COMS E6998 010

Practical Deep Learning Systems Performance

Lecture 9 10/12/20

Logistics

- Homework 4 has been posted Nov. 9. Due Nov. 22 by 11:59 PM.
- Final project presentation
  - Possible dates: Dec 14, 16,18
  - Record your presentation (15 min/student) and submit.
  - We will replay presentation from each team and ask questions in zoom meeting.

Recall from last lecture

- Distributed ML performance modeling in Hemingway; Decomposition into system level model and algorithm level model
- DL performance modeling in Optimus; Training loss model (training loss as a function of number of epochs) and Resource-speed model (time per iteration as a function of number of workers and PS shards
- Reinforcement learning and its formulation **33**
- Q-learning, DQN algorithm, Experience replay
- Distributed DQN using GORILA
- Prioritized experience replay and Ape-X
- Deep RL simulators like OpenGym
- RLLib open source library for RL

Deep Reinforcemnt Learning will be covered in a later lecture

Deep Learning Benchmarks

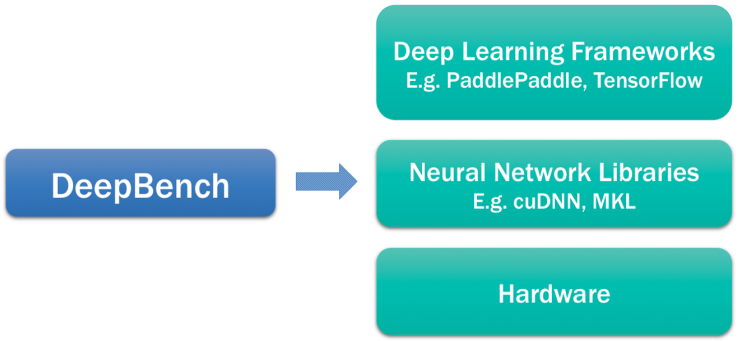
- DL jobs (training and inference) are computationally expensive
- Research targeted at reducing computation time and cost
  - software systems, training algorithms, communication methods, hardware
- Different optimizations target different performance metrics
- Lack of standard evaluation criteria (datasets and metrics) to objectively study and compare the benefits of different results

Tensorflow benchmarks

- https://github.com/tensorflow/benchmarks
- Two projects:
  - Perfzero: benchmarking framework for tensorflow
  - Tensorflow CNN benchmarks (/tf_cnn_benchmarks)
- Problem with throughput based benchmarks like tf_cnn_benchmarks
  - Just looking at throughput or time to process one mini-batch of data can be misleading
    - Speed-up techniques like larger batch sizes, reduced precision, and asynchronous updates can prevent an algorithm from converging to a good result, or increase the time to do so
- Need metrics capturing end-to-end performance of DL jobs
  - Not just throughput !

DeepBench

- https://github.com/baidu-research/DeepBench
- Released in 2016 by Baidu Research
- Open source benchmarking tool
- Measures performance of basic operations involved in training and inference of deep neural networks
  - Dense matrix multiplication, convolution, recurrent layers (with different activations), communication (AllReduce)



- Uses neural network libraries to benchmark the performance of basic operations on different hardware
- Does not work with deep learning frameworks or deep learning models built for applications
- Can identify bottlenecks in different operations
- Cannot benchmark end-to-end training time or inference time

"Which hardware provides the best performance on the basic operations used for training deep neural networks?"

Kernel level and Model level benchmarking

Kernel level
- Hardware vendors
- DeepBench

Application level
- AI system engineers
- DAWNBench, MLPerf

DAWNBench

- Open benchmark for end-to-end deep learning training & inference
- Measures *end-to-end* performance of training (e.g., time, cost) and inference (e.g., latency, cost) at a specified accuracy level
- Developed at Stanford University; predecessor of MLPerf
- DAWNBench evaluates deep learning systems on different tasks based on several objective metrics, using multiple datasets.
  - Allows experimentation of new model architectures and hardware

Tasks	Metrics	Datasets
Image classification	Training time Training cost	ImageNet CIFAR10
Question answering	Inference latency Inference cost	SQuAD

Table 1: Dimensions evaluated in the first version of DAWNBench. All metrics are for a near-state-of-the-art accuracy.

Stanford **Qu**estion **A**nswering **D**ataset (SQuA

DAWNBench First Release

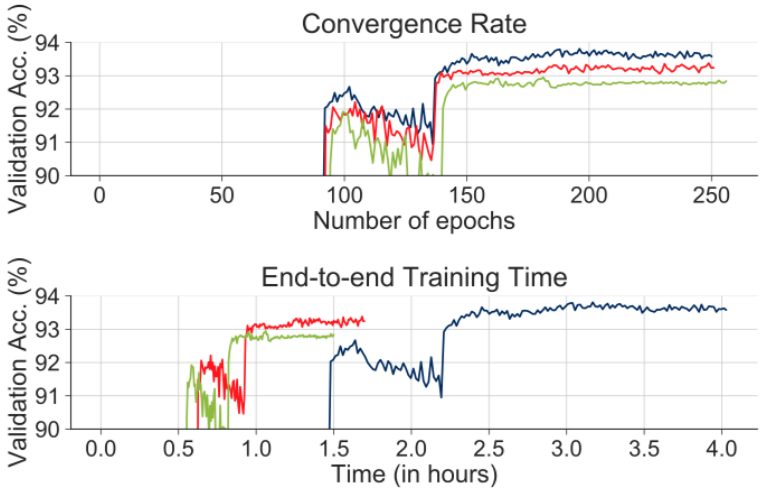
- **Two DL tasks**: image classification on CIFAR10 and ImageNet, and question answering on SQuAD
- **Four Metrics**:
  1. Training time to a specified validation accuracy (TTA)
  2. Cost (in USD) of training to a specified validation accuracy using public cloud instances
  3. Average latency of performing inference on a single item (image or question)
  4. Average cost of inference for 10,000 items
- Provided reference implementations and seed entries in PyTorch and Tensorflow
  - Adapted from official repositories on Github
  - Produce similar accuracy numbers as reported in the original research papers
  - Conforming to performance recommendations published with these frameworks

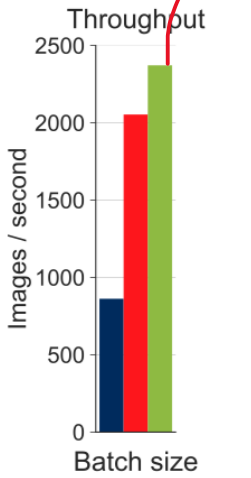
DAWNBench Vs other DL Benchmarks

- Benchmarks prior to DAWNBench
  - Baidu DeepBench
  - Tensorflow CNN benchmarks
  - Fathom
- DAWNBench focuses on end-to-end performance
  - Tensorflow CNN benchmarks use the time needed to train on a single minibatch of data as the key metric, while disregarding the resulting accuracy of the trained model
  - DeepBench and Fathom focuses on timing individual low-level operations (e.g. convolutions, matrix multiplications) utilized in deep learning computations
- Rolling submissions to DAWNBench ended on **3/27/2020**

TTA vs Minibatch Size

ResNet56 CIFAR10 model on a P100.





Batch size = 32 Batch size = 256 Batch size = 2048

large batch size better saturate GPU cores
requires less weight updates

Accuracy-Time tradeoff

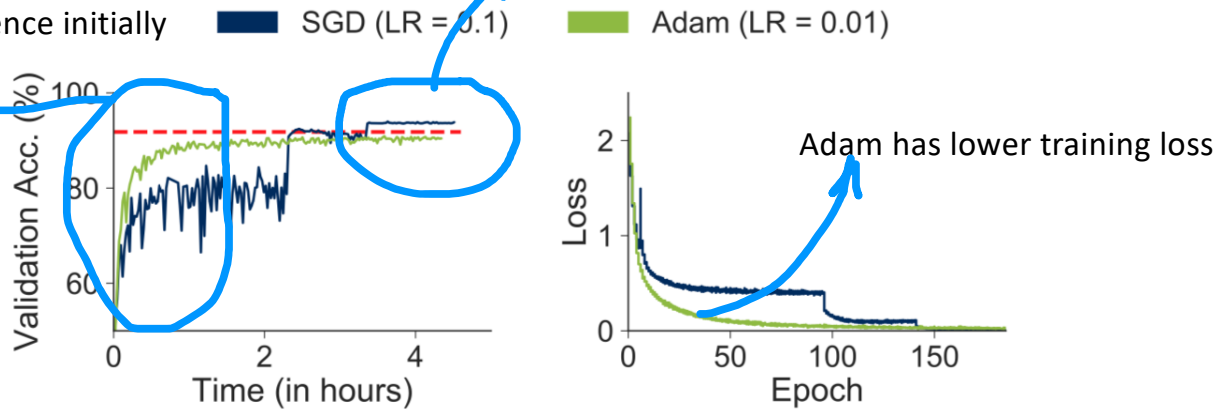
Minibatch size of 32 produces the best convergence rate (least number of epochs to highest accuracy), and a minibatch size of 2048 produces the best throughput (number of images processed divided by total time taken). Minibatch size of 256 represents a reasonable trade-off between convergence rate and throughput. Minibatch size of 256 reaches an accuracy of 93.38%, which is only 0.43% less than the maximum accuracy achieved with a minibatch size of 32, in 1.9x less time

Coleman et al. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. NIPS 2017

Why accuracy and not loss threshold ?

SGD (with momentum) converges to a better accuracy

Adam has faster convergence initially



Adam has lower training loss

Figure 2: Impact of different optimizers while training a ResNet56 model on CIFAR10. The graph on the left plots the top-1 validation accuracy with respect to time, and the graph on the right plots a rolling average of the loss with respect to the epoch. We see that Adam initially outperforms SGD with momentum, but falls short around epoch 100 (~2 hours in the graph on the left), and does not reach a validation accuracy of 93%.

Coleman et al. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. NIPS 2017

Interaction Between Optimizations

- Do benefits from different optimizations compose?
  - Will the benefits add up (e.g., 2x with O1 and 3x with O2 will translate to 6x with O1+O2)
- Example scenario considering 3 candidate optimizations:
  - ADAM optimizer
  - Single-node multi-GPU training
  - Stochastic Depth*
    - Regularization similar to dropout
    - Entire layers are randomly dropped during training,
    - Full network is used to perform inference
  - Larger batch size

*Huang et al. Deep Networks with Stochastic Depth

Composing Optimizations

- Do different optimizations compose well ?

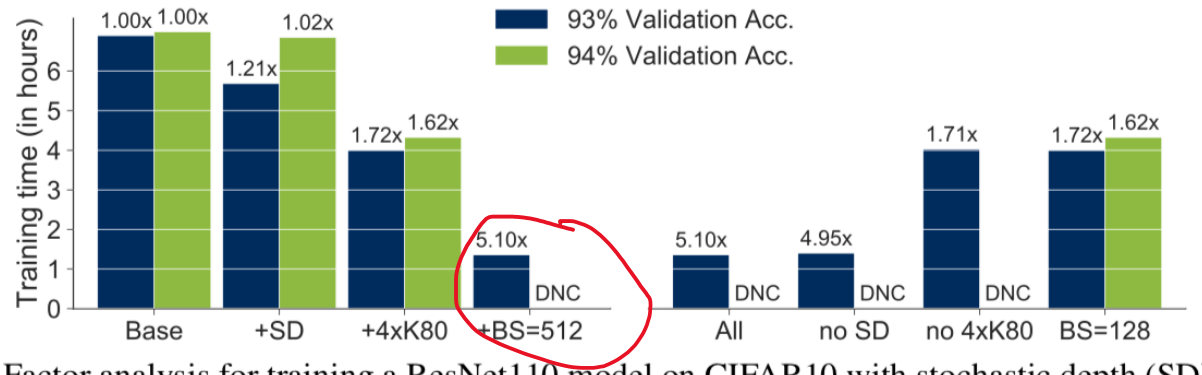


Figure 3: Factor analysis for training a ResNet110 model on CIFAR10 with stochastic depth (SD), 4 Nvidia K80 GPUs on a single node (4xK80), and a minibatch size (BS) of 512. Cumulatively enabling each optimization reduces the time to 93% top-1 accuracy, but combined, the model does not converge (DNC) to the 94% accuracy threshold. By removing the larger minibatch size, the model reaches the higher accuracy target.

Optimizations can interact in non-trivial ways when used in conjunction, producing lower speed-ups and less accurate models

Coleman et al. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. NIPS 2017

Variability Due to Hardware and Software

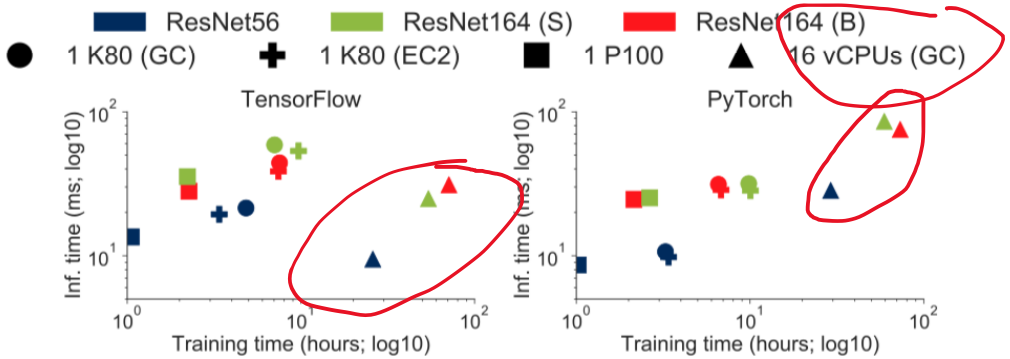
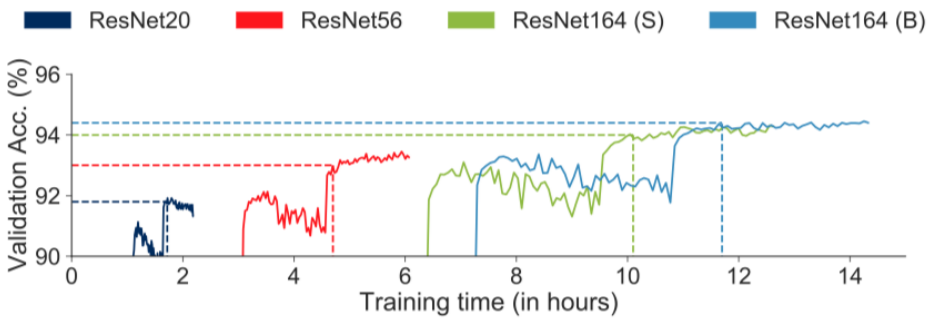


Figure 5: Inference time vs. training time to 93% validation accuracy, for different hardware, frameworks, and model architectures in DAWNBench's seed entries. ResNet164 (S) uses a simple building block, while (B) uses a bottleneck building block. Amazon EC2 instances use a `p2.xlarge` instance type (4vCPUs, 61 GB memory).

- TensorFlow is faster than PyTorch on CPUs, but slightly slower on GPUs, both for training and inference
- Training and inference time are proportional to the depth of the model

Coleman et al. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. NIPS 2017

Identifying Optimal DL Architecture



Accuracy threshold	Time to threshold	Fastest model
91.8%	1h 43m	ResNet20
93%	4h 42m	ResNet56
94%	10h 6m	ResNet164 (simple)
94.4%	11h 42m	ResNet164 (bottleneck)

Figure 6: Validation accuracy vs. training time for different ResNet architectures on CIFAR10. Horizontal lines indicate accuracy thresholds of 91.8%, 93%, 94%, and 94.4%. ResNet20, ResNet56, ResNet164 (with simple building blocks), and ResNet164 (with bottleneck building blocks) are fastest to the corresponding accuracy thresholds.

For lower accuracy thresholds, shallower architectures reach the threshold faster.

Coleman et al. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. NIPS 2017

Training Cost vs Training Time

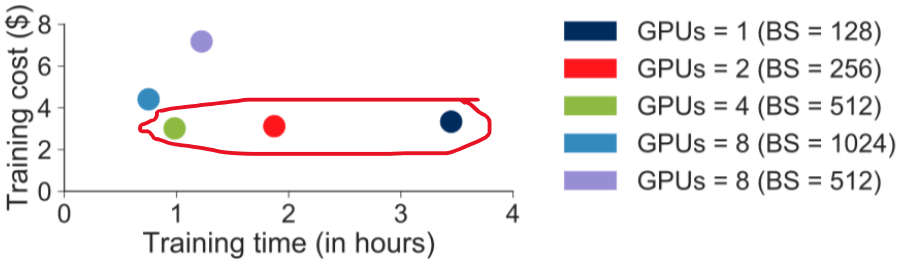


Figure 7: Training cost vs. training time for ResNet56 on the CIFAR10 dataset, using different numbers of GPUs, with an accuracy threshold of 92.5%. The cost of training stays roughly the same, regardless of the number of GPUs used, until 8 GPUs. Training time scales almost linearly with the inverse of the number of GPUs.

- Scaling from 1 to 4 GPUs
  - Training time scales perfectly linearly with the inverse of the number of GPUs used
  - Cost remains constant despite training time going down
- Scaling to 8 GPUs
  - Increase in the cost of training
  - Training time does not decrease enough to counter the doubling in instance cost per unit time (related to scaling efficiency)

Coleman et al. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. NIPS 2017

MLPerf

- https://mlperf.org
- Training: https://mlperf.org/training-overview/#overview
- Inference: https://mlperf.org/inference-overview/#overview

Time to Accuracy (TTA) Metric

- TTA measures time for a system to train to a target, near-state-of-the-art accuracy level on a held-out dataset
- TTA combines both generalization and speed
- Dawnbench was the first multi-entrant benchmark competition to use the TTA metric
- MLPerf benchmark also uses TTA as its primary metric
- Entries compete to achieve target accuracy in the fastest time
  - Imagenet training from 30 mins to less than 2 mins
  - Very large-scale distributed training, with large batch sizes, GPUs, CPUs, TPUs
  - Major companies Google, Intel, NVIDIA compete with optimized solutions with the goal to reduce TTA
  - Entries provide an opportunity to study ML systems optimized heavily for *training performance*

DL Training

- Tradeoffs in Speed and Generalization
  - Increasing batch size can increase hardware efficiency, but may prevent or slow down convergence
  - Naively reducing floating point precision to 16 or 8 bit can prevent convergence
  - Synchronous vs asynchronous SGD. Synchronous is slower but converges faster, asynchronous is faster but converges slower
- Stochasticity in Training
  - Randomness in model initialization and data traversal
  - System introduced stochasticity for improved hardware efficiency, e.g., reordering floating point operations.
  - TTA will differ across multiple trials of the same job

Questions about TTA and DL systems performance

- Is TTA metric stable or do the entries to these metrics only represent the best result out of many trials?
- Do models optimized for TTA still generalize well ?
  - Are they implicitly adapting to the held-out dataset used in the benchmark through extensive hyperparameter tuning?
- How close are these entries from fully utilizing hardware platforms and what are the computational bottlenecks?

Variability of TTA

Coefficient of Variation % = (standard deviation/mean)x100

Coefficient of Variation of TTA for DAWNBench

Entry name	Coeff. of variation	Frac. of runs
ResNet-50, `p3.16xlarge`	5.3%	80%
ResNet-50, 4x`p3.16xlarge`	11.2%	60%
ResNet-50, 8x`p3.16xlarge`	9.2%	100%
ResNet-50, 16x`p3.16xlarge`	12.2%	100%
ResNet-50, 1xTPU	4.5%	100%
AmoebaNet-D, 1xTPU	2.3%	100%
ResNet-50, 1/2 TPU Pod	2.5%	100%

Coefficient of Variation of TTA for MLPerf

Entry	Coeff. of variation
ResNet, NVIDIA, 1xDGX-1	6.7%
SSD, NVIDIA, 1xDGX-1	0.5%
SSD, NVIDIA, 8xDGX-1	6.7%
Mask, NVIDIA, 1xDGX-1	3.9%
Mask, NVIDIA, 8xDGX-1	0.8%
GNMT, NVIDIA, 1xDGX-1	0.2%
Transformer, NVIDIA, 1xDGX-1	13.8%

Low coefficient of variation => low variability in TTA across runs

TTA is quite stable across different runs

Coleman et al. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. 2019

Generalization of Optimized Models

- Evaluation on new data for image classification
  - Labeled set of 2,864 images from Flickr
  - Only images posted after January 1st, 2014 were used (no overlap with Imagenet images)
  - Images spanned 886 (out of 1000) classes.

Model	Accuracy (top-5, unseen data)
ResNet-18 (pretrained)	89.5%
ResNet-50 (pretrained)	92.2%
ResNet-152 (pretrained)	93.2%
ResNet-50, 1xTPU	92.6%
ResNet-50, p3.16xlarge	91.9%
ResNet-50, 4xp3.16xlarge	91.3%
ResNet-50, 8xp3.16xlarge	91.5%
ResNet-50, 16xp3.16xlarge	91.3%
AmoebaNet-D, 1xTPU	91.3%

(a) DAWNBENCH submissions, top-5 accuracy. ResNet-50 on p3.16xlarge instances used non-standard optimizations such as progressive resizing.

Model	Accuracy (top-1, unseen data)
ResNet-18 (pretrained)	71.7%
ResNet-50 (pretrained)	77.4%
ResNet-152 (pretrained)	79.4%
ResNet-50, DGX-1	77.6%

(b) MLPERF submission, top-1 accuracy.

- Models optimized for TTA achieve nearly the same accuracy or higher than the pre- trained ResNet-50
- Optimizing for TTA does not sacrifice generalization performance

Coleman et al. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. 2019

System Scale vs TTA vs System Throughput

Model	System scale	BSes	Epochs	Thpt. speedup	TTA speedup
Trans.	1, 24	10k, 492k	2, 6	10.9×	3.6×
GNMT	1, 32	1k, 8.2k	3, 5	10.9×	6.5×
ResNet	1, 80	4k, 16k	63, 82	28.2×	21.6×
SSD	1, 8	1.2k, 2k	49, 55	4.6×	4.1×
Mask R-CNN	1, 8	32, 128	13, 14	4.2×	3.9×

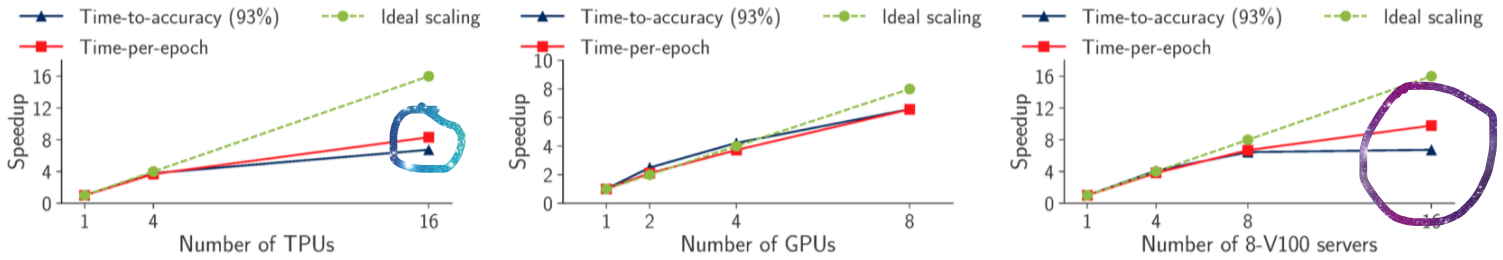
Table 7: Model, system scale (in number of DGX-1s), batch size (BS), number of epochs for convergence, throughput speedup, and TTA speedup. Numbers are given for two system scales per model using official MLPERF entries. As shown, throughput does not directly correlate with TTA and speedups can differ by up to 3× (10.9× vs 3.6× for transformer).

Increasing the batch size to increase throughput may increase the number of epochs required for convergence

System scale does not correlate with Thpt speedup and TTA speedup

Coleman et al. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. 2019

Scaling of TTA with Distributed Training



(a) AmoebaNet across TPUs, TPU pod. **(b)** ResNet-50 within `p3.16xlarge` server. **(c)** ResNet-50 across `p3.16xlarge` servers.

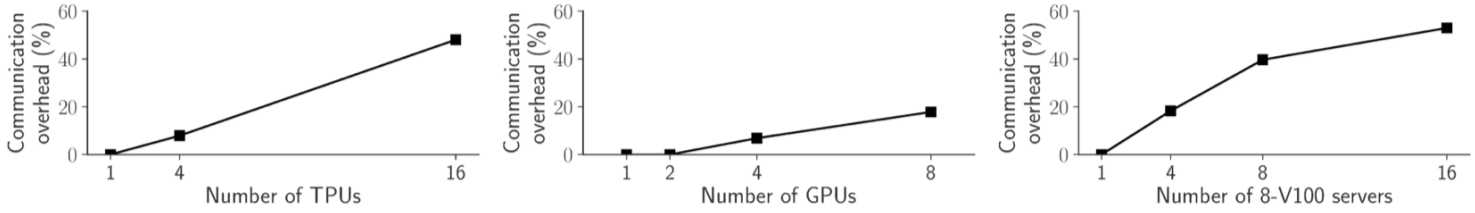
Figure 3: Speedup with respect to a single worker vs. number of workers for three ImageNet models, one on a TPU pod, another on a single `p3.16xlarge` instance with 8 NVIDIA V100 GPUs, and a third on multiple `p3.16xlarge` instances for selected official DAWNBENCH entries. As the number of workers increases, the scaling performance drops off (over 2× gap from ideal scaling).

Within a TPU pod, TPU devices (TPUs) are connected connected to each other over dedicated high-speed networks

- Both time-per-epoch and TTA scale almost linearly with the number of workers *within* a server
- Both time-per-epoch and TTA do not scale as well for training that spans multiple servers
- TTA scales worse than time-per-epoch
  - Greater number of epochs are needed to converge to the same accuracy target for the larger minibatch size.

Coleman et al. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. 2019

Communication Overhead



(a) AmoebaNet across TPUs, TPU pod. **(b)** ResNet-50 within `p3.16xlarge` server. **(c)** ResNet-50 across `p3.16xlarge` servers.

Figure 5: Percentage of time in an epoch spent communicating vs. number of workers for three ImageNet models, one on a TPU pod, another on a single `p3.16xlarge` instance, and a third on multiple `p3.16xlarge` instances. Within a 8-V100 server, communication overhead is low (17.82%), but cross-machine communication is more expensive (53%).

- Communication is a bottleneck in DL at scale
- Communication optimized libraries like Horovod are helpful
- Gradient compression/quantization techniques are helpul; Need to be integrated with standard frameworks

Coleman et al. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. 2019

Single Worker Utilization: Roofline Analysis

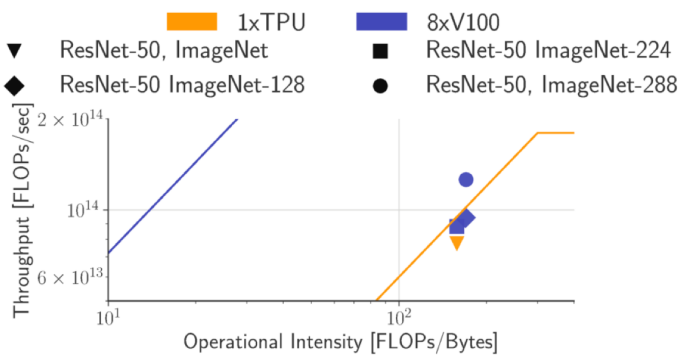


Figure 7: Roofline models for the various DAWNBENCH entries. All of the entries under-utilize the hardware resources, by up to 10×.

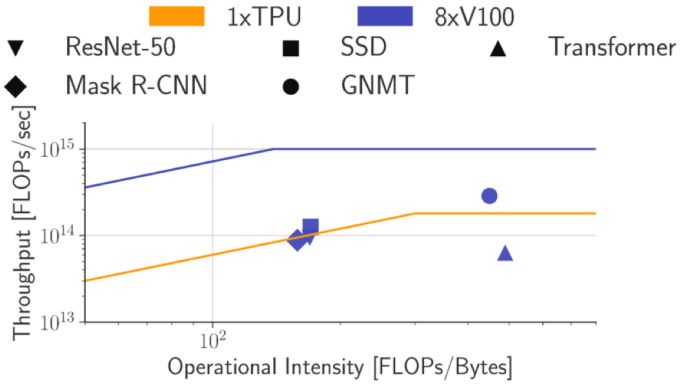
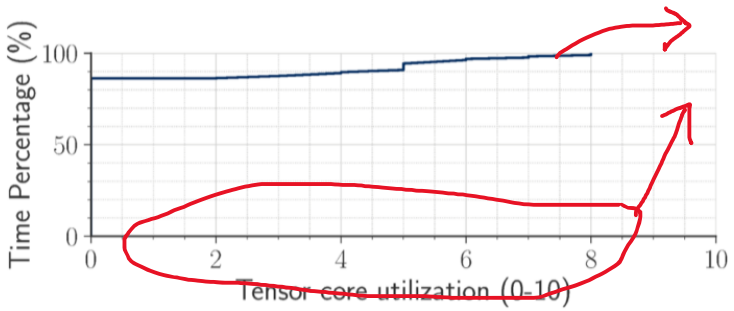


Figure 8: Roofline models for the various MLPERF entries. All of the entries under-utilize the hardware resources, by up to 10×.

Single Worker Utilization: Training Bottleneck

- Tensor core utilization of each GPU kernel in Pytorch



85% of the time is spent on kernels which have less than 8% tensor core utilization

Figure 9: CDF of tensor core utilization for the `fast.ai` ResNet50 model trained with `fp16` precision submitted to the DAWNBENCH competition. About 85% of time is spent on kernels that don't utilize the NVIDIA Tensor Cores *at all*, and no kernel achieves full utilization of the Tensor Core units.

Coleman et al. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. 2019

Single Worker Utilization: Kernel throughput

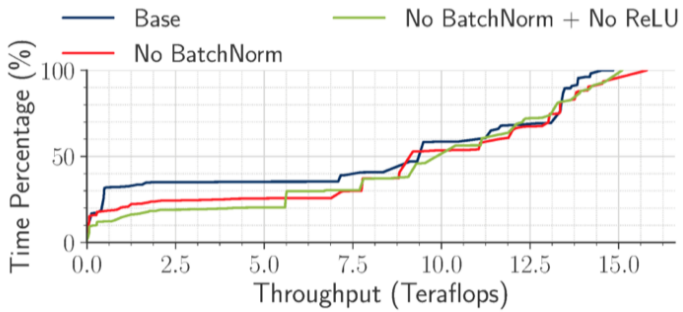


Figure 11: CDF of per-kernel throughput for ResNet50 models trained with `fp32` precision. The CDF is computed by percentage of time spent executing each GPU kernel. A standard ResNet-50 model spends about 40% time in low-throughput kernels (< 6 Teraflops). Removing the BatchNorm layer from the ResNet50 model decreases the percentage of time in low-throughput kernels to about 30%; removing the ReLU layers decreases this further.

- Memory-bound kernels like BatchNorm and ReLU take a significant percentage of total runtime.
- Optimizations like loop and kernel fusion can help reduce the impact of memory-bound kernels by reducing the number of DRAM reads and writes made

Coleman et al. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. 2019

Kaggle

- Online open community of data scientists; Owned by Google
- Huge repository of community published data and code
- Share data, review kernels
- What are kernels ?
- Features:
  - Datasets
  - Competitions
  - Support GPU enabled Jupyter notebooks
- Visit https://www.kaggle.com
  - Review two ongoing competitions
  - Kaggle members collaborate and compete in competitions
- Participation in competition can also earn 10%

OpenML

- https://www.openml.org

ONNX

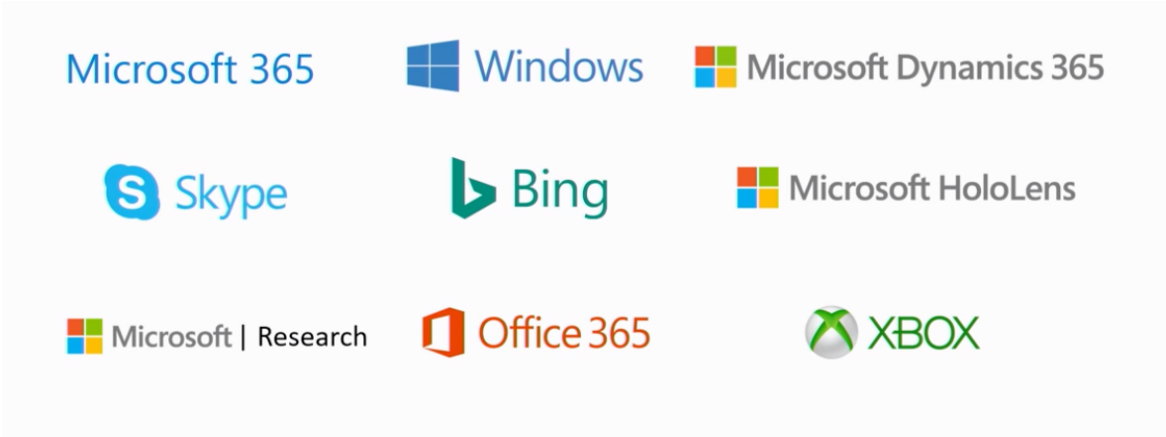
- Open Neural Network Exchange
- An open format to represent deep learning models
- ONNX Features
  - Framework interoperability
    - Models trained in one DL framework to be transferred to another for inference
  - Hardware optimizations
    - ONNX-compatible runtimes and libraries designed to maximize performance on specific DL hardware
- Supported by a community of over 20 leading companies
- Visit http://onnx.ai

ONNX Capabilities

- Supported Tools
  - Visit http://onnx.ai/supported-tools
- ONNX Tutorials
  - Visit https://github.com/onnx/tutorials
- Model Zoo
  - A collection of pre-trained, state-of-the-art models in the ONNX format contributed by community members
  - Visit https://github.com/onnx/models

ONNX in Enterprise: Microsoft

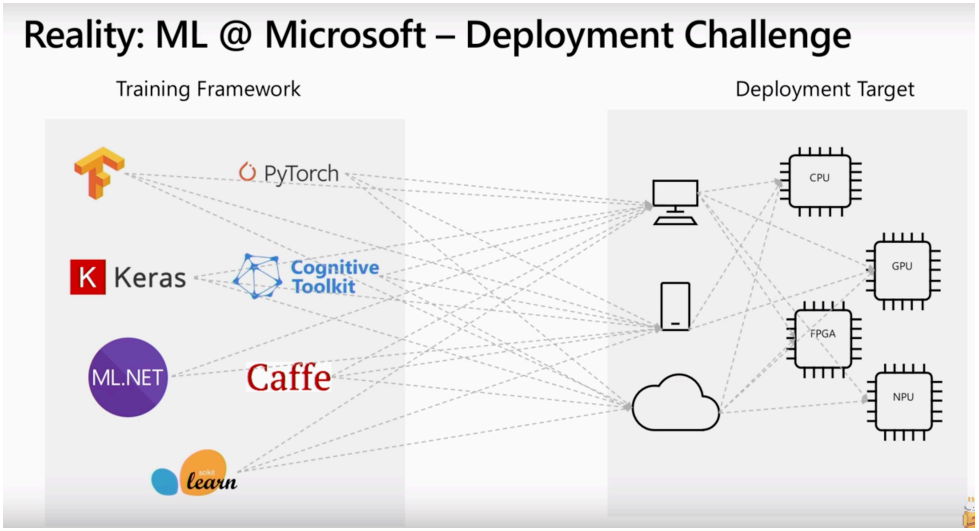
ML used in many products to improve performance and productivity



- AI developers work with different frameworks
- Deployment of trained models to production
  - Different deployment targets: cloud, IoT devices, edge devices
  - Different hardware: CPUs, GPUs, TPUs, FPGAs

Youtube Video: Open Neural Network Exchange (ONNX) in the enterprise: how Microsoft scales ML

Multiple ML frameworks + multiple deployment targets



- Maintaining multiple frameworks in deployment
  - Not scalable
  - Hard to maintain
  - Degrades application performance
    - Frameworks compete for system resources
- Decouple training and deployment frameworks

Youtube Video: Open Neural Network Exchange (ONNX) in the enterprise: how Microsoft scales ML

Bridge Model Training and Deployment

Open and Interoperable industry wide standard

Build and train model in framework A → Convert trained model in framework A to ONNX model → Convert ONNX model to framework B and deploy

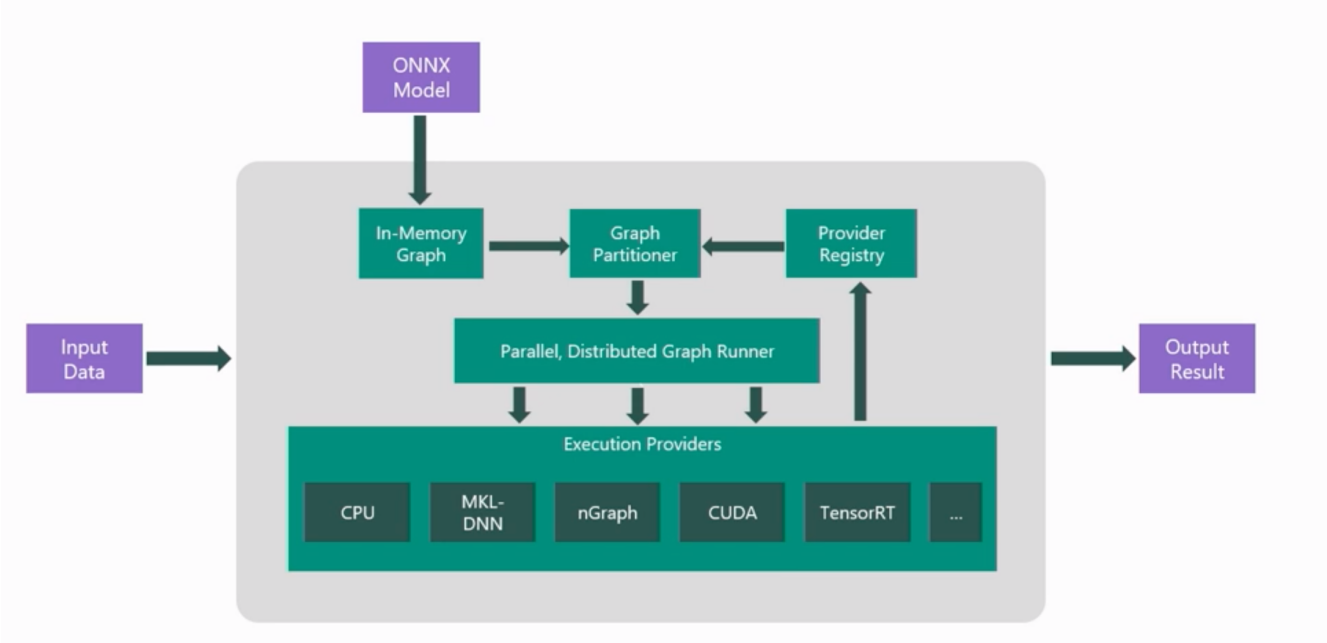
Example Use Case

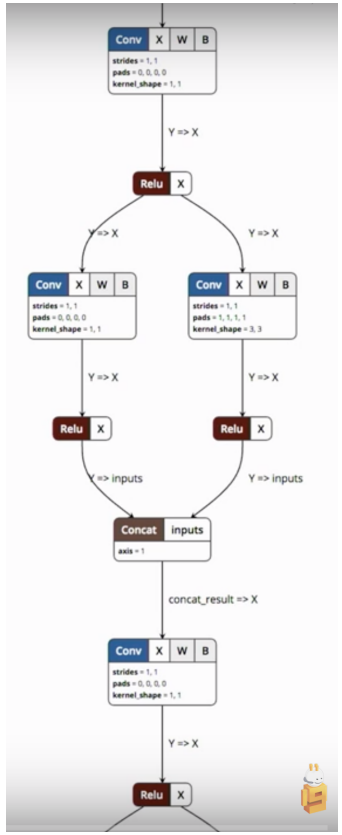
- Pytorch-ONNX-CoreML
  - https://attardi.org/pytorch-and-coreml

ONNX Runtime

- Microsoft open source high performance inference engine for ONNX models
- Available on Mac, Windows, Linux platforms ; x84 and ARM architectures
- Support full ONNX-ML spec
- Open-sourced https://github.com/microsoft/onnxruntime

ONNX Runtime Architecture





ONNX Runtime has a graph parser which takes ONNX model, parses the graph, applies runtime optimizations like fusion ops, executes portions of the graph on specific hardware, provides the inference output result

Youtube Video: Open Neural Network Exchange (ONNX) in the enterprise: how Microsoft scales ML

Factors Contributing to Democratization of AI

- Open benchmarking communities
  - Kaggle, Open ML
- ONNX
  - Faster and broader reuse of models; Shorten time to move to production
  - Hardware optimizations can be leveraged by many developers easily
  - Makes deep learning models portable thus preventing vendor lock in
- Cloud based AI solutions

Reading List

- **DAWNBench and TTA**
  - Coleman et al. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. NIPS 2017
  - Coleman et al. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. 2019

Blogs, Code Links

- Optimize TensorFlow performance using the Profiler
- Video: Intro to Kaggle (Cloud Next '18)
- Visualizing an ONNX Model https://github.com/onnx/tutorials/blob/master/tutorials/VisualizingAModel.md
- Video: Open Neural Network Exchange (ONNX) in the enterprise: how Microsoft scales ML
- Deep Learning AMI. Using Frameworks with ONNX
- Video: ONNX Runtime