Abstract

Despite considerable research on systems, algorithms and hardware to speed up deep learning workloads, there is no standard means of evaluating end-to-end deep learning performance. Existing benchmarks measure proxy metrics, such as time to process one minibatch of data, that do not indicate whether the system as a whole will produce a high-quality result. In this work, we introduce DAWNBench, a benchmark and competition focused on end-to-end training time to achieve a state-of-the-art accuracy level, as well as inference time with that accuracy. Using time to accuracy as a target metric, we explore how different optimizations, including choice of optimizer, stochastic depth, and multi-GPU training, affect end-to-end training performance. Our results demonstrate that optimizations can interact in non-trivial ways when used in conjunction, producing lower speed-ups and less accurate models. We believe DAWNBench will provide a useful, reproducible means of evaluating the many trade-offs in deep learning systems.

1 Introduction

Deep learning methods are effective but computationally expensive, leading to a great deal of work to optimize their computational performance. Researchers have proposed new software systems [29, 18, 12, 9, 46], training algorithms [33, 48, 45, 28, 44, 18, 27, 50, 42, 43, 14], communication methods [32, 49, 11, 9, 12, 22] and hardware [20, 30, 37, 19, 7, 21] to decrease this cost. However, despite these significant advances, it is difficult to measure or compare the utility of these results due to a lack of standard evaluation criteria. Most existing benchmarks for deep learning performance [16, 10, 5, 2, 41, 8, 3] only measure proxy metrics such as the time to process one minibatch of data. In reality, deep learning performance is far more complex. In some settings, techniques such as using larger batch sizes [18, 30], reduced precision [21, 9, 25, 11] and asynchronous updates [39, 12, 9, 49] can stop an algorithm from converging to a good result, or increase the time to do so. Moreover, these approaches interact in nontrivial ways and may require updating the underlying optimization algorithms [33, 18, 56], further affecting end-to-end performance.

This lack of standard evaluation criteria results in a set of poorly-understood trade-offs. For example, minimal effort back propagation delivers a 3.1x speed up over back propagation on MNIST [43]. Using 8-bit precision gives a 3x speed up on MNIST [11]. Does combining minimal effort back propagation with 8-bit precision give a 9.3x speed up? Would that speed translate to a larger model on a dataset like ImageNet, and combine with accurate, large minibatch SGD [18] to train an ImageNet model in 7 minutes? Currently, these questions can only be answered via tedious and time-consuming experimentation. Moreover, which previous techniques should one build on when evaluating the efficiency of a new optimization?

To provide an objective means of quantifying end-to-end deep learning performance, we introduce DAWNBench, an open benchmark and competition for end-to-end deep learning training and inference. Instead of simply measuring time per iteration or throughput, DAWNBench measures end-to-end
Table 1: Dimensions evaluated in the first version of DAWNBench. All metrics are for a near-state-of-the-art accuracy.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Metrics</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image classification</td>
<td>Training time</td>
<td>ImageNet</td>
</tr>
<tr>
<td></td>
<td>Training cost</td>
<td>CIFAR10</td>
</tr>
<tr>
<td>Question answering</td>
<td>Inference latency</td>
<td>SQuAD</td>
</tr>
<tr>
<td></td>
<td>Inference cost</td>
<td></td>
</tr>
</tbody>
</table>

performance of training (e.g., time, cost) and inference (e.g., latency, cost) at a specified accuracy level. This provides an objective means of normalizing across differences in computation frameworks, hardware, optimization algorithms, hyperparameter settings, and other factors that affect real-world performance. Our initial release of DAWNBench provides end-to-end learning and inference tasks including image classification on CIFAR10 [34] and ImageNet [40], and question answering on SQuAD [38], and reference implementations for each task.

In this paper, we present DAWNBench’s benchmark specification and goals, and show how one can use DAWNBench to perform a simple performance study that measures the impact of choice of optimizer, batch size, multi-GPU training, and stochastic depth [26]. Over time, and with community input, we plan to expand DAWNBench’s scope to include both additional benchmark tasks (e.g., segmentation, machine translation, video classification) and metrics (e.g., energy, sample complexity).

2 Benchmark Overview and Goals

DAWNBench evaluates deep learning systems on different tasks based on several metrics, using multiple datasets. The benchmark allows innovation in software, algorithms, communication methods, etc. By only specifying the task, DAWNBench also allows experimentation of new model architectures and hardware. In the initial release, we seed entries for two tasks: image classification on CIFAR10 and ImageNet, and question answering on SQuAD, and evaluate on four metrics: training time to a specified validation accuracy, cost (in USD) of training to a specified validation accuracy using public cloud instances, average latency of performing inference on a single item (image or question), and average cost of inference for 10,000 items (Table 1). We also provide reference implementations and seed entries, implemented in two popular deep learning frameworks, PyTorch and TensorFlow.

These reference implementations were collected and adapted from official repositories on Github, and produce accuracy numbers on par with those reported in the original research papers [24], while also conforming to the various performance recommendations published with these frameworks [17].

DAWNBench is not the first benchmark to compare and evaluate deep learning systems on standard tasks; previous efforts include Baidu DeepBench [5], Fathom [2], and TensorFlow Benchmark [16]. Our benchmark differs in two key aspects. First, DAWNBench focuses on end-to-end performance. Many prior 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 [2, 16, 10, 41, 4]. Other benchmarks [2, 5] focus on timing individual low-level operations (e.g., convolutions, matrix multiplications) utilized in deep learning computations. DAWNBench, instead, measures time to a pre-specified (and high) level of accuracy, taking into account both hardware and statistical performance.

Second, we view DAWNBench as open and evolving. Prior benchmarks were often defined at a snapshot in time, using fixed model architectures and tasks. However, advances in deep learning have quickly invalidated architectures. For example, recent approaches in machine translation using recurrent layers have been quickly superseded by convolutional and attention layers [13, 47]. Deep learning is a rapidly evolving field, and needs benchmarks that adapt at the same pace. We plan to allow DAWNBench to grow and evolve, and hope to foster a community discussion around advances in the field, including new tasks.

3 Early Results

To demonstrate the utility of an end-to-end benchmark, we present DAWNBench results that aim to answer three questions: (1) Is training time to a specified validation accuracy a useful metric to evaluate deep learning systems? (2) Do different optimizations in deep learning actually compose? (3) Using DAWNBench’s seed entries, what trade-offs between training time, training cost, and inference latency do different software frameworks and hardware present?
Figure 1: Effect of minibatch size on convergence rate, throughput, and end-to-end training time of a ResNet56 CIFAR10 model on a P100. Learning rates are tuned as per [18]. We follow the same basic training procedure as the original ResNet paper [24] for all experiments in this section. For a batch size of 128, we use SGD with a weight decay of 0.0005 and momentum of 0.9. We use the weight initialization procedure from [23] and use Batch Normalization [28] without dropout. We warm-up training with a learning rate of 0.01 for five epochs, then proceed with an initial learning rate of 0.1 for 90 epochs; we decay the learning rate by a factor of 10 every 45 epochs thereafter, and terminate training after 185 epochs. We follow the same data augmentation process as [24]. For larger batch sizes, we linearly scale the initial learning rate according to [18], using a base learning rate of 0.1 (corresponding to a batch size of 128).

Unless stated otherwise, the K80 machines used in this section are Google Cloud Engine instances based on the n1-standard-8 instance type (8 vCPUs, 30 GB memory), with Haswell CPUs and Google’s standard HDD for persistent storage. The P100 machines used have 512 GB of memory and 28 CPU cores, and are in a private cluster. We used TensorFlow 1.2 and PyTorch 0.1.12, compiled from source with CUDA 8.0 and CuDNN 5.1. DAWNBench does not currently include distributed training results, but we plan to explore this as future work, and also welcome outside submissions. All experiments in this section compute validation accuracy on the CIFAR10 test set (10,000 images) [34].

3.1 Evaluating Time to Accuracy for Various Minibatch Sizes

To illustrate the value of DAWNBench’s end-to-end performance metric, we study how minibatch size impacts both the convergence rate and hardware performance (FLOPS) of a deep learning workload. Prior work [32, 18, 15, 6, 35] has shown that picking a minibatch size too small or too large can lead to poor convergence, i.e. minibatch size affects convergence. Additionally, larger minibatch sizes better saturate hardware execution units [6, 15]. In choosing the minibatch size that minimizes total time to a target accuracy, we must balance these two factors.

As we show in Figure 1 for a ResNet56 model trained on the CIFAR10 dataset on a Nvidia P100 GPU, a 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). A minibatch size of 256 represents a reasonable trade-off between convergence rate and throughput. A 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. Benchmarks that focus exclusively on convergence rate and throughput are unable to surface these practical trade-offs for optimizations even as simple as modifying the minibatch size.

3.2 Composing Optimizations

To show how DAWNBench can be used for performance studies in practice, we run a small study that evaluates the effect of composing several different optimizations for training Convolutional Neural Networks (CNNs) on the CIFAR10 dataset. Evaluating deep learning systems is challenging, in part because it’s unclear how different optimizations interact with each other. Given the complexity of deep learning systems and plethora of hyperparameters involved, seemingly complementary optimizations can interact in non-trivial ways and stop an algorithm from converging to a good result, or increase the time to do so.
Here, we demonstrate that different optimizations in fact do not compose while training a ResNet110 model on CIFAR10. Our goal here is not to provide blanket statements about any particular training method. Instead, we showcase the trade-offs and interactions between methods and their non-trivial impact on end-to-end accuracy and performance. We consider the following three optimizations,

- **Adam.** An adaptive optimizer for gradient descent, which reports considerable speed-ups over other adaptive optimizers when training CNNs on CIFAR10 [33].

- **Single-node multi-GPU training,** using 4 GPUs in two different settings: 1) With the same minibatch size (128) as that used in the baseline approach, but distributed across the 4 GPUs 2) With effectively a minibatch size multiplied by 4, where every GPU is given a minibatch size of 128. For the larger minibatch size, we tune the learning rate as per [18].

- **Stochastic Depth [26],** a technique that can be thought of as a form of regularization similar to dropout. Entire layers are randomly dropped during training, helping to regularize against co-adaptation, while still using the full network to perform inference. Stochastic Depth reports improvements in both training time and accuracy.

**Adam.** After evaluating each optimization in isolation, we discovered Adam with multiple initial learning rates (0.0001, 0.001, 0.01, 0.1) was unable to come close to the maximum validation accuracy we were able to achieve with our baseline. Figure 2 shows the best performance using Adam (learning rate = 0.01) compared to SGD with momentum using the standard learning rate schedule from [24]. Despite the better training loss and faster initial convergence on both the training and validation dataset, Adam’s promising performance plateaued well below SGD. Because Adam failed to reach a reasonable validation accuracy in isolation, we eliminated it from the remainder of our factor analysis. This shows that training loss is only a proxy for accuracy and generalization, and that “time to an accuracy threshold” is a valuable metric when evaluating deep learning systems.

**Single-node Multi-GPU Training.** We observe that the other two optimizations each provide a reasonable speed-up, but the net speed-up from applying all the optimizations at once is less than the product of the individual speed-ups as shown Figure 3. All speed-ups and accuracy thresholds are relative to the performance of ResNet110 on 1 Nvidia K80 GPU with a minibatch size of 128 and initial learning rate of 0.1. The top accuracy ResNet110 achieved was 94.33%. Using this as a baseline, we chose two threshold points, 93% and 94% top-1 accuracy, representing different points in the trade-off space between

---

**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%.

**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.
Figure 4: Comparison of maximum top-1 validation accuracy and training times of different combination of optimizations. While reducing training time, each optimization decreases the maximum accuracy.

Figure 5: Inference time vs. training time to 93% validation accuracy, for different hardware, frameworks, and model architectures in DAWN-Bench’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).

training time and accuracy. 94% represents human-level accuracy on the CIFAR10 dataset [31]. 93%, while considerably lower, helps demonstrate both the magnitude of speed-up possible when accuracy constraints are relaxed, and the difficulty of achieving human-level or super-human-levels of accuracy.

Scaling the number of GPUs while keeping the minibatch size the same produces a 1.68x and 1.59x speed-up to reach 93% and 94% accuracy respectively, compared to the baseline. A larger batch size of 512 reached an accuracy of 93% much faster (speed-up of 4.95x over baseline), but failed to reach the higher threshold. Interestingly, the speed-up in terms of time to 93% is larger than the increase in throughput alone (3.57x). In addition to scaling the learning rate according to [18], we also tried a batch size of 512 with multiple initial learning rates (0.1, 0.2, 0.3) with the same learning rate decay schedule. Surprisingly, the learning rate of 0.1 outperformed the linearly scaled learning rate of 0.4 on CIFAR10 in terms of maximum accuracy. We thus used a learning rate of 0.1 for all experiments with a batch size of 512 in Figure 3.

Stochastic Depth. Adding stochastic depth to ResNet110 provided a speed-up for both thresholds. Stochastic depth was much faster at reaching the lower threshold (1.21x speed-up) than the higher threshold (1.02x speed-up). While the total training time speed-up was close to the 1.25x speed-up reported in [26], the number of epochs to reach the higher threshold was larger for stochastic depth than the baseline, which led to the decreased speed-up in terms of time to accuracy.

Multi-GPU Training and Stochastic Depth Combined. By combining stochastic depth and multi-GPU training, we see that the maximum accuracy reached by applying all the optimizations is about a percent lower than the maximum accuracy reached by the baseline (Figure 4). We note that there are many possible ways to tune learning rates, batch size and optimization algorithms that may achieve better performance here. For this experiment, we chose to follow the standard recommendations in the literature [24, 18, 26] to illustrate what a practitioner following these approaches may achieve, and we performed parameter sweeps where possible to explore alternative settings. Without standard ways to run and tune different optimizations jointly, composing optimizations may affect the convergence rate of the algorithm, making performance hard to reason about.

3.3 Comparing Hardware and Software Frameworks

We seeded DAWN-Bench with single-GPU and CPU results for TensorFlow and PyTorch, using reference implementations of models when possible. We show some of the variability present across DAWN-Bench’s metrics even from simple factors such as the model, software framework, and hardware type, in Figure 5. This figure presents training time to 93% validation accuracy, and single-image inference la-
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.

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.

tency for various ResNet architectures for the CIFAR10 dataset, on different hardware platforms (1 K80 GPU on two cloud providers [Google Compute Engine and Amazon EC2], 1 P100 GPU on a private cluster, and a 16vCPU machine on Google Cloud with Broadwell microarchitecture) and frameworks.

As the figure illustrates, TensorFlow is faster than PyTorch on CPUs, but slightly slower on GPUs, both for training and inference. This is partly due to data format: TensorFlow supports both NCHW and NHWC layouts (N: Number of Samples, C: Number of Channels, H: Height, W: Width), which give better performance on GPUs and CPUs respectively, while PyTorch only supports NCHW. K80 performance is similar on both cloud providers, but with spot pricing for GPU instances, Amazon is cheaper. Training and inference time are proportional to the depth of the model, as expected.

We also see that for different target validation accuracies, different ResNet architectures are “optimal” (that is, fastest to reach target validation accuracy). Figure 6 shows how the top-1 validation accuracy varies with training time for different ResNets for CIFAR10 on a Nvidia K80 GPU. For lower accuracy thresholds, shallower architectures reach the threshold faster.

We also demonstrate how training cost relates to training time for ResNet56 on CIFAR10 as the number of GPUs used changes (Figure 7). Prior to running this experiment, we expected training cost to increase as the number of GPUs increases (and training time decreases), leading to a Pareto frontier between cost and time. In practice, we did not observe this to be the case for CIFAR10 experiments run on 1 to 4 GPUs – training time scales perfectly linearly with the inverse of the number of GPUs used, and hence cost remains constant despite training time going down. However, when scaling this computation to 8 GPUs, we did see an increase in the cost of training, because training time does not decrease enough to counter the doubling in instance cost per unit time.

4 Conclusion

DAWNBench is a benchmark that measures the end-to-end time to train a model with state-of-the-art accuracy, and inference time with that accuracy. By focusing on time to accuracy, we show how different optimizations can interact; using different optimizations in conjunction can prevent models from converging, or increase the time to do so. Reasoning about deep learning systems in this way exposes valuable trade-offs between training time, training cost, and inference time. We intend to keep DAWNBench up to date with new tasks and goals to foster a quantitatively-informed dialogue around progress in deep learning systems.
Acknowledgments

We thank Toby Boyd, Doug Burger, Soumith Chintala, Ramesh Illikal, Mario Srouji, Jian Zhang, Bill Zhao, Xiaoxiang Zheng, the anonymous reviewers, and the many members of the Stanford InfoLab for their valuable feedback on this work. 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 – Intel, Microsoft, Teradata, and VMware – as well as industrial gifts and support from Toyota Research Institute, Juniper Networks, Keysight Technologies, Hitachi, Facebook, Northrop Grumman, NetApp, and the NSF under grants DGE-1656518, DGE-114747, and CNS-1651570.

References


