Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark

Cody Coleman*, Daniel Kang*, Deepak Narayanan*, Luigi Nardi, Tian Zhao, Jian Zhang,
Peter Bailis, Kunle Olukotun, Chris Ré, Matei Zaharia
Stanford DAWN

Abstract

Researchers have proposed hardware, software, and algorithmic optimizations to improve the computational performance of deep learning. While some of these optimizations perform the same operations faster (e.g., increasing GPU clock speed), many others modify the semantics of the training procedure (e.g., reduced precision), and can impact the final model's accuracy on unseen data. Due to a lack of standard evaluation criteria that considers these trade-offs, it is difficult to directly compare these optimizations. To address this problem, we recently introduced DAWNBENCH, a benchmark competition focused on *end-to-end* training time to achieve near-state-of-the-art accuracy on an unseen dataset—a combined metric called time-to-accuracy (TTA). In this work, we analyze the entries from DAWNBENCH, which received optimized submissions from multiple industrial groups, to investigate the behavior of TTA as a metric as well as trends in the best-performing entries. We show that TTA has a low coefficient of variation and that models optimized for TTA generalize nearly as well as those trained using standard methods. Additionally, even though DAWNBENCH entries were able to train ImageNet models in under 3 minutes, we find they still underutilize hardware capabilities such as Tensor Cores. Furthermore, we find that distributed entries can spend more than half of their time on communication. We show similar findings with entries to the MLPERF v0.5 benchmark.

1 Introduction

Machine learning (ML) training has become an increasingly expensive computational workload. In particular, deep learning (DL) enables users to train high-capacity models with billions of parameters [10, 17, 39] from massive datasets that improve in accuracy as the dataset grows [8, 61]. Because modern DL methods are computationally expensive, researchers have proposed many hardware, software, and algorithmic optimizations for DL, ranging from new hardware platforms [15, 38, 53] and software systems [5, 19, 20, 25, 36] to novel distributed optimization algorithms [23, 28, 31–33, 41, 51, 60, 62, 66].

Unfortunately, performance evaluation for ML training systems is significantly more challenging than performance evaluation for traditional software. The main goal of ML training is to build a statistical model that *generalizes* well to new data, i.e., makes accurate predictions on it, but many techniques that increase throughput can adversely affect generalization. On the hardware side, large minibatch training [31, 38] and reduced precision [20, 23, 48] can help run iterations of the optimization algorithm faster and speed up "proxy" metrics such as time to process an epoch ("time-per-epoch"), but can prevent models from reaching the same accuracy on unseen data [24, 46, 47]. On the algorithmic side, techniques such as the Adam optimizer [41] were shown to accelerate the minimization of training loss ("time-to-training-loss") but sometimes lead to models with lower accuracy on unseen data [64]. These proxy metrics do not consider runtime and final model accuracy jointly, making it hard to evaluate proposed computational optimizations.

To address this lack of standard evaluation criteria, we ran the DAWNBENCH [22] competition in 2018 to measure the end-to-end performance of ML systems using a *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. Unlike prior work that focused solely on throughput metrics such as time-per-epoch [6, 11, 12, 21, 30, 58], TTA combines both generalization and speed. While several papers had previously used TTA for evaluation [7, 31, 42, 59], DAWNBENCH was the first multi-entrant benchmark competition to use the TTA metric. During the initial competition that ran in April 2018, Google, Intel, fast.ai, and others submitted optimized entries that could train to 93% top-5 accuracy on ImageNet in less than 30 minutes, which subsequently dropped to under 3 minutes with rolling submissions. Later that year, the MLPERF [3] benchmark launched using TTA as its primary metric as well.

Despite the impressive speedups achieved by DAWNBENCH and MLPERF entries, many questions remain about

*Equal Contribution

the performance of ML training systems and TTA as a metric. For example, is the 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 or are they implicitly adapting to the held-out dataset used in the benchmark through extensive hyperparameter tuning? Finally, how close are these entries from fully utilizing hardware platforms and what are the computational bottlenecks?

In this paper, we evaluate entries from DAWNBENCH and from MLPERF v0.5 to understand the behavior of TTA as an ML performance metric and identify bottlenecks in the best performing entries. Both benchmarks received professionally optimized entries from leading industry groups, such as the Google TPU team, Intel, and NVIDIA, creating one of the first opportunities to study ML systems optimized heavily for *training performance*, as opposed to traditional ML competitions that only evaluate accuracy [26]. Fortunately, most of the top entries were open source. Using these top-performing, open-source benchmark entries, we find that:

1. Despite the stochasticity of ML training procedures, TTA is a relatively stable metric that can reliably distinguish between systems on tasks that include image classification, object detection, and machine translation (§4.1).
2. Even though accuracy in TTA is measured on a fixed, held-out evaluation set, models optimized for TTA generalize to unseen data nearly as well as off-the-shelf models (§4.2).
3. Distributed training often bottlenecks on communication (often $> 50\%$ of total time spent on communication), both on publicly available cloud infrastructure and optimized on-premise deployments with fast networks (§5.1).
4. Some of the top-performing benchmark entries *severely* underutilize hardware capabilities such as Tensor Cores by up to $10\times$.
5. Training is bottlenecked by operators previously thought to be inexpensive, such as rectified linear units (ReLUs) [50] (§5.2).

2 Background: ML Training

In this section, we describe the ML training workload and how it differs in performance goals from other applications.

The Goal of ML Training: Generalization. The main goal of ML is to train a model that makes high quality predictions on unseen data, which is referred to as *generalization* [29]. An optimization algorithm minimizes a problem-specific loss function to find a model that not only performs well on the training data, but is also likely to *generalize* to unseen data from a similar distribution. This goal is different from pure mathematical optimization, as shown in Figure 1: for example, when the ML algorithm can propose a large range of functions as models, it is possible to *overfit* the training data and return a model that generalizes less well to unseen data than a simpler model. Deep learning models in particular have the capacity

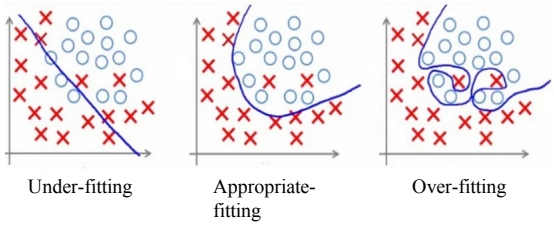


Figure 1: Examples of underfitting, appropriate fitting, and overfitting for ML models. The overfit model classifies the training data perfectly, but will perform worse on unseen data than the middle model. Figure adapted from [13].

to represent a very wide range of functions, so much of DL research focuses on finding methods that generalize well [29].

To quantify how well a model generalizes, a separate dataset is held out from training and used for periodic evaluation. This dataset is referred to the validation dataset. To avoid overfitting, most systems stop training when performance on the validation set has plateaued.[1]

However, even with a held-out validation set, repeated experiments could lead to overfitting, even though the model was never explicitly trained on the held-out data [55]. While tuning hyperparameters to optimize for TTA, entries could implicitly be learning about and adapting to the validation set rather than achieving the principal goal of generalization. Fortunately, this form of overfitting does not seem to occur in the existing DAWNBENCH and MLPERF entries (§ 4.2).

Typical Training Processes. Most deep learning models are trained using Stochastic Gradient Descent [56] or one of its accelerated variants, such as Adam [41]. These methods iterate over the training data in *minibatches*, which are small batches of records (e.g., 32 records) drawn at random. The training algorithm updates the weights of the model after processing each batch. In total, the optimization method may make multiple passes over the entire dataset during training, where each complete pass is called an *epoch*.

Tradeoffs in Speed and Generalization. Unlike more traditional workloads, many optimizations that improve how fast the ML system processes data affect the quality of the solution, either changing how many updates it takes for the model to converge or preventing the model from converging to the same quality. For example:

1. Increasing the number of records used for each update can increase hardware efficiency, but prevent or slow down convergence [47] (§ 4.3).
2. Naively reducing floating point precision to 16 bits prevents convergence, but using "loss scaling" allows for convergence [48]. Further reducing to 8 bits generally prevents convergence with current methods [24].

[1] Some texts also use the term "validation set" to refer to data held out for hyperparameter tuning, and use "test set" for evaluation data.

Hardware	# of entries
GPU	8
TPU	8
CPU	3

Framework	# of entries
TensorFlow	11
PyTorch	4
Caffe	3
MXNet	1

(a) Overview of hardware platforms and software frameworks for DAWNBENCH ImageNet submissions.

Hardware	# of entries
GPU	28
TPU	7
CPU	6

Framework	# of entries
PyTorch	22
TensorFlow	10
Caffe	3
MXNet	5
Big DL	1

(b) Overview of the hardware platforms and software frameworks for MLPERF entries. We excluded "research" submissions, which include frameworks and hardware not publicly available.

Table 1: Summary of infrastructure used for DAWNBENCH and MLPERF entries.

Model	Area	Problem	Dataset	Dataset size	Quality target
ResNet	Vision	Image classification	ImageNet	1.2M images	74.9%
SSD, ResNet-34 backbone	Vision	Object detection	MS-COCO	127K images	21.1 mAP
Mask R-CNN, ResNet-50 backbone	Vision	Object detection and instance segmentation	MS-COCO	127K images	37.7 box mAP, 33.9 mask mAP
GNMT	Language	Translation (recurrent)	WMT English-German	3.5M sent.	21.8 BLEU
Transformer	Language	Translation (non-recurrent)	WMT English-German	4.6M sent.	25.0 BLEU
NCF	Commerce	Recommendation	MovieLens-20M	20M ratings	0.635 HR@10
Minigo	RL	Go	Go	Self-play	40.00% accuracy

Table 2: Overview of tasks, models, and problem areas for the MLPERF v0.5 training benchmark.

3. In the multi-accelerator case, SGD can be performed synchronously or asynchronously [51]. Synchronicity ensures that each update uses the most up-to-date weights of the model to accurately assess performance, but requires more overhead to copy the model's weights between accelerators after each update. Asynchronous SGD can remove this synchronization at the cost of data efficiency [42, 49].

Stochasticity in Training. Training via SGD is inherently stochastic. Stochasticity enters in several ways, including randomness in model initialization and data traversal. Furthermore, many DL systems introduce stochasticity for improved hardware efficiency, e.g., by reordering floating point operations. Thus, multiple trials of the same optimization procedure can reach the same target validation accuracy in a different number of epochs.

3 Overview of Benchmarks

This section overviews the rules, training procedures, and models from DAWNBENCH and MLPERF. We also detail the entries we leverage in our subsequent analysis.

3.1 DAWNBENCH Overview

DAWNBENCH was introduced in November 2017 and concluded in April 2018. DAWNBENCH evaluates the time and cost (in USD) of popular deep learning training and inference workloads. The initial release included two tasks: image classification on ImageNet and CIFAR10, and question answering on SQuAD, and four metrics: training time to a specified validation accuracy, cost of training to that accuracy for submissions that use hardware in the public cloud, average latency of performing inference on a single item (image or question), and average inference cost.

Entries were required to submit a description of their submission and the validation accuracy after every epoch. While source code was optional, every submission for image classification included a link to all code needed to reproduce runs, assuming access to the appropriate hardware. For question answering on SQuAD, some submissions did not include code until well after the DAWNBENCH deadline; because of this and the general lack of submissions, we focus exclusively on image classification *training* submissions in this paper. While our analysis applies to both ImageNet and CIFAR10, we do not include results for CIFAR10 in this paper since CIFAR10 does not reflect the scale of production workloads. As a result our analysis of DAWNBENCH focuses solely on ImageNet, where DAWNBENCH used a top-5 accuracy target of 93%.

3.2 MLPERF Overview

MLPERF v0.5 is a more recent benchmark that concluded in December 2018. MLPERF evaluates TTA on a broader range of tasks, including image classification, object detection, translation, and recommendation, as shown in Table 2. Unlike DAWNBENCH, MLPERF used a fixed model and optimization algorithm. There was some flexibility for choosing SGD hyperparameters to allow submissions of different computational scales. Submissions were also allowed to submit results for a subset of tasks, so the majority of hardware targets did not include entries for every task. For example, the reinforcement learning task had no entries with accelerators, as game simulation was the bottleneck. As such, we do not analyze the reinforcement learning entries. Similarly, we do

not analyze the results on the recommendation task because it does not reflect production usage and will be replaced [14].

3.3 Summary of Entries

Entries to DAWNBENCH and MLPERF v0.5 came from many organizations, including Google, NVIDIA, and Intel, which had teams of engineers optimize their submissions. The entries spanned GPUs, TPUs, and CPUs on the hardware side and TensorFlow [5], PyTorch [52], Caffe [36], MXNet [18], and Big DL [34] on the software side. The number of compute units (which we refer to as compute scale) ranged from 2 to 640 processors, and speedups over reference implementations ranged from 1.6× to over 1,400×. In MLPERF v0.5, every entry with an accelerator used mixed precision training [48], and large batch sizes [31]. DAWNBENCH submissions were allowed to use a wider range of optimizations, including progressive resizing of images [40, 43] and novel model architectures [54], in addition to mixed-precision training and large minibatch training. In our analysis, we used all pre-February 2019 submissions that were reproducible with public cloud infrastructure or included sufficient information for analysis (e.g., training logs).

4 Analysis of Time-to-Accuracy

In this section, we evaluate the TTA metric along three axes, using publicly available code and results from DAWNBENCH and MLPERF submissions. First, we demonstrate that TTA has a low coefficient of variation ($< 14\%$) over several runs with fixed hyperparameters, even with some statistical optimizations (e.g., cyclic learning rates, progressive resizing) that result in higher variance. Second, we provide evidence that models optimized for TTA generalize nearly as well as regular, unoptimized models. Third, we compare TTA against other metrics and show that the alternative metrics do not capture the complexity of DL training.

4.1 Variability of Time-to-Accuracy

To understand the stability of TTA, we computed the coefficient of variation (the ratio of the variance to the mean) for the top DAWNBENCH entries available on public cloud (by rerunning them several times) and *official* MLPERF entries (which contained multiple trials). We chose this metric as the mean is a natural scale for comparing systems. For example, a coefficient of variation of 14% means that systems that achieve a TTA within 14% of each other are not easily distinguished, but a system that is two times faster than another is easy to distinguish.

As shown in Table 3a, the coefficient of variation of TTA for the reproduced DAWNBENCH entries is at most 4.5% for entries that do not use novel statistical optimizations, but 12.2% for all entries. This indicates that TTA is largely stable despite the randomness in DL training.

We also found that several entries failed to consistently achieve the given accuracy threshold. In particular, progressive resizing used by several of the DAWNBENCH ImageNet entries appear to make validation convergence less robust as seen in Table 3a.

The coefficient of variation was similarly low for the official MLPERF results. Table 3b shows the coefficient of variation for the official MLPERF results. We find that TTA is largely stable; the coefficient of variation is always less than 14% and generally less than 7%. We additionally reproduced the majority of available MLPERF entries on stable public cloud hardware. We found that these reproduced MLPERF entries were in line with the official entries.

Source of Variation. To understand the source of variation in TTA, we analyzed the validation accuracy curves per epoch for MLPERF entries. Figure 2 shows the variance in quality metric per epoch across several tasks and machine scales. Validation accuracy is less stable at the beginning of training but becomes more stable as training continues. This variance early in training grows with the system scale because larger entries start training with large learning rates. Additionally, the variation in the number of epochs is high due to the different machine scales. For selected large scale entries, Table 4 shows low variation in time-per-epoch, with a coefficient of variation less than 3%. The variation in the number of epochs to reach the target quality metric is up to 45× higher than the variation in time-per- epoch. Thus, most of the variation in TTA comes from variation in the number of epochs.

4.2 Generalization of Optimized Models

To measure the generalization performance of models optimized for TTA in image classification and translation, we collect unseen data, i.e., data that is not in the validation and training sets, and test the accuracy on this unseen data. We used reproduced DAWNBENCH and MLPERF entries since neither benchmark provided checkpoints.

Evaluation on New Data for Image Classification. To test image classification, we scraped and labeled a set of 2,864 images from Flickr. The images were scraped based on the WordNet keywords associated with each class in the ImageNet dataset. The top five images based on relevance were shown to a human labeler and labeled correct or incorrect. To ensure no overlap with ImageNet, only images posted after January 1st, 2014 were used. The images spanned 886 (out of 1000) classes. While these images are not entirely representative of ImageNet, we believe they reflect a reasonable distribution.

We computed the relevant accuracy metric (top-1 or top-5 accuracy) for DAWNBENCH entries, an optimized MLPERF entry, and pre-trained ResNet-50 weights provided by PyTorch on the images from Flickr. The results are summarized in Table 5. As shown, the models optimized for TTA achieve nearly the same accuracy or higher than the pre-trained ResNet-50, indicating that optimizing for TTA does not sacrifice generalization performance.

Evaluation on Unseen Data for Translation Tasks. For the MLPERF GNMT and Transformer models, we additionally

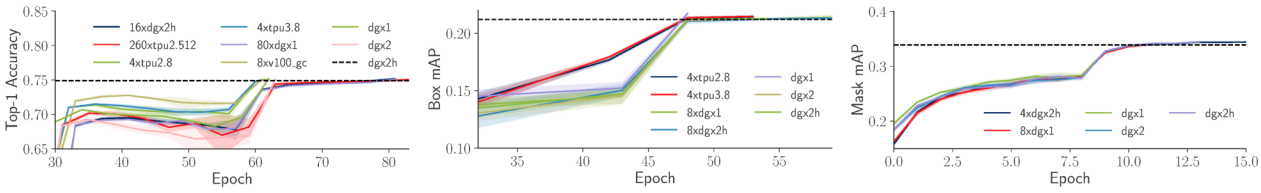
Entry name	Coeff. of variation	Frac. of runs
ResNet-50, `p3.16xlarge`	5.3%	80%
ResNet-50, 4x`p3.16xlarge`	11.2%	60%
ResNet-50, 8x`p3.16xlarge`	9.2%	100%
ResNet-50, 16x`p3.16xlarge`	12.2%	100%
ResNet-50, 1xTPU	4.5%	100%
AmoebaNet-D, 1xTPU	2.3%	100%
ResNet-50, 1/2 TPU Pod	2.5%	100%

(a) Coefficient of variation and fraction of runs that reached the desired target accuracy of the top DAWNBENCH entries for image classification on ImageNet (5 runs). `p3.16xlarge` entries were from fast.ai and used progressive resizing. We also include the coefficient over 4 runs of 1/2 a TPU Pod for ResNet-50.

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%

(b) Coefficient of variation for selected official MLPERF entries. All of the displayed runs achieved the target accuracy 100% of the time. We exclude recommendation as the model and dataset are being replaced for the next version of MLPERF.

Table 3: Coefficient of variation and fraction of runs that achieved the target accuracy for MLPERF and DAWNBENCH entries. As shown, the coefficient of variation is less than 14% for all runs, with the exception of multi-accelerator Transformer entries (not shown). However, MLPERF plans to expand the dataset size for Transformer, which we believe will improve stability.



(a) ResNet **(b)** SSD **(c)** Mask

(d) GNMT **(e)** Transformer

Figure 2: Variation in validation quality metric per epoch for official MLPERF entries. As shown, the variation decreases closer to the final target score. The variation within an epoch is typically smaller than the variation between epochs close to the target score. Best seen in color.

Entry	Epochs	Time per epoch (s)	Ratio of std. devs
ResNet, 80xDGX-1	82 ± 0	4.6 ± 0.01	$0\times$
SSD, 8xDGX-2	53.6 ± 4.2	6.3 ± 0.03	$14.8\times$
Mask, 4xDGX-2H	14.4 ± 0.8	310.0 ± 2.9	$5.9\times$
GNMT, 16xDGX-2H	4.2 ± 0.4	39.2 ± 1.0	$3.9\times$
Transf., 24xDGX-1	7.2 ± 2.1	56.2 ± 0.4	$44.9\times$

Table 4: The deviation in number of epochs, time per epoch, and the ratio of the standard deviations for selected official MLPERF entries. We selected the largest scale entries. The variation largely comes from the variation in the number of epochs, not the time per epoch.

used new data to test generalization performance. We used the WMT'17 English-German `newstest2017` test set [1], which is not in the training or validation sets for GNMT or Transformer. Table 6 shows the optimized implementations generalize as well as the reference implementations, despite being over 50× faster. This indicates that optimizing for TTA does not sacrifice generalization performance.

4.3 Comparison to Alternative Metrics

Comparison to Throughput. To demonstrate that throughput (and equivalently time-per-epoch and achieved FLOPS for a fixed model and dataset) is not sufficient for measuring DL system performance, we show the batch size, number of epochs, throughput speedup, and TTA speedup in Table 7. As shown, TTA speedups and throughput speeds can differ by up to 3×, as increasing the batch size to improve throughput can increase the number of epochs required for convergence. Thus, throughput is insufficient to characterize DL system performance even though it has been used extensively in prior benchmarks [12, 21, 30, 58].

Comparison to Peak Device FLOPS. As shown in Table 7, system scale does not correlate with throughput or TTA speedup. We further show in § 5.2 that existing accelerators are severely underutilized in many cases. Additionally, other

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, 4x`p3.16xlarge`	91.3%
ResNet-50, 8x`p3.16xlarge`	91.5%
ResNet-50, 16x`p3.16xlarge`	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.

Table 5: Performance of pre-trained models and models optimized for TTA on unseen data for DAWNBENCH and MLPERF ImageNet entries. The models optimized for TTA perform nearly as well as or better than the PyTorch pre-trained model. We expect the pre-trained ResNet-18 and ResNet-152 to be lower and upper bounds respectively on generalization performance.

Model	BLEU score
GNMT, reference	23.44 ± 0.08
GNMT, DGX-1	23.63 ± 0.20
Transformer, reference	26.60 ± 0.44
Transformer, DGX-1	26.78 ± 0.45

Table 6: BLEU scores on unseen data for the reference and optimized GNMT and Transformer models. As shown, models optimized for TTA generalize as well as the reference models. We show the average of three runs and the standard deviation.

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).

costs (e.g., communication overhead) can dominate runtimes and can add a $> 2\times$ overhead. Thus, peak device FLOPS is a poor proxy for observed DL system performance.

5 Hardware Utilization and Scaling

In this section, we evaluate how well highly optimized DAWNBENCH and MLPERF entries utilize available hardware. First, we demonstrate that distributed entries can spend more than half of their time on communication overhead. Second, we study the utilization of these entries on a single worker. Through a roofline analysis [63], we provide evidence that despite near state-of-the-art training performance across a range of tasks, many submissions still *severely* underutilize the available hardware resources. We also show that memory-bound kernels take a significant percentage of total runtime, leading to lower observed FLOPS.

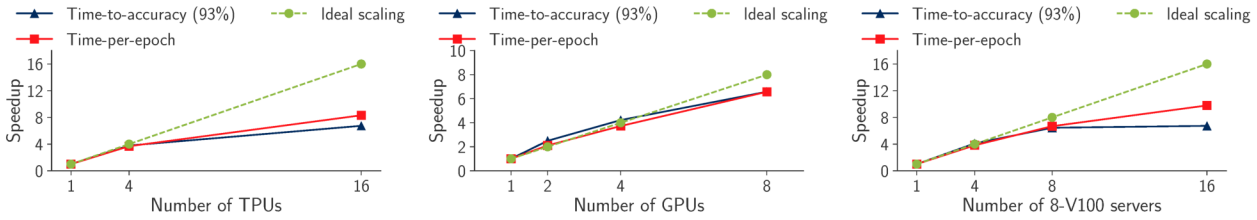
5.1 Scaling of Distributed Training

With an increase in model size and complexity, distributed training has become imperative to train models in reasonable timeframes. However, distributed training requires expensive cross-accelerator communication [37]. To better quantify these communication overheads, we trained the same models with different accelerator counts, and studied the scaling behavior of end-to-end training.

Scaling of Time-to-Accuracy. To scale up to hundreds of accelerators, every large-scale DAWNBENCH and MLPERF entry used large minibatches to saturate the available hardware. This includes the machine translation and object detection tasks, even though the original large minibatch training technique was only tested on the ResNet-50 image classification model [31]. Table 7 shows batch sizes and throughputs of various MLPERF official entries. As shown, the batch size can be scaled from 4 to nearly 50× the base batch size.

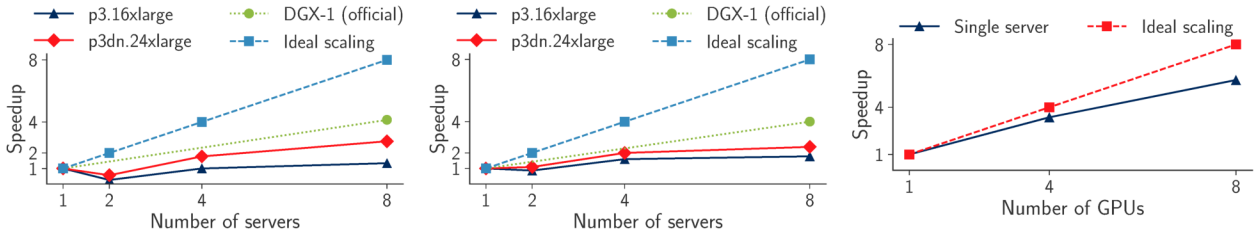
We find that both time-per-epoch and TTA scale almost linearly with the number of workers *within* a server, across a range of models for image classification, object detection, and language translation in both the DAWNBENCH and MLPERF benchmarks (Figures 3b and 4c).

However, we found that both time-per-epoch and TTA do not scale as well for training that spans multiple servers. In Figure 3a, we show the speedup relative to one worker of per-epoch time for an AmoebaNet model trained in a TPU Pod with 64 TPUs on the ImageNet dataset. Figure 3c shows the speedups when scaling ResNet-50 training up to 16 `p3.16xlarge` instances (each server has 8 NVIDIA V100 GPUs) on Amazon Web Services (AWS). Time-per-epoch shows as much as a 38.9% gap from linear scaling. Time-to-accuracy scales even worse, since a greater number of epochs are needed to converge to the same accuracy target for the larger minibatch size. We see similar results for the SSD and Mask R-CNN models using both `p3.16xlarge` instances on AWS, and DGX-1 servers (NVIDIA's optimized server with 8 V100 GPUs) in a private cloud deployment with Infiniband network communication in Figures 4a and 4b .



(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).



(a) SSD across 8-V100 servers. **(b)** Mask R-CNN across 8-V100 servers. **(c)** Mask R-CNN within 8-V100 server.

Figure 4: Speedup of TTA with respect to a single worker vs. number of workers for an SSD model on multiple 8-V100 servers (`p3.16xlarge` and `p3dn.24xlarge` instances on AWS, and a NVIDIA DGX-1 server in an on-premise deployment), a Mask R-CNN model on multiple 8-V100 servers, and a Mask R-CNN model within a `p3.16xlarge` instance.

Model	Machine Config.	Comm. Overhead (%)
ResNet-50	80xDGX-1	64.947%
ResNet-50	16xDGX-2H	25.859%
SSD	8xDGX-1	42.043%
SSD	8xDGX-2H	68.231%
Mask R-CNN	8xDGX-1	47.674%
Mask R-CNN	4xDGX-2H	42.131%
GNMT	32xDGX-1	71.146%
GNMT	16xDGX-2H	67.436%
Transformer	24xDGX-1	35.127%

Table 8: Percentage of time in an epoch spent communicating for official optimized distributed MLPERF entries.

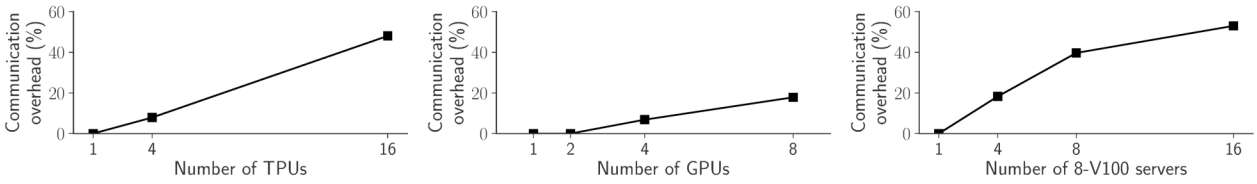
Model	Machine Config.	Comm. Overhead (%)
ResNet-50	4xV100 (AWS)	4.528%
ResNet-50	8xV100 (AWS)	13.400%
SSD	4xV100 (AWS)	5.364%
SSD	8xV100 (AWS)	14.999%
Mask R-CNN	4xV100 (AWS)	17.167%
Mask R-CNN	8xV100 (AWS)	26.163%
GNMT	4xV100 (AWS)	9.921%
GNMT	8xV100 (AWS)	15.832%
Transformer	4xV100 (AWS)	26.692%
Transformer	8xV100 (AWS)	15.546%

Table 9: Percentage of time in an epoch spent communicating for reproduced single-server MLPERF entries.

Communication Overhead. To further understand the impact of networking on distributed training, we computed the communication overheads of both official DAWNBENCH and MLPERF entries. These results are shown in Figures 5 and 6, and Tables 8 and 9. For the MLPERF entries, we show communication overheads for both the official entries run on private on-premise deployments, and reproduced entries run on public cloud deployments to quantify the impact of optimized network interconnects like Infiniband and RDMA on end-to-end training time. For the DAWNBENCH entries, we show communication overhead numbers on the public cloud.

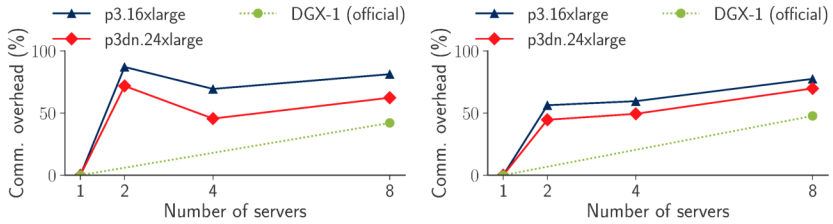
As shown, communication remains a *significant* overhead. Even on on-premise deployments with 100Gb/s InfiniBand EDR interconnect can have communication overheads as high as 71.15%, for the GNMT model using 32 DGX-1 servers. This overhead can rise to 77.46% when using Amazon's `p3.16xlarge` instances with a 25 Gigabits/second interconnect for Mask R-CNN.

Figure 5 shows the communication overhead as the number of workers increase for DAWNBENCH entries; we see that within a DGX-1, communication overhead is much lower (17.82%) compared to across servers (53%), since NVIDIA's `nvlink` interconnect has far higher bandwidth than the 25Gbps provided by Amazon EC2 across `p3.16xlarge` instances. As DAWNBENCH did not have entries for Amoe-



(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%).



(a) SSD across 8-V100 servers. **(b)** Mask R-CNN across 8-V100 servers.

Figure 6: Percentage of time per epoch spent communicating vs. number of workers for various MLPERF entries in both on-premise and public cloud deployments. Communication overheads are as high as 47.67% in on-premise deployments and up to 77.46% in the public cloud.

baNet on GPUs, we were unable to make a completely fair apples-to-apples comparison of the scaling properties between AmoebaNet and ResNet-50.

Discussion. These results suggest that despite the work in scaling ML training to many multi-GPU servers [31,59], communication remains a bottleneck, for large machine counts and for certain models in public cloud deployments. The work by Goyal et al. [31] shows far better scaling than we have observed in the DAWNBENCH entries; we believe this is due to the fact that the results presented in this paper used faster V100 GPUs (compared to P100 GPUs), and had slower network interfaces (up to 25 Gigabits/second on AWS compared to 50 Gigabits/second in a private Facebook cluster).

To address this, highly optimized communication libraries like Horovod [57] have been developed. Other work [44] has explored techniques to reduce the amount of data sent over the network. However, these techniques need to be evaluated on more models and in more hardware settings for widespread adoption. Integration into widely-used deep learning frameworks like PyTorch and TensorFlow would also help with usability. Additionally, exploring parallelization schemes other than data parallelism that do not require all-to-all communication among all workers could be helpful.

5.2 Single-worker Utilization

To study the utilization of the compute units of the accelerators themselves, we analyzed the performance of some DAWNBENCH and MLPERF submissions on a single accelerator, without network overhead.

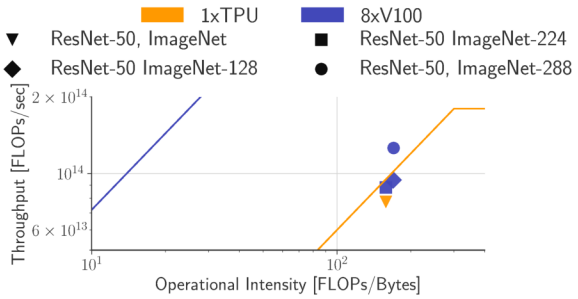


Figure 7: Roofline models for the various DAWNBENCH entries. All of the entries under-utilize the hardware resources, by up to 10×.

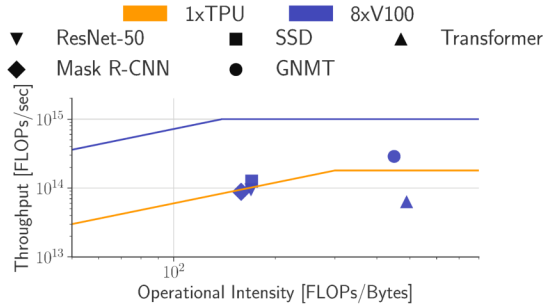


Figure 8: Roofline models for the various MLPERF entries. All of the entries under-utilize the hardware resources, by up to 10×.

Roofline Analysis. To understand the hardware performance of single-worker training, we used the roofline model [63],

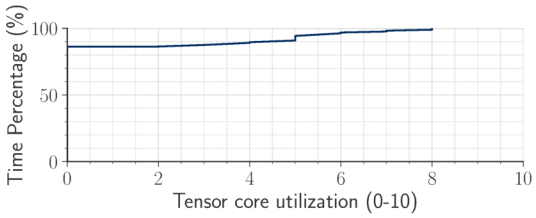


Figure 9: CDF of tensor core utilization for the `fast.ai` ResNet50 model trained with `fp16` precision submitted to the DAWNBENCH competition. About 85% of time is spent on kernels that don't utilize the NVIDIA Tensor Cores *at all*, and no kernel achieves full utilization of the Tensor Core units.

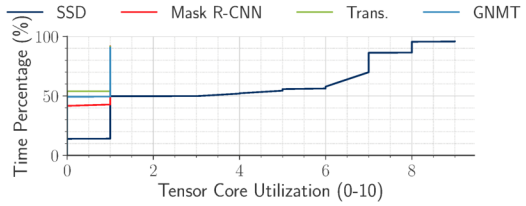


Figure 10: CDF of tensor core utilization for different MLPERF models trained with `fp16` precision.

which can highlight causes of performance bottlenecks. The roofline model plots computational throughput (in floating point operations per second) against the operational intensity of the application (number of floating-point operations performed per DRAM byte accessed). Applications with high operational intensity are "compute-bound" (the flat line in Figure 7) and bottlenecked on the device's computation units, while applications with low operational intensity are "memory-bound" (the slanting line in Figure 7) and bottlenecked on memory accesses.

We show results in Figures 7 and 8. Each point in these figures represents a DAWNBENCH or MLPERF entry. For entries which used progressive image resizing [40, 43], where different image sizes are used through training, we show each image size used. Operational intensities and throughputs are approximated by instrumenting training code and using profiling tools like `nvprof`.

As shown, all entries analyzed *severely* underutilize the available compute resources – each plotted point achieves a throughput significantly lower than peak device throughput.

Bottlenecks in Training. To investigate the source of underutilization on the V100 GPU, we measured the `fp32` throughput and Tensor Core utilization of each GPU kernel in PyTorch's implementation of ResNet-50. The V100s have peak throughputs of 15.7 Teraflops of `fp32` arithmetic and 125 Teraflops of half-precision arithmetic via Tensor Cores [45].

Figures 9 and 10 show that the GPU kernels taking the majority of time when using `fp16` precision utilize the Tensor Cores poorly, with a time-averaged Tensor Core utilization of

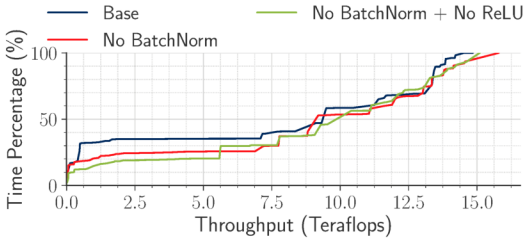


Figure 11: CDF of per-kernel throughput for ResNet50 models trained with `fp32` precision. The CDF is computed by percentage of time spent executing each GPU kernel. A standard ResNet-50 model spends about 40% time in low-throughput kernels (< 6 Teraflops). Removing the BatchNorm layer from the ResNet50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

0.71 (on a scale of 1-10 as reported by `nvprof`).

Training the same model even with standard `fp32` precision only achieves a throughput of 7.6 Teraflops, compared to peak device throughput of 15.7 Teraflops. This is largely due to memory-bound kernels like BatchNorm [35] and ReLU [50], that take a significant percentage of total runtime. This is illustrated in Figure 11, which show that a non-trivial portion of kernels underutilize the GPU, and that removing BatchNorm and ReLU layers improves `fp32` throughput by about 20%.

Discussion. Compilers for deep learning like TVM [4] and XLA [2] try to automatically generate code given a higher-level description of the deep learning computation being performed. 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. For example, consider code that performs the forward pass of a BatchNorm followed by the forward pass of a ReLU. Naively, this code would be executed by code that resembles the following,

```
1  // BatchNorm.
2  for (int i = 0; i < n; i++) {
3      y[i] = gamma * ((x[i] - mu) / sigma) + beta;
4  }
5  // ReLU.
6  for (int i = 0; i < n; i++) {
7      z[i] = max(y[i], 0);
8  }
```

In the above listing, a DRAM write is performed for `y[i]` and `z[i]`, and a DRAM read is performed to compute `z[i]`.

However, for training, we could optimize the above code by fusing the two loops, saving on DRAM reads of `y[i]` since the intermediate result is written to a local variable instead.

In addition, we believe that co-design of model architectures with modern hardware could be useful as well. For example, as we have shown, the BatchNorm and ReLU operations are memory bound. It may be possible to develop alternatives to these operations that are less memory-bound, but provide similar statistical effects, resulting in faster training.

6 Related Work

Benchmarking DL Training. Many prior ML benchmarks use throughput (either per-kernel or per-iteration) as a metric [6, 11, 12, 21, 30, 58]. While throughput can inform the development of ML algorithms and systems, we show throughput alone cannot fully characterize ML systems.

Several ML benchmarks have done static workload characterizations on systems that do not contain state-of-the-art hardware with FP16 support [6, 67]. Furthermore, several benchmarks, including Fathom, do not benchmark distributed DL training [6,12,21]. TBD [67] benchmarks distributed training on older accelerators that do not contain FP16 support, which significantly changes the proportion of total runtime spent on computation and communication. In contrast to prior work, we analyze code that has been optimized by teams of engineers on state-of-the-art hardware. We additionally analyze distributed DL systems that uses this hardware. We show that Tensor Cores can be severely underutilized and that communication overheads are as high as 71%, even in optimized on-premise deployments.

Benchmarking High Performance Computing Systems. Researchers have developed many methods for benchmarking computer systems and HPC systems [9, 16, 27]. The majority of these systems measure deterministic workloads (e.g., DRAM, key-value stores), but measuring DL systems requires a more nuanced analysis to reason about both runtime and the generalizability of the final model. While these systems could be used to improve individual components of DL training systems (e.g., faster convolution algorithms), they are not sufficient to measure end-to-end DL training.

High Performance DL. Researchers have developed many optimizations for high performance DL training [7,31,42,65]. Unfortunately, many such optimizations are closed-source. To the best of our knowledge, DAWNBENCH and MLPERF are the first open-entrant benchmarks with open-source entries for optimizing TTA on a range of tasks. We take advantage of the open-source code to study TTA and analyze these workloads.

Some work on high performance DL [7, 31, 65] used TTA. However, these systems largely used TTA as a metric to optimize, but do not study the metric in detail. In this work, we analyze TTA as a metric and show that it is largely stable and models optimized for TTA generalize well.

7 Conclusion

In this paper, we perform the first in-depth analysis of DAWNBENCH entries to investigate the behavior of TTA as a metric and trends in the best-performing entries. We corroborate our results by analyzing entries from MLPERF v0.5, which also adopted TTA. Both benchmarks received professionally optimized entries from leading industry groups, creating one of the first opportunities to study ML systems optimized heavily for training performance. We find that TTA is usually stable to the randomness in ML training with a low coefficient of variation ($< 14\%$) across image classification, machine translation, and object detection. We also find that models optimized for TTA generalize nearly as well as unoptimized models. Finally, we find that entries highly optimized for TTA still underutilize available hardware, leaving significant room for further improvement.

Acknowledgments

We thank Jeremy Howard, the Google Cloud TPU team (including Sourabh Bajaj, Frank Chen, Brennan Saeta, and Chris Ying), and the many other teams that submitted to DAWNBENCH. We thank Juan Manuel Camacho, Shoumik Palkar, Kexin Rong, Keshav Santhanam, Sahaana Suri, Pratiksha Thaker, and James Thomas for their assistance in labeling. We also thank Amazon and Google for cloud credits. This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, Infosys, Intel, Microsoft, NEC, SAP, Teradata, and VMware—as well as Toyota Research Institute, Keysight Technologies, Amazon Web Services, Cisco, and the NSF under grants DGE-1656518, DGE-1147470, and CNS-1651570. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Second conference on machine translation, 2017.

[2] Tensorflow xla overview. https://www.tensorflow.org/performance/xla, 2017.

[3] MLPerf. https://mlperf.org/, 2018.

[4] TVM: An automated end-to-end optimizing compiler for deep learning. In *OSDI*, Carlsbad, CA, 2018. USENIX Association.

[5] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*, volume 16, pages 265–283, 2016.

[6] Robert Adolf, Saketh Rama, Brandon Reagen, Gu-Yeon Wei, and David Brooks. Fathom: Reference Workloads for Modern Deep Learning Methods. In *IISWC*, pages 1–10. IEEE, 2016.

[7] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

[8] Dario Amodei and Danny Hernandez. Ai and compute, 2018.

[9] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload analysis of a large-scale key-value store. In *SIGMETRICS*, volume 40, pages 53–64. ACM, 2012.

[10] Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853*, 2018.

[11] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. Comparative Study of Deep Learning Software Frameworks. *arXiv preprint arXiv:1511.06435*, 2015.

[12] Baidu. DeepBench: Benchmarking Deep Learning Operations on Different Hardware. https://github.com/baidu-research/DeepBench, 2017.

[13] Anup Bhande. What is underfitting and overfitting in machine learning and how to deal with it, 2018.

[14] Victor Bittorf. Making ncf reflect production usage, 2019.

[15] Doug Burger. Microsoft unveils Project Brainwave for Real-time AI. *Microsoft Research, Microsoft*, 22, 2017.

[16] Kevin K Chang, A Giray Yağlıkçı, Saugata Ghose, Aditya Agrawal, Niladrish Chatterjee, Abhijith Kashyap, Donghyuk Lee, Mike O'Connor, Hasan Hassan, and Onur Mutlu. Understanding reduced-voltage operation in modern dram devices: Experimental characterization, analysis, and mechanisms. *SIGMETRICS*, 1(1):10, 2017.

[17] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

[18] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[19] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient Primitives for Deep Learning. *arXiv preprint arXiv:1410.0759*, 2014.

[20] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project Adam: Building an Efficient and Scalable Deep Learning Training System. In *OSDI*, volume 14, pages 571–582, 2014.

[21] Soumith Chintala. Convnet-Benchmarks: Easy Benchmarking of All Publicly Accessible Implementations of Convnets. https://github.com/soumith/convnet-benchmarks, September 2017.

[22] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. *NIPS ML Systems Workshop*, 2017.

[23] Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. Understanding and Optimizing Asynchronous Low-precision Stochastic Gradient Descent. In *ISCA*. ACM, 2017.

[24] Christopher De Sa, Megan Leszczynski, Jian Zhang, Alana Marzoev, Christopher R Aberger, Kunle Olukotun, and Christopher Ré. High-accuracy low-precision training. *arXiv preprint arXiv:1803.03383*, 2018.

[25] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large Scale Distributed Deep Networks. In *NIPS*, 2012.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[27] Saugata Ghose, Abdullah Giray Yaglikçi, Raghav Gupta, Donghyuk Lee, Kais Kudrolli, William X Liu, Hasan Hassan, Kevin K Chang, Niladrish Chatterjee, Aditya Agrawal, et al. What your dram power models are not telling you: Lessons from a detailed experimental study. *SIGMETRICS*, 2(3):38, 2018.

[28] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In *AISTATS*, pages 315–323, 2011.

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[30] Google. TensorFlow Benchmarks. https://www.tensorflow.org/performance/benchmarks, 2017.

[31] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[32] Aaron Harlap, Henggang Cui, Wei Dai, Jinliang Wei, Gregory Ganger, Phillip Gibbons, Garth Gibson, and Eric Xing. Addressing the Straggler Problem for Iterative Convergent Parallel ML. In *SoCC*. ACM, 2016.

[33] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and $<$ 0.5 MB Model Size. *arXiv preprint arXiv:1602.07360*, 2016.

[34] Intel. Bigdl: Distributed deep learning library for apache spark, 2019.

[35] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[36] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. In *ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[37] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. In *SysML*, 2019.

[38] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter Performance Analysis of a Tensor Processing Unit. In *ISCA*, pages 1–12. ACM, 2017.

[39] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[40] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *arXiv preprint arXiv:1710.10196*, 2017.

[41] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *ICLR*, 2015.

[42] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 14, pages 583–598, 2014.

[43] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced Deep Residual Networks for Single Image Super-Resolution. In *CVPR Workshops*, volume 1, page 3, 2017.

[44] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *ICLR*, 2018.

[45] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S Vetter. Nvidia tensor core programmability, performance & precision. *arXiv preprint arXiv:1803.04014*, 2018.

[46] Dominic Masters and Carlo Luschi. Revisiting Small Batch Training for Deep Neural Networks. *arXiv preprint arXiv:1804.07612*, 2018.

[47] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

[48] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaev, Ganesh Venkatesh, et al. Mixed Precision Training. *arXiv preprint arXiv:1710.03740*, 2017.

[49] Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. Asynchrony begets momentum, with an application to deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 997–1004. IEEE, 2016.

[50] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[51] Feng Niu, Benjamin Recht, Christopher Re, and Stephen Wright. Hogwild: A Lock-free Approach to Parallelizing Stochastic Gradient Descent. In *NIPS*, pages 693–701, 2011.

[52] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[53] Dexmont Pena, Andrew Forembski, Xiaofan Xu, and David Moloney. Benchmarking of CNNs for Low-Cost, Low-Power Robotics Applications. 2017.

[54] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized Evolution for Image Classifier Architecture Search. *arXiv preprint arXiv:1802.01548*, 2018.

[55] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do CIFAR-10 classifiers generalize to cifar-10? *CoRR*, abs/1806.00451, 2018.

[56] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[57] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[58] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking State-of-the-Art Deep Learning Software Tools. In *Cloud Computing and Big Data (CCBD)*. IEEE, 2016.

[59] Samuel L Smith, Pieter-Jan Kindermans, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[60] Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast Large-scale Optimization by Unifying Stochastic Gradient and Quasi-Newton Methods. In *ICML*, pages 604–612, 2014.

[61] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. *CoRR*, abs/1707.02968, 2017.

[62] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the Importance of Initialization and Momentum in Deep Learning. In *ICML*, pages 1139–1147, 2013.

[63] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[64] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *NIPS*, pages 4148–4158, 2017.

[65] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. In *ICPP*, page 1. ACM, 2018.

[66] Ce Zhang and Christopher Ré. Dimmwitted: A Study of Main-memory Statistical Analytics. *PVLDB*, 7(12):1283–1294, 2014.

[67] Hongyu Zhu, Mohamed Akrout, Bojian Zheng, Andrew Pelegris, Amar Phanishayee, Bianca Schroeder, and Gennady Pekhimenko. Tbd: Benchmarking and analyzing deep neural network training. *arXiv preprint arXiv:1803.06905*, 2018.