A New Golden Age in Computer Architecture: Empowering the Machine Learning Revolution

Jeff Dean, David Patterson, and Cliff Young
Google Brain
jeff@google.com, davidpatterson@google.com, cliffy@google.com

Abstract: The ending of Moore's Law and Dennard scaling has led to the ending of rapid improvement in general-purpose program performance. Machine learning, and in particular deep learning, is an attractive alternative for architects to work on. It has recently revolutionized vision, speech, language understanding, and many other fields, and promises to help with the grand challenges facing our society. The computation at its core is low-precision linear algebra. Thus, machine learning is both broad enough to apply to many domains, yet narrow enough to benefit from domain-specific architectures, such as Google's TPU. Moreover, the growth in demand for machine learning computing exceeds Moore's Law at its peak just as it is fading. Hence, machine learning experts and computer architects must work together to design the computing systems required to deliver on the potential of machine learning. This paper offers motivation, suggestions, and warnings to computer architects on how to best contribute to the machine learning revolution.

As scientists and engineers, it's our responsibility to address society's biggest problems, such as those in Figure 1. Many of the items on that list might seem to be beyond the skill set of *computer* scientists and engineers, but happily, that is not the case. We believe advances in machine learning will lead to significant advances on most of these grand challenges.

Make solar energy affordable	Reverse-engineer the brain
Provide energy from fusion	Prevent nuclear terror
Develop carbon sequestration methods	Secure cyberspace
Manage the nitrogen cycle	Enhance virtual reality
Provide access to clean water	Advance personalized learning
Restore and improve urban infrastructure	Engineer the tools for scientific discovery
Advance health informatics	Engineer better medicines
Enable universal communication	Build flexible general-purpose AI systems

Figure 1. Societal grand challenges. The National Academy of Engineering proposed the first 14 in 2008 [1]. Since it's been a decade, the authors added two new challenges in the last row, which seem appropriate for today.

The deep neural network branch of machine learning is a transformational technology that in the last five years has started revolutions in several fields. In 2012, image recognition didn't work well. AlexNet [2] broke previous accuracy records of the ImageNet competition [3], and has led to a sequence of further improvements such that machines now exceed human accuracy in image recognition tasks [4,5,6,7]. Similar breakthroughs have accelerated fields such as speech recognition [8], web search [9], human language translation [10,11], medical

Digital Object Identifier 10.1109/MM.2018.112130030 0272-1732/$26.00 2018 IEEE

diagnosis [12,13,14], and even playing the game of Go [15]. We expect this trend of breakthroughs to continue, and to broaden to provide new avenues of research and development for today's grand challenge problems.

The current machine learning revolution requires two kinds of scale: in the data sets that are available and the computing resources used to analyze them. Large data sets contain the raw material to understand the world around us. Thus far, large-scale machine learning computing has been done in large datacenters containing hordes of GPUs, which were originally designed to accelerate graphics. We are transitioning to an era where datacenters will be filled with computers designed solely for machine learning computations. They will come not just from GPU manufacturers, but also from startups, from new product lines by traditional microprocessor suppliers, and from more established Internet companies that had not previously built their own processors.

For example, Google began designing custom ML ASICs in 2013 and has deployed them in its datacenters since 2015. Our first-generation Tensor Processing Unit (TPU) [16] (Figure 2) focused on the production phase of ML, because deploying ML models at Google scale demanded unprecedented computing power. We projected that deep neural networks would become so popular that they would double computation demands on our datacenters (as indeed, they have). The production phase is called *inference* or *prediction*. The earlier development phase is called *training* or *learning*. The second-generation Cloud TPU [17] (Figure 3), deployed in 2017, trains large ML models at scale quickly. Both datacenters and edge devices (such as cars, phones, and watches) have custom hardware for inference, but the majority of training cycles will likely reside in datacenters.

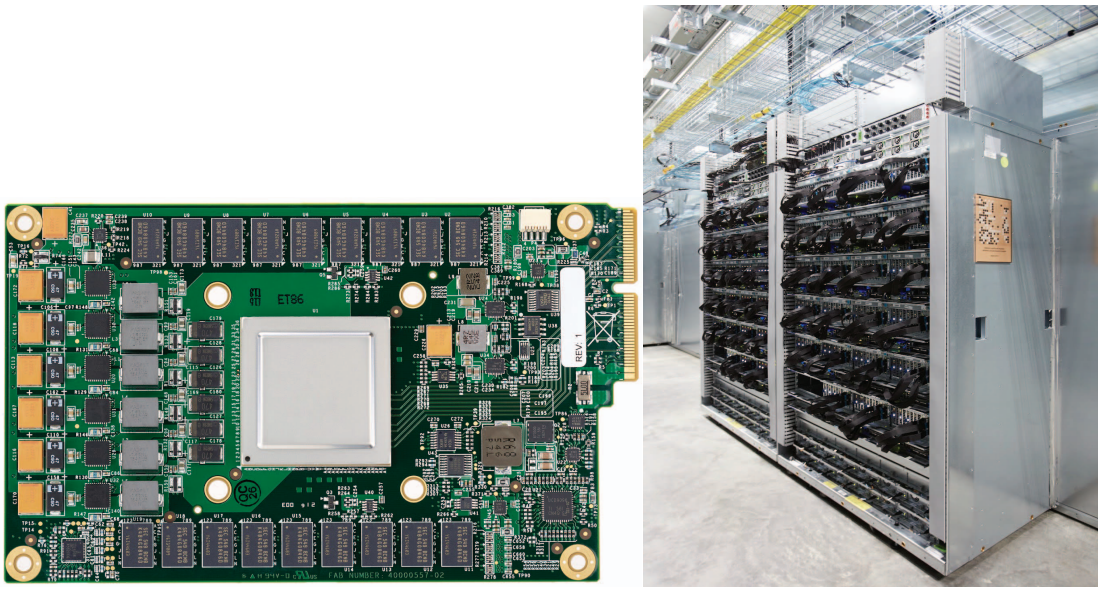

Figure 2. The first generation Google TPU chip, board, and system. AlphaGo ran on the TPUs in these racks to defeat Lee Sedol at Go in 2016.

Digital Object Identifier 10.1109/MM.2018.112130030 0272-1732/$26.00 2018 IEEE

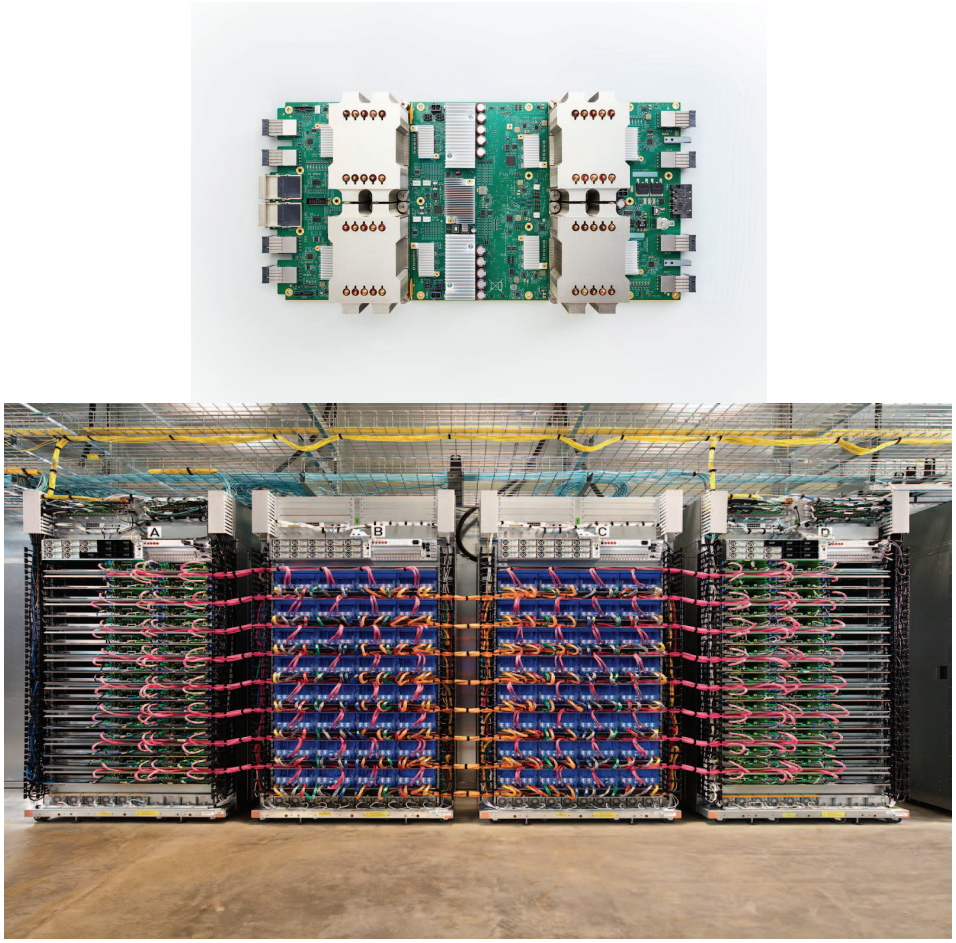

Figure 3. The second generation Google TPU chip, board, and system. The board has 4 chips. The collection of racks includes 64 boards and a custom high-speed network.

Even with these new custom accelerators, our ML researchers remain hungry for more. We observe that no matter what hardware is available, ML researchers have a “patience threshold” that limits how long they are willing to wait for the results of an experiment (much as computer architects have a threshold for how long they are willing to wait for simulations to run). The threshold varies between a few hours and a week, depending on the researcher. Faster hardware enables larger, more ambitious experiments within the confines of this patience threshold; smaller experiments run the risk that they won’t explore a large enough space to improve ML accuracy. What this means for computer systems designers is that demand for ML cycles is effectively unbounded. Speeding up model evaluation by any amount is valuable; factors of 10 or 100 would likely increase the rate of recent breakthroughs such as those mentioned above.

Alas, just when ML provides a burst of seemingly inexhaustible demand for computing cycles, computer architecture is nearing the end of the Moore’s Law and Dennard scaling. Their demise raises the stakes even higher for system designers: we need to solve exponentially harder problems without the exponentially more resources that Moore’s Law used to deliver. We will need to find better, more efficient solutions at the same time as we tackle larger, more difficult problems.

Advice for Hardware Engineers

Digital Object Identifier 10.1109/MM.2018.112130030 0272-1732/$26.00 2018 IEEE

ML applications span the entire spectrum of hardware design points, from the datacenter and supercomputers, to automotive applications, mobile devices, and the internet-of-things. Each of these design points comes with a different package of constraints on cost, power, size, and the mixture of compute, communication, and storage required for a solution.

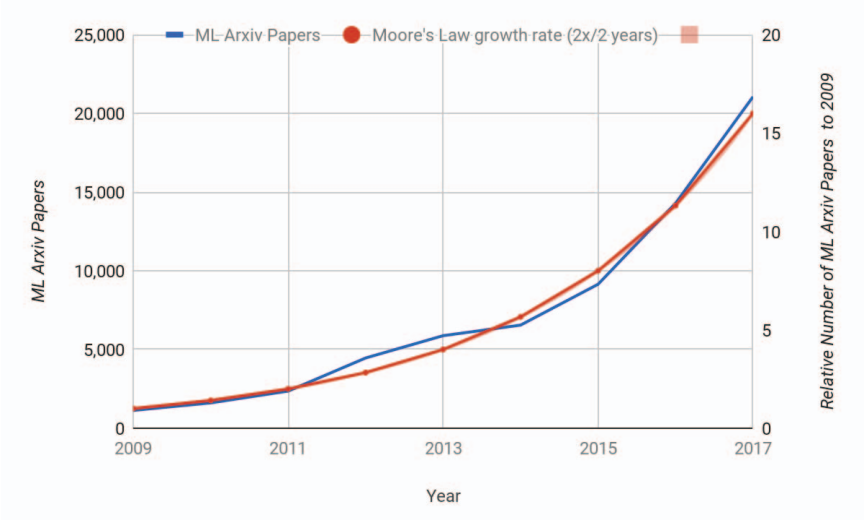

Figure 4. ML arXiv papers per year. They have more than doubled every two years, which was Moore's prediction for chips in 1975. We counted papers in the cs.CV, cs.CL, cs.LG, cs.AI, cs.NE, and stat.ML topics.

Moreover, ML algorithms are changing rapidly. Figure 4 shows the number of ML papers posted on arXiv.org per year, graphed over time. They grow faster than the Moore's Law rate of doubling every two years. More than 50 ML papers now appear daily on Arxiv alone! The good news is that these algorithms are commonly expressed in higher-level frameworks, such as TensorFlow [18], Caffe [19], or PyTorch [20], which raise the programming abstraction and make it easier to map the program to innovative architectures.

Hardware designs must remain relevant over at least two years of design time plus a three-year deployment window (assuming standard depreciation schedules). As designers, we need to pick a good system-level balance point between specialization for the currently popular techniques and flexibility to handle changes in the field over the lifetime of a design. Designing appropriate hardware for a five year time window in a field changing as rapidly as machine learning is quite challenging. Compilation techniques, possibly themselves enhanced by machine learning, will be central to meeting this challenge. It would be ideal if users could express their ML problems in the most natural way, but then rely on compilation to give interactive response times for researchers and scalability for production users.

Because of this wide range of constraints, it seems hard to imagine that a single solution is the best overall, in the same way that the x86 architecture came to dominate general-purpose computation in the late 1990s. The end of Moore's Law might also mean an end to the "trickle-down" effect in computer architecture, where this year's high-end part can be better integrated and cost-reduced to be the mainstream part of two years from now, and the embedded part of four years from now.

This section discusses six issues that impact ML hardware design, roughly sorted on a spectrum from purely architectural to mostly ML-driven concerns.

1. **Training**. Both training and inference are important, but insufficient computer architectural work is going into training. Inference is easier, not only because it takes ⅓ as many arithmetic operations as training. Storage requirements are larger in training

Digital Object Identifier 10.1109/MM.2018.112130030 0272-1732/$26.00 2018 IEEE

because activation values must be saved for *back-propagation*, where the feedback about the model's prediction accuracy is applied to update the model's parameters. Assuming good enough latency, inference workloads can be scaled out. Training doesn't necessarily scale out: it currently requires many expensive, sequential steps. It's not surprising that many companies' (including Google's) first forays into deep learning hardware have aimed at inference first. Tackling inference is only solving the easier, more scalable piece of the end-to-end problem, however.

2. **Batch size.** Batch size [21] has fundamental implications for architecture—it enables an important form of operand reuse—but it is surprisingly poorly understood, especially for training. The perceived knowledge in the deep learning field says that the maximum accuracy improvement per computational step comes from stochastic gradient descent with momentum at a minibatch size of 1. Today's GPUs and TPUs operate efficiently at minibatch sizes of 32 or larger, raising questions about whether architectures for minibatch=1 might be more effective. Empirical results are confusing: a recent sequence of papers [22,23] has shown that image-oriented convolutional models can train effectively at minibatch sizes of 8192 and 32768. We are unaware of similar results for multi-layer perceptron classifiers or for LSTM-based models. If batch size could be made arbitrarily large while still training effectively, then training is amenable to standard weak scaling approaches. However, if the training rate of some models is restricted to small batch sizes, then we will need to find other algorithmic and architectural approaches to their acceleration.
3. **Sparsity and Embeddings.** Sparsity in machine learning takes many forms, which lend themselves to very different approaches to hardware support. Many recent architectural papers address fine-grain sparsity, exploiting zeros and small values to reduce work, and indeed, because of the use of Rectified Linear Units (ReLU) [24] as activation functions, many models exhibit significant levels of fine-grained sparsity in their activation values. However, we think that coarse-grain sparsity, where an example touches only a fraction of the parameters of huge model, has even more potential: *Mixture of Experts* (MoE) models [25] consult a learned subset of a panel of experts as part of their network structure. Thus, MoE models train more weights using fewer FLOPS for higher accuracy than previous approaches. *Embeddings*, which transform huge sparse spaces (such as vocabularies) into more compact dense representations suitable for linear algebra operations, have received relatively little attention from the architecture community, yet they are key to textual applications like web search and translation. Unlike other kinds of accesses to parameters in neural networks, accesses to embedding tables often involve many relatively small random accesses in very large data structures (e.g., hundreds of 100-1000 byte reads in multi-hundred-gigabyte data structures per training or inference example).
4. **Quantization and Distillation.** A number of techniques have been deployed to great effect to provide cost-effective inference, but might also greatly benefit training. *Quantized* or reduced-precision arithmetic representations [26] have proven themselves in several inference accelerators and are now available in GPUs. Sadly, there is very little work on training in reduced-precision arithmetic, and the little published work focuses on toy-sized problems like MNIST and CIFAR-10 (see the Pitfall in the next section). *Distillation* [27] uses a larger model to bootstrap the training of a smaller model, while achieving higher accuracy than directly training the smaller model on the same inputs. It's not clear why this is the case: could better training methods allow us to directly train the smaller models (and perhaps all models) to higher accuracy? Is there something fundamental about the more degrees of freedom in the larger model that enables better training?

Digital Object Identifier 10.1109/MM.2018.112130030 0272-1732/$26.00 2018 IEEE

5. **Networks with Soft Memory**. Some deep learning techniques provide functionality akin to random-access memory; doing so is complicated by preserving differentiability so the techniques can support back-propagation. Examples include Neural Turing Machines [28,29], Memory Networks [30], and *attention* [31]. Attention allows Google Translate to map from a word in the target language to the corresponding word or phrase in the source language. Such soft mechanisms are expensive compared to traditional "hard" memories, because the soft memory computes a weighted average over all entries of a table. A hard memory simply loads a single entry from a table. We haven't seen research into efficient or sparse implementations of these soft memory models.
6. **Learning to Learn (L2L)**. Recent results from Zoph et al. [32] suggest that neural network models can themselves search for and develop better neural network models. They used the CIFAR-10 image recognition competition to show that L2L could match the accuracy of the best models developed by ML experts. The L2L search used the smaller CIFAR-10 dataset instead of ImageNet to make the computational workload plausible on today's hardware, and nevertheless required hundreds of GPU cards for a week. Despite huge growth in the field, there is a shortage of experts on Deep Learning technologies. L2L offers the possibility that we might use far more computing resources but require considerably less human machine-learning expertise in designing machine-learning solutions, which seems like a utilitarian optimization of marginal returns on capital and labor. For those doing hardware-software co-design, L2L gives the further possibility to construct better systems by simultaneously automatically searching for new ML model architectures *and* new computer architectures.

ML Hardware Fallacies and Pitfalls

We complement the six issues above with four warnings, following the Fallacies and Pitfalls style [33].

Fallacy: Given the large size of the ML problems, the hardware focus should be operations per second (throughput) rather than time to solution (latency).

You need to know the right target if you are going to hit it. Proper metrics are important to understand and apply. Because it is easy to collect, many ML benchmarks compare input examples/second (i.e., throughput) when measuring the performance of ML systems. The original TPU paper [16] pointed out that latency is even more important than throughput for user-facing inference workloads. The best measure of a training system is *time to converged accuracy*, i.e., the wall-clock time to fully train a model to a desired accuracy.

Fallacy: Given a sufficiently large speedup, ML researchers would be willing to sacrifice a little accuracy.

Take great care when trading accuracy for performance. It's sometimes okay, but it depends a lot on specifics. Using a grand challenge from Figure 1, loss of accuracy on "*preventing nuclear terror*" is probably not good. For many ML researchers, a high-throughput training system that loses 0.5% in accuracy is uninteresting. At the other extreme, some mobile applications accept higher accuracy losses in order to fit on-device.

Pitfall: Designing hardware using last year's models.

Figure 2 suggests the rapid rate of change in ML. Recent architecture papers for ML have evaluated their proposals using MNIST, CIFAR-10 and AlexNet. While MNIST is important

Digital Object Identifier 10.1109/MM.2018.112130030 0272-1732/$26.00 2018 IEEE

historically, it is 20 years old and useful primarily for student programming assignments. CIFAR-10 was the image recognition competition that ImageNet replaced in 2010 because it became too easy. Even AlexNet, which kickstarted the CNN revolution by winning ImageNet in 2012, is now old fashioned. It has only 6 neural network layers, while the 2017 champion has 150. Furthermore, it has the bulk of its parameters in two fully-connected layers at the top of the model, a characteristic that is missing from nearly every other accurate image model since Alexnet. Designing hardware or evaluating proposed optimizations based on MNIST, CIFAR-10, or AlexNet is no more effective or persuasive than designing general-purpose processors evaluated with quicksort, hashing, or binary search. Ideally, the field would develop benchmark suites for deep learning accelerators, similarly to the way that workstation manufacturers developed the SPEC [34] benchmarks. Until such benchmarks emerge, using the recent winners of ML competitions such as ImageNet ensures up-to-date if narrow results.

Pitfall: Designing ML hardware assuming the ML software is untouchable.

In the heydays of Moore's Law and ample instruction level parallelism, an architect could contribute without understanding what the program was trying to do. In fact, the ground rules of benchmark suites like SPEC2006 disallowed changes to the source code, so studying it didn't help. A hardware team that takes the time to learn some of the constraints and paradigms of the ML community is much more likely to produce an effective solution than one that sticks to familiar territory. There is no requirement to run the identical model; a faster but different computation with the same accuracy is a welcome result. The goal is a hardware/software system that more effectively solves the problem.

Conclusion: A Grand Opportunity

ML advances in the next decade may help solve some of the grand challenges in Figure 1. For example, a car that never crashes was science fiction in 2000, but thanks to advances in ML, might be a reality in 2020. These past successes and exciting future prospects have accelerated the demand for compute cycles for ML.

In the past, specialized hardware rarely made sense, because it often applied to narrow pieces of our overall problems. However, ML hardware is an exception: we can build specialized machines uniquely tailored to it—very fast low-precision linear algebra—and then apply them to a huge and growing swath of the world's computing needs going forward: vision, speech, language understanding, and so on. Given that performance of general-purpose code on traditional microprocessors has plateaued along with Moore's Law and Dennard scaling, perhaps domain-specific architectures for ML should be the next major hardware focus. ML applies to a vast and growing range of useful computations and exploration of the hardware design space for ML has only just begun.

One critical shortage in the race to rollout exciting applications of machine learning is availability of ML experts. Learn-to-learn (L2L) systems may answer this shortage. Suppose we wanted to run L2L on a big problem instead of CIFAR-10. L2L trained about 14,000 experimental models, each of which took about one hour on one GPU for CIFAR-10. If we instead trained 100,000 experimental models (because the more complex problem needs more data points to learn from), and each such experiment required 200 GPUs to train for a week (which is the size of some of the translation models trained today), the total is 20M GPU weeks. That is, doing L2L to develop a state-of-the-art model on a problem of significant scale might require 100,000 times more computing problem than what our earliest L2L experiments on CIFAR-10 used.

Digital Object Identifier 10.1109/MM.2018.112130030 0272-1732/$26.00 2018 IEEE

From an architect's viewpoint, the next decade promises to be exciting. The demand for machine learning computing is growing much faster than Moore's Law just when Moore's Law itself is fading. Architects need to deliver innovative machines to empower ML experts, and together, we will be able to address the grand challenges that our society faces.

Acknowledgments

We thank the Hot Chips and IEEE Micro reviewers for their excellent suggestions. Our colleagues in Google's Brain and Platforms teams have helped us immensely in understanding and mapping these codesign problems.

References

1. 14 Grand Challenges for Engineering in the 21st Century, http://www.engineeringchallenges.org/challenges.aspx.
2. A. Krizhevsky, I. Sutskever, and G. Hinton, " ImageNet Classification with Deep Convolutional Neural Networks", *Advances in Neural Information Processing Systems 25 (NIPS 2012)*; https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.
3. O. Russakovsky et al., "Imagenet large scale visual recognition challenge." *Int'l Journal of Computer Vision*, 2015, 115(3).
4. C. Szegedy et al., "Going Deeper with Convolutions," *arXiv.org*, 17 Sept. 2014; https://arxiv.org/abs/1409.4842.
5. S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv.org*, 11 Feb. 2015; https://arxiv.org/abs/1502.03167.
6. C. Szegedy et al., "Rethinking the Inception Architecture for Computer Vision," *arXiv.org*, 2 Dec. 2015; https://arxiv.org/abs/1512.00567.
7. K. He et al., "Deep Residual Learning for Image Recognition," *arXiv.org*, 10 Dec. 2015; https://arxiv.org/abs/1512.03385.
8. G. Hinton et al., "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups", *IEEE Signal Processing Magazine*, vol. 29, no. 6, 2012, pp. 82-97; http://ieeexplore.ieee.org/document/6296526/.
9. J. Clark, "Google Turning Its Lucrative Web Search Over to AI Machines," *Bloomberg Technology*, 26 Oct. 2015, https://www.bloomberg.com/news/articles/2015-10-26/google-turning-its-lucrative-web-search-over-to-ai-machines.
10. Y. Wu et al., "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," *arXiv.org*, 26 Sept. 2016; https://arxiv.org/abs/1609.08144.
11. M. Johnson et al., "Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation," *arXiv.org*, 14 Nov 2016; https://arxiv.org/abs/1611.04558.
12. V. Gulshan, L. Peng, and M. Coram, "Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs," *JAMA.* 2016;316(22):2402-2410. doi:10.1001/jama.2016.17216; http://jamanetwork.com/journals/jama/fullarticle/2588763.
13. Y. Liu et al., "Detecting Cancer Metastases on Gigapixel Pathology Images," *arXiv.org*, 3 Mar. 2017; https://arxiv.org/abs/1703.02442.
14. J. Olczak et al., "Artificial intelligence for analyzing orthopedic trauma radiographs," *Acta Orthopaedica*, DOI: 10.1080/17453674.2017.1344459; http://www.tandfonline.com/doi/full/10.1080/17453674.2017.1344459.

Digital Object Identifier 10.1109/MM.2018.112130030 0272-1732/$26.00 2018 IEEE

15. D. Silver et al., “Mastering the game of Go with deep neural networks and tree search,” *Nature* vol. 529, pp. 484-489, 28 Jan. 2016; http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html
16. N. Jouppi et al., “In-Datacenter Performance Analysis of a Tensor Processing Unit,” Proc. 44th Annual Intl Symp on Computer Architecture, pp. 1-12; https://dl.acm.org/citation.cfm?id=3080246.
17. Google.ai, “Cloud TPUs: Google’s second-generation Tensor Processing Unit is coming to Cloud,” 2017, http://g.co/tpu
18. M. Abadi et al., “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *arXiv.org preprint arXiv:1603.04467*.
19. Y. Jia et al., “Caffe: Convolutional Architecture for Fast Feature Embedding,” *Proc. 22nd ACM Int’l Conf. on Multimedia*, 2014, pp. 675-678.
20. A. Paszke and S. Chintala. "PyTorch." (2017); http://www.pytorch.org.
21. https://developers.google.com/machine-learning/glossary/#batch size.
22. P. Goyal et al., “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour,” *arXiv.org*, 8 Jun 2017; https://arxiv.org/abs/1706.02677.
23. Y. You et al., “100-epoch ImageNet Training with AlexNet in 24 Minutes,” *arXiv.org*, 20 Sep 2017, https://arxiv.org/pdf/1709.05011.pdf.
24. M.D. Zeiler et al., “On rectified linear units for speech processing,” Proc. *IEEE Int’l Conf* on *Acoustics, Speech and Signal Processing (ICASSP),* 2013,pp. 3517-3521. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/40811.pdf
25. M. Shazeer et al., “Outrageously large neural networks: The sparsely-gated mixture-of-experts layer,” *arXiv.org*, 23 Jan 2017https://arxiv.org/abs/1701.06538.
26. P. Warden, “What I’ve learned about neural network quantization,” blog, 22 June 2017; https://petewarden.com/2017/06/22/what-ive-learned-about-neural-network-quantization/
27. G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network.” *arXiv preprint arXiv:1503.02531*; https://arxiv.org/abs/1503.02531.
28. A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines." *arXiv* preprint arXiv:1410.5401, 20 Oct. 201); https://arxiv.org/abs/1410.5401.
29. A. Graves, et al., “Hybrid computing using a neural network with dynamic external memory,”, *Nature* vol. 538, pp. 471–476, 27 Oct. 2016, doi:10.1038/nature20101.
30. J. Weston, S. Chopra, and A. Bordes, “Memory Networks,” *arXiv.org*, 15 Oct. 2014; https://arxiv.org/abs/1410.3916.
31. D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,”, *arXiv.org*, 1 Sep. 2014; https://arxiv.org/abs/1409.0473.
32. B. Zoph and Q. Le, “Neural Architecture Search with Reinforcement Learning,” *arXiv.org*, 5 Nov. 2016; https://arxiv.org/abs/1611.01578.
33. J. Hennessy, and D.A. Patterson, *Computer Architecture: a Quantitative Approach, 6th edition*. Elsevier, 2018.
34. http://spec.org

Digital Object Identifier 10.1109/MM.2018.112130030 0272-1732/$26.00 2018 IEEE