

Opportunities and Challenges Of Machine Learning Accelerators In Production

Rajagopal Ananthanarayanan, Peter Brandt, Manasi Joshi, and Maheswaran Sathiamoorthy, *Google, Inc.*

https://www.usenix.org/conference/opml19/presentation/ananthanarayanan

This paper is included in the Proceedings of the 2019 USENIX Conference on Operational Machine Learning (OpML '19).

May 20, 2019 • Santa Clara, CA, USA

ISBN 978-1-939133-00-7

Open access to the Proceedings of the 2019 USENIX Conference on Operational Machine Learning is sponsored by USENIX.

Opportunities and Challenges Of Machine Learning Accelerators In Production

Rajagopal Ananthanarayanan, Peter Brandt, Manasi Joshi, Maheswaran Sathiamoorthy, *Google, Inc.*[1]
{ananthr, pbrandt, manasi, nlogn}@google.com

Abstract

The rise of deep learning has resulted in tremendous demand for compute power, with the FLOPS required for leading machine learning (ML) research doubling roughly every 3.5 months since 2012 [1]. This increase in demand for compute has coincided with the end of Moore's Law [2].

As a result, major industry players such as NVIDIA, Intel, and Google have invested in ML accelerators that are purpose built for deep learning workloads.

ML accelerators present many opportunities and challenges in production environments. This paper discusses some high level observations from experience internally at Google.

1. Products Enabled by ML Accelerators

ML accelerators have had transformational impact on consumer tech products. Many of the recent AI-inspired products developed across the industry would not have been possible without gains in compute via ML accelerators. Notable examples at Google include computational photography features in Photos and Lens, breakthroughs in Translate quality, Gmail SmartCompose, and improvements to Search and Assistant [3, 4]. Similarly, ML accelerators have powered neural architecture search [15, 20] with hyperparameter exploration to pick the best of the breed of large set of models for a given task.

DeepMind's WaveNet model is particularly illustrative [5]. WaveNet enabled a dramatic jump in text-to-speech quality, which for the first time approached truly human-sounding voice. However, the initial version of this model took 1 second to generate just .02 seconds of audio. Through optimizations enabled by Google's TPU ML accelerators, it was possible to achieve a 1000X speed improvement in audio generation and to launch both in Assistant and as a Cloud product offering [6].

ML accelerators have led to the launch of new large scale compute products e.g. NVIDIA's DGX-2 2 petaFLOPS system and Google's TPU v3 pod 100 petaFLOPS system.

2. Overview of ML Accelerator Specialization

ML accelerator chips and the systems which incorporate them are characterized by a variety of specializations compared to general purpose CPUs and distributed systems [2, 13]. These specializations have led to order of magnitude gains in performance and cost [16], and in turn led to significant breakthroughs in AI research e.g. AmoebaNet [17], AlphaGo [18], and BERT [19].

Below, we summarize key specializations tailored to deep learning, encompassing both supervised and unsupervised learning with neural networks (NN) [10].

2.1. Instruction Sets

The main instructions for an ML accelerator implement linear algebra operations such as matrix multiplication and convolutions. Supported data types allow variable precision tailored for deep learning workloads such as bfloat16 [11] and quantized or low-precision arithmetic [2, 13], leading to advantages in memory use and power savings.

2.2. Memory Hierarchy

ML accelerator instructions operate over block-oriented data to fully utilize memory and computation capacity. The memory hierarchy consists of on-chip buffers, on-board high bandwidth memory to efficiently feed data, and host memory to hold state across multiple ML accelerators.

2.3. Host Systems and Networks

To enable access to file systems for input/output, debugging and development workflows, language runtimes, and general purpose computing stack, ML accelerators are connected to a host CPU system. Hosts connect to each other through networks such as Gigabit Ethernet.

ML accelerators connect to hosts through off-the-shelf networking such as PCIe. Accelerator boards also incorporate customized high speed interconnects that connect multiple cores on and across boards. This allows for fast synchronization of state, e.g. by using AllReduce.

3. Software Stack Design

Software stacks for ML accelerators generally strive to abstract away hardware complexity from developers. However, the supported ops, data types, and tensor shapes can differ greatly across hardware platforms and create limitations. These differences can render parts of a model's architecture unsuitable for accelerators. It can be challenging to adapt models trained on one platform, e.g. CPU, to run on a different platform for production inference e.g. TPUv1 or mobile devices. Many bespoke solutions exist, but a good general purpose approach remains an active area of research, including compilers and runtimes that abstract away hardware details [7]. In practice, API design emphasizing helpful error messages greatly improves developer experience and enables broader adoption.

[1]Alphabetical by last name

4. Developer Experience

Model developers want to accelerate training and inference across a variety of models. Below we summarize a few key considerations in porting these computations to accelerators.

4.1 Model Suitability and Decomposition

Operations used by a model must be implemented using the instruction set of the accelerator e.g. to launch CUDA kernels. For a modeler, it is a crucial first step to know which of a model's ops are not supported on the accelerator and whether alternatives exist. Beyond compatibility, it is also important to consider the suitability of ops to run the accelerator (e.g. matmuls) vs. the host CPU (e.g. I/O).

A common decomposition is to place the input ops on the host CPU, with its access to the operating system stack, including file systems, and feed the data to the accelerator. APIs such as tf.data enable this decomposition [25, 26].

4.2 Batch Sizes and Learning Rates

Large batch sizes help to fully exploit the data parallelism available in accelerators. However, increasing the batch size without additional tuning may increase the out-of-sample error [12]. Hyper-parameter tuning, and warm-up techniques where learning rate is slowly increased, may be necessary to obtain quality comparable to lower batch sizes.

4.3 Toolchain - Reproducibility, Performance, and Tests

For a model developer, A/B diff tools integrated into the workflow are essential to compare metrics around model convergence (e.g. accuracy, recall, per batch weight distribution at every step of training) and performance (e.g. latency, throughput, resource utilization). The diff tools can quantify model prediction equivalence between CPU and accelerator based models. Comparing two model versions both using accelerators is important to track benefits and trade offs between cost, speed, and utilization. Finally, continuous quality tests and performance benchmarks across models must be used to gate models rolling into production.

4.4 Other Considerations

Contemporary large models deployed on multiple computers use asynchronous stochastic gradient descent (SGD) [12] to remain efficient on loosely coupled clusters found in data centers. With dedicated high performance interconnects, an accelerator-based system can use Synchronous SGD, which can be beneficial in terms of accuracy [10].

Other performance optimizations include support for batching, switching between model versions, model multi-tenancy to drive higher throughput for inference, and optimizing lookups of variables in the model graph from host CPUs to reduce per query cost.

5. Production Deployment

5.1 System Design for Balanced I/O and Compute

It is important to pay attention to bottlenecks that can creep into various stages of the training pipeline. For example, we need to ensure that the CPU host(s) connected to the ML accelerators can perform data processing, shuffling, transformation, etc. at a high throughput. If any of these stages is slow, the entire pipeline will be slow.

5.2. Diverse Hardware Pool Utilization

Traditionally, large-scale data processing used algorithms such as MapReduce [8] on large clusters of fungible, commodity hardware sharing the x86 architecture [13].

Accelerators add significant heterogeneity into data center environments. Ensuring efficient use of diverse hardware pools to achieve maximum value for an organization is an open problem. Implementing a dominant resource fairness policy [14] works well in practice for some training workloads. However, complications arise while considering data and traffic proximity, inference latency, and query cost.

5.3. Resource Planning

Timelines for designing new hardware and deploying it to data centers often stretch over several years. Pricing and accessibility to resources can significantly determine scale of adoption and benefit to the organization. Forecasting the future ML compute needs is uncertain, and driven by research progress, e.g. new computationally intensive models such as BERT [19]. When trends are accurately reflected in ASIC design and data center planning, it can drive enormous performance improvements [13].

6. Future Work

Several trends point to areas that will be increasingly important for accelerators over the next few years.

Multitask learning [21, 22], where a single model performs multiple tasks (e.g. CTR prediction and user like prediction in a recommender system), is growing in popularity. Today, the number of tasks is limited and generally related. Models with orders of magnitude more tasks of greater diversity are an active area of research [23]. Such models will consume more data of wider variety, posing I/O challenges. It may be necessary to distill [24] such giant models and leverage ML accelerators to make production serving possible.

Transfer learning is a related approach where pre-trained models are fine tuned to create many different models for different tasks, often with significant quality wins. If this grows in prevalence, it will dramatically increase the need for ML software stacks to consistently run inference on

model architectures across diverse HW platforms, regardless of the HW platform used to train the models.

References

[1] AI and Compute. https://blog.openai.com/ai-and-compute/

[2] Jouppi, N. P., Young, C., Patil, N., & Patterson, D. (2018). A domain-specific architecture for deep neural networks. *Communications of the ACM*, 61(9), 50-59.

[3] Google I/O 2018 Keynote, Sundar Pichai https://www.youtube.com/watch?v=ogfYd705cRs

[4] Cloud TPU. https://cloud.google.com/tpu/

[5] Oord, A.V.D., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. and Kavukcuoglu, K., 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

[6] Cloud Text to Speech. https://cloud.google.com/blog/products/gcp/introducing-cloud-text-to-speech-powered-by-deepmind-wavenet-technology

[7] Compilers for ML. https://www.c4ml.org/

[8] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107-113.

[9] Jouppi, N.P. *et al* (2017), June. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*(pp. 1-12). IEEE.

[10] Ian Goodfellow, Yoshua Bengio, Aaron Courville, (2016) Deep Learning, The MIT Press.

[11] Bfloat16 format. https://en.wikipedia.org/wiki/Bfloat16_floating-point_format

[12] Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., & Dahl, G. E. (2018). Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*.

[13] Wang, E., Davis, J.J., Zhao, R., Ng, H.C., Niu, X., Luk, W., Cheung, P.Y. and Constantinides, G.A., 2019. Deep Neural Network Approximation for Custom Hardware: Where We've Been, Where We're Going. *arXiv preprint arXiv:1901.06955*.

[14] Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., & Stoica, I. (2011, March). Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In NSDI (Vol. 11, No. 2011, pp. 24-24).

[15] Using Evolutionary AutoML to Discover Neural Network Architectures. https://ai.googleblog.com/2018/03/using-evolutionary-automl-to-discover.html

[16] DAWNBench v1 Deep Learning Benchmark Results. https://dawn.cs.stanford.edu/2018/04/30/dawnbench-v1-results/

[17] Real, E., Aggarwal, A., Huang, Y. and Le, Q.V. (2018). Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*.

[18] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. and Chen, Y., 2017. Mastering the game of go without human knowledge. *Nature*, *550*(7676), p.354.

[19] Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[20] Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

[21] Caruna, R. (1993). Multitask learning: A knowledge-based source of inductive bias. In *Machine Learning: Proceedings of the Tenth International Conference*, 41–48.

[22] Caruana, R. (1998). Multitask learning. In *Learning to learn*. Springer. 95–133.

[23] Kaiser, L., Gomez, A.N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L. and Uszkoreit, J., (2017). One model to learn them all. *arXiv preprint arXiv:1706.05137*.

[24] Hinton, G., Vinyals, O. and Dean, J., (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

[25] Keras on TPUs in Colab. https://medium.com/tensorflow/tf-keras-on-tpus-on-colab-674367932aa0

[26] Data Input Pipeline Performance https://www.tensorflow.org/guide/performance/datasets