Survey of Progress in Deep Neural Networks for Resource-Constrained Applications

Morgan Stuart, Milos Manic
Department of Computer Science
Virginia Commonwealth University
Richmond, Virginia
stuartms@vcu.edu, mmanic@ieee.org

Abstract—**Artificial neural networks and deep learning methodologies have had growing interest across industry domains, including IoT and mobile systems. However, in low-power applications, resource limitations and operating environment restrictions make implementations difficult. This survey examines efforts that target the data and compute challenges of implementing energy efficient, low cost, and accurate neural network models. Approaches come in many forms, with solutions ranging from software optimization to hardware reorganization. We examine three avenues of approach - binary neural networks, application specific circuit designs, and neuromorphic computing. For each methodology, we summarize progress, use-cases, and inherent challenges.**

I. Introduction

Deep learning [1] has brought promising advances in artifical neural networks, renewing considerable interest in the field. Models built with deep learning techniques have demonstrated state-of-the art performance in classification of images [2] [3], videos [4] [5], and text [6] [7]. Other areas of work include generative models [8], reinforcement algorithms [9], and neural turing machines [10].

Growth in deep learning has coincided with heightened interest in mobile and the Internet-of-Things (IoT) [11]. These and other resource restricted domains have been pursuing intelligent algorithms and controls for several decades [12] [13]. It follows that efforts in robotics [14] and control systems [15], low-power consumer devices [16], and sensor networks [17] stand to benefit from deep learning models being applied to their applications.

However, resource limited applications generally demand low-power and low-latency operation to be viable. Instead, current state of the art deep learning models can require billions of floating-point operations to perform inference on a single batch of inputs. Training procedures in deep learning are more burdensome and can require many server-class machines, often equipped with Graphics Processing Units (GPU), in order to train within days or weeks [18] [19] [20]. While modern GPUs offer high throughput for the operations needed by neural networks, GPUs consume considerable power and generally have higher response latency.

In order to bring deep learning to more domains, efforts have sought to decouple the specialized operations of neural network systems from their high-cost host systems. This survey examines three gradations of these efforts: application specific integrated circuits (ASIC), binarized neural networks, and neuromorphic architectures. These approaches vary in whether they target software, hardware, or both, but each serve to greatly reduce the cost of utilizing neural network models in resource-constrained environments. Looking forward, near-future computer organizations are likely to experiment with hybrid approaches, combining biologically inspired organizations with more structured and familiar digital designs. As efforts zero-in on these new computing paradigms, intelligent devices and systems will become a mainstay in everyday life.

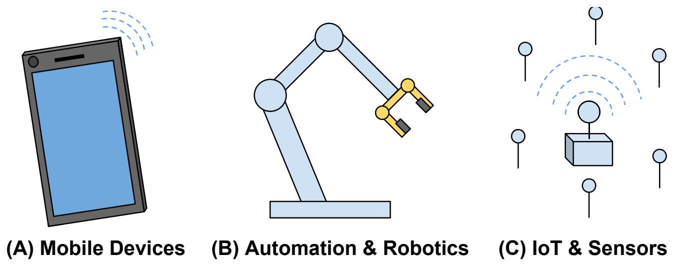

Fig. 1. Example applications for deep learning models in low power and restrictive environments. **(A)** AI-assisted functionality in mobile systems. **(B)** Improved task automation for industrial systems and robotics. **(C)** Intelligent IoT and wireless sensor networks (WSN) reporting on their local environment.

The remainder of this work is organized as follows. Section II will provide a refresher on deep neural learning models. Sections IV, III, and V summarize the major motivations, considerations, and approaches within each method. Then in Section VI, these techniques are compared and discussed within the problem context.

II. Overview of Neural Networks

This section outlines fundamental neural network and deep learning methodologies used today by researchers and practitioners. In later sections, these core concepts will arise again in the consideration of low-cost techniques.

Artificial neural networks arose from efforts to mimic biological neural systems: each artificial neuron is said to be connected to many other neurons via synapses, with each neuron *firing* (producing output) based on the inputs received from other neurons. Efforts in deep learning still utilizes these ideas, but has further abstracted the artificial neural network model type into *layers* of non-linear transformations.

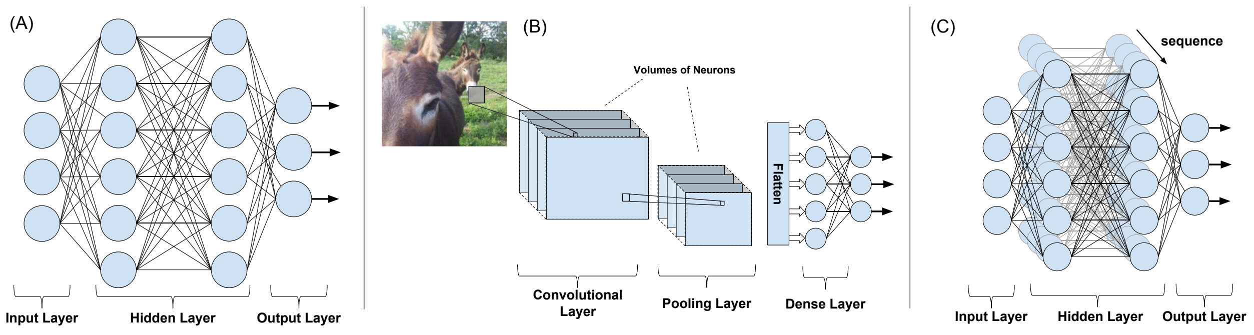

Fig. 2. Popular artificial neural network architectures: (A) Feedforward dense networks, (B) Convolutional Neural Networks, (C) Recurrent Neural Networks

Each layer accomplishes its transformation through *units*, a generalization of the artificial neuron.

A. Model Organization

Artificial neural network models can be separated into several categories, but for this work we consider the three most dominant organizations. These are illustrated in Figure 2 as *feedforward*, *convolutional*, and *recurrent* architectures.

A ***feedforward*** neural network is a type of artificial neural network that cascades values through layers of units. If values are transferred backwards through layers or across samples, the network is considered a ***recurrent*** neural network. Illustrations of feedforward and recurrent networks are provided in Figure 2a and 2c, respectively. [21] [22]

A ***convolutional*** neural network is a distinct organization of units, such that the layers implement n-dimensional filters. These convolutional layers are in contrast to *dense* layers that fully connect all units between layers. Given the properties of digital filtering, convolutional neural networks work well with highly correlated data and require fewer parameters than a fully-connected counter-part. When trained on image classification tasks, the resulting filters illustrate a hierarchy of 2-dimensional convolutional filters. Hidden layers deeper in the model, closer to the output, build more abstract filters capable of matching complex relationships. [23] [21]

B. Formulations

Neural networks can be defined as a method of approximating some function through a *training* procedure. In generalized terms, a network with parameters θ, input features x, and target variable y is described by the mapping

$$y = \mathbf{f}(x; \theta)$$

A traditional artificial neuron performs a weighted sum of its inputs and passes this value through an *activation* function. Formulation of a neuron receiving i features is given by:

$$Z = \sum_i x_i \times w_i = \boldsymbol{W}^T \boldsymbol{X} \tag{1}$$

$$y = \phi(Z) \tag{2}$$

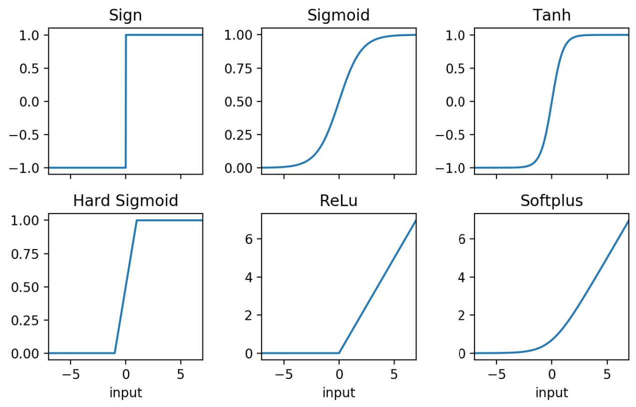

Fig. 3. Activation functions for neural networks. After weights and other operations are applied to a units inputs, the results are passed through the unit's activation function. Nonlinearity and differentiability are important characteristics of activation functions. However, more complex activation require more compute power, potentially limiting applications.

Where $\boldsymbol{X}$ is a vector of input features, $\boldsymbol{W}$ is a vector of weights or parameters, and ϕ is the neuron's activation function. This style of neuron is still at the center of most deep architectures' units, but these units often integrate other regularization or transformation operations.

Without a non-linear activation function, a neural network becomes a linear model, reducing model capacity [21]. A wide range of activation functions have been used historically, see Figure 3 for a sample of the more prominent nonlinearities used for activation. In biological systems, neurons *spike* over time. Within this context, artificial activation functions in Figure 3 represent the average response of the neuron over time with respect to a sample. The choice of activation function can significantly impact both model accuracy and training performance. Deep learning has pioneered the use of a rectified-linear activation (ReLu), which helps to prevent saturation while also being fast to compute. [24] [21]

Training procedures attempt to optimize the network's parameters for the mapping $x \rightarrow y$ with a low error. Training is typically performed through a *cost* function and *gradient descent*. The cost function represents the error in the network's output relative to the actual output. Minimas in the cost surface are pursued by calculating the error gradient and adjusting parameters to descend the cost surface. A key challenge with gradient descent applied to complex models like neural networks is determining the error gradient. *Error back-propagation* [25] is still widely used as a method to unroll the contribution of the error onto each neuron's parameters.

This section outlined the basics of current deep learning techniques in artificial neural networks. These methods are almost exclusively applied on standard desktop or server class systems, which may not be well-suited for many low-cost environments. The remainder of this paper will explore how the basic biological model of neural networks, made powerful with these techniques, are entering lower cost domains.

III. Application Specific Integrated Circuits

The generalization power of deep neural networks has renewed efforts in application specific integrated circuit (ASIC) designs for neural networks, since a single neural network chip can potentially support many separate applications. ASIC-based neural computing approaches reorganize traditional functional units in order to more directly target the needs of deep learning models. Implementations are typically realized as separate devices for use by a larger host machine.

A. Progress in ASICs

ASIC implementations are especially interesting since they've been evaluated in production commercial environments running deep learning algorithms. Designed and deployed in only 15 months, Google's *tensor processing unit* [26] achieves 14x-16x and 17x-34x improvement in performance/watt in comparison to GPU and CPU versions, respectively. Microsoft's Catapult project [27] previously leveraged the flexability of FPGAs in their datacenter to achieve 2x performance improvement in their search ranking system. Recent publications demonstrate the organization's push for FPGA-accelerated deep learning in hyperscale datacenters [28] [29].

Summary of ASICs:

- Large-scale systems already in production [26][27]
- High complexity and development effort
- High performance-per-watt

B. Considerations & Techniques

Deep learning's success at building generalized models has encouraged highly specialized designs, intended to target the specific needs of neural network execution. Still, high flexibility is desirable, both from a cost and end-user perspective.

1) ***Systolic Arrays****:* The overhead of general-purpose processors and memory systems have limited utility in efficiently executing neural network models. For this reason, efforts have instead focused on specialized organizations that are more readily amenable to the characteristics of neural network models.

Illustrated in Figure 4, systolic arrays enable architectures better suited for neural systems through use of tightly coupled sequences of homogeneous processing elements (PE) pipelined together. Each element performs a limited set of instructions, such as multiplication and addition. The goal of a PE is to operate as part of a sequence of PEs. The sequence operates synchronously, pumping results, partial results, and inputs through the architecture. This pneumatic action is what earned these systems the name systolic, likening them to the rhythmic action of the biological heart [30].

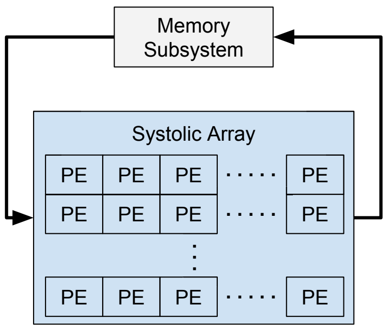

Fig. 4. Organization and scaling of a 2D systolic array system. Processing Elements (PE) perform simple operations before passing results to adjacent PEs. The organization reduces the need for memory access and improves the systems throughput.

The hierarchical nature of neural networks makes systolic arrays a fitting design choice for ASIC implementations [31][32][33]. Neural computing with systolic arrays generally focus on inference applications rather than training. The guiding principle being that offline training suffices in the majority of cases, and can be performed on larger and more flexible machines. By offloading the training phase, the hardware design can be simplified, reducing costs for the system by focusing only on inference [33].

2) ***Dynamic Architectures using FPGAs****:* Exploring new hardware architectures for neural systems can be time consuming. Furthermore, while many deep learning models have demonstrated impressive reusuability, some platforms and applications may still need to support a variety of different models, each with differing hardware requirements.

Field programmable gate arrays (FPGA) allow for software-defined hardware architectures, providing a route for compromise between the flexability of general-purpose processors and the performance of ASIC hardware implementations. The primary advantage being the ability to rapidly iterate on organizational schemes that better target neural computing.

In addition to rapid iteration of designs, FPGAs can also enable on-the-fly model reorganization in order to better suit the problem at hand. This can be extended to a system that automatically rearranges itself, adapting to a changing environment or problem. However, this is still hampered by long synthesis times, which can take minutes or even hours to complete depending on the system's complexity [34] [35].

While FPGA-based neural networks have undergone decades of sporadic research, only recently have implementations shown valuable improvements in performance/watt. Microsoft's acceleration of deep learning using FPGAs achieved 134 images/second classification speed at only 25 watts. Comparatively, GPU implementations can accomplish well over 500 images/second, but with over 200 watts of required power. The continual advancement of FPGA technology will see this performance improve even further [28] [35].

IV. Binarized Neural Networks for Embedded Systems

The high cost of floating point operations encouraged parameter sparsity and reduced precision arithmetic in deep learning. More extreme approaches build neural systems that operate entirely on binary operations, such as XOR, shifts, and bit counting. Networks implemented in this way greatly reduce memory footprint and computational cost, enabling low-powered implementations.

A. Progress in Binary Neural Networks

Efforts towards binary neural networks in relation to deep learning methodologies are relatively recent. Techniques focus on using traditional training methodologies on large scale machines, with restrictions imposed such that the resulting model can be represented and executed as a binary network atop low-power systems.

BinaryConnect [36] presents deep learning models that propgate data through binary weights, but still require real-valued inputs and temporaries. The authors followup work expanded this effort with BinaryNets [37] by binarizing the network's activations. Additional efforts have since sought to binarize both weights and inputs by using bitwise approximations of convolutional arithmetic [38]. These efforts have shown promise, achieving varying levels of comparable accuracy to their 32-bit precision counter-parts, but no low-cost/embedded implementation efforts have been widely published at this time.

Summary of binary neural networks:

- Reduce floating point dependency for embedded systems
- Improves performance and capacity for large machines
- No published experimentation on embedded hardware

B. Considerations & Techniques

Reducing the precision of parameters and training operations has the potential to negatively impact accuracy. Further precision reduction to binary networks requires reconsideration of core operations such as a model's activation function.

1) ***Reduced Precision Models****:* Precise floating-point operations are expensive and are often avoided when implementing algorithms for low-power systems. Large scale systems can also improve performance by reducing the precision of their arithmetic arithmetic, if the application allows for the error.

Neural network algorithms heavily rely on precision arithmetic in both training and inference phases of execution. However, experiments have demonstrated that low precision multipliers, such as half precision float, are sufficient for training deep networks for image classification [39][40]. In some cases, reduced precision and binary networks have been shown to help regularize a neural model, improving generalizeability [36]. Binary networks can be considered an extreme version of reduced precision, but specialized bit-wise operations make them distinct.

2) ***Forward-Pass Activation Binarization****:* A primary consideration when approaching binarization is how to translate a real-valued activation function into a binary space. Most popular activation functions such as *sigmoid*, *tanh*, or *ReLu* are designed for real valued inputs and outputs - binary activation must use different techniques to determine their output.

Creating a binary network requires binarization of the activation outputs by restricting outputs to +1 and -1. A *hard threshold* or *sign* function can serve this purpose. Furthermore, a stochastic methodology can instead map real value inputs to a probability σ, which is then used to draw a random value with probability $p = \sigma(x)$. The authors in [37] found success with the sign function due to its simple implementation and reduced overhead.

3) ***Optimization Binarization****:* Error backpropagation can't readily translate to the discontinuous gradient of a discrete activation implicit in a binary activation function. In order determine appropriate weighting schemes, the model most be made to work with error backpropagation or another training scheme must be used.

Approaches generally opt to compute the gradients of a binary activation using a straight-through estimator, which equates the gradient to the units discrete output. Then parameter gradients are accumulated into real valued temporaries during backpropagation, resulting in binary version of the network for forward pass inference [41][36][37]. Other work demonstrated that binary networks were feasible and could be reached by replacing *error backpropagation* with *expectation backpropagation*, a techinque using variational Bayes methods [42].

V. Neuromorphic Computing Architectures

Neuromorphic systems are guided by the capacity and efficiency of biological neuronal systems, and are therefore closely aligned with deep learning systems implemented atop traditional hardware architectures. Neuromorphic designs can be seen as simulations of biological brains, but the recent successes in deep learning has further motivated the pursuit of practical applications for these designs.

A. Progress in Neuromorphic Computing

Neuromorphic computing architectures are more closely aligned with biological brains, with models distributed across a large fabric of primitive *neurons* intercommunicating via *synapses* [43]. New architectures are required since large scale implementations of this concept are difficult with traditional computing hardware and digital representations.

While there are several avenues to pursue, spiking neural networks (SNNs) [44] have seen considerable focus as a key approach [45][46][47][48]. Methodologies have begun to focus on memristor-based designs [49] [50], but all focus on remaining biologically inspired [51]. Simulation results have shown promise with speedups approaching 200x that of a conventional CPU [49]. Other real-world implementations have demonstrated various approahces that offer low power consumption while maintaining accuracy [52] [53] [47] [54] [55] [56]. Online training procedures remain challenging, with many hardware learning rules considered over the years [57] [45] [52]. Offline training is typically used, but a mapping between the neuromorphic architecture and the software neural net must exist for this approach.

Summary of neuromorphic computing:

- Low-power and accurate using event-driven architectures
- Few real-world implementations in operation today
- High complexity and effort with current technology

B. Considerations & Techniques

In pursuit of biologically inspired computing architectures, researchers largely focus on asynchronous communications between neurons. The asynchronous nature removes the need for a digital clock signal. This simplifies in some ways the time-dependent information transfer through pulse trains. These principles are difficult to implement with traditional digital circuitry, leading to exploration of new dynamic circuitry.

*1) **Spiking Neurons**:* Biological brains *spike* changes in potential over time, creating a series of pulses that efficiently transfers information between endpoints [58] [59]. In contrast, digital systems use binary encoded values, represented as a collection of $log_2(X)$ bits. This scheme makes implementing many neurological ideas difficult or error-prone. Therefore, artificial spiking neurons are pursued in place of traditional c*omplementary metal-oxide-semiconductor* (CMOS) digital designs in many neuromorphic approaches.

Spiking neurons leverage univariate encoding to simplify operations. Rather than represent information as a parallel sequence of bits, as in traditional architectures, spiking neurons instead imbue information in the frequency of pulses emitted. With careful consideration to stimuli response and input values, spiking neurons such as Leaky Integrate and Fire (LIF) model can be use to implement complex signal operations such as integration [60]. In using a frequency domain representation of data, spiking neuron systems can more readily receive and operate on input data from sensors and other real-world inputs, since these are often time varying and pulse-oriented themselves [61].

*2) **Memristors and Crossbar Circuits**:* The latest passive electrical component, the *memristor* [62] [63], has aided both in explaining biologically intelligent systems [64], and in implementing them [65][66][71][72]. A memristor refers to a two terminal component with a time-varying resistance. The memristor's resistance value only adjusts if the charge flowing over it changes. Furthermore, the direction of charge flow can be used to increase or decrease resistance. The resistance of the memristor can be measured and used as stored memory, which is maintained once the element is no longer powered. In this way, the memristor can be considered a memory element, holding a single value.

Existing CMOS devices have limitations in footprint size and leakage - smaller devices tend to waste more of the energy they consume. The crossbar circuit has been formulated as possible replacement, but limitations in the switching speed at it's interconnects delayed practical implementations. The advent of memristors, with their small footprint and fast switching speeds has made worthwhile crossbar designs a reality [67] [68]. These traits have made crossbar circuits using hybrid CMOS/memristors circuits [69][70] an important organization for neuromorphic computing efforts [71][72]. Architectures can create distinct training phases and inference phases by exploiting the responsive voltage ranges of the memristor - neutral voltages are used for inference while larger pulses adjust the memristor's stored weight [73].

*3) **Training Procedures**:* Neuromorphic systems are still pursuing online, or *on-chip*, training procedures in order to enable self-reliant, low-power, and intelligent systems. However, the discrete spiking nature of neuromoprhic systems makes backpropagation challenging since it requires a differentiable model in order to calculate error contributions.

Recent work addresses this challenge by restricting the training procedure of a real-valued deep neural network in software, such that the resulting model is more readily mapped to neuromorphic SNNs. Primary challenges include representing negative weights without adding inhibitory neurons, representing layer biases, and implementing spatial pooling operations without additional layers. Approaches include adopting ReLu activations to avoid negative values, zeroing biases, and substituting spatial linear subsampling in place of pooling [46]. Still, converting networks trained in this way showed some performance loss, but this can be ragained through weight-normalization techniques [45].

VI. Discussion

Deep learning methodologies are poised to tackle new challenges outside the datacenter, but first, new levels of energy and compute efficiency must be reached. In pursuit of this, neural network techniques are being adapted both through software and hardware reconsiderations. Software-based implementations are altering their idealized methods to better suit the underlying compute infrastructure, and hardware designs are being rethought to better support existing algorithms or to realize larger scale and complexity.

Enabling complex models in low-cost environments stands to bring a wide range of benefits. Foremost is offline autonomy for mobile and IoT devices. Today, solutions would likely require remote connectivity back to a datacenter or base station in order to receive model outputs from larger machines. Decoupling these devices reduces network load and latency for the application. This is especially important as these devices are often directly interacting with their environment or an end-user - the type of application where rapid response is key.

Low-power versions of these complex models will also encourage experimentation in domains such as compression and data transformations. For instance, a sensor may use a neuromorphic chip to classify incoming samples. When an interesting or otherwise important sample is discovered, the neuromorphic chip can be reconfigured to encode the sample to a lower dimensionality. The compressed sample can then be transmitted directly from the neuromorphic chip using the serial pulses of the popular SNN models.

As neural networks continue to develop, they are confronted with the various performance barriers of modern general-purpose computers. Neural network models seem especially challenging since they demand both large amounts of fast memory (both in parameters and input data) as well as complementary compute architectures. These challenges are not new in the field of computer engineering. The road to understanding how to scale machines is paved by the considerations of Amdahl [75], Gustafason [76], and more recent memory bound models [77] [78]. It seems then that massively parallel processing can be highly scalable, but is ultimately determined by algorithm's degree of parallelism and its memory access requirements. Neural networks will continue to be challenged by these properties as researchers and practitioners push for larger scale and complexity.

Each of the approaches discussed still rely heavily on traditional machines and tools to train a model. After training is complete, the model could be adapted to it's final platform, whether it be an ASIC, embedded system, or SSN hardware. This illustrates both how early these techniques are, and also how these methods are focused on the real-world, inference-heavy applications. Put another way, it appears that model execution is currently a higher priority than model training. This is intuitive, since a well-trained model undergoes training once, and can then be used many times for inference.

As software and hardware domains converge to more pervasive neural computing, industries are likely to entertain a series of partial and hybrid measures towards its realization. Techniques such as binary networks will serve to allow organizations to leverage existing embedded and sensory networks to create topologies of intelligent devices. Similar to the rapid adoption of GPUs to offload burdensome graphics processing from the CPU, future enterprise and home computer systems may come with discrete neural network processing units, enabling fast and local artifical intelligence. Specialized hardware seems likely to emerge first in large scale compute providers already oriented around provinding interfaces to artificial intelligence applications [28] [26], but heightened demands may bring this processing closer to the user. Privacy advocates may further demand this transition.

Neuromorphic computing faces an additional challenge - that of composing programs that utilizing a complex fabric of neuron-like processors. Already, work in this area has begun to revolve around hybrid archtectures, creating memristor-CMOS neuromorphic circuits to better enable exploration and practical use [74]. It's reasonable to expect this approach to further yield a hybrid hierarchy. A digital neural processing core can pass outputs to larger neuromorphic fabric to extract deeper features and perform more abstract inferences. A fitting analogy being the current memory and computing performance hierarchy pervasive in today's general-purpose compute systems.

VII. Conclusion

The success of deep learning methodologies has encouraged their entrance into a multitude of problem domains - web services, automotive systems, manufacturing, and personal devices are all seeing benefits from these advances. However, high demands for compute and memory pose a challenge to practitioners. In turn, efforts are growing to reduce costs and to decouple these techniques from the large platforms that bore them. ASIC designs, binary neural networks, and neuromorphic organizations all serve to approach this issue. Practitioners in need of solutions now will continue to utilize standard large machine techniques, but as more efficient methods mature, hybrid architectures will likely emerge. This progression will only serve to further proliferate these algorithms and their adoption into mainstream computing.

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[4] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[5] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[7] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[10] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[11] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[12] FW Lewis, Suresh Jagannathan, and A Yesildirak. *Neural network control of robot manipulators and non-linear systems*. CRC Press, 1998.

[13] Panos J Antsaklis. Neural networks for control systems. *IEEE Transactions on Neural Networks*, 1(2):242–244, 1990.

[14] K. Isa and M. R. Arshad. Neural networks control of hybrid-driven underwater glider. In *2012 Oceans - Yeosu*, pages 1–7, May 2012.

[15] N. L. Lu Wang, S. Li, and K. Li. Neural network based model predictive control performance monitoring-data-driven approach. In *2013 9th Asian Control Conference (ASCC)*, pages 1–6, June 2013.

[16] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[17] L. M. Borges, F. J. Velez, and A. S. Lebres. Survey on the characterization and classification of wireless sensor network applications. *IEEE Communications Surveys Tutorials*, 16(4):1860–1890, Fourthquarter 2014.

[18] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, MarcAurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *NIPS*, 2012.

[19] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with cots hpc systems. In *Proceedings of the 30th international conference on machine learning*, pages 1337–1345, 2013.

[20] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan C. Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595, 2015.

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

[22] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[23] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 253–256. IEEE, 2010.

[24] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[25] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[26] 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.

[27] Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. A reconfigurable fabric for accelerating large-scale datacenter services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 13–24. IEEE, 2014.

[28] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric S Chung. Accelerating deep convolutional neural networks using specialized hardware. *Microsoft Research Whitepaper*, 2(11), 2015.

[29] Eric Chung. Deep learning in the enhanced cloud. In *Proceedings of the 2017 ACM on International Symposium on Physical Design*, ISPD '17, pages 5–5, New York, NY, USA, 2017. ACM.

[30] HT Kung and Charles E Leiserson. Systolic arrays (for vlsi). In *Sparse Matrix Proceedings 1978*, volume 1, pages 256–282. Society for Industrial and Applied Mathematics, 1979.

[31] Jaehyeong Sim, Jun-Seok Park, Minhye Kim, Dongmyung Bae, Yeongjae Choi, and Lee-Sup Kim. 14.6 a 1.42 tops/w deep convolutional neural network recognition processor for intelligent ioe systems. In *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*, pages 264–265. IEEE, 2016.

[32] Yeongjae Choi, Dongmyung Bae, Jaehyeong Sim, Seungkyu Choi, Minhye Kim, and Lee-Sup Kim. Energy-efficient design of processing element for convolutional neural network. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2017.

[33] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 92–104. ACM, 2015.

[34] Clément Farabet, Yann LeCun, Koray Kavukcuoglu, Eugenio Culurciello, Berin Martini, Polina Akselrod, and Selcuk Talay. Large-scale fpga-based convolutional networks. *Scaling up Machine Learning: Parallel and Distributed Approaches*, pages 399–419, 2011.

[35] Griffin Lacey, Graham W Taylor, and Shawki Areibi. Deep learning on fpgas: Past, present, and future. *arXiv preprint arXiv:1602.04283*, 2016.

[36] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *CoRR*, abs/1511.00363, 2015.

[37] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[38] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[39] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Low precision arithmetic for deep learning. *CoRR*, abs/1412.7024, 2014.

[40] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *ICML*, pages 1737–1746, 2015.

[41] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[42] Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems*, pages 963–971, 2014.

[43] Don Monroe. Neuromorphic computing gets ready for the (really) big time. *Commun. ACM*, 57(6):13–15, June 2014.

[44] Wolfgang Maass and Henry Markram. On the computational power of circuits of spiking neurons. *Journal of computer and system sciences*, 69(4):593–616, 2004.

[45] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE, 2015.

[46] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, 2015.

[47] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[48] Giacomo Indiveri. Neuromorphic bistable vlsi synapses with spike-timing-dependent plasticity. In *NIPS*, volume 15, pages 1091–1098, 2002.

[49] Carlos Zamarreño-Ramos, Luis A Camuñas-Mesa, Jose A Pérez-Carrasco, Timothée Masquelier, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex. *Frontiers in neuroscience*, 5, 2011.

[50] Andy Thomas. Memristor-based neural networks. *Journal of Physics D: Applied Physics*, 46(9):093001, 2013.

[51] Shih-Chii Liu and Tobi Delbruck. Neuromorphic sensory systems. *Current opinion in neurobiology*, 20(3):288–295, 2010.

[52] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, pages 1117–1125, 2015.

[53] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V Arthur, Paul A Merolla, and Kwabena Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.

[54] Daniel Neil and Shih-Chii Liu. Minitaur, an event-driven fpga-based spiking network accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(12):2621–2628, 2014.

[55] Eustace Painkras, Luis A Plana, Jim Garside, Steve Temple, Francesco Galluppi, Cameron Patterson, David R Lester, Andrew D Brown, and Steve B Furber. Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits*, 48(8):1943–1953, 2013.

[56] Thomas Pfeil, Andreas Grübl, Sebastian Jeltsch, Eric Müller, Paul Müller, Mihai A Petrovici, Michael Schmuker, Daniel Brüderle, Johannes Schemmel, and Karlheinz Meier. Six networks on a universal neuromorphic computing substrate. *Frontiers in neuroscience*, 7, 2013.

[57] Perry Moerland and Emile Fiesler. Neural network adaptations to hardware implementations. Technical report, IDIAP, 1997.

[58] Rodolfo R Llinás. Central nervous system function. 1988.

[59] Bruce Hutcheon and Yosef Yarom. Resonance, oscillation and the intrinsic frequency preferences of neurons. *Trends in neurosciences*, 23(5):216–222, 2000.

[60] Bryan P Tripp and Chris Eliasmith. Population models of temporal differentiation. *Neural computation*, 22(3):621–659, 2010.

[61] Bilel Belhadj, Antoine Joubert, Zheng Li, Rodolphe Héliot, and Olivier Temam. Continuous real-world inputs can open up alternative accelerator designs. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 1–12, New York, NY, USA, 2013. ACM.

[62] L. Chua. Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519, September 1971.

[63] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *nature*, 453(7191):80–83, 2008.

[64] Frank Zhigang Wang, Leon O Chua, Xiao Yang, Na Helian, Ronald Tetzlaff, Torsten Schmidt, Caroline Li, Jose Manuel Garcia Carrasco, Wanlong Chen, and Dominique Chu. Adaptive neuromorphic architecture (ana). *Neural Networks*, 45:111–116, 2013.

[65] Djaafar Chabi, Weisheng Zhao, Damien Querlioz, and Jacques-Olivier Klein. On-chip universal supervised learning methods for neuro-inspired block of memristive nanodevices. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 11(4):34, 2015.

[66] Ahmad Muqeem Sheri, Hyunsang Hwang, Moongu Jeon, and Byung-geun Lee. Neuromorphic character recognition system with two pcmo memristors as a synapse. *IEEE Transactions on Industrial Electronics*, 61(6):2933–2941, 2014.

[67] Dmitri B Strukov and Konstantin K Likharev. Cmol fpga: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *Nanotechnology*, 16(6):888, 2005.

[68] A. Bhola and G. Kanitkar. Memristors and crossbar latches. In *Proceedings of the International Conference and Workshop on Emerging Trends in Technology*, ICWET '10, pages 915–918, New York, NY, USA, 2010. ACM.

[69] Konstantin K Likharev. Hybrid cmos/nanoelectronic circuits: Opportunities and challenges. *Journal of Nanoelectronics and Optoelectronics*, 3(3):203–230, 2008.

[70] Dmitri B Strukov and R Stanley Williams. Four-dimensional address topology for circuits with stacked multilayer crossbar arrays. *Proceedings of the National Academy of Sciences*, 106(48):20155–20158, 2009.

[71] Konstantin K Likharev. Crossnets: Neuromorphic hybrid cmos/nanoelectronic networks. *Science of Advanced Materials*, 3(3):322–331, 2011.

[72] Mirko Prezioso, Farnood Merrikh-Bayat, BD Hoskins, GC Adam, Konstantin K Likharev, and Dmitri B Strukov. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521(7550):61–64, 2015.

[73] Djaafar Chabi, Damien Querlioz, Weisheng Zhao, and Jacques-Olivier Klein. Robust learning approach for neuro-inspired nanoscale crossbar architecture. *J. Emerg. Technol. Comput. Syst.*, 10(1):5:1–5:20, January 2014.

[74] Giacomo Indiveri, Bernabé Linares-Barranco, Robert Legenstein, George Deligeorgis, and Themistoklis Prodromakis. Integration of nanoscale memristor synapses in neuromorphic computing architectures. *Nanotechnology*, 24(38):384010, 2013.

[75] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

[76] John L Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[77] Xian-He Sun and Lionel M Ni. Scalable problems and memory-bounded speedup. *Journal of parallel and distributed computing*, 19(1):27–37, 1993.

[78] Xian-He Sun and Yong Chen. Reevaluating amdahls law in the multicore era. *Journal of Parallel and Distributed Computing*, 70(2):183–188, 2010.