Training distributed deep recurrent neural networks with mixed precision on GPU clusters

Alexey Svyatkovskiy
Princeton University
Princeton, New Jersey

Julian Kates-Harbeck
Harvard University
Cambridge, Massachusetts

William Tang Princeton
University Princeton, New
Jersey

Presented by Ryan Barton

Matsuoka Laboratory, Tokyo Institute of Technology

Thursday, December 20 2018

Abstract

- Goal: implement a **mixed precision, highly scalable (multi-node)** approach to train & evaluate Recurrent Neural Networks (RNNs)
  - CPU + GPU heterogeneous system
  - Distributed training, synchronous SGD, data-parallel
  - Tensorflow & CUDA-aware MPI
  - Custom learning rate scheduler
- Datasets:

  Joint European Torus (JET)
  - (~4300 plasma shots from a tokamak fusion experiment)

Large Movie Review
  - (~50,000 IMDB reviews, each movie <= 30 reviews)

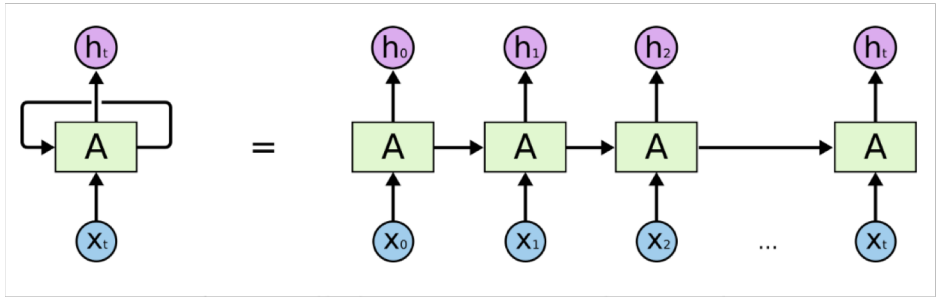
Results: near-linear strong scaling $0\left(\frac{\log(N)}{N}\right)$, where N=num of GPUs

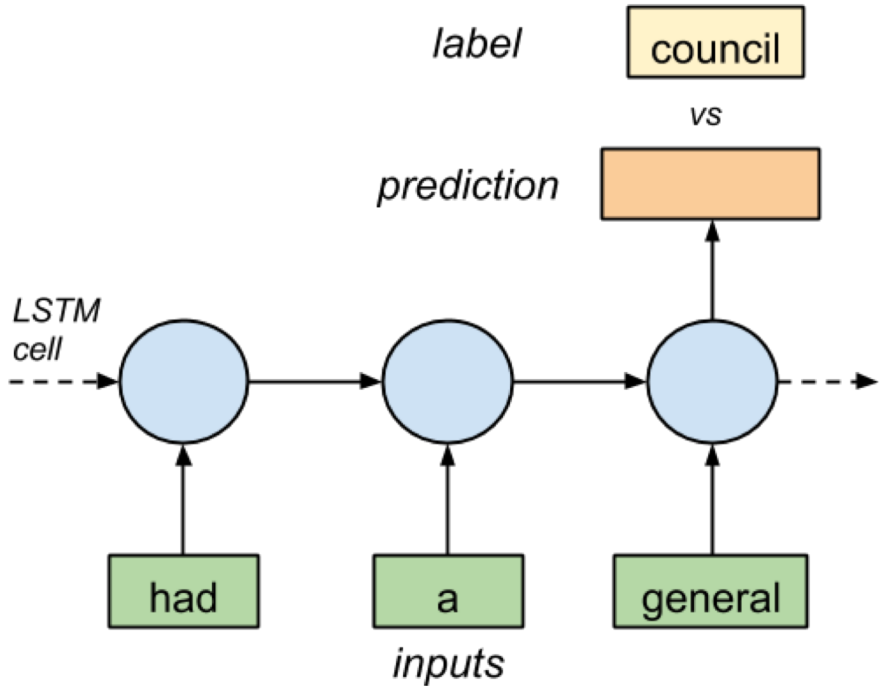
Motivation

- FP16 (half precision) -> less memory & bandwidth
  - Data can fit more neatly in GPU memoryspace
  - Allows for 2x parameters & more (deeper) layers
  - Improves max computational throughput
- At time of publication, no good study on distributing & scaling RNNs
  - Specifically, Long Short-Term Memory (LSTM) ones

Refresher: what is an LSTM?

- A type of RNN where each node has potential to ”remember” information over time.
- Flow to and from nodes can loop back onto themselves (”unrolling” reveals temporal aspect)
- Inputs are ”mini-batches” of vectors.
- Widely used in decision-making situations (continuous speech recognition, etc).





Hardware Environment

“Tiger”, at Princeton University

- 27 petaFlops
- 80 nodes, each w/ 28 cores and 16GB HBM2 memory
- 320 P100 GPUs
- Intel Broadwell E5-2680v4 CPUs
- Intel Omnipath interconnects

Deep Learning Distribution Method

- Based on Tensorflow
- Synchronous SGD w/ parameter averaging (70 mil in total)

STEP	ACTION
Step 1	Initialize parameters randomly
Step 2	MPI *broadcast* parameters to each worker GPU
Step 3	Distribute mini-batch of data m_i to each worker GPU. Each perform forward and backward propagations.
Step 4	MPI *allreduce* to gather gradients (in lock-step). Worker 0 averages them.
Step 5	Worker 0 updates optimizer (if exists) and global weights using these gradients.
Step 6	MPI *broadcast* updated parameters, repeat steps 2-5 for mini-batch m_{i+1}

Learning Rate Scheduler

- After each epoch, update the learning rate

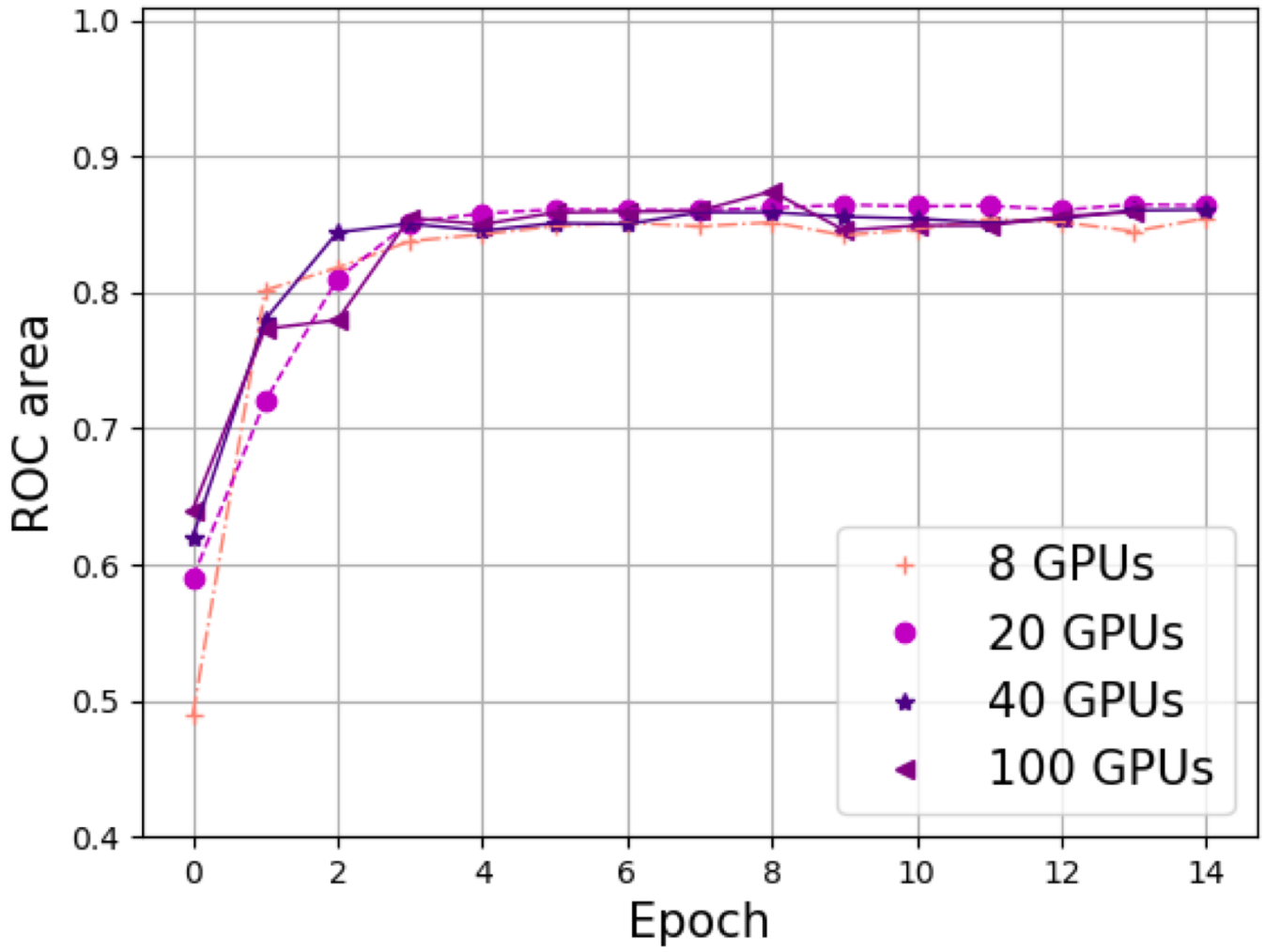
$$\lambda_i = \lambda_0 \delta^{\mathrm{i}}$$

  - λ_0 is the base learning rate (constant)
  - δ^{i} is the a learning rate “decay” (constant)
  - i is the epoch number

- λ_0 is reduced in a distributed configuration, for reliable convergence.
- Value depends on:
  - N (total num of GPUs)
  - n (num of GPUs at which base learning rate is halved).

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

Learning Rate Scheduler (2)



Base learning rate λ_0 = 0.0004
Batch size β = 256.
SGD optimizer with momentum, loss scaling factor: 10.0.

AUTHORS' NOTES

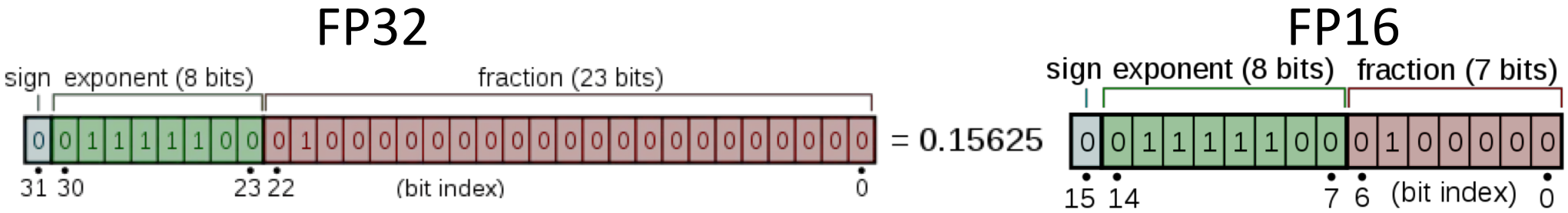
Single (32-bit) precision used.

Effective batch size β proportional to N (num of GPUs)

For larger N = convergence is affected (authors unsure how)

Precision Change

- At the bit level, what is FP16 like compared to FP32?



- Authors' claim FP16 accuracy comparable to FP32 baseline.
- For FP16 training, implemented custom MPI data type
  - 2 contiguous bytes and *allreduce* reduction operation
- Cases where precision is changed:
  - Matrix multiply
  - Parameter averaging and weights/gradients sent across network
  - Weight updating

Timings and Scalability

- Several areas of performance improvement considered
- Synchronous SGD can be split into two timing components:

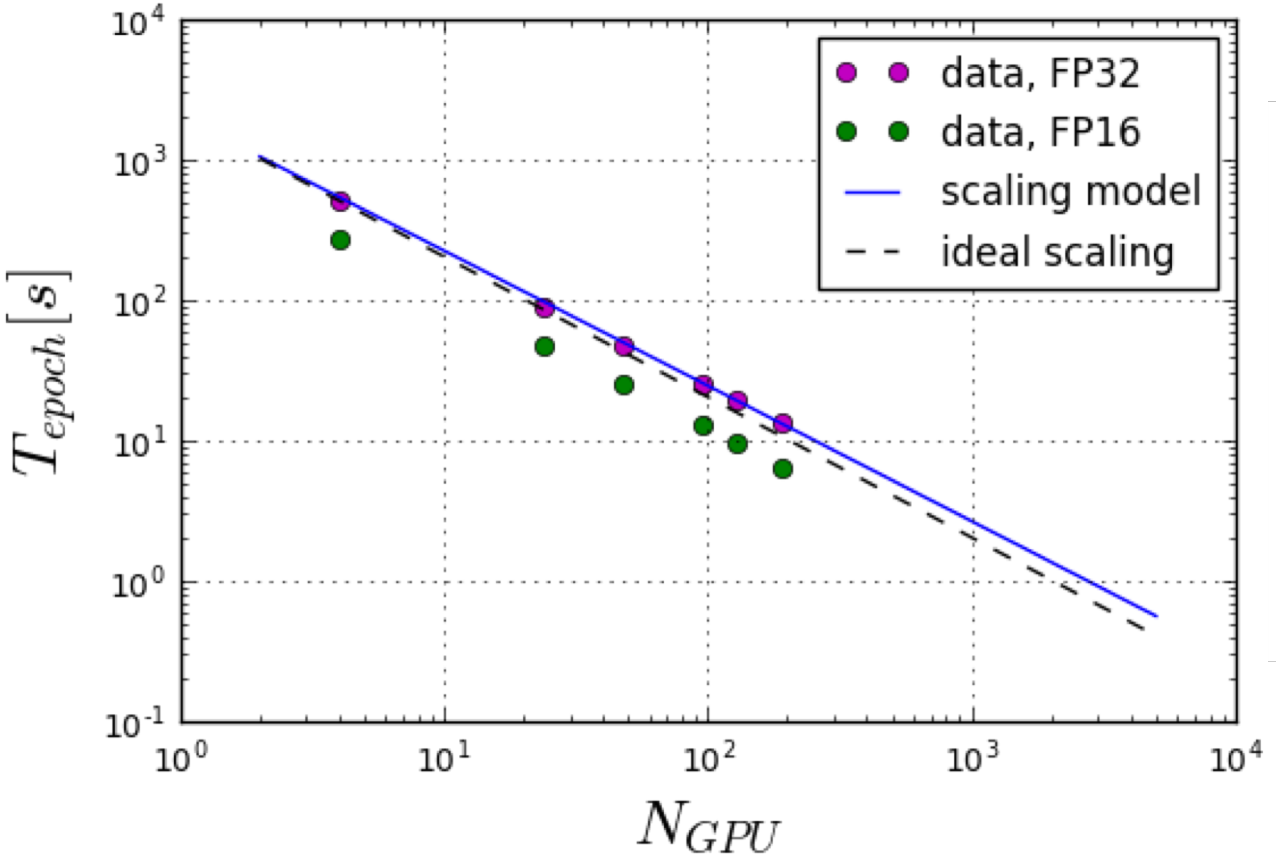
  computation (T_{batch}) and **synchronization (T_{sync})**
- MPI *allreduce* is a tree-like operation, so T_{sync} is O(log(N))
- For 1 mini-batch step, processed data **increases** linearly with increasing N.
  - More workers = more assigned data.
  - Thus, there is less fetching for mini-batches. Implies that for 1 epoch, number of mini-batches required **decreases** linearly with increasing N.

Epoch time scales really well:

$$T_{epoch} \propto \frac{1}{N}(T_{batch}+T_{sync}) = \frac{1}{N}(A+B \cdot log(N)) = O(\frac{log(N)}{N})$$

Timings and Scalability (2)

Thus, claiming epoch timings get **linear scaling** with increasing N



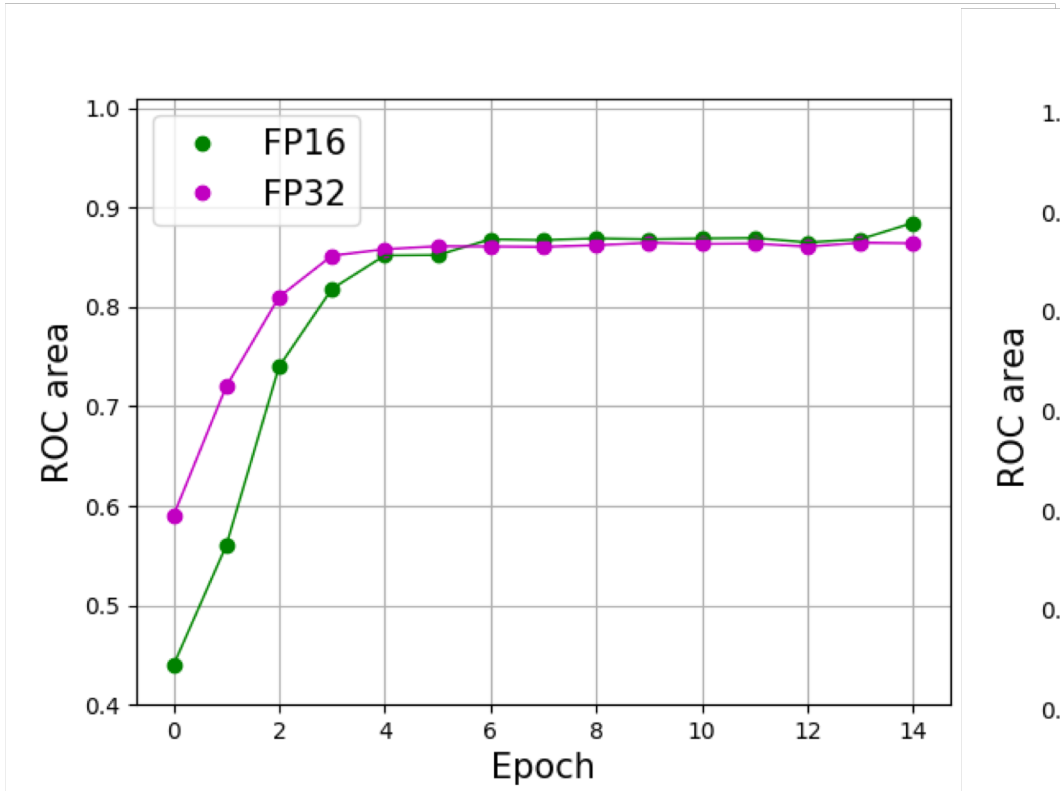
Dataset: plasma disruptivity signals
Npar and Nlayers are max values

Precision	N_{par}	N_{layers}	Batch size
FP64	$4.6 \cdot 10^6$	15	256
FP32	$9.2 \cdot 10^6$	29	256
FP16	$18.2 \cdot 10^6$	58	256
FP64	$18.2 \cdot 10^6$	58	64
FP32	$36.3 \cdot 10^6$	118	64
FP16	$72.1 \cdot 10^6$	234	64

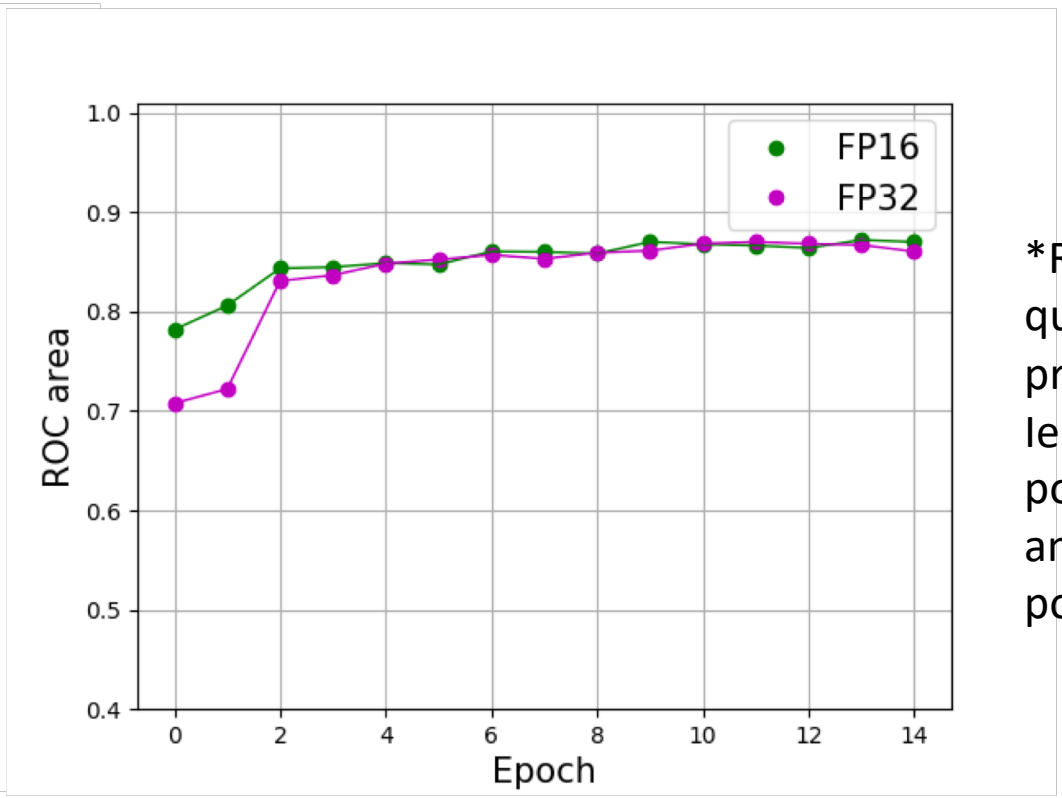
*My criticism: for >100 GPUs, would speedup level off due to so much communication taking place (temporally and spatially)?

Timings and Scalability (3) JET Dataset

- Epoch vs Accuracy, both FP16 and FP32 (baseline) scenarios
- Both cases: plateau @ epoch 6, ROC area =0.87. 8~9 passes to finish training.



Base learning rate λ_0 = 0.0004

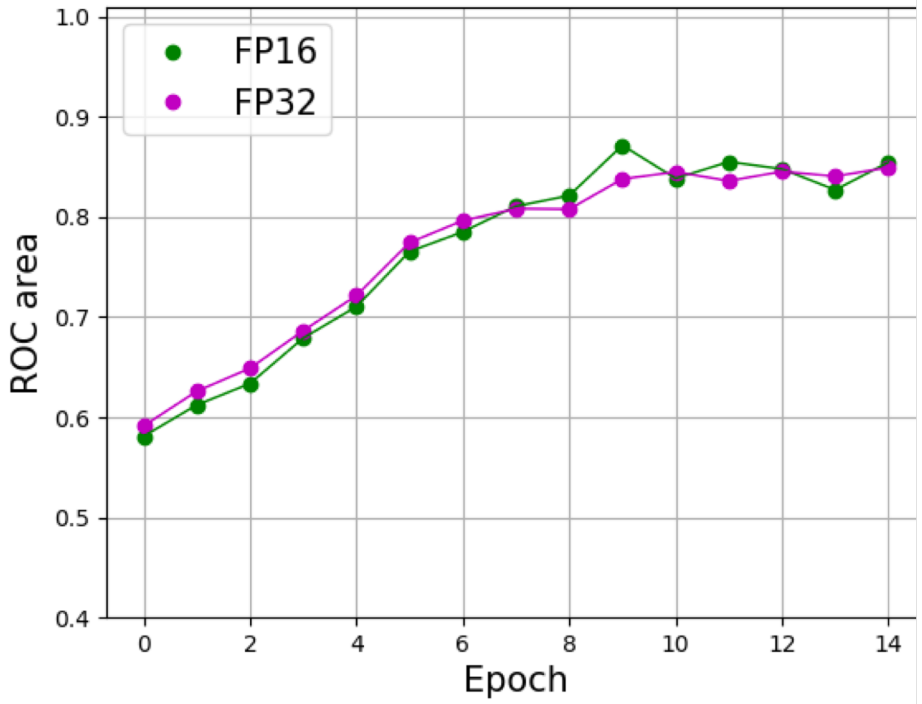


Base learning rate λ_0 = 0.0001

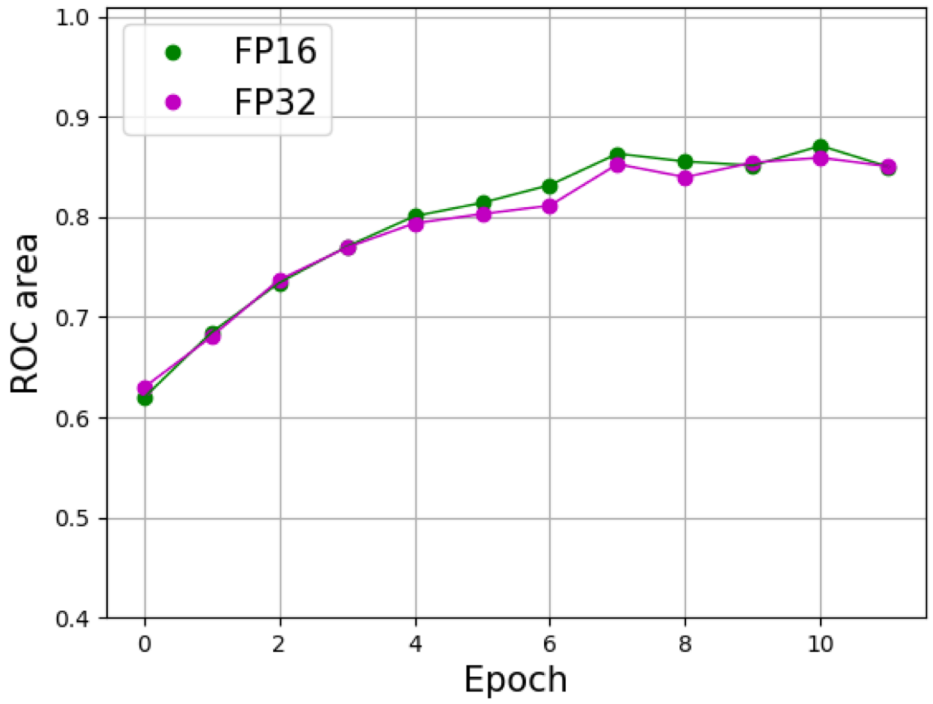
*ROC area is quality of a binary predictor.
Ie. Maximize true positives
and minimize false positives

Timings and Scalability (4) IMDB Dataset

- Epoch vs Accuracy, both FP16 and FP32 (baseline) scenarios
- Left case: plateau @ epoch 9, ROC area = 0.86
  Right case: plateu @ epoch 6, ROC area = 0.86



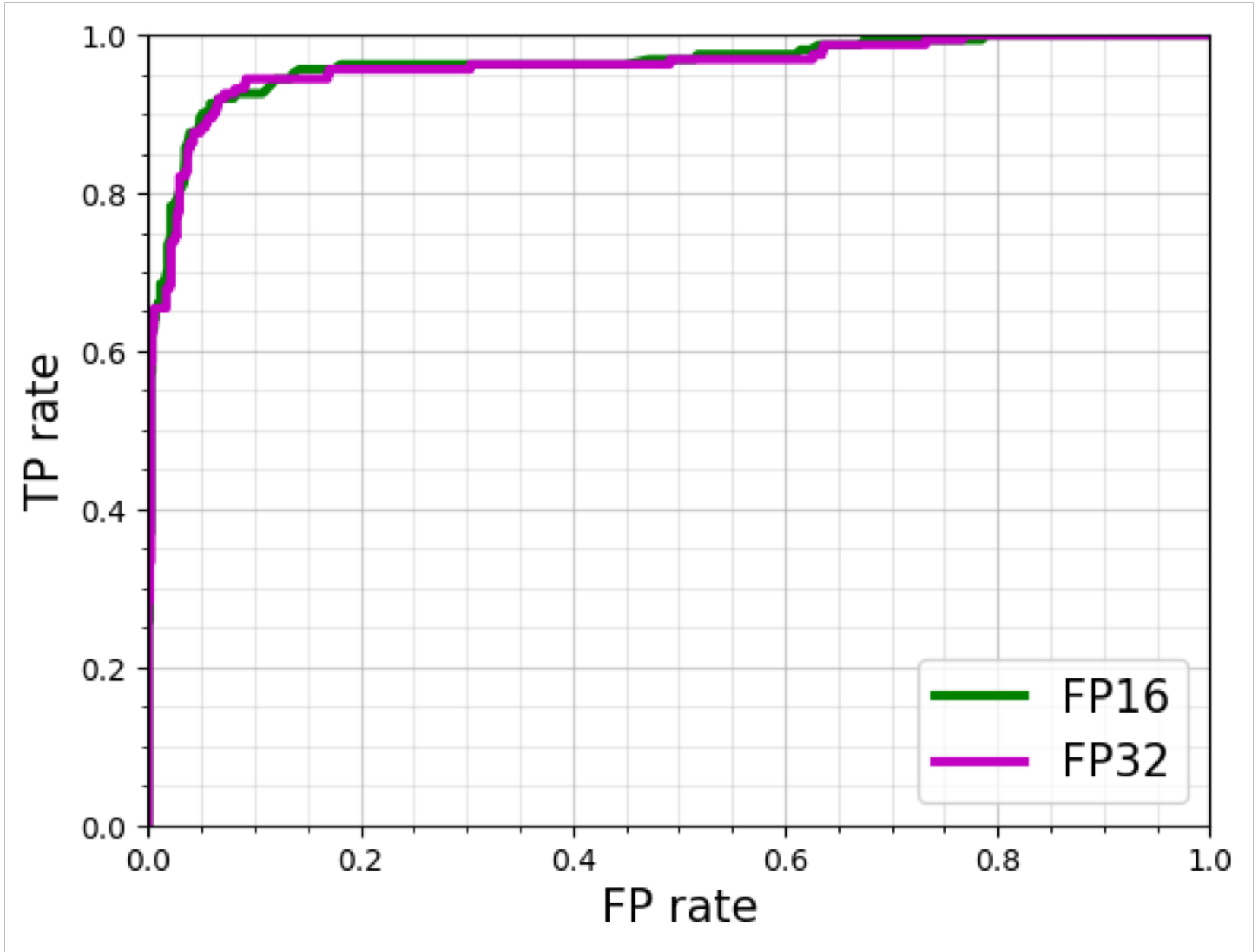
Base learning rate λ_0 = 0.02



Base learning rate λ_0 = 0.05

*ROC area is quality of a binary predictor.
Ie. Maximize true positives
and minimize false positives

Timings and Scalability (5) FP vs TP rate



Other important factors (drawbacks?)

- OpenMPI provides theoretical max ~10GB/s bandwidth (through GPUDirect Random Device Memory Access)
- Team's Omnipath environment yielded ~6.25GB/s. SLOW!
- Thus, 1 Synchronized SGD iteration :

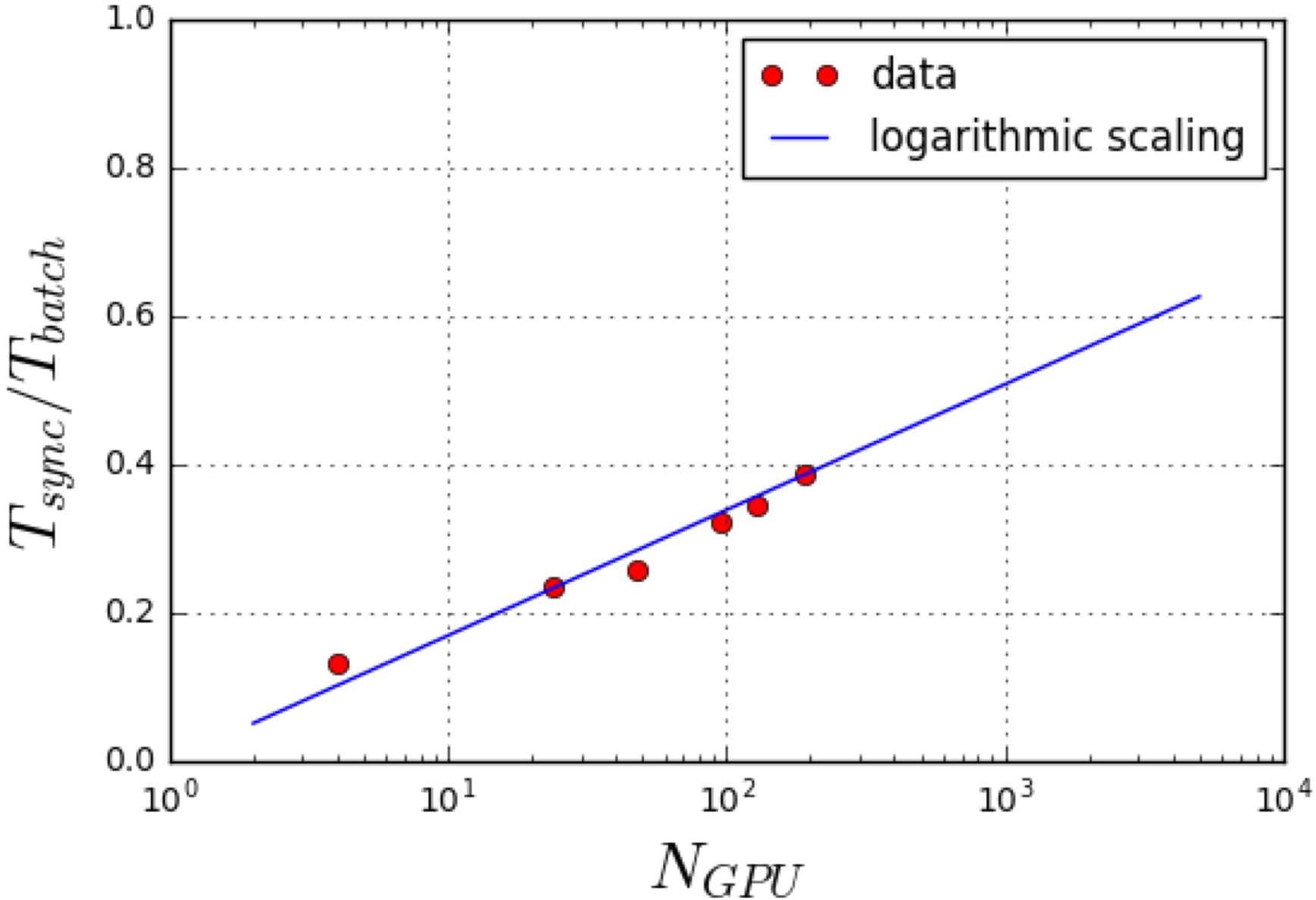
$$\frac{Net\ gradient\ size}{Bandwidth} = \frac{9.4\ GB/iter}{6.25\ GB/s} \approx 1500\ ms/iter$$

- To maintain accuracy between FP16 and FP32 (baseline) runs, loss function includes scalar multiplier a
  - *Before* back propagation partial derivatives
  - Without this, convergence at FP16 would **not be possible**

$$L(y) = \alpha \cdot max(0, 1 - t \cdot y)$$

t = classification label
y = predicted outcome of RNN

The Great Finding



Number of GPUs vs Ratio of Sync time per mini-batch

Conclusion

- Employing mixed precision (ie. reducing FP32 to FP16) in a multi-GPU distributed synchronous SGD LSTM revealed:
  - Surprising strong scalability (linear)
  - Maintained accuracy (ROC) from FP32 -> FP16.
  - Thanks to FP16, ability to accommodate
    - larger neural networks
    - larger batch sizes
    - >>70 mil parameters (?)

Thank you!