DAWNBench: An End-to-End Deep Learning Benchmark and Competition

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

Stanford DAWN Project
http://dawn.cs.stanford.edu/benchmark

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 with that accuracy. We have seeded the benchmark with entries for image classification on CIFAR10 and ImageNet, and question answering on SQuAD, showing differences across models, software and hardware. We believe DAWNBench will provide a useful, reproducible means of evaluating the many tradeoffs 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 [1, 7, 8, 11, 23, 39], training algorithms [12, 15, 21, 22, 26, 35–38, 40, 42], communication methods [8, 10, 11, 19, 32, 41] and hardware [6, 16–18, 24, 30] to decrease this cost. Despite significant advances, it is hard to measure or compare the utility of these results due to a lack of standard evaluation criteria. Most existing benchmarks for deep learning performance [2–4, 7, 9, 14, 34] only measure proxy metrics such as the time to process one minibatch of data. In reality, deep learning performance is far more complex. Approaches such as using larger batch sizes [15, 24], reduced precision [8, 10, 18, 20] and asynchronous updates [8, 11, 32, 41] can stop an algorithm from converging to a good result, or increase the time to do so. These approaches also interact in nontrivial ways and may require updating the underlying optimization algorithm [15, 26, 29], further affecting performance.

This lack of standard evaluation criteria leaves deep learning practitioners having to navigate these trade-offs. For example, minimal effort back propagation (meProp) delivers a 3.1x speed up over back propagation on MNIST [36]. Using 8-bit precision gives a 3x speed up on MNIST [10]. Does combining meProp 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 [15] to train an ImageNet model in 7 minutes? Currently, these questions can only be answered via tedious and time-consuming experimentation. Researchers face a similar challenge: when they have a new idea for an optimization, which previous techniques should they consider combining in evaluating their results?

Tasks	Metrics
Image classification	Training time Training cost
Question answering	Inference latency Inference cost

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

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* performance in training (e.g., time, cost) and inference (e.g., latency, cost) at a specified *state-of-the-art* level of accuracy. 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 [27] and ImageNet [33], and question answering on SQuAD [31], and reference implementations for each task. Over time, with community input, we plan to expand the set of benchmark tasks (e.g., segmentation, machine translation, video classification) and metrics (e.g., power, sample complexity).

2 Benchmark Structure

DAWNBench evaluates deep learning systems on different *tasks* based on several *metrics*. The benchmark allows innovation in software, algorithms, communication methods, etc. By only specifying the task, DAWNBench also allows

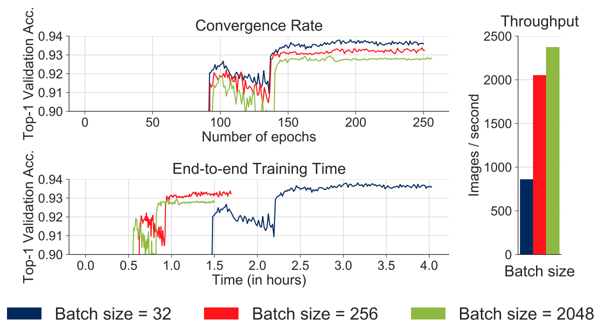

Figure 1: Effect of minibatch size on convergence rate, throughput, and end-to-end training time of a ResNet56 CIFAR10 model on a P100. Learning rates are tuned as per [15].

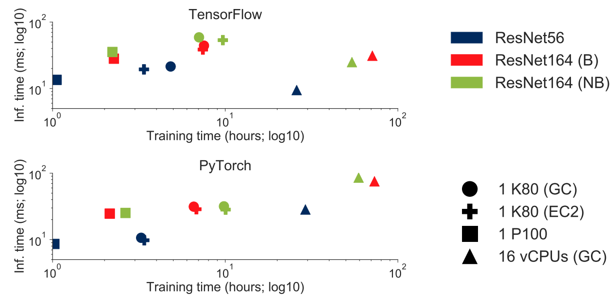

Figure 2: Inference time vs. training time to 93% val. acc., for different hardware, frameworks, and model architectures in DAWNBench's seed entries. ResNet164 (B) uses a bottleneck building block, while (NB) uses a simple building block.

experimentation of new model architectures and hardware. In the v0 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 of training to a specified validation accuracy, latency of performing inference on a single item (image or question), and cost of inference for a single item (Table 1). We intend to update the benchmark with more tasks and new accuracy targets over time.

3 Example Results

In this section, we offer preliminary results that seek to answer two questions: (1) Is training time to a specified validation accuracy a useful metric to evaluate deep learning systems? (2) What type of insights can DAWNBench surface?

Evaluating Impact of Minibatch Size. To illustrate the value of DAWNBench's end-to-end performance metric, we use it to study how minibatch size impacts *both* the convergence rate and hardware performance (FLOPS) of a deep learning workload, making it hard to reason about end-to-end performance from either metric alone. Prior work [5, 13, 15, 25, 28] 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 [5, 13]. 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 factors even as simple as minibatch size.

Comparison of DAWNBench Seed Entries. We seeded DAWNBench 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 DAWNBench's metrics even from simple factors such as the model, software framework and hardware type in Figure 2. This figure presents training time to 93% validation accuracy, and single-image inference latency for various ResNet architectures for the CIFAR10 dataset, on different hardware platforms (1 K80 GPU on two cloud providers [Google and Amazon], 1 P100 GPU on a private cluster, and a 16vCPU machine on Google Cloud) 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.

4 Conclusion

DAWNBench proposes a simple, living benchmark for the performance metrics practitioners care about most: *end-to-end* time to train a model with *state-of-the-art* accuracy, and inference time with that accuracy. We hope that this collection of tasks, seed entries and our ongoing competition will provide a simple way to test and validate a wide variety of new ideas, spanning systems, algorithms, and hardware, to optimize deep learning. We intend to keep DAWNBench up to date with new tasks and goals to help the community track progress in deep learning systems.

Acknowledgments

We thank the many members of the Stanford InfoLab for their valuable feedback on this work. 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

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

[2] Robert Adolf, Saketh Rama, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2016. Fathom: Reference Workloads for Modern Deep Learning Methods. In *Workload Characterization (IISWC), 2016 IEEE International Symposium on*. IEEE, 1–10.

[3] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. 2015. Comparative study of deep learning software frameworks. *arXiv preprint arXiv:1511.06435* (2015).

[4] Baidu. 2017. DeepBench: Benchmarking Deep Learning operations on different hardware. (Aug. 2017). https://github.com/baidu-research/DeepBench

[5] Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*. Springer, 437–478.

[6] Microsoft Research Blog. 2017. Microsoft unveils Project Brainwave for real-time AI. https://www.microsoft.com/en-us/research/blog/microsoft-unveils-project-brainwave/. (2017). Accessed: 2017-09-04.

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

[8] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an Efficient and Scalable Deep Learning Training System. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*. USENIX Association, Berkeley, CA, USA, 571–582. http://dl.acm.org/citation.cfm?id=2685048.2685094

[9] Soumith Chintala. 2017. Convnet-benchmarks: Easy Benchmarking of All Publicly Accessible Implementations of Convnets. (Sept. 2017). https://github.com/soumith/convnet-benchmarks original-date: 2014-07-12T03:18:46Z.

[10] Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. 2017. Understanding and Optimizing Asynchronous Low-Precision Stochastic Gradient Descent. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 561–574.

[11] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large Scale Distributed Deep Networks. In *Advances in neural information processing systems*. 1223–1231.

[12] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 315–323.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[14] Google. 2017. TensorFlow Benchmarks. (2017). https://www.tensorflow.org/performance/benchmarks

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

[16] Graphcore. 2017. Accelerating Next Generation Machine Intelligence. https://www.graphcore.ai/technology. (2017). Accessed: 2017-09-04.

[17] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. *SIGARCH Comput. Archit. News* 44, 3 (June 2016), 243–254. https://doi.org/10.1145/3007787.3001163

[18] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[19] Aaron Harlap, Henggang Cui, Wei Dai, Jinliang Wei, Gregory R Ganger, Phillip B Gibbons, Garth A Gibson, and Eric P Xing. 2016. Addressing the straggler problem for iterative convergent parallel ML. In *SoCC*. 98–111.

[20] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. (2017). arXiv:1704.04861 http://arxiv.org/abs/1704.04861

[21] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. (2016). https://arxiv.org/abs/1602.07360

[22] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*. 448–456.

[23] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia (MM '14)*. ACM, New York, NY, USA, 675–678. https://doi.org/10.1145/2647868.2654889

[24] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/3079856.3080246

[25] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On large-batch training for deep learning: Generalization gap and sharp minima. *ICLR* (2017).

[26] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[27] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).

[28] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. 2014. Efficient mini-batch training for stochastic optimization. In *SIGKDD*. ACM, 661–670.

[29] Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. 2016. Asynchrony begets Momentum, with an Application to Deep Learning. In *Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on*. IEEE, 997–1004.

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

[31] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. *arXiv preprint arXiv:1606.05250* (2016).

[32] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*. 693–701.

[33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[34] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. 2016. Benchmarking state-of-the-art deep learning software tools. In *Proceedings of the International Conference on Cloud Computing and Big Data*. IEEE.

[35] Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. 2014. Fast large-scale optimization by unifying stochastic gradient and quasi-Newton methods. In *International Conference on Machine Learning*. 604–612.

[36] Xu Sun, Xuancheng Ren, Shuming Ma, and Houfeng Wang. 2017. meProp: Sparsified Back Propagation for Accelerated Deep Learning with Reduced Overfitting. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 3299–3308. http://proceedings.mlr.press/v70/sun17c.html

[37] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*. 1139–1147.

[38] T Tieleman and G Hinton. 2014. RMSprop Gradient Optimization. *http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf* (2014).

[39] Leonard Truong, Rajkishore Barik, Ehsan Totoni, Hai Liu, Chick Markley, Armando Fox, and Tatiana Shpeisman. 2016. Latte: A Language, Compiler, and Runtime for Elegant and Efficient Deep Neural Networks. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '16)*. ACM, New York, NY, USA, 209–223. https://doi.org/10.1145/2908080.2908105

[40] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer. 2017. SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2017-07). 446–454. https://doi.org/10.1109/CVPRW.2017.60

[41] Ce Zhang and Christopher Ré. 2014. Dimmwitted: A study of main-memory statistical analytics. *Proceedings of the VLDB Endowment* 7, 12 (2014), 1283–1294.

[42] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2017. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. (2017). arXiv:1707.01083 http://arxiv.org/abs/1707.01083