DNN-Train: Benchmarking and Analyzing DNN Training

Hongyu Zhu, Bojian Zheng, Bianca Schroeder, Gennady Pekhimenko
University of Toronto
{serailhydra,bojian,bianca,pekhimenko}@cs.toronto.edu

Amar Phanishayee
Microsoft Research
amar@microsoft.com

ABSTRACT

We aim to build a new benchmark pool for deep neural network training and to analyze how efficient existing frameworks are in performing this training. We will provide our methodology and develop proper profiling tools to perform this analysis.

1 INTRODUCTION AND MOTIVATION

The recent success of deep learning algorithms has attracted a lot of attention from the system and the architecture communities. For example, the opening session topics of ISCA 2016 and ISCA 2017 are both about machine/deep learning; ISCA 2016 has three sessions and nine accepted papers related to deep neural networks (DNN) optimizations; ISCA 2017 and MICRO 2016 also have two sessions and six accepted papers in this area. While this attention leads to interesting recent work on how to perform DNN computation more efficiently, the primary focus is usually on (i) *inference* – i.e. how to efficiently execute already trained models, and (ii) *image classification* applications as the primary benchmark to evaluate DNN computation efficiency.

While inference is arguably an important problem, we observe that as machine learning is applied to an ever growing number of domains, e.g., speech recognition, machine translation, automobile industry, advertising, efficiently training new models is becoming equally important. Moreover, these new applications employ types of layers that differ from those used for image classification (e.g., machine translation models usually use recurrent neural network (RNNs) layers rather than convolutional layers). These new layer types have very different compute and memory bandwidth characteristics, and frameworks that optimize for convolutions might not perform as well on RNNs.

Our **primary goal** is to bridge this gap by (i) selecting a representative set of DNN models that cover different machine learning applications: image classification, machine translation, speech recognition, adversarial networks, reinforcement learning, and by (ii) performing an extensive performance analysis on training these different applications on many different deep learning frameworks (such as TensorFlow, MXNet, CNTK) and on different hardware configurations (single-GPU, multi-GPU, and multi-machine).

How is Training Different from Inference? The algorithmic differences between training and inference lead to differences in requirements for the underlying system and architecture. For example, due to the presence of the *backward pass* and *weight updates*, the training procedure needs to save/stash a huge amount of intermediate results, e.g., outputs of the inner layers (often called feature maps) in GPU memory. This puts a significant pressure on the memory subsystem of modern DNN accelerators (usually GPUs) – in some cases the model might need tens of gigabytes of main memory [17]. Another difference is that training usually needs to process large amount of data to avoid overfitting, making throughput a major performance metric of concern. On the other hand, inference is relatively light-weight with respect to computation, but much more latency sensitive. The memory footprint of inference is usually significantly smaller (in the order of tens of megabytes), and the major memory consumer are the model weights rather than activations.

The Need for Diversity. Although deep learning has achieved state-of-the-art results in various application domains, image classification so far is still the dominant benchmark used in most evaluations, and convolutional neural networks (CNNs) remain the most widely-used models for system/hardware researchers. As a result, many important non-CNN models are ignored. Among the system/architecture papers that employed machine learning benchmarks from other application domains, only a handful evaluated other types of neural networks such as recurrent neural networks [1, 11, 13]. Papers that cover unsupervised learning or deep reinforcement learning are extremely rare. Since the computation characteristics of image classification models are very different from these networks, a wider benchmark suite for DNN training is needed.

Finding the Right Metrics and Bottlenecks. Machine learning (ML) researchers usually perform the training passes multiple times to identify the best hyper-parameter configurations, the best network topology, and for fine-tuning the subset of model weights. The desired flexibility is usually provided by the high-level frameworks such as TensorFlow [1] or MXNet [5]. These DNN frameworks make DNN programming much more convenient. Unfortunately, their complexity makes the performance analysis and tuning of the resulting applications much more challenging, as the performance bottlenecks could come from different sources. In our work we aim to develop the proper methodology to detect those bottlenecks, characterize them, and either automatically optimize the code or suggest potential improvements to ML developers.

Typical CNNs are usually very computationally intensive, therefore *computation* is normally one of the primary bottlenecks. Modern GPUs provide significant compute power to alleviate this bottleneck, but they have to be programmed properly (usually using one of the CUDA libraries) to deliver high performance. As we will show in Section 3, even state-of-the-art frameworks do not always efficiently utilize modern GPUs for DNN training. For example, the performance of training machine translation models with RNNs is significantly limited by the maximum mini-batch size allowed (number of inputs processed in parallel in one iteration), which is bounded by the physical memory capacity of a single GPU (usually 8–16GBs).

Training DNNs in a distributed environment with multiple GPUs and multiple machines, requires fast communication between many CPUs and GPUs. The communication overhead (*network and interconnect bandwidth*) significantly affects the training performance and limits DNN training scalability as we will further show in Section 3.

Identifying whether the training performance is bounded by computation, memory or communication is not easy. Existing profiling tools (e.g., vTune, nvprof, etc.) have no domain-specific knowledge about the algorithm logic, and can only capture low-level information within their own scopes. Therefore to do a holistic analysis, the profiling results generated by these tools have to be correctly collected, merged, analyzed, and interpreted using domain-specific knowledge of DNN training. In the case of memory footprint analysis, no open-source tools are currently available and we built them ourselves for three major frameworks: TensorFlow, MXNet and CNTK. We also develop a tool chain for an end-to-end analysis of DNN training. We believe that our benchmarks, new methodology for their analysis, related tools, and our observations/insights

using those tools will be interesting for system and architecture researchers, and ML practitioners.

2 METHODOLOGY

Applications. To create a representative benchmark pool with sufficient diversity, we surveyed several application areas where deep learning has been regarded promising. We discussed our choices with machine learning researchers and industry developers to ensure that we do not miss major deep learning domains. The chosen areas are: image classification, object detection, machine translation, speech recognition, generative adversarial nets, and deep reinforcement learning. Table 1 shows our collection. The models we choose are able to produce state-of-the-art results in each of their corresponding domains, ensuring that they reflect the most popular DNNs used for training.

Application	Model(s)	Dataset
image classification	ResNet[9], GoogleNet[19]	imagenet1k[18]
object detection	Faster-RCNN[16]	Pascal VOC[8]
machine translation	Seq2Seq[22], Transformer[21]	iwslt15
speech recognition	Deep Speech 2[2]	LibriSpeech[15]
unsupervised learning	WGAN[3]	Cifar10
reinforcement learning	A3C[14]	Atari 2600

Table 1: Applications from *DNN-Train* benchmark pool.

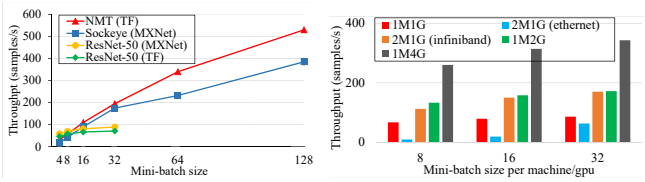
Frameworks. Choosing frameworks and comparing them can be tricky. There are many open-source DNN frameworks (e.g. Tensorflow[1], Theano[4], MXNet[5], CNTK[23], Caffe[12], Chainer[20], Torch[7], Keras[6], etc). Each of them applies some special optimization techniques of their own. Fortunately, some high-level system designs are generally applied to most frameworks. Almost all frameworks allow users to use GPUs for acceleration. The CPUs are then mostly responsible for scheduling, data copying, kernel launching, etc. High-level APIs are provided in either a declarative or imperative way (or both). For one model trained using different frameworks, the invoked GPU kernels are functionally the same although the framework code bases are usually quite different. All these common features provide us a basis to compare different frameworks. In this work we choose TensorFlow[1], MXNet[5], CNTK[23] as the frameworks for all major evaluations, since all three platforms have a large number of users and are actively evolving.

Performance Analysis Pipeline and Tools. To analyze our benchmarks we create a pipeline with several major steps. (1) Match implementation from different networks that includes matching their topologies, key algorithm properties/parameters, and all key hyperparameters. (2) Measure the GPU occupation rate as it is usually the primary compute engine used in training. (3) Analyze CPU runtime: CUDA vs. non-CUDA (e.g., synchronization, memory transfers etc.) (4) Analyze GPU runtime: usually ALU utilization is the major metric to look at. (5) (If needed) analyze memory footprint – how much memory is used by weights, activations, workspace etc.

Hardware/Software Configurations. We run the single-GPU experiments with NVidia GeForce GTX 1080Ti and NVidia TITAN Xp GPU. To carry out multi-GPU and multi-machine experiments, we use a cluster of 4 machines, each of them equipped with one or more NVidia Quadro P4000 GPUs. All experiments are executed on Ubuntu 16.04 OS with Tensorflow v1.3, MXNet v0.11.0, CNTK v2.0, CUDA v8 and cuDNN v6.

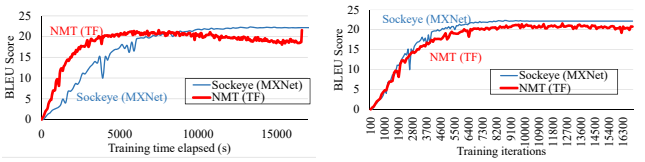
3 KEY RESULTS AND OBSERVATIONS

Observation 1: the importance of application diversity. We observe that different applications from our pool have very different performance characteristics. Machine translation models that use RNN layers do not map efficiently on modern GPUs. ResNet-50 model used for image classification has GPU occupation over 90% for all three frameworks; in contrast LSTM-based models have less than 40%



(a) Performance for different DNN models. **(b) ResNet-50 on MXNet with multiple GPUs(G)/machines(M).**

Figure 1: DNN training for different mini-batch sizes.



(a) Quality over time. **(b) Quality per epoch.**

Figure 2: Machine translation training.

occupation rate. This happens because current GPU implementations of LSTM cells used in RNN layers generate many small kernels. Increasing the mini-batch size (as we show in Figure 1a) helps reduce this overhead, but only until we saturate memory capacity (mini-batch size of 256 does not fit into 12GB of GPU memory).

Observation 2: the importance of framework diversity. Based on results in Figure 1a, we observe that MXNet performs better on image classification (ResNet-50 model) than TensorFlow, but TensorFlow is much better on machine translation tasks. Figure 2a shows how this performance difference affects the training quality (BLEU Score) over time for a maximum possible mini-batch size of 128. Semantically both models (NMT[22] and Sockeye[10]) are very close using our methodology (same topology, hyper-parameters, and algorithms). Figure 2b shows that if we look at the training process per epoch (training iterations) then the training quality of both models is very similar.

Observation 3: LSTM saturation. Another observation from Figure 1a is that LSTM-based models do not saturate the compute resources of the GPU even at the maximum possible batch size. The reasons include both very low GPU occupation rate (how much GPU cores are busy) – less than 40% and also extremely low GPU utilization (the efficiency of using GPU compute resources) – 20%–30%.

Observation 4: Memory capacity matters. Figure 1a shows that larger batches could improve training performance even further and the bottleneck could be memory capacity. The memory profiling tools we built show us what data structures are responsible for high memory consumption and it can vary cross different models. This will provide ML practitioners with some intuition on how adjustments made to model parameters would affect the memory usage of their models.

Observation 5: Network bandwidth matters. Inter-machine communication is a bottleneck for scale-out training. Figure 1b shows how GPU training scales with the number of GPUs and machines.[1] We observe that going from one machine (1M1G) to two machine (2M1G (ethernet)) configuration the performance degrades significantly. This is because DNN training requires constant synchronization between GPUs in distributed training. Hence faster networking is required to improve the situation (2M1G configuration has 100Gb/s InfiniBand Mellanox networking). DNN training on a single machine (1M1G, 1M2G, 1M4G) scales reasonably well.

[1] *M stands for the number of machines, and *G for the number of GPUs.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[4] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

[5] 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.

[6] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[7] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, Idiap, 2002.

[8] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. Sockeye: A toolkit for neural machine translation. *arXiv preprint arXiv:1712.05690*, 2017.

[11] Yu Ji, YouHui Zhang, ShuangChen Li, Ping Chi, CiHang Jiang, Peng Qu, Yuan Xie, and WenGuang Chen. Neutrams: Neural network transformation and co-design under neuromorphic hardware constraints. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–13. IEEE, 2016.

[12] 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 *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[13] 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. *arXiv preprint arXiv:1704.04760*, 2017.

[14] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[15] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5206–5210. IEEE, 2015.

[16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[17] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W Keckler. vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–13. IEEE, 2016.

[18] 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. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[19] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[20] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, volume 5, 2015.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[22] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[23] Dong Yu, Adam Eversole, Mike Seltzer, Kaisheng Yao, Zhiheng Huang, Brian Guenter, Oleksii Kuchaiev, Yu Zhang, Frank Seide, Huaming Wang, et al. An introduction to computational networks and the computational network toolkit. *Microsoft Technical Report MSR-TR-2014–112*, 2014.