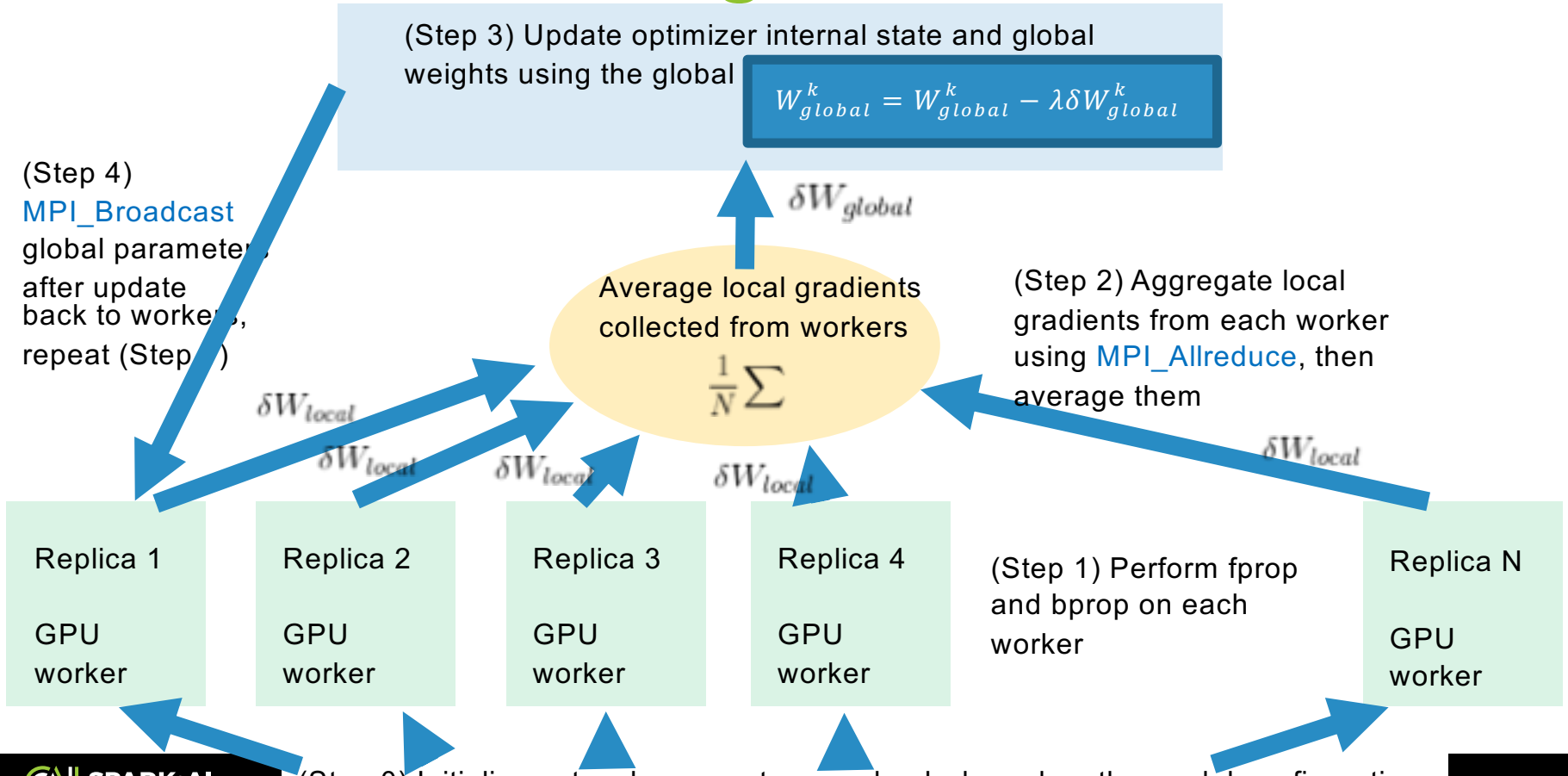
- To train batch-wise we need M independent (stemming from different shots) time slices of equal length to feed to the GPU
  - Maintain a buffer of M separate shots. At every training step, the first T time slices of the buffer are fed as the next batch, then shift the buffer by T time steps
- When shot is finished in the buffer, a new shot is loaded and the RNN internal states of the corresponding batch index are reset for training
- Internal states of the other batch indices are maintained. Thus, the internal state is persisted during learning for the entire length of any given shot
  - Allow the RNN to learn temporal patterns much longer than unrolling length T and potentially as long as the entire shot
- Random offsets of shots against each other and random shuffling of train set provides a mixture of disruptive and non-disruptive samples for the network at every batch to stabilize training



Introducing FRNN

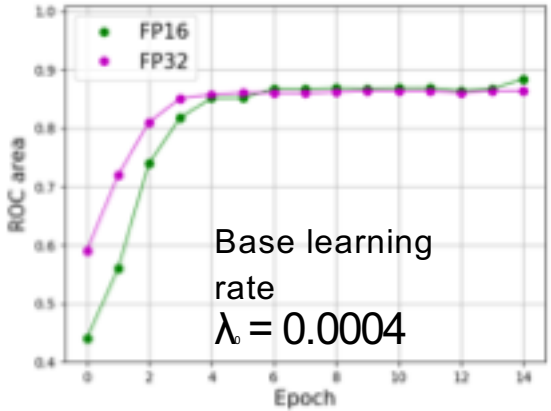
- FRNN represents a combined workflow implemented in Python
  - Keras for layer definitions
  - TensorFlow optimizers
  - Custom parameter averaging, global weights update in CUDA-aware MPI
  - Not a general purpose tool: geared toward Fusion energy scientific community, but certain parts can be useful for a wider community
  - https://github.com/PPPLDeepLearning/plasma-python
- We demonstrate how to enable FP16 for distributed training with this combined Keras/TensorFlow/MPI based workflow
  - Weight update is sensitive to floating point precision
  - Volta/FP16 optimized algorithms are disabled by default in TF, chosen by CuDNN autotuner

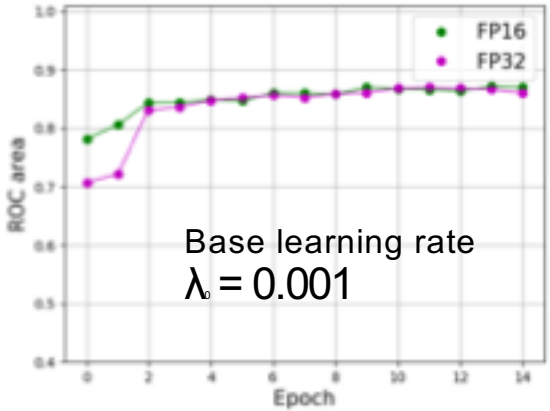
Distributed training flow

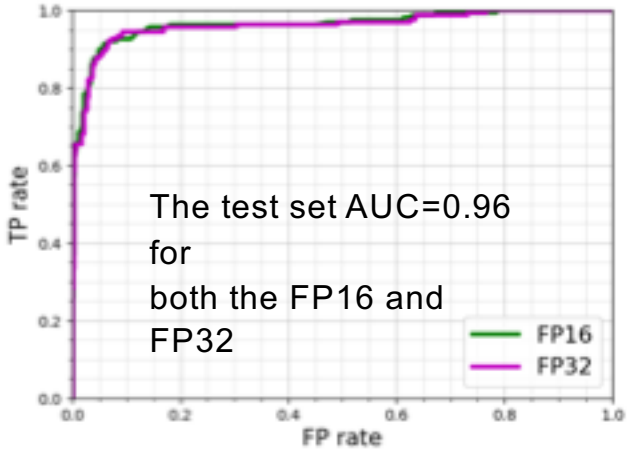


No loss in training accuracy

- Use validation level area under the ROC and AUC (area under curve) to characterize the quality of FRNN and applicability of the half-precision training
- Validation level AUC as a function of epoch show similar shapes for both FP16 and FP32
  - Reaching the plateau at around AUC=0.87 by the epoch 6
  - SGD optimizer with momentum, loss scaling factor=10



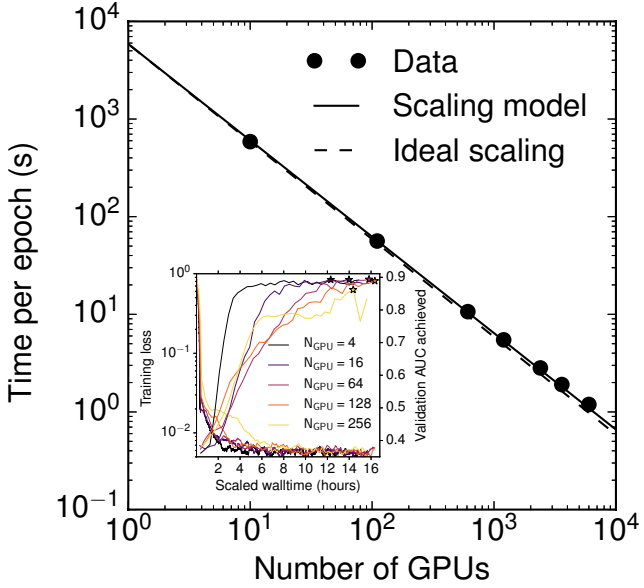


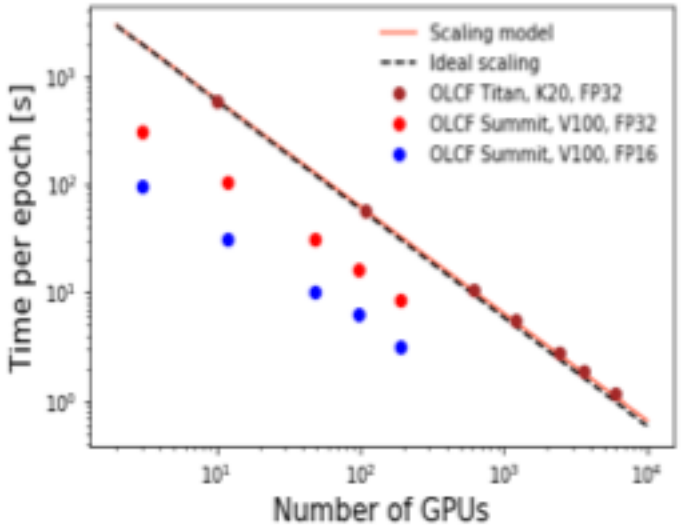


Performance and scaling properties (I)

- FRNN shows nearly-linear runtime scaling and logarithmic communication

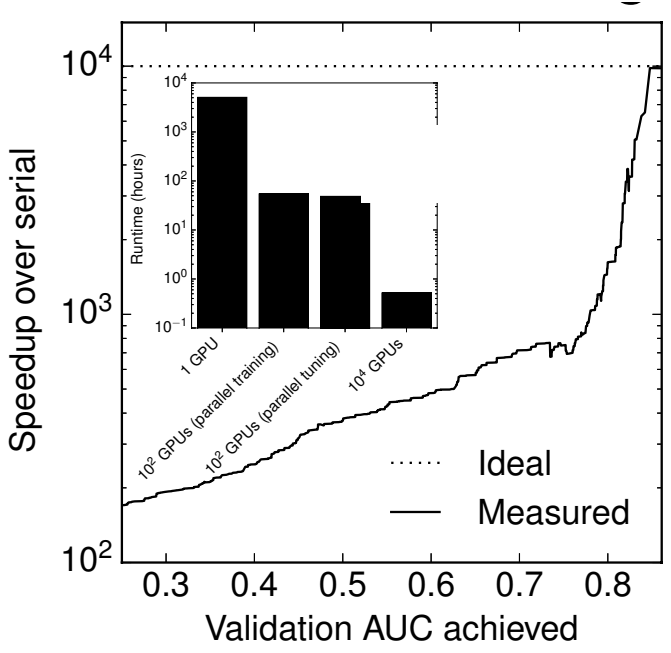
$$T \sim \frac{1}{N}\left(A + B\ log(N)\right) = O\left(\frac{log(N)}{N}\right)$$





Performance and scaling properties (II)

- Results for hyperparameter tuning with10000 GPUs with parallel random search across
  100 models, trained on 100 GPUs each.
- The time to solution for finding a model of given validation AUC is compared to the scenario of using a single GPU.
  - The inset shows the time required for finding the best model in the scenarios of using 1, 100 and 10000 total GPUs.
  - For 100 GPUs, we distinguish training the 100 models in serial, but using 100 GPUs for each model (“parallel training”) vs. running 100 models in parallel, trained on 1 GPU each (“parallel tuning”), which both achieve nearly 100x speedup.



Conclusions

- As neural network models are getting larger, reduced floating point precision training approaches become crucial
  - Ability to optimize computational throughput and reduce memory and network bandwidths, allowing for faster training
- Develop a domain-specific deep learning framework integrating Keras and TensorFlow with custom parameter averaging and global weight update routines implemented with CUDA-aware MPI intended for disruption forecasting in tokamaks
  - Strong nearly linear runtime scaling on GPU clusters
  - Hyperparameter tuning with O(10000) of nodes
- Evaluated training of deep RNNs with half-precision floats on a real scientific dataset
  - Use scientific JET plasma disruption time series dataset
  - Use loss scaling approach to facilitate model convergence at FP16
  - Obtained comparable validation level performances at the end of each epoch for half and single floating point precisions

Microsoft Cloud+AI SMART Data Science Team is hiring!

Contact: neels@microsoft.com

Answering Documentation Questions using a Seq-to-Seq Deep Learning Model

ML-Enhanced Coding Assistant based on a Markov Chain Model

AI Infused Analytics

Automatic Code Reviews Generation through Deep Learning

Code Risk Analysis using Graph Learning

Backup

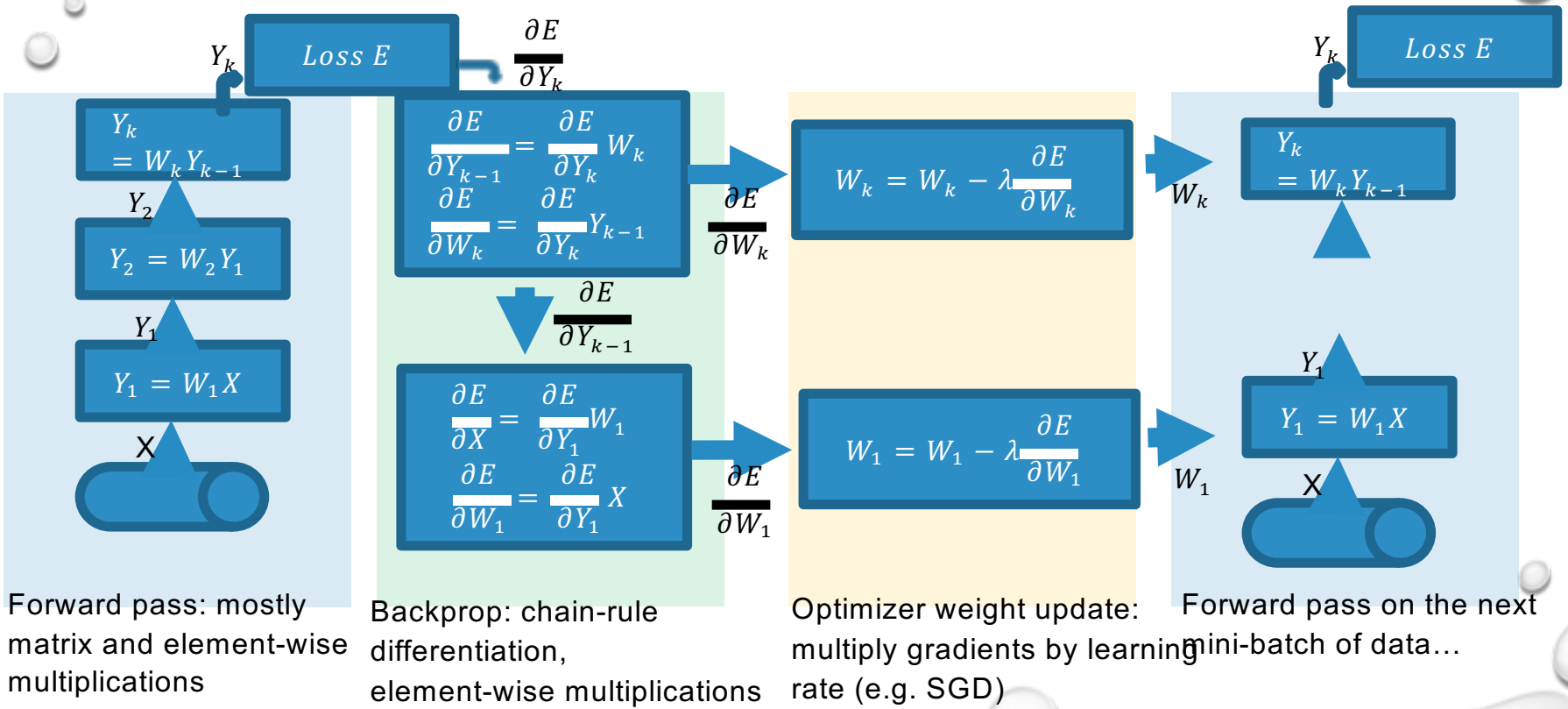
NVIDIA Volta architecture: TensorCores

- What are TensorCores?
  - Programmable matrix-multiply-and-accumulate units. 8 TensorCores per streaming multiprocessor

https://devblogs.nvidia.com/tensor-core-ai-performance-milestones/

  - Each tensor core can perform operations on small 4x4 matrices
  - 1 matrix-multiply-and-accumulate operation per GPU clock
  - During program execution, multiple Tensor Cores are used concurrently by a full warp of execution in GPU

Training flow



Forward pass: mostly matrix and element-wise multiplications

Backprop: chain-rule differentiation, element-wise multiplications

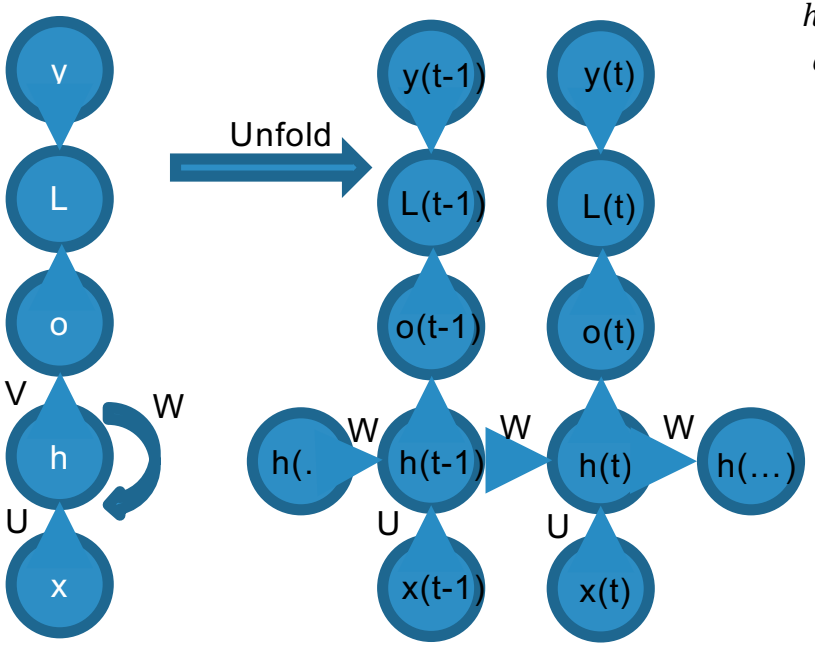
Optimizer weight update: multiply gradients by learning rate (e.g. SGD)

Forward pass on the next mini-batch of data…

Recurrent Neural Nets: basic description

- RNNs are a family of neural networks to process sequential data
- Feed forward equations are recurrent:

$$a(t) = b + Wh(t-1) + Ux(t)$$
$$h(t) = \tanh\big(a(t)\big)$$
$$o(t) = c + Vh(t)$$



Notations:

x – input sequence,
U – is the input to hidden weight matrix,
W - hidden to hidden,
V – hidden to output weights
b,c are the biases
tanh() is the activation function (non-linearity)
o – output sequence
Loss **L** and target values are denoted as **y**

Model convergence at large N

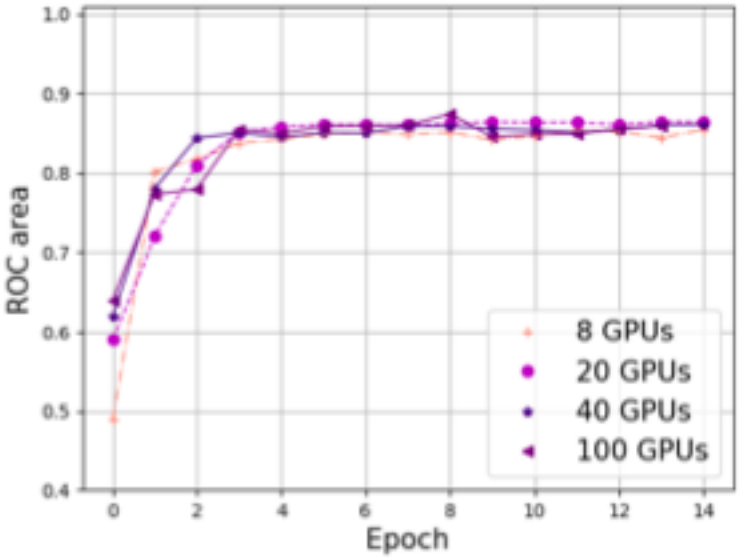
- Learning rate schedule is crucial to facilitate model convergence during distributed training when the effective mini-batch size is large
- We use exponential learning rate decay with adjustable base learning rate

$$\lambda_i = \lambda_0 \gamma^i$$

- Effective batch size is multiplied by the number of workers
- In a distributed regime, the base learning rate is reduced as the number of workers N is increased

$$\lambda_0(N, n) = \frac{\lambda_0}{1.0 + N/n}$$

- Learning rate is clipped



Half-precision training (I)

Half-precision training:

- Store weights, activations and gradients in FP16
- Perform matrix and element-wise multiplications, reduction ops in FP16 (including during fprop, bprop and weight update)

- You already know it is possible to perform FP16 math, arithmetics and comparisons on NVIDIA GPUs with compute capability 5.3 and greater
  - CUDA half intrinsics (round to nearest even): http://docs.nvidia.com/cuda/cuda-math-api/group__CUDA__MATH__INTRINSIC__HALF.html#group__CUDA__MATH__INTRINSIC__HALF
  - TensorFlow has DT_HALF registered type: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/framework/types.cc
  - We add a custom MPI datatype of 2 contiguous bytes: https://github.com/PPPLDeepLearning/plasma-python/blob/master/plasma/primitives/ops.py
- **This talk:**
  - Review techniques to enable FP16 and mixed precision training
  - Introduce FRNN framework for disruption forecasting in tokamak fusion plasmas , discuss strong scaling
  - Evaluate FP16 training on FRNN framework and a real scientific dataset from JET experiment
  - Evaluate FP16 training on the benchmark IMDB dataset

RNN Predictions

True positive example