DOI:10.1145/3467017

After decades of incentivizing the isolation of hardware, software, and algorithm development, the catalysts for closer collaboration are changing the paradigm.

BY SARA HOOKER

The Hardware Lottery

HISTORY TELLS US that scientific progress is imperfect. Intellectual traditions and available tooling can prejudice scientists away from some ideas and towards others.²⁴ This adds noise to the marketplace of ideas and often means there is inertia in recognizing promising directions of research. In the field of artificial intelligence (AI) research, this article posits that it is tooling which has played a disproportionately large role in deciding which ideas succeed and which fail.

What follows is part position paper and part historical review. I introduce the term “hardware lottery” to describe when a research idea wins because it is compatible with available software and hardware, not because the idea is superior to alternative research directions. The choices about software and hardware have often played decisive roles in deciding the winners and losers in early computer science history.

These lessons are particularly salient as we move into a new era of closer collaboration between the hardware, software, and machine-learning research communities. After decades of treating hardware,

software, and algorithm as separate choices, the catalysts for closer collaboration include changing hardware economics, a “bigger-is-better” race in the size of deep-learning architectures, and the dizzying requirements of deploying machine learning to edge devices.

Closer collaboration is centered on a wave of new-generation, “domain-specific” hardware that optimizes for the commercial use cases of deep neural networks. While domain specialization creates important efficiency gains for mainstream research focused on deep neural networks, it arguably makes it even more costly to veer off the beaten path of research ideas. An increasingly fragmented hardware landscape means that the gains from progress in computing will be increasingly uneven. While deep neural networks have clear commercial use cases, there are early warning signs that the path to the next breakthrough in AI may require an entirely different combination of algorithm, hardware, and software.

This article begins by acknowledging a crucial paradox: machine-learning researchers mostly ignore hardware despite the role it plays in determining which ideas succeed. The siloed evolution of hardware, software, and algorithm has played a critical role in early hardware and software lotteries. This article considers the ramifications of this siloed evolution

» key insights

- The term hardware lottery describes a research idea that wins due to its compatibility with available software and hardware, not its superiority over alternative research directions.
- We may be in the midst of a present-day hardware lottery. Hardware design has prioritized delivering on commercial use cases, while built-in flexibility to accommodate the next generation of ideas remains a secondary consideration.
- Any attempt to avoid future hardware lotteries must be concerned with making it cheaper and less time consuming to explore different hardware/software/algorithm combinations.

IMAGE BY ANDRIJ BORYS ASSOCIATES

BONUS

TECH YOUR HARDWARE

NEW

**ARE YOU A
TECH WINNER?**

**BIG
THE BETTER**

SCALE

WILL YOUR IDEA WIN

with examples of early hardware and software lotteries. And, while today's hardware landscape is increasingly heterogeneous, I posit that the hardware lottery has not gone away, and the gap between the winners and losers will grow. After unpacking these arguments, the article concludes with thoughts on how to avoid future hardware lotteries.

Separate Tribes

For the creators of the first computers, the program was the machine. Due to both the cost of the electronics and a lack of cross-purpose software, early machines were single use; they were not expected to be repurposed for a new task (Figure 1). Charles Babbage's "difference engine" (1817) was solely intended to compute polynomial functions.⁹ IBM's Harvard Mark I (1944) was a programmable calculator.²² Rosenblatt's perceptron machine (1958) computed a step-wise single-layer network.⁴⁸ Even the Jacquard loom (1804), often thought of as one of the first programmable machines, was, in practice, so expensive to re-thread that it was typically threaded once to support a pre-fixed set of input fields.³⁶

In the early 1960s, joint specialization of hardware and software went vertical. IBM was an early pioneer in the creation of instruction sets, which were portable between computers. A growing business could install a small IBM 360 computer and not be forced to relearn everything when migrating to a bigger 360 machine. Competitors Burroughs, Cray, and Honeywell all developed their own systems—compatible with their own machines but not across manufacturers. Programs could be ported between different machines from the same manufacturer, but not on competitive machines. The design itself remained siloed, with hardware and software developed jointly in-house.

Today, in contrast to the specialization necessary in computing's very early days, machine-learning researchers tend to think of hardware, software, and algorithms as three separate choices. This is largely due to a period in computer science history that radically changed the type of hardware that was produced and incentivized

the hardware, software, and machine-learning research communities to evolve in isolation.

The general-purpose computer era crystallized in 1969, when a young engineer named Gordon Moore penned an opinion piece in *Electronics* magazine titled, "Cramming More Components onto Circuit Boards."³³ In it, Moore predicted that the number of transistors on an integrated circuit could be doubled every two years. The article and subsequent follow-up were originally motivated by a simple desire—Moore thought it would sell more chips. However, the prediction held and motivated a remarkable decline in the cost of transforming energy into information over the next 50 years.

Moore's law combined with Dennard scaling¹² enabled a factor of three-magnitude increase in micro-processor performance from 1980 to 2010. The predictable increases in computing power and memory every two years meant hardware design became risk averse. Why experiment on more specialized hardware designs for an uncertain reward when Moore's law allowed chip makers to lock in predictable profit margins? Even for tasks which demanded higher performance, the benefits of moving to specialized hardware could be quickly eclipsed by the next generation of general-purpose hardware with ever-growing computing power.

The emphasis shifted to universal processors, which could solve myriad different tasks. The few attempts to deviate and produce specialized supercomputers for research were financially unsustainable and short-lived. A few

Figure 1. Early computers were single use and were not expected to be repurposed. These machines could not be expected to run the variety of programs our modern-day machines do.

very narrow tasks, such as mastering chess, were an exception to this rule because the prestige and visibility of beating a human adversary attracted corporate sponsorship.³⁴

Treating the choice of hardware, software, and algorithm as independent has persisted until recently. It is expensive to explore new types of hardware, both in terms of time and capital required. Producing a next-generation chip typically costs \$30-\$80 million and takes two to three years to develop.¹⁴ These formidable barriers to entry have produced a hardware research culture that might feel odd or perhaps even slow to the average machine-learning researcher. While the number of machine-learning publications has grown exponentially in the last 30 years, the number of hardware publications has maintained a fairly even cadence.⁴² For a hardware company, leakage of intellectual property can make or break the survival of the firm. This has led to a much more closely guarded research culture.

In the absence of any lever with which to influence hardware development, machine-learning researchers rationally began to treat hardware as a sunk cost to work around rather than something fluid that could be shaped. However, just because we have abstracted hardware away does not mean it has ceased to exist. Early computer science history tells us there are many hardware lotteries where the choice of hardware and software has determined which ideas succeed and which fail.

The Hardware Lottery

The first sentence of Tolstoy's *Anna Karenina* reads, "All happy families are alike; every unhappy family is unhappy in its own way."⁴⁷ Tolstoy is saying that it takes many different things for a marriage to be happy—financial stability, chemistry, shared values, healthy offspring. However, it only takes one of these aspects to not be present for a family to be unhappy. This has been popularized as the Anna Karenina principle, "a deficiency in any one of a number of factors dooms an endeavor to failure."³²

Despite our preference to believe algorithms succeed or fail in isolation, history tells us that most computer science breakthroughs follow the Anna

Karenina principle. Successful breakthroughs are often distinguished from failures by benefiting from multiple criteria aligning surreptitiously. For AI research, this often depends upon winning what we have named the *hardware lottery*—avoiding possible points of failure in downstream hardware and software choices.

An early example of a hardware lottery is the analytical machine (1837). Charles Babbage was a computer pioneer who designed a machine that could be programmed, at least in theory, to solve any type of computation. His analytical engine was never built, in part because he had difficulty fabricating parts with the correct precision.²⁵ The electromagnetic technology required to actually build the theoretical foundations laid down by Babbage only surfaced during WWII. In the first part of the 20th century, electronic vacuum tubes were heavily used for radio communication and radar. During WWII, these vacuum tubes were repurposed to provide the computing power necessary to break the German enigma code.¹⁰

As noted in the TV show *Silicon Valley*, often “being too early is the same as being wrong.” When Babbage passed away in 1871, there was no continuous path between his ideas and modern computing. The concept of a stored program, modifiable code, memory, and conditional branching were rediscovered a century later because the right tools existed to empirically show that the idea worked.

The Lost Decades

Perhaps the most salient example of the damage caused by not winning the hardware lottery is the delayed recognition of deep neural networks as a promising direction of research. Most of the algorithmic components needed to make deep neural networks work had already been in place for a few decades: backpropagation was invented in 1963,⁴³ reinvented in 1976,²⁹ and then again in 1988,³⁹ and was paired with deep convolutional neural networks¹⁵ in 1989.²⁷ However, it was only three decades later that deep neural networks were widely accepted as a promising research direction.

The gap between these algorithmic advances and empirical success

While domain specialization creates important efficiency gains for mainstream research focused on deep neural networks, it arguably makes it even more costly to veer off the beaten path of research ideas.

is due in large part to incompatible hardware. During the general-purpose computing era, hardware such as central processing units (CPUs) was heavily favored and widely available. CPUs are very good at executing an extremely wide variety of tasks; however, processing so many different tasks can incur inefficiency. CPUs require caching intermediate results and are limited in the concurrency of tasks that can be run, which poses limitations for an operation such as matrix multiplication, a core component of deep neural-network architectures. Matrix multiplies are very expensive to run sequentially but far cheaper to compute when parallelized. The inability to parallelize on CPUs meant matrix multiplies quickly exhausted memory bandwidth, and it simply wasn't possible to train deep neural networks with multiple layers.

The need for hardware that supported tasks with lots of parallelism was pointed out as far back as the early 1980s in a series of essays titled, “Parallel Models of Associative Memory.”¹⁹ The essays argued persuasively that biological evidence suggested massive parallelism was needed to make deep neural-network approaches work.

In the late 1980s/90s, the idea of specialized hardware for neural networks had passed the novelty stage. However, efforts remained fractured due to a lack of shared software and the cost of hardware development. Without a consumer market, there was simply not the critical mass in end users to be financially viable. It would take a hardware fluke in the early 2000s, a full four decades after the first paper about backpropagation was published, for the insights about massive parallelism to be operationalized in a useful way for connectionist deep neural networks.

A graphical processing unit (GPU) was originally introduced in the 1970s as a specialized accelerator for video games and developing graphics for movies and animation. In the 2000s, GPUs were repurposed for an entirely unimagined use case—to train deep neural networks.⁷ GPUs had one critical advantage over CPUs: they were far better at parallelizing a set of simple, decomposable instructions, such as matrix multiples. This higher number

of effective floating-point operations per second (FLOPS), combined with clever distribution of training between GPUs, unblocked the training of deeper networks.

The number of layers in a network turned out to be the key. Performance on ImageNet jumped with ever-deeper networks. A striking example of this jump in efficiency is the now-famous 2012 Google research that required 16,000 CPU cores to classify cats; just a year later, a published paper reported the same task had been accomplished using only two CPU cores and four GPUs.⁸

Software Lottery

Software also plays a role in deciding which research ideas win and which ones lose. Prolog and LISP were two languages heavily favored by the AI community until the mid-90s. For most of this period, AI students were expected to actively master at least one, if not both. LISP and Prolog were particularly well suited to handling logic expressions, which were a core component of reasoning and expert systems.

For researchers who wanted to work on connectionist ideas, such as deep neural networks, no clearly suited language of choice existed until the emergence of MATLAB in 1992. Implementing connectionist networks in LISP or Prolog was cumbersome, and most researchers worked in low-level languages such as C++. It was only in the 2000s that a healthier ecosystem began to take root around software developed for deep neural-network approaches, with the emergence of LUSH and, subsequently, TORCH.

Where there is a loser, there is also a winner. From the 1960s through the mid-1980s, most mainstream research focused on symbolic approaches to AI. Unlike deep neural networks, where learning an adequate representation is delegated to the model itself, symbolic approaches aimed to build up a knowledge base and use decision rules to replicate the ways in which humans would approach a problem. This was often codified as a sequence of logic ‘what-if’ statements that were well suited to LISP and PROLOG.

Symbolic approaches to AI have yet to bear fruit, but the widespread

Machine-learning researchers mostly ignore hardware despite the role it plays in determining which ideas succeed.

and sustained popularity of this research direction for most of the second half of the 20th century cannot be seen as independent of how readily it fit into existing programming and hardware frameworks.

The Persistence of the Hardware Lottery

Today, there is renewed interest in collaboration between the hardware, software, and machine-learning communities. We are experiencing a second pendulum swing back to specialized hardware. Catalysts include changing hardware economics, prompted by both the end of Moore’s law and the breakdown of Dennard scaling; a “bigger is better” race in the number of model parameters;¹ spiraling energy costs;²⁰ and the dizzying requirements of deploying machine learning to edge devices.⁵⁰

The end of Moore’s law means we are not guaranteed more computing power and performance; hardware will have to earn it. To improve efficiency, there is a shift from task-agnostic hardware, such as CPUs, to domain-specialized hardware that tailors the design to make certain tasks more efficient. The first examples of domain-specialized hardware released over the last few years—tensor processing units (TPUs),²³ edge-TPUs,¹⁶ and Arm Cortex-M55²—optimize explicitly for costly operations common to deep neural networks, such as matrix multiplies.

In many ways, hardware is catching up to the present state of machine-learning research. Hardware is only economically viable if the lifetime of the use case is longer than three years.¹¹ Betting on ideas that have longevity is a key consideration for hardware developers. Thus, co-design efforts have focused almost entirely on optimizing an older generation of models with known commercial use cases. For example, ‘matrix multiplies’ are a safe target to optimize because they are here to stay—anchored by the widespread use and adoption of deep neural networks in production systems. Allowing for unstructured sparsity and weight-specific quantization is also a safe strategy because there is wide consensus that these will enable higher compression levels.

There is still the separate question of whether hardware innovation is versatile enough to unlock or keep pace with entirely new machine-learning research directions. This question is difficult to answer; the data points here are limited, making it hard to model whether this idea would succeed given different hardware. However, despite this task's inherent challenge, there is already compelling evidence that domain-specialized hardware makes it more costly for research ideas that stray outside of the mainstream to succeed.

The authors of a 2019 published paper titled, "Machine Learning Is Stuck in a Rut"³ consider the difficulty of training a new type of computer vision architecture called capsule networks⁴² on domain-specialized hardware. Capsule networks include novel components, such as squashing operations and routing by agreement. These architecture choices aim to solve for key deficiencies in convolutional neural networks (lack of rotational invariance and spatial hierarchy understanding) but stray from the typical architecture of neural networks. As a result, while capsule network operations can be implemented reasonably well on CPUs, performance falls off a cliff on accelerators such as GPUs and TPUs, which have been overly optimized for matrix multiplies.

Whether or not you agree that capsule networks are the future of computer vision, the authors say something interesting about the difficulty of trying to train a new type of image-classification architecture on domain-specialized hardware. Hardware design has prioritized delivering on commercial use cases, while built-in flexibility to accommodate the next generation of research ideas remains a distant secondary consideration.

While hardware specialization makes deep neural networks more efficient, it also makes it far more costly to stray from accepted building blocks. It prompts the questions: how much will researchers implicitly be overfit to ideas that operationalize well on available hardware, rather than take a risk on ideas that are not currently feasible? What are the failures we still don't have the hardware to see as a success?

The Likelihood of Future Hardware Lotteries

It is an ongoing, open debate within the machine-learning community about how much future algorithms will differ from models such as deep neural networks. The risk you attach to depending on domain-specialized hardware is tied to your position on this debate. Betting heavily on specialized hardware makes sense if you think that future breakthroughs depend on pairing deep neural networks with ever-increasing amounts of data and computation.

Several major research labs are making this bet, engaging in a "bigger is better" race in the number of model parameters and collecting ever-more-expansive datasets. However, it is unclear whether this is sustainable. An algorithm's scalability is often thought of as the performance gradient relative to the available resources. Given more resources, how does performance increase?

For many subfields, we are now in a regime where the rate of return for additional parameters is decreasing.⁴⁶ The cost of throwing additional parameters at a problem is becoming painfully obvious. Perhaps more troubling is how far away we are from the type of intelligence humans demonstrate. Despite their complexity, human brains remain extremely energy efficient. While deep neural networks

may be scalable, it may be prohibitively expensive to do so in a regime of comparable intelligence to humans. An apt metaphor is that we appear to be trying to build a ladder to the moon.

Biological examples of intelligence differ from deep neural networks in enough ways to suggest it is a risky bet to say that deep neural networks are the only way forward. While general-purpose algorithms such as deep neural networks rely on global updates to learn a useful representation, our brains do not. Our own intelligence relies on decentralized local updates which surface a global signal in ways that are not well understood.⁵

In addition, our brains can learn efficient representations from far fewer labeled examples than deep neural networks (Figure 2). Humans have highly optimized and specific pathways developed in our biological hardware for different tasks.⁴⁹ This suggests that the way a network is organized, and our inductive biases, are as important as the network's overall size.¹⁸

For example, it is easy for a human to walk and talk at the same time. However, it is far more cognitively taxing to attempt to read and talk.⁴⁴ Our brains are able to fine-tune and retain human skills across our lifetimes.⁴ In contrast, deep neural networks that are trained upon new data often evidence catastrophic forgetting, where performance deteriorates on the original task because the new information interferes with previously learned behavior.³⁰

There are several highly inefficient assumptions about how we train models. For example, during typical training, the entire model is activated for every example, leading to a quadratic explosion in training costs. In contrast, the brain does not perform a full forward and backward pass for all inputs; it simulates what inputs are expected against incoming sensory data. What we see is largely virtual reality computed from memory.⁶

The point of these examples is not to convince you that deep neural networks are not the way forward, but rather that there are clearly other models of intelligence, which suggests it may not be the only way. It is possible that the next breakthrough will require a fundamentally different way of

Figure 2. Our own cognitive intelligence is inextricably both hardware and algorithm. We do not inhabit multiple brains over our lifetime.

modeling the world with a different combination of hardware, software, and algorithm. We may very well be amid a present-day hardware lottery.

The Way Forward

Scientific progress occurs when there is a confluence of factors that allows scientists to overcome the “stickiness” of the existing paradigm. The speed at which paradigm shifts have happened in AI research have been disproportionately determined by the degree of alignment between hardware, software, and algorithm. Thus, any attempt to avoid hardware lotteries must be concerned with making the exploration of different hardware/software/algorithm combinations cheaper and less time-consuming.

This is easier said than done. Expanding the search space of possible hardware/software/algorithm combinations is a formidable goal. It is expensive to explore new types of hardware, both in terms of time and capital required. Producing a next-generation chip typically costs \$30–\$80 million and takes two to three years to develop.¹⁴ The fixed costs alone of building a manufacturing plant are enormous, estimated at \$7 billion in 2017.⁴⁵

Experiments using reinforcement learning to optimize chip placement (Figure 3) may help decrease costs.³¹ There is also renewed interest in reconfigurable hardware, such as field-programmable gate array (FPGAs)¹⁷ and coarse-grained reconfigurable arrays (CGRAs).³⁷ These devices allow chip logic to be reconfigured to avoid being locked into a single use case. However, the tradeoff for flexibility is much higher FLOPS and the need for tailored software development. Coding even simple algorithms on FPGAs remains very painful and time-consuming.⁴¹

Hardware development in the short- to medium-term is likely to remain expensive and prolonged. The cost of producing hardware is important because it determines the amount of risk and experimentation hardware developers are willing to tolerate. Investment in hardware tailored to deep neural networks is assured because neural networks are a cornerstone of enough commercial use cases. The widespread profitabil-

Figure 3. Hardware design remains risk averse due to the large amount of capital and time required to fabricate each new generation of hardware.

ity of downstream uses of deep learning has spurred a healthy ecosystem of hardware startups aiming to further accelerate deep neural networks and has encouraged large companies to develop custom hardware in-house.

The bottleneck will continue to be funding hardware for use cases that are not immediately commercially viable. These riskier directions include biological hardware, analog hardware with in-memory computation, neuromorphic computing, optical computing, and quantum computing-based approaches. There are also high-risk efforts to explore the development of transistors using new materials.

An interim goal is to provide better feedback loops to researchers about how our algorithms interact with the hardware we do have. Machine-learning researchers do not spend much time talking about how hardware chooses which ideas succeed and which fail. This is primarily because it is hard to quantify the cost of being concerned. At present, there are no easy-and-cheap-to-use interfaces to benchmark algorithm performance against multiple types of hardware at once. There are frustrating differences in the subset of software operations supported on different types of hardware, which prevents the portability of algorithms across hardware types.²¹ Software kernels are often overly optimized for a specific type of hardware, leading to huge lags in efficiency when used with different hardware.

These challenges are compounded by an ever-more-formidable and heterogeneous landscape of hardware.³⁸ As the hardware landscape becomes

increasingly fragmented and specialized, writing fast and efficient code will require more niche and specialized skills.²⁸ This means that there will be increasingly uneven gains from progress in computer science research. While some types of hardware will benefit from a healthy software ecosystem, progress on other languages will be sporadic and often stymied by a lack of critical end users.⁴⁵

One way to mitigate this need for specialized software expertise is through the development of domain-specific languages that focus on a narrow domain. While you give up expressive power, domain-specific languages permit greater portability across different types of hardware. They allow developers to focus on the intent of the code without worrying about implementation details.³⁵ Another promising direction is automatically autotuning the algorithmic parameters of a program based upon the downstream choice of hardware. This facilitates easier deployment by tailoring the program to achieve good performance and load balancing on a variety of hardware.¹³

In parallel, we need better profiling tools to empower researchers with a more-informed opinion about how hardware and software should evolve. Ideally, software should surface recommendations about what type of hardware to use given the configuration of an algorithm. Registering what differs from our expectations remains a key catalyst in driving new scientific discoveries. Software needs to do more work, but it is also well positioned to do so. We have neglected efficient software throughout the era of Moore’s Law, trusting that predictable gains in computing performance would compensate for inefficiencies in the software stack. This means there are many low-hanging fruit as we begin to optimize for more efficient software.²⁶

Conclusion

George Gilder, an American investor, powerfully described the computer chip as inscribing worlds on grains of sand. The performance of an algorithm is fundamentally intertwined with the hardware and software it runs on. This article proposes the

term ‘hardware lottery’ to describe how these downstream choices determine whether a research idea succeeds or fails.

Today the hardware landscape is increasingly heterogeneous. This article posits that the hardware lottery has not gone away, and the gap between the winners and losers will grow. To avoid future hardware lotteries, we need to make it easier to quantify the opportunity cost of settling for the hardware and software we have.

Acknowledgments

Thank you to many of my wonderful colleagues and peers who took time to provide valuable feedback on earlier versions of this contributed article. In particular, I would like to acknowledge the valuable input of Utku Evci, Erich Elsen, Melissa Fabros, Amanda Su, Simon Kornblith, Aaron Courville, Hugo Larochelle, Cliff Young, Eric Jang, Sean McPherson, Jonathan Frankle, Carles Gelada, David Ha, Brian Spiering, Stephanie Sher, Jonathan Binas, Pete Warden, Sean Mcpherson, Lara Florescu, Jacques Pienaar, Chip Huyen, Razi Alvaréz, Dan Hurt, and Kevin Swersky. Thanks for the institutional support and encouragement of Natacha Mainville and Alexander Popper.

References

1. Amodei, D., Hernandez, D., Sastry, G., Clark, J., Brockman, G., and Sutskever, I. AI and compute. *OpenAI* (2018), <https://openai.com/blog/ai-and-compute/>.
2. ARM. Enhancing AI performance for IoT endpoint devices. (2020), <https://www.arm.com/company/news/2020/02/new-ai-technology-from-arm>
3. Barham, P. and Isard, M. Machine learning systems are stuck in a rut. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS '19)*, (Bertinoro, Italy), ACM, New York, NY, USA, 177–183. <https://doi.org/10.1145/3317550.3321441>
4. Barnett, S. and Ceci, S. When and where do we apply what we learn? A taxonomy for far transfer. *Psychological Bulletin* 128, 4 (2002), 612–37.
5. Bi, G. and Poo, M. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience* 18, 24 (1998), 10464–10472. <https://doi.org/10.1523/JNEUROSCI.18-24-10464.1998> arXiv:https://www.jneurosci.org/content/18/24/10464.full.pdf
6. Bubic, A., Cramon, D., and Schubotz, R. Prediction, cognition, and the brain. *Frontiers in Human Neuroscience* 4 (2010), 25. <https://doi.org/10.3389/fnhum.2010.00025>
7. Chellapilla, K., Puri, S., and Simard, P. High performance convolutional neural networks for document processing. *Tenth International Workshop on Frontiers in Handwriting Recognition* (2006).
8. Coates, A., Huval, B., Wang, T., Wu, D., Catanzaro, B., and Andrew, N. Deep learning with COTS HPC systems. In *Proceedings of the 30th Intern. Conf. on Machine Learning* (2013), Sanjoy Dasgupta and David McAllester (Eds.), PMLR, Atlanta, GA, USA, 1337–1345. <http://proceedings.mlr.press/v28/coates13.html>
9. Collier, B. Little engines that could've: The calculating

- machines of Charles Babbage. *Garland Publishing, Inc.* (1991) USA.
10. Computer history 1949-1960: Early vacuum tube computers overview. *Computer History Archives Project* (2018), https://www.youtube.com/watch?v=WnNm_uJYWhA
11. Dean, J. The deep learning revolution and its implications for computer architecture and chip design. *IEEE International Solid-State Circuits Conference* (2020), 8–14.
12. Dennard, R., Gaensslen, F., Yu, H., Rideout, V., Bassous, E., and LeBlanc, A. Design of ion implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits* 9, 5 (1974), 256–268.
13. Dongarra, J., Gates, M., Kurzak, J., Luszczek, P., and Tsai, Y. Autotuning numerical dense linear algebra for batched computation with GPU hardware accelerators. In *Proceedings of the IEEE* 106, 11 (2018), 2040–2055.
14. Feldman, M. The era of general-purpose computers is ending. *The Next Platform* (2019), <https://bit.ly/3hP8XJh>
15. Fukushima, K. and Miyake, S. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition* 15, 6 (1982), 455–469. <http://www.sciencedirect.com/science/article/pii/0031320382900243>
16. Gupta, S. and Tan, M. EfficientNet-EdgeTPU: Creating accelerator-optimized neural networks with AutoML. *Google AI Blog* (2019), <https://ai.googleblog.com/2019/08/efficientnet-edgegpu-creating.html>
17. Hauck, S. and DeHon, A. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. (2017), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
18. Herculano-Houzel, S., et al. The elephant brain in numbers. *Frontiers in Neuroanatomy* 8 (2014).
19. Hinton, G. and Anderson, J. *Parallel Models of Associative Memory*. (1989), L. Erlbaum Associates Inc., USA.
20. Horowitz, M. Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 10–14.
21. Hotel, H., Johansen, H., Bernholdt, D., Héroux, M., and Hornung, R. Software productivity for extreme-scale science. *U.S. Department of Energy Advanced Scientific Computing Research* (2014).
22. Isaacson, W. Grace Hopper, computing pioneer. *The Harvard Gazette* (2014). <https://news.harvard.edu/gazette/story/2014/12/grace-hopper-computing-pioneer/>
23. Joupri, N., et al. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News* 45, 2 (June 2017), 1–12. <https://doi.org/10.1145/3140659.3080246>
24. Kuhn, T. *The Structure of Scientific Revolutions*. (1962), University of Chicago Press, Chicago.
25. Kurzweil, R. *The Age of Intelligent Machines*. (1990), MIT Press, Cambridge, MA, USA.
26. Larus, J. Spending Moore's dividend. *Commun. ACM* 52, 5 (May 2009), 62–69. <https://doi.org/10.1145/1506409.1506425>
27. LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. Backpropagation applied to handwritten zip code recognition. *Neural Computation* 1, 4 (1989), 541–551. <https://doi.org/10.1162/neco.1989.1.4.541>
28. Lee, H., Brown, K., Sujeeth, A., Chafi, H., Rompf, T., Odersky, M., and Olukotun, K. Implementing domain-specific languages for heterogeneous parallel computing. *IEEE Micro* 31, 5 (2011), 42–53.
29. Linnainmaa, S. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics* 16 (1976), 146–160.
30. McClelland, J., McNaughton, B., and O'Reilly, R. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review* 102 (Aug. 1995), 419–57. <https://doi.org/10.1037/0033-295X.102.3.419>
31. Mirhoseini, A., et al. A graph placement methodology for fast chip design. *Nature* 594 (June 9, 2021), 207–212. <https://www.nature.com/articles/s41586-021-03544-w>
32. Moore, D. The Anna Karenina Principle applied to ecological risk assessments of multiple stressors. *Human and Ecological Risk Assessment: An International Journal* 7, 2 (2001), 231–237. <https://doi.org/10.1080/20018091094349>

33. Moore, G. 1965. Cramming more components onto integrated circuits. *Electronics* 38, 8 (April 1965). <https://www.cs.utexas.edu/~fussell/courses/cs352h/papers/moore.pdf>
34. Moravec, H. When will computer hardware match the human brain. *Journal of Transhumanism* 1 (1998).
35. Olukotun, K. Beyond parallel programming with domain specific languages. *SIGPLAN Not.* 49, 8 (Feb. 2014), 179–180. <https://doi.org/10.1145/2692916.2557966>
36. Possett, E.A. *The Jacquard Machine Analyzed and Explained: The Preparation of Jacquard Cards and Practical Hints to Learners of Jacquard Designing*. (1888).
37. Prabhakar, R., et al. Plasticine: A reconfigurable architecture for parallel patterns. In *2017 ACM/IEEE 44th Annual Intern. Symposium on Computer Architecture (ISCA)*. 389–402.
38. Reddi, V., et al. MLPerf inference benchmark. In *2020 ACM/IEEE 47th Annual Intern. Symposium on Computer Architecture* (2020), 446–459.
39. Rumelhart, D., Hinton, G., and Williams, R. Learning representations by back-propagating errors. *MIT Press* (1988), 696–699.
40. Sabour, S., Frost, N., and Hinton, G. Dynamic routing between capsules. (2017), 3856–3866. <http://papers.nips.cc/paper/6975-dynamic-routing-between-capsules.pdf>
41. Shalf, J. The future of computing beyond Moore's law. *Philosophical Transactions of the Royal Society A*, 378 (2020).
42. Singh, V., Perdigones, A., Garcia, J., Cañas, I., and Mazarrón, F. Analyzing worldwide research in hardware architecture, 1997–2011. *Commun. ACM* 58 (January 2015), 76–85. <https://doi.org/10.1145/2688498.2688499>
43. Steinbuch, K. and Piske, U.A.W. Learning matrices and their applications. *IEEE Transactions on Electronic Computers EC-12*, 6 (1963), 846–862.
44. Stroop, J. Studies of interference in serial verbal reactions. *J. of Experimental Psychology* 18, 6 (1935), 643. <https://doi.org/10.1037/h0054651>
45. Thompson, N. and Spanuth, S. The decline of computers as a general purpose technology: Why deep learning and the end of Moore's Law are fragmenting computing. (November 2018).
46. Thompson, N., Greenewald, K., Lee, K., and Manso, G. The computational limits of deep learning. *arXiv e-prints*, Article arXiv:2007.05558 (July 2020), arXiv:2007.05558 pages. arXiv:2007.05558 [cs.LG]
47. Tolstoy, L. and Bartlett, R. *Anna Karenina*. Oxford University Press (2016), <https://books.google.com/books?id=1DooDwAAQBAJ>
48. Van Der Malsburg, C. Frank Rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. *Brain Theory* (1986), 245–248.
49. Von Neumann, J., Churchland, P.M., and Churchland, P.S. *The Computer and the Brain*. Yale University Press (2000), <https://books.google.com/books?id=Q30MqJRV1gC>
50. Warden, P. and Situnayake, D. *TinyML: Machine Learning With TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. (2019), O'Reilly Media, Inc. <https://books.google.com/books?id=Sb3mxQEACAAJ>

Sara Hooker (shooker@google.com) is a research scholar at Google Brain, Mountain View, CA, USA, with a focus on deep learning.

This work is licensed under a Creative Commons Attribution-NonCommercial International 4.0 License.

Watch the author discuss this work in the exclusive Communications video. <https://cacm.acm.org/videos/the-hardware-lottery>