HARDWARE DESIGN FOR MACHINE LEARNING

Pooja Jawandhiya

School of Electrical and Electronic Engineering,
Nanyang Technological University, Singapore

ABSTRACT

Things like growing volumes and varieties of available data, cheaper and more powerful computational processing, data storage and large-value predictions that can guide better decisions and smart actions in real time without human intervention are playing critical role in this age. All of these require models that can automatically analyse large complex data and deliver quick accurate results – even on a very large scale. Machine learning plays a significant role in developing these models. The applications of machine learning range from speech and object recognition to analysis and prediction of finance markets. Artificial Neural Network is one of the important algorithms of machine learning that is inspired by the structure and functional aspects of the biological neural networks. In this paper, we discuss the purpose, representation and classification methods for developing hardware for machine learning with the main focus on neural networks. This paper also presents the requirements, design issues and optimization techniques for building hardware architecture of neural networks.

KEYWORDS

Artificial intelligence (AI), application specific integrated circuit (ASIC), artificial neural network (ANN), central processing unit (CPU), field programmable gate array (FPGA), graphics processing unit (GPU), machine learning (ML), neurochip

1. INTRODUCTION

From self-driving cars to SIRI, Artificial Intelligence (AI) is progressing rapidly. Science fiction often portrays AI as robots with human characteristics (example, Ava in Ex Machina and Skynet in Terminator) but the truth is that AI can encompass anything. The tech giants are racing to build their own AI software and products. We currently have Google's Tensorflow and AlphaGo, Nvidia's DGX, Amazon's Alexa, Microsoft's Azure, IBM's Watson and Intel's Nervana. A survey by McKinsey & Company showed that the total investments in AI development tripled between 2013 and 2016 [1]. Most of that — $20 billion to $30 billion — came from these tech giants [2]. These companies expect that machine learning (ML), and other AI models that descend from it, will be very critical to their customers in the future, just like networking and mobility. While many ML algorithms have been used for years, the ability to automatically apply complex mathematical calculations to big data – over and over, faster and faster – is a recent development. Online recommendation offers like those from Flipkart and Amazon, real-time advertisements on web pages and mobile devices, web search results are some of the examples of

ML applications that we are familiar with. Few other applications are Optical Character Recognition (OCR) [3], speech recognition [4,5] and pattern classification [6].

Google has become a part and parcel of our daily lives. But we have hardly cared to think on how it works [7]. The millions of images uploaded to the internet are sorted by image recognition which results in accurate classification and in turn give users better search results. One of the breakthroughs of Google in deep learning is in image enhancement. It involves restoring or filling in details missing from images, by extrapolation, as well as by using what it knows about other similar images. Google has implemented AI in language processing too. Google's Assistant speech recognition AI uses deep neural networks (DNN) to learn how to better understand spoken questions and commands. [8] Google's Neural Machine Translation also works in deep learning environment. Another way Google uses deep learning today on its services is to provide more useful recommendations on Youtube^{TM}. Google Brain^{TM} monitors and records our viewing habits as we stream content from their servers. DNNs are made to study and learn everything about viewers' habits and preferences, and work out what would keep them glued to their screens.

Hence, we see that machine learning plays a significant role in the advances of technology today. There have been literature surveys previously done on this subject. The authors in [9] present the opportunities and challenges in designing hardware for machine learning while the study in [10] specifically talks about neural networks. A detailed survey of neural networks in hardware is done in [11] whereas the authors in [12] present a brief survey of FPGA implementation of neural networks. This paper presents the latest review of the hardware architectures for machine learning focussing mainly in the aspects of neural networks.

The rest of the paper is organized as follows. Section 2 talks about the architectural design of the neural networks in both software and hardware keeping in the contrast between them. The different types of hardware for ANN are discussed in Section 3 with detailed explanation of each type. Section 4 finally talks about the hardware architecture in detail including CPU, GPU, FPGA and ASIC whereas section 5 focusses on the various issues in the design architecture and related optimization techniques. Some other approaches and further extension in the ML hardware architecture as advanced technologies are discussed in sections 6 and 7. Section 8 is a case study upon the Google's Tensor Processing Unit (TPU), and the final concluding remarks are presented in section 9.

2. ARCHITECTURE DESIGN

In recent years, there have been massive advances in implementing ML algorithms with application-specific hardware (e.g., FPGA, ASIC, etc.) due to their inherent parallelism. ML algorithms, such as those for specialised applications like image processing, speech synthesis and analysis, face recognition, multi-category classification, and data analysis, are being developed that will fundamentally alter the way individuals and organizations live, work, and interact with each other. Chipmakers are racing to build hardware for AI. Technology giants and governments are investing heavily in neuromorphic chips (prominent examples include the EU's BrainScaleS project, the UK's SpiNNaker brain simulation machine, IBM's "synaptic chips", DARPA's SyNAPSE program, and Brain Corporation, a research company funded by Qualcomm). There is a timely need to map the latest learning algorithms to physical hardware, in order to achieve significant improvements in speed, performance, area and energy efficiency. However, their computational complexity still challenges the state-of-the-art computing platforms, especially when the application of interest is tightly constrained by the requirements of low power, high throughput, less latency, small size, etc.

2.1. Hardware vs Software Implementation of ANNs

Implementation of ANNs falls into two categories: software implementation in conventional computers and hardware implementation, capable of dramatically decreasing execution time [10]. ANNs are implemented in software, and are trained and simulated on general-purpose computers for emulating a wide range of neural networks models. Software implementations offer flexibility and can be used to develop and debug new algorithms. However, hardware implementations are essential for applicability to large networks and for taking the advantage of ANN's inherent parallelism. The main purpose of building dedicated hardware for AI is to provide a platform for efficient adaptive systems, capable of updating their parameters in the course of time. Specific-purpose hardware implementations are dedicated to a particular ANN model. VLSI implementations of ANNs provide compact architecture and high speed in real time applications.

A significant amount of work has been done in developing software and simulation environments for ANNs. Standard implementations of ML algorithms are readily and widely available through libraries/packages/APIs (e.g. scikit-learn [13], Theano [14], Spark MLlib [15], H2O [16] , TensorFlow [17] etc.) but applying them effectively involves choosing a suitable model (decision tree, nearest neighbour, neural net, support vector machine, etc.), a learning procedure for fitting the data (linear regression, gradient descent, genetic algorithms and other model-specific methods), as well as understanding how hyper-parameters affect learning [18]. Specialized applications have motivated the use of hardware in ANN. For example, cheap dedicated devices used for speech recognition in consumer products, and analog neuromorphic devices, such as silicon retinas, directly implement the desired functions [11].

Generally, neural network hardware designs are of two types. The first is - a general, but probably costly, system that can be re-programmed for many kinds of tasks - such as Adaptive Solutions CNAPS [19]. The second is - a specialized, but relatively cheaper, chip that does single task quickly and efficiently, such as IBM ZISC [20].

2.2. Measurement Units

The traditional approach for quantifying ANN hardware performance is to measure the number of MAC operations performed in the unit time, i.e., Millions of Connections Per Second (MCPS) and the rate of weight updates, i.e., Millions of Connection Update Per Second (MCUPS) [10, 11]. These two measurements somewhat correspond to the Million Instructions per second (MIPS) or the Mega Floating-point Operations per Second (MFLOPS) measured on conventional systems. The common speed measurement units of today's computers are GFLOPS (billions of flops) or TFLOPS (trillions of flops) [21].

2.3. Precision and Number Formats

During the hardware implementation of ANNs, two important considerations need to be made. Firstly, there should be balance between the need of reasonable **precision** (number of bits) and the cost of logic area. Secondly, a suitable **number format** should be chosen so that dynamic range is large enough for general-purpose application [11]. So, before beginning the hardware design of ANN model, the number format (floating point, fixed point etc.) and precision to be used for inputs, weights and activation functions must be considered. The precision is mostly limited to 16-bit fixed point for the weights of ANN and 8-bit fixed point for the outputs. In case of the standard error backpropagation and the multi-layer perceptron learning algorithm this precision was shown to be sufficient in most cases [22]. Using 16 bits as precision of calculation instead of 32 bits results in faster computation, because processors tend to have more throughput at lower resolution [23]. Reducing the precision also increases the amount of available bandwidth, because smaller amounts of data is being fetched for each computation. Kohonen's SOM algorithm can learn well with only 6-bit weights [24]. An arithmetical precision of more than 16 bits may be required by recurrent neural networks [25]. However, the precision cannot be reduced too much because then the network will not train and will never achieve the accuracy needed or it will become unstable. It was found that the discretization process degraded the performance of the NN algorithm in [26]. Since precision has great impact in the learning phase, it is important to keep the precision of numbers as high as possible during training phase. However, propagation phase requires the use of low precision.

According to researchers, it is possible to train ANNs with **integer** weights [22]. The advantage of using integer weights is that integer multipliers can be implemented more efficiently on hardware than the floating-point ones. There are some special learning algorithms which use powers-of-two integers as weights [27]. The advantage of powers-of-two integer weight learning algorithms is that the required multiplications in an ANN can be reduced to a series of shift operations. Floating point offers the maximum dynamic range, making it suitable for any application. However, floating-point operations require more cycles for computation than integer operations (unless extremely complex designs are used) [10]. This is why most neurocomputer designers consider fixed-point representations in which only integers are used and the position of the decimal point is handled by some simple additional circuits or software. Appropriate word length must be found for using such representations. The convergence of the learning algorithms should not be affected and enough resolution should be provided during normal operation. The classification capabilities of the trained networks depend on the length of the bit representation. Another method for representation called **Bit-Stream arithmetic** is described in [12].

Deeper networks have improved accuracy but they greatly increase the number of parameters and model sizes. This increases the storage demands and computational memory bandwidth. As such, the trends have shifted towards more efficient DNNs. An emerging trend is the adoption of *compact low precision data types*, much less than 32-bits [28]. 16-bit and 8-bit data types are being used, as they are supported by DNN software frameworks (example, TensorFlow). Furthermore, researchers have shown continued improvements in accuracy for extremely low precision two-bit ternary DNNs where the values are constraints to (0,+1,-1), and one-bit binary DNNs where the values are constraints to (+1,-1).

3. CLASSIFICATION OF NEURAL NETWORK HARDWARE

The range of neural network hardware lies from single stand-alone neurochips to full-fledged neurocomputers. A block level architectural representation for almost all neurochips and neurocomputer processing elements has been presented in [11]. A variety of attributes have been used to classify NN hardware, such as system architecture, inter-processor communication networks, on-chip or off-chip learning, degrees of parallelism, general purpose or special purpose devices, and so on. NN hardware can be categorized into 4 classes by the degree of parallelism:

coarse-grained, medium-grained, fine-grained and massive parallelism [11]. The number of processing elements yields the degree of parallelism of a system. Parallelism increases data processing speed but is expensive in terms of chip area. Therefore, highly parallel systems usually employ simpler processing elements. Parallel processing elements only speed up the computation when they are not idle. Thus, for better system performance it is necessary that the inter-processor communication network provides the processing elements with sufficient data.

Neurocomputers are divided into two major categories. Standard chips consist of sequential accelerators, which speed up conventional computers like PC or workstation, and parallel multiprocessors, which are mostly stand alone and can be monitored by a host computer. Neurochips are built from dedicated neural ASICs (Application Specific Integrated Circuits) and can be digital, analog or hybrid.

3.1. Standard chips

3.1.1. Accelerator Boards

Accelerator boards are the most frequently used neural network hardware because they are widely available, relatively cheap, simple to connect to the workstation, and typically provided with user-friendly software tools. They are used to increase the speed of neural network computations. Accelerator boards are usually based on NN chips but some just use fast digital signal processors (DSP). A disadvantage of accelerator boards is that they are only specialized for certain tasks, and thus lack flexibility. Examples of accelerator boards are IBM ZISC ISA and PCI Cards. Other accelerator systems include SAIC SIGMA-1, Neuro Turbo, HNC, etc. Various types of neural network architectures have been studied and developed using accelerators in [29-32].

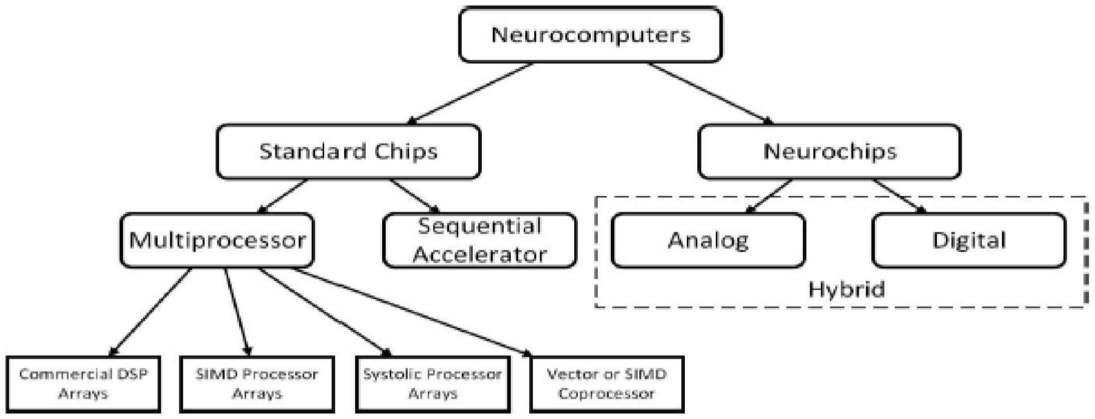

Figure 1. Neural network hardware categories

3.1.2. Neurocomputers Built from General Purpose Processors

Programmable neurocomputers were built to meet the need for high performance on large ANN simulations with reasonable cost/performance and flexible software control. These architectures can be simple, low-cost elements or rather sophisticated processors like transputers, which are unique for their parallel I/O lines or DSPs. These transputers were primarily developed for correlators and discrete Fourier transforms. A problem associated with neurocomputers is to find an interconnection strategy for large numbers of processors. Fortunately, knowledge about the architectures of these massively parallel computers can be directly applied in the design of neural architectures.

Multiprocessor neurocomputers can be classified into four categories [33]. The first category uses commercial digital signal processors (DSP) while the other three categories are based on custom-designed silicon.

3.1.2.1. Commercial DSP Arrays

Several neurocomputers have been built using commercial DSP arrays. Some notable examples are the RAP (Ring Array Processor) [34] developed at the International Computer Science Institute and the MUSIC system [35] developed at the Swiss Federal Institute of Technology. Both of these arrays connect the DSPs in a unidirectional ring topology with communication circuitry built from field-programmable gate arrays (FPGAs). The RAP supports up to 40 Texas Instruments TMS320C30 floating-point DSPs, with a peak performance of 32 MFLOPS per node. The MUSIC system connects up to 45 Motorola DSP96002 floating-point DSPs, with a peak performance of 60 MFLOPS per node. Both of these systems have distributed memories. They are programmed using a Single Program Multiple Data (SPMD) model, where all nodes operate on different portions of the data but run identical programs. A separate host computer handles data input and output and manages the overall program flow.

3.1.2.2. SIMD Processor Arrays

A popular approach in neurocomputer design is a SIMD (Single Instruction Multiple Data) array of processors which have limited form of processor interconnect. Instructions are broadcast by a common sequencer and executed simultaneously by all processors in these designs. The processors in SIMD system are much simpler than those in SPMD system because they do not have to fetch and decode instructions. Examples of SIMD neurocomputers include the CNAPS systems [19] from Adaptive Solutions and the SNAP [36] system from HNC. The neurochip N6400 is the basic building block of the CNAPS system and consists of 64 processing elements (or processing nodes PN) connected by a broadcast bus. The HNC system is built from SNAP chips each of which contains four 32-bit floating-point multiply-add datapaths with access to local off-chip memory. Multiple chips can be interconnected and controlled by the same central sequencer in both of these systems.

3.1.2.3. Systolic Processor Arrays

Several neurocomputers have been built using systolic processor arrays that perform the matrix operations for most neural algorithms. A systolic processor contains an array of interconnected pipelines through which operands flow in a regular manner. The most advanced of these systems is the SYNAPSE-1 [37]. The basic building block for this neurocomputer is the Siemens' MA-16 neurochip. It consists of eight MA-16 chips connected in two parallel rings controlled by two Motorola MC68040 processors. Systolic arrays can be formed by cascading multiple MA-16 chips. This ensures that inputs and outputs are passed from one MA-16 chip to another in a pipelined manner leading to an optimal throughput.

3.1.2.4. Vector or SIMD Co-processor

All the neurocomputers mentioned above rely on some form of off-chip control sequencer or host computer for managing the matrix computations occurring on the parallel processor arrays. Alternatively, the control processor can be integrated with the parallel execution units on the same die. Two examples of this type of design are the T0 vector microprocessor [38] and the L-Neuro 2.3 multi-DSP [39]. The T0 vector microprocessor integrates an industry-standard MIPS-II 32-bit integer scalar RISC processor with a tightly-coupled fixed-point vector coprocessor. The L-Neuro 2.3 design contains a 16-bit RISC controller along with an array of 12 DSP datapaths. The DSP datapaths are controlled using a writable microinstruction store, indexed by the RISC controller macroinstructions.

3.2. Neurochips

For multiprocessor neurocomputers the neural functions are programmed on general-purpose processors. Neurochips contain dedicated circuits devised in special purpose chips for the neural functions. This speeds up the neural iteration time by about two orders of magnitude as compared to general-purpose processor implementations. Neurochips can be designed using several implementation technologies. Defining a taxonomy of neurosystems requires consideration of three important factors:

• the kind of signals used in the network,
• the implementation of the weights, and
• the integration and output functions of the units.

The signals transmitted through the network can be coded using an analog or a digital model [10]. In the analog approach, a signal is represented by the magnitude of a current or voltage difference whereas in the digital approach, discrete values are stored and transmitted. If the signals are represented by currents or voltages, it is easier to implement the weights using resistances or transistors with a linear response function for certain range of values. In the case of a digital implementation, each transmission through one of the network's edges requires a digital multiplication. Hybrid neurocomputers are built combining analog and digital circuits. Analog systems require less power and offer higher implementation density on silicon. But digital systems offer programming flexibility, greater precision, and the possibility of working with virtual networks, that is, networks which are not physically mapped to the hardware, making it possible to deal with more units. Due to limited precision, direct implementation in circuits may alter the exact functioning of the original (simulated or analysed) computational elements. In order to build large-scale implementations many neurochips have to be interconnected. Some chips are therefore supplied with special communication channels. Other neurochips are to be interconnected by specially designed communication elements. A detailed study of digital, analog and hybrid neurochips is given in [40].

3.2.1. Digital Neurochips

Digital Neural ASICs are the most powerful neurochips. Digital techniques offer high computational precision, programmability and reliability. Furthermore, powerful design tools are available for full-custom and semi-custom design. Its shortcoming is the relatively large circuit size compared to analog implementations. Synaptic weights can be stored on or off chip which is determined by the trade-off between speed and size. Two well-known digital neurochips are

CNAPS [19] and SYNAPSE-1 [37]. VLSI microprocessors, vector and signal processors, slice architectures, SIMD and systolic arrays can be used for digital implementation. Greater speedup of linear algebraic operations can be achieved using systolic arrays which are regular structures of VLSI units, mainly one or two-dimensional, and which can communicate only locally [10].

3.2.2. Analog Neurochips

Analog electronics have characteristics that can directly be used for neural network implementation. For instance, operational amplifiers (Opamps), are built from single transistors and easily perform neuron-like functions, such as integration and sigmoid transfer. Some VLSI circuits work with field effect transistors (FETs) made of semiconductors. These are materials with a nonlinear voltage-current response curve which makes them especially suitable for the implementation of digital switches. Floating gate transistors can be used for an analog implementation of the multiplication operation. They represent the weights by statically stored charges or dynamically with the help of charge coupled devices (CCDs). Analog electronics are compact and offer high speed at low energy dissipation. The drawbacks of analog electronics are susceptibility to noise and process-parameter variations. Chips built according to the same design will never function in exactly the same way. Another limitation of the applicability of analog circuits is the problem of representing adaptable weights. Although analog chips will never reach the flexibility attainable with digital chips, their compactness and speed make them attractive for neural network research, especially when they adopt the adaptive properties of the original NN paradigms. Another advantage is that they more directly interface with the real, analog world, whereas digital implementations will always require fast analog-to-digital converters to read in data and digital-to-analog converters to put their data back into the world. Extensive implementations have been done using analog neurochips in [41-44]. Intel's ETANN is an example of analog neurochip [45].

3.2.3. Hybrid Neurochips

Both digital and analog techniques offer unique advantages but they also have drawbacks. The main shortcomings of digital techniques are the large amount of silicon and power required for multiplication circuits and the relatively slow computations. The shortcomings of analog techniques are the sensitivity to noise and susceptibility to interference and process variations. Therefore, the right combination of analog and digital techniques for the implementation of these processes is advantageous. Several research groups have implemented hybrid systems in order to gain advantages of both techniques and avoid the major drawbacks.

The ANNA (Analog Neural Network Arithmetic and Logic Unit) [46] chip can be used for a wide variety of ANN architectures but is optimized for locally connected, weight-sharing networks and time-delay neural networks (TDNNs). The Epsilon (Edinburgh Pulse Stream Implementation of a Learning Oriented Network) chip is a hybrid neurochip that uses pulse coding techniques [47].

4. HARDWARE ARCHITECTURE

There are two aspects of machine learning: training the network with massive amounts of sample data and then using the trained network to infer some attributes about new data sample. Training is typically done in large data centres. Figure 2 lays out the wide range of hardware targeting machine learning from leading vendors.

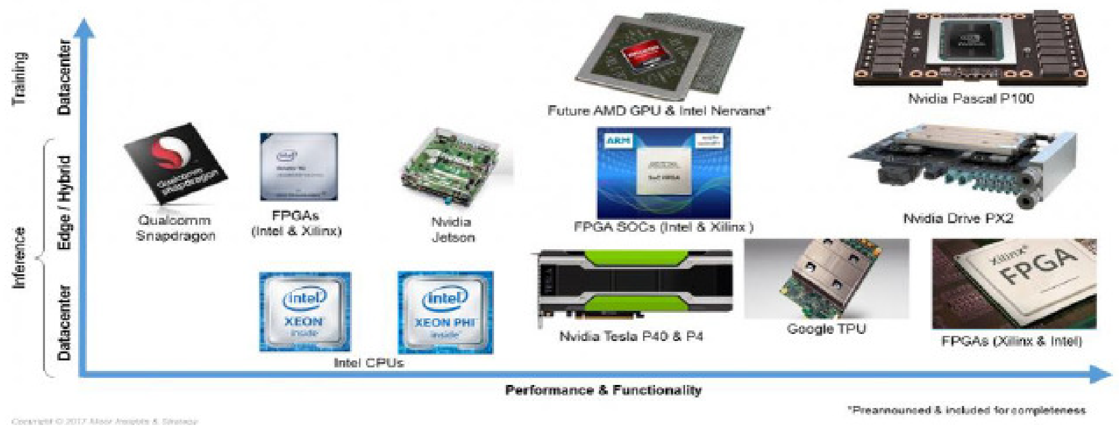

Figure 2. Hardware across the Machine Learning landscape [48]

Most of the applications today are hosted in public clouds with companies like Amazon, Google, Microsoft, etc. These companies run their online services from data centres packed with thousands of servers, each driven by a chip called a central processing unit, or CPU. Now these companies are supplementing CPUs with other processors for using Deep Learning Networks. Neural networks can learn tasks by analysing huge amounts of data, and they require more than just CPU power. So Google built the Tensor Processing Unit, or TPU while Microsoft is using a FPGA processor. Myriad companies employ machines equipped with large numbers of graphics processing units, or GPUs. The hardware used for machine learning today mainly consists of one or more of the following:

- CPU – Central Processing Units
- GPU – Graphic Processing Units
- FPGA – Field Programmable Gate Arrays
- ASIC – Application Specific Integrated Circuits

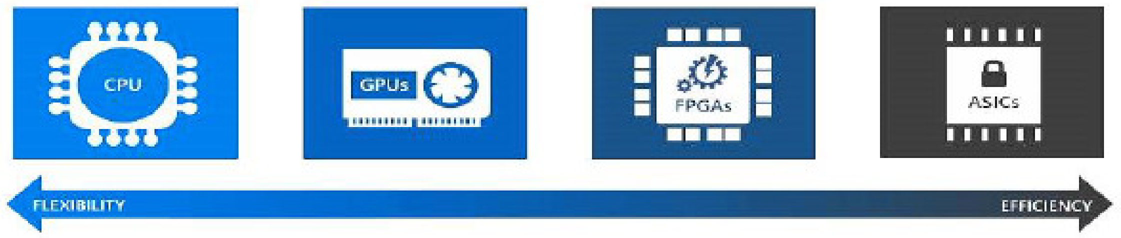

Figure 3. Overview of ML hardware available today [49]

Each step in this progression of technologies produces tremendous performance advantages. Each has its advantages for specific type of application or data, that is being deployed and in a specific environment. The velocity and data complexity determine the amount of processing needed, while the environment typically determines the power budget and latency demands. Performance can be measured in a number of ways [50]:

- computational capacity (or throughput)
- energy-efficiency (computations per Joule)
- cost-efficiency (throughput per dollar)

4.1. CPU

Central Processing Units or CPUs are often referred to as the “brains” of a computing system – whether it is for mobile, tablet, consumer (laptop/desktop), or enterprise servers. They are extremely flexible in terms of programmability, and handling workloads. They do fast calculations and have dynamic circuitry. However, they have cost and heat issues. Nearly all CPUs use these four steps in their operation: fetch, decode, execute, and write-back. They are good at fetching small amounts of memory quickly and the best ones have about 50GB/s memory bandwidth. Typical consumer CPUs have <10 cores, while server CPUs may go all the way up to 28. Intel is the dominant CPU manufacturer compared to others (ARM, AMD, IBM POWER, Oracle SPARC, Fujitsu). Intel’s Xeon and Xeon Phi [51] in datacentres and the Qualcomm Snapdragon in mobile devices are some examples of CPUs. Today, CPUs are mostly used for classic machine learning problems and sometimes for Deep Learning Inference [49].

4.2. GPU

Graphics Processing Units or GPUs are currently the most widely used hardware option for machine and deep learning [50]. GPUs are designed for high parallelism and memory bandwidth. They are considered to be the best option for training. They were originally designed to accelerate the large number of multiply and add computations performed in graphics rendering. Packaged as a video card attached to the PCI bus, they offloaded these numerically intensive computations from the CPU. As the demand for high performance graphics grew, so did the GPU, eventually becoming far more powerful than the CPU.

Machine and deep learning involves lot of matrix multiplications and convolutions. GPUs can provide an energy-efficient means of juggling the complex array of calculations required to train a neural network. This means they can train more neural networks with less hardware. GPU is good at fetching large amounts of memory. But companies also need chips that can rapidly execute neural networks through a process called inference. Google built the TPU specifically for doing this job. Microsoft uses FPGAs while Baidu is using GPUs, which are more suitable for training than for inference, but can do the job with the right software in place.

CPUs contain few cores with a large cache memory, and each core capable of handling a few software threads at a time. In contrast, a GPU contains hundreds of cores that can handle thousands of threads simultaneously. For example, a 16-core CPU processor running at 3.0 GHz performing fused multiply-add instructions has a peak performance of 96 Gflops, and a 56 processor GPU having 32 cores per processor containing 1792 cores and running at 1.48 GHz performing fused multiply-add instructions has a peak performance of 5300 Gflops [52]. The superior floating-point performance provided by GPUs is due to the large number of cores. That’s why the GPU can take on many multimedia tasks, such as accelerating Adobe Flash video, transcoding (translating) video between different formats and some really hard problems to solve that have an inherent parallel nature – video processing, image analysis, signal processing. Also, GPUs are now being used to accelerate computational workloads in areas such as cutting-edge scientific research, oil and gas exploration, and financial modelling [53].

Computers may contain multiple CPUs and GPUs for achieving good efficiency and very high speed processing [54]. Popular configurations include 2 CPUs and 1 to 8 GPUs. Each GPU provides an order of magnitude or more in performance over general purpose CPU processors resulting in faster solution times and the ability to solve large problems. Also, GPUs provide an order of magnitude or more in processing power for the same capital cost. The efficient architecture of GPUs perform more floating-point operations per watt of power consumed. Performance comparison of CPUs and GPUs is provided in [55].

For training purpose, using clusters of 8-16GPUs gives easy parallelism leading to best performance, cost and energy efficiency, and memory bandwidth [54]. For inference in data centres or in mobile devices (Automotive, IoT), single GPUs are used. GPUs are often far away from the main memory of the server, thus sending all the data to GPU takes time. This can pose a problem. Hence, companies like NVIDIA have come up with a faster interconnect called NVLink [56]. Titan X and Tegra X1 [57] are examples of GPUs.

NVIDIA and AMD are expanding both the sophistication of their processors and the software development tools for developing, porting, and debugging GPU code [58]. NVIDIA has an intriguing software tool called Nexus [59] that helps software developers to trace and debug application code from the CPU running on Windows into the GPU, including parallel applications on the GPU, and back to the CPU. These enhancements mean it will be easier to get existing software running on GPUs, although it will still require a software development effort.

NVIDIA's Compute Unified Device Architecture (CUDA) parallel computing architecture is developed for GPU computing. CUDA is a key to getting high performance out of certain computations that are important in engineering analysis and simulation. Many systems using GPUs and CUDA have a single industry-standard processor, usually running on Windows or Linux.

An ideal configuration is one that has one or more CPUs and a set of GPUs, known as hybrid computing [60], that use CUDA or similar parallel computation architecture thus delivering the best value of system performance, price, and power. All support applications, such as word processing, email and web browsing use the CPU. And with tools such as NVIDIA Nexus, engineering software will eventually take advantage of both to speed up complex computations.

4.3. FPGA

Field Programmable Gate Arrays (FPGAs) are a type of hardware that can be programmed and reconfigured using a hardware descriptive language (HDL). FPGAs have recently become a target appliance for machine learning researchers, and companies like Microsoft and Baidu have invested heavily in FPGAs. Even though they do not offer top peak floating-point performance, it is observed that FPGAs' performance/watt is higher than the GPUs'. This is because they have much less power usage (often 10s of Watts). This metric is important for applications in IoT and self-driving cars. As FPGAs can provide quick results for a pre-trained machine learning model (stored on the FPGA memory), they are also being used for inference [49].

To increase the number of operations processed and hence the compute performance researchers are looking for ways to leverage CPU and GPU architectures. FPGAs are concerned with system performance. By controlling the data path, they accelerate and aid the compute and connectivity required to collect and process the massive quantities of information. Also, they can directly receive data and process it inline without going through the host system. This frees the processor to manage other system events and provide higher real-time system performance. AI often relies on real-time processing to draw instantaneous conclusions and respond accurately. The work in [61] shows the procedure to design and develop AI hardware using FPGA.

FPGAs' flexibility aids in delivering deterministic low latency and high bandwidth. The authors in [62] demonstrate an approach that uses an on-chip stream buffer that efficiently stores input and output feature maps on FPGAs and improves bandwidth efficiency. It enables users to update the hardware capabilities for the system without requiring a hardware refresh as in case of CPUs and GPUs, thus resulting in longer lifespans of deployed products. FPGAs support the creation of custom hardware for individual solutions in an optimal way. Regardless of the custom or standard data interface, topology, or precision requirement, an FPGA can implement the exact architecture defined, which allows for fixed data paths and unique solutions. This is also equivalent to excellent power efficiency and future proofing. With such a dynamic technology as machine learning, which is evolving and changing constantly, FPGAs provide the flexibility unavailable in fixed devices. An FPGA has the flexibility to instantly support changes such as precision-drop from 32-bit to 8-bit and even binary/ternary networks. No layout, masks or other manufacturing steps are needed for FPGAs, thus making the time-to-market faster. The design cycle is simpler due to software that handles much of the routing, placement, and timing. The project cycle of FPGAs is more predictable due to elimination of potential re-spins, wafer capacities, etc.

All FPGA implementations of ANNs try to use the re-configurability of FPGA hardware in one way or another. Intel and Xilinx produce FPGA that has the ability to reconfigure the hardware [49]. Identifying the purpose of reconfiguration highlights the motivation behind different implementation approaches [63].

- **Prototyping** exploits the fact that FPGA-based hardware can be quickly reconfigured an unlimited number of times. This apparent hardware flexibility allows rapid prototyping of different ANN implementation strategies and learning algorithms for initial simulation. Also, due to the dynamic nature of FPGA devices they have modifiable topologies. Hence, iterative construction of ANNs can be realized through topology adaptation. A digital architecture for classification using FPGAs' re-programmability feature is described in [64].

- **Density enhancement** refers to methods which increase the amount of effective functionality per unit circuit area through FPGA reconfiguration. This is attained by using FPGA run-time / partial re-configurability in one of the two ways. Firstly, an FPGA chip can be time-multiplexed for each of the sequential steps in an ANN algorithm. Secondly, an FPGA chip can be time-multiplexed for each of the ANN circuits that is specialized with a set of constant operands at different stages during execution. This technique is also called *dynamic constant folding*.

A feed-forward ANN algorithm is implemented based on the FPGA technology in [26]. A method of implementing a fully connected feed forward network with Xilinx FPGAs for image processing in a way that a single processing node is partitioned into two XC3090 chips is proposed in [63]. It explores the way to implement fully parallel ANN and efficiently use 32-bit floating-point numeric representation in FPGA-based ANNs by making use of the features of SpartanIIE series FPGAs. Pedro Ferreira et. al. [65] proposed a hardware implementation of ANN using FPGA and piece-wise linear approximation. In [66], a digital hardware-implementation strategy for feedforward ANNs with step activation functions has been reported. The algorithm treats each neuron as a special case of Boolean functions with properties that can be exploited to achieve compact implementation. This is accomplished by means of VHDL code that can be easily translated into an FPGA implementation, using suitable electronic-design-automation software. The work in [67] describes the hardware implementation of a real-time neural network controller with a DSP and an FPGA for nonlinear systems.

Recent studies say that FPGAs have outperformed GPUs in many fields and are expected to beat GPUs in accelerating Deep Learning [58]. The emerging sparse and low precision DNN algorithms offer orders of magnitude algorithmic efficiency improvement over the traditional dense FP32 DNNs, but they introduce custom data types and irregular parallelism which are difficult for GPUs to handle. In contrast, FPGAs are designed for extreme customizability when running irregular parallelism and custom data types. Such trends make future FPGAs a viable platform for running DNN, ML and AI applications. Intel's evaluation of various emerging DNNs on two generations of FPGAs (Intel Arria 10 [68] and Intel Stratix 10 [69] and Titan X GPU shows that current trends in DNN algorithms may favour FPGAs, and that FPGAs may even offer superior performance than GPUs. In [70], the authors showed the implementation of Binarized Neural Networks (BNN) using CPU, GPU, FPGA, and ASIC and compared their performances. FPGAs may be used for other irregular applications beyond DNNs, and on latency sensitive applications like ADAS and industrial uses [28].

Generally, FPGA is about an order of magnitude less efficient than ASIC. However, modern FPGAs contain "hardened" resources, such as DSPs for arithmetic operations and M20Ks (in Altera FPGA) for on-chip RAMs which reduce the efficiency gap between FPGA and ASIC.

Dedicated ASICs can provide a higher total cost of ownership (TCO) in the long run, and with such a dynamic technology, there is a higher threshold to warrant building them, especially if FPGAs can meet a system's needs.

We can have a fusion of these hardware to suit the applications. This enables the hardware to be used to their fullest capacity and gives optimal network architectures. For example, in [48] the accelerators do good job of running the AI inference engine; sensor fusion, data pre-processing and post-scoring policy execution require a lot of special I/O and fast traditional logic which is best suited for CPUs. Hybrid hardware platforms are offered by NVIDIA with an ARM / GPU combination in NVIDIA's Jetson [71] and DrivePX2 [72], while Intel and Xilinx offer SoCs (System on Chips) that bring ARM and FPGAs into a single, elegant low-power package. All of these products are finding their way into drones, automobiles and factory robots / cobots where the right combination of flexibility, speed and low power demand innovative approaches.

4.4. ASIC

Whilst GPUs and FPGAs perform far better than CPUs, a factor of 10 in efficiency can still be gained with a more specific design, via an Application-Specific Integrated Circuit (ASIC) [49]. ASICs are the least flexible but highest performing hardware options. They have full custom capability for design since devices are manufactured to design specifications. They are also the most efficient in terms of performance/dollar and performance/watt, but require huge investment and NRE (non-recurring engineering) costs that make them cost-effective only in very high volume designs. They have smaller form factor since devices are manufactured to design specifications. ASICs can be designed for either training or inference.

Google is the best example of successful machine learning ASIC deployments. The first generation of Google's Tensor Processing Unit (TPU) [73] originally focused on 8-bit integers for inference workloads. The newer generation ASICs offers floating point precision and can be used for training, too. Unlike CPUs and GPUs, they are designed for a specific purpose (for e.g., mining bitcoins) and cannot be reprogrammed. Their lack of extraneous logic makes them extremely high in performance and economic in their power usage – but very expensive. Intel's Nervana [74], a low-latency, high-memory bandwidth chip built for deep learning, is another example of an ASIC.

5. ISSUES IN HARDWARE DESIGN AND OPTIMIZATION TECHNIQUES

The key metrics for embedded machine learning are accuracy, energy consumption, throughput/latency, and cost [9]. The accuracy of the ML algorithm should be measured on a sufficiently large dataset. The weights have to be updated when the application changes, thus making programmability important. For DNNs, the processor should also be able to support different networks with varying number of channels, layers, filters and filter sizes. The need for programmability and higher dimensionality both result in an increase in data movement and computation. Programmability means that the weights also need to be read and stored and higher dimensionality increases the amount of data generated. This can pose a challenge for energy-efficiency since data movement costs more than computation. The throughput is dictated by the amount of computation, which also increases with the dimensionality of the data. The cost is dictated by the amount of storage required on the chip while maintaining low off-chip memory bandwidth. Finally, training requires a significant amount of labelled data (particularly for DNNs) as well as computation for multiple iterations of back-propagation for determining the value of weights. Currently, state-of-the-art DNNs consume higher energy than other forms of embedded processing (e.g., video compression). We must exploit opportunities at different levels of hardware design to address all these issues and remove this energy gap.

When implementing ANNs, selecting weight precision is one of the important choices. Weight precision is used to trade-off the capabilities of the realized ANNs against the implementation cost. A higher weight precision results in fewer quantization errors in the final implementations, while a lower precision leads to greater speed, simpler designs, and reductions in area requirements and power consumption. One way of resolving the trade-off is to determine the "minimum precision" required to solve a given problem [12].

Direct implementation for non-linear sigmoid transfer functions is very expensive. There are two practical approaches to approximate sigmoid functions [12]. Piece-wise linear approximation describes a combination of lines in the form of $y = ax + b$ which is used for approximating the sigmoid function. The sigmoid functions can be realized by a series of shift and add operations if the coefficients for the lines are chosen to be powers of two. The second method is lookup tables, in which uniform samples are taken from the centre of a sigmoid function and stored in a table for look up. The regions outside the centre of the sigmoid function are still approximated in a piece-wise linear fashion.

Researchers are modifying the ML algorithms to make them more hardware-friendly while maintaining accuracy. The main focus lies on reducing computation, data movement and storage requirements. The optimization techniques are as follows:

5.1. Precision reduction

The default size for programmable platforms such as CPUs and GPUs is often 32 or 64 bits with floating-point representation during training [9]. While during inference, it is possible to use a fixed-point representation and substantially reduce the bit-width for savings in energy and area, and increase in throughput. For instance, in [58], input and feature vectors are 4 bits while weight is 6 bits.

5.2. Pruning

For DNNs, the number of multiply and accumulate operations, and weights can be reduced by removing weights with small or minimal impact on the output through a process called pruning. However, removing weights does not necessarily lead to energy reduction. Hence, weights are removed based on an energy-model to directly minimize energy consumption. In [31], performance and energy efficiency are improved by a factor of 2.7x and 2.3x respectively by network pruning during training.

5.3. Compression

Data movement and storage are important factors in both cost and energy. Feature extraction can result in sparse data and the weights used in classification can also be made sparse by pruning. As a result, compression can be applied to exploit data statistics to reduce data movement and storage cost.

6. USING MIXED-SIGNAL CIRCUITS FOR ML HARDWARE ARCHITECTURE

Most of the data movement is in between the memory, processing element, and sensor. Since the training often occurs in the digital domain, the analog-to-digital conversion and digital-to-analog conversion overhead should be accounted for when evaluating the system. While spatial architectures bring the memory closer to the computation (i.e., into the processing element), there have also been efforts to integrate the computation into the memory itself. For instance, in [75] and [76], the classification is embedded in the SRAM. Recently, use of mixed-signal circuits to reduce computation cost of the MAC have been explored. Authors in [77] study the trade-off between energy and accuracy in neural networks, and present the ways to incorporate mixed-signal design techniques to achieve low power dissipation in a semi-programmable ASIC implementation.

7. ADVANCED TECHNOLOGIES FOR ML HARDWARE ARCHITECTURE

Conventional CPUs/GPUs, which are based on the sequential von Neumann architecture, are inadequate for fast training with large data set due to limited power constraints and various other reasons. Even the computing speed of custom-designed ASIC lags behind the requirement of real-time online learning. Hence, to speed this process researchers are looking for ideas beyond the traditional CMOS designs. For example, in [78] Chen et. al. have implemented machine learning algorithms on a chip using Synaptic Device Model and Technology-design Co-optimization Methodologies of the Resistive Cross-point Array. Reverse scaling rules have been used for sizing the array geometrical dimensions such as the wire width and the cell spacing to achieve high learning accuracy. It realizes fully parallel operations of the weighted sum and the weight update with the help of parallel read and write scheme. The digital spike encoding scheme and the analog voltage encoding scheme have been used in terms of learning accuracy. It achieves 10^3 speed-up and 10^6 energy efficiency improvement, enabling real-time image feature extraction and learning.

An analog deep learning system has been developed in [79] which overcomes the limitations of conventional digital implementations by exploiting the efficiency of analog signal processing. Reconfigurable current-mode arithmetic realizes parallel computation. A floating-gate analog memory compatible with digital CMOS provides non-volatile storage. Algorithm-level feedback mitigates the effects of device mismatch. System-level power management applies power gating to inactive circuits. The online cluster analysis with accurate parameter learning, and feature extraction in pattern recognition with dimension reduction by a factor of 8 has been demonstrated. The system features unsupervised online trainability, non-volatile memory and good efficiency and scalability, making it a general-purpose feature extraction engine ideal for autonomous sensory applications as well as a building block for large-scale learning systems.

The use of advanced memory technologies such as embedded DRAM (eDRAM) is explored in [80] to reduce the energy cost in memory access of the weights in DNN. In [81], memristors are used to compute a 16-bit dot product operation with 8 memristors each storing 2-bits. In [82], ReRAM is used to compute the product of a 3-bit input and 4-bit weight. Similar to the mixed-signal circuits, the precision is limited, and the analog-to-digital conversion and digital-to-analog conversion overhead must be considered in the overall cost, especially when the weights are trained in the digital domain.

An analog neural network suitable for building large scale systems has been developed using a learning procedure called contrastive backpropagation learning in [83]. In [84], the components of VLSI implementation of a spiking neural network is presented while [85] demonstrates a highly configurable neuromorphic chip with integrated learning for a network of spiking neurons which can be used in pattern classification, recognition, and associative memory tasks. A spatial architecture named ‘Eyeriss’ for energy-efficient dataflow for Convolutional Neural Networks is implemented in [86].

8. CASE STUDY

CNAPS [19] and SYNAPSE-1[37] have been studied extensively in [11]. Here, a study of the TPU has been presented.

A Tensor Processing Unit (TPU) is an ASIC developed by Google specifically for machine learning. Compared to a GPU, it is designed explicitly for a higher volume of reduced precision computation (e.g. 8-bit precision) with higher input/output operations per second per watt. The chip has been specifically designed for Google's TensorFlow™ framework. However, Google still uses CPUs and GPUs for other types of machine learning.

Google has stated that its proprietary TPUs were used in the AlphaGo versus Lee Sedol series of man-machine Go games. Google has also used TPUs for Google Street View text processing, and was able to find all the text in the Street View database in less than five days [87]. In Google

Photos, an individual TPU can process over 100 million photos a day. It is also used in RankBrain which is used by Google to provide search results.

8.1. First generation

The first generation TPU is an 8-bit matrix multiply engine, driven with CISC instructions. It is manufactured on a 28nm process with a die size $\leq 331\ mm^2$. The clock speed is 700 MHz and has a thermal design power of 28-40 W. It has 28 MiB of on chip memory, and 4 MiB of 32-bit accumulators taking the results of a 256x256 array of 8-bit multipliers. Within the TPU package is 8 GiB of dual-channel 2133 MHz DDR3 SDRAM offering 34GB/s of bandwidth. Google's first-generation TPUs made it dramatically faster to run ML models that had not been trained a lot, but the training had to be performed separately.

8.2. Second generation

Training state-of-the-art machine learning models requires an enormous amount of computation, due to which researchers, engineers, and data scientists often wait weeks for results. To solve this problem, an all-new ML accelerator was designed from scratch, a second-generation TPU or Tensor Processing Unit, that can accelerate both training and running ML models. The second generation TPU was announced in May 2017. Google stated the first generation TPU design was memory bandwidth limited, and using 64 GB of high bandwidth memory in the second-generation design increased bandwidth to 600GB/s and performance to 45 TFLOPS. The TPUs are arranged into 4-chip 180 TFLOPS modules. These modules are then assembled into 256 chip pods (64-TPU pods) with 11.5 PFLOPS of performance. Notably, while the first generation TPUs were limited to integers, the second generation TPUs can also calculate in floating point. This makes the second generation TPUs useful for both training and inference of machine learning models. Google's second-generation Cloud TPUs are even more powerful, designed to accelerate the training of ML models as well as running them. The TPU features [88] are:

- The TPU is 15x to 30x faster than contemporary GPUs and CPUs for AI workloads that utilize neural network inference.
- As compared to conventional chips, the TPU achieves much better energy efficiency gaining 30x to 80x improvement in TOPS/Watt measure (tera-operations [trillion or 10^{12} operations] of computation per Watt of energy consumed).
- The neural networks powering these applications require a surprisingly small amount of code: just 100 to 1500 lines. The code is based on TensorFlow, which is an open-source machine learning framework.

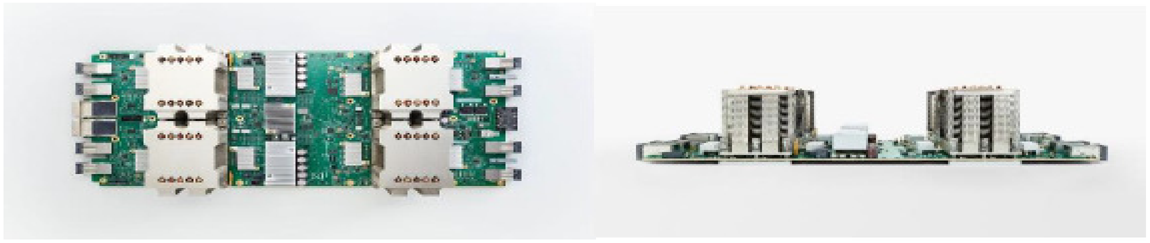
Figure 4. Google's new Cloud TPUs deliver machine learning acceleration [89]

9. CONCLUSION

Machine learning has covered significant areas of computing and information processing in today's world. There is a timely need to map the latest ML algorithms to physical hardware, in order to achieve significant advances in performance, speed, area and energy efficiency. ANNs, one of the important learning algorithms of machine learning, are implemented in software, and are trained and simulated on general-purpose computers for testing a wide range of neural networks models. The main objective of building dedicated hardware for ML is to provide a platform for efficient adaptive systems, capable of updating their parameters in the course of time. Deep learning networks are playing a critical role in most AI-based technologies today.

Hardware implementation of ANNs is essential for applicability to large networks and for taking advantage of their inherent parallelism which is less efficient in their software implementation. While designing hardware for neural networks, careful consideration should be made for the choice of precision, number format and type of neurocomputer. Machine learning hardware is also designed using CPUs, GPUs, FPGAs and ASICs depending on the required performance. The hardware built using these technologies may have structural and behavioural issues and hence optimization and careful design is necessary. Advanced technologies such as eDRAM and ReRAM are being used for speed-up and to overcome conventional design problems. As AI applications expand, the demand for ML-specialized devices will drive hardware into the next phases of evolution. It will be fascinating to experience the impact of these technologies applied in healthcare, transportation, and robotics. Many exciting steps in the evolution of machine learning still remain yet to be explored.

REFERENCES

[1] Jacques Bughin et. al., "How Artificial Intelligence Can Deliver Real Value to Companies", McKinsey. [Online] Available: https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/how-artificial-intelligence-can-deliver-real-value-to-companies.

[2] Kevin Fogarty, (2017, Nov. 9), "The next Phase of Machine Learning", Semiconductor Engineering. [Online] Available: https://semiengineering.com/the-next-phase-of-machine-learning/.

[3] Eduard Sackinger et. al., "Application of the ANNA Neural Network Chip to High-Speed Character Recognition", IEEE Transactions on Neural Netsworks, Vol. 3, No. 3, May 1992.

[4] Patrick Bourke, Rob A. Rutenbar, "A High-Performance Hardware Speech Recognition System for Mobile Applications", 2005.

[5] Sergiu Nedevschi, Rabin K. Patra, Eric A. Brewer, "Hardware Speech Recognition for User Interfaces in Low Cost, Low Power Devices", Design Automation Conference, 2005.

[6] B.E. Boser et al, "Hardware requirements for neural network pattern classifiers", IEEE Micro (Volume: 12, Issue: 1, Feb. 1992), pp. 32-40.

[7] Bernard Marr, (2017, August 8). Forbes [Online]. Available: https://www.forbes.com/sites/bernardmarr/2017/08/08/the-amazing-ways-how-google-uses-deep-learning-ai/#711a9ea43204.

[8] Ryan Whitwam (2017, October 16). ExtremeTech [Online]. Available: https://www.extremetech.com/extreme/257110-deepminds-wavenet-voice-synthesizer-live-google-assistant.

[9] Vivienne Sze, Yu-Hsin Chen, Joel Emer, Amr Suleiman, Zhengdong Zhang, "Hardware for Machine Learning: Challenges and Opportunities", CICC 2017.

[10] R. Rojas, "Neural Networks", Springer-Verlag, Berlin, 1996.

[11] Liao, Yihua, "Neural networks in hardware: A survey", Davis, CA, 2017.

[12] Jihan Zhu and Peter Sutton, "FPGA Implementations of Neural Networks – A Survey of a Decade of Progress", Y. K. Cheung P., Constantinides G.A. (eds) Field Programmable Logic and Application, FPL 2003, Lecture Notes in Computer Science, vol. 2778. Springer, Berlin, Heidelberg.

[13] "Scikit-learn" [Online] Available: http://scikit-learn.org/stable/, Accessed on: Dec. 19, 2017.

[14] "Theano" [Online] Available: http://deeplearning.net/software/theano/, Accessed on: Dec. 19, 2017.

[15] "Apache Spark MLlib" [Online] Available: https://spark.apache.org/mllib/, Accessed on: Dec. 19, 2017.

[16] "H2O" [Online] Available: https://www.h2o.ai/, Accessed on: Dec. 19, 2017.

[17] "Tensorflow" [Online] Available: https://www.tensorflow.org/, Accessed on: Dec. 19, 2017.

[18] Arpan Chakraborty, (2016, April 7). Udacity [Online]. Available: https://blog.udacity.com/2016/04/5-skills-you-need-to-become-a-machine-learning-engineer.html.

[19] McCartor, H., 1991, "A Highly Parallel Digital Architecture for Neural Network Emulation", Delgado-Frias, J. G. and Moore, W. R. (eds.), VLSI for Artificial Intelligence and Neural Networks, pp. 357- 366, Plenum Press, New York, 1991.

[20] Lindsey, C. S., Lindblad, Th., Sekniaidze, G., Minerskjold, M., Szekely, S., and Eide, A., "Experience with the IBM ZISC Neural Network Chip". Proceedings of 3rd Int. Workshop on Software Engineering, Artificial Intelligence, and Expert Systems, for High Energy and Nuclear Physics, Pisa, Italy, April 3-8, 1995.

[21] Nvidia, "Why GPUs?". [Online] Available: http://www.fmslib.com/mkt/gpus.html,
Accessed on: Dec. 20, 2017.

[22] Holt, J. and Hwang, J., "Finite Precision Error Analysis of the Neural Network Hardware Implementations". IEEE Trans. on Computers, 42:281-290, 1993.

[23] Dany Bradbury, (2017, July 24), "What sort of silicon brain do you need for artificial intelligence?", The Register. [Online]. Available:
https://www.theregister.co.uk/2017/07/24/ai_hardware_development_plans/.

[24] Thiran, P., Peiris, V., Heim, P. and Hochet, B., "Quantization Effects in Digitally Behaving Circuit Implementations of Kohonen Networks". IEEE Trans. on Neural Networks, 5(3):450-458, 1994.

[25] Strey, A. and Avellana, N., "A New Concept for Parallel Neurocomputer Architectures". Proceedings of the Euro-Par'96 Conference, Lyon (France), Springer LNCS 1124, Berlin, 470-477, 1996.

[26] E. Won, "A hardware implementation of artificial neural networks using field programmable gate arrays", Elsevier, Nuclear Instruments and Methods in Physics Research A 581 (2007) pp. 816–820, 2007.

[27] Marchesi, M., et al., "Fast neural networks without multipliers". IEEE Transactions on Neural Networks, 1993. 4(1): p. 53-62.

[28] Linda Barney, (2017, March 21), "Can FPGAs beat GPUs in accelerating next-generation deep learning?", The Next Platform. [Online]. Available: https://www.nextplatform.com/2017/03/21/can-fpgas-beat-gpus-accelerating-next-generation-deep-learning/.

[29] Andre Xian Ming Chang, Eugenio Culurciello, "Hardware accelerators for Recurrent Neural Networks on FPGA", Circuits and Systems (ISCAS), 2017 IEEE International Symposium, ISSN: 2379-447X, 2017.

[30] Chao Wang, Qi Yu, Lei Gong, Xi Li, Yuan Xie, Xuehai Zhou, "DLAU: A Scalable Deep Learning Accelerator Unit on FPGA", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (Volume: 36, Issue: 3, March 2017), pp. 513 – 517.

[31] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, William J. Dally, "SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks", ISCA'17, Proceedings of the 44th Annual International Symposium on Computer Architecture, pp. 27-40.

[32] Yijin Guan, Zhihang Yuan, Guangyu Sun, Jason Cong, "FPGA-based Accelerator for Long Short-Term Memory Recurrent Neural Networks", Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific, ISSN: 2153-697X, 2017.

[33] Krste Asanovic, "Programmable Neurocomputing", MIT Laboratory for Computer Science, Cambridge, MA 02139. [Online]. Available:
https://people.eecs.berkeley.edu/~krste/papers/neurocomputing.pdf, Accessed on: Sept. 26, 2017.

[34] N. Morgan, J. Beck, P. Kohn, J. Bilmes, E. Allman, and J. Beer, “The Ring Array Processor (RAP): A multiprocessing peripheral for connectionist applications”, Journal of Parallel and Distributed Computing, 14:248–259, April 1992.

[35] U. A. Muller, B. Baumie, P. Kohler, A. Gunzinger, and W. Guggenbuhl, “Achieving supercomputer performance for neural net simulation with an array of digital signal processors”, IEEE Micro, 12(5):55–64, October 1992.

[36] R. Means and L. Lisenbee, “Extensible linear floating-point SIMD neurocomputer array processor”, Proceedings of the International Joint Conference on Neural Networks, pages I–587–592, New York, 1991. IEEE Press.

[37] Ramacher, U., Raab, W., Anlauf, J., Hachmann, U., Beichter, J., Bruls, N., Webeling, M. and Sicheneder, E., 1993, “Multiprocessor and Memory Architecture of the Neurocomputers SYNAPSE-1”, Proceedings of the 3rd International Conference on Microelectronics for Neural Networks (MicroNeuro), pp. 227-231, 1993.

[38] J. Wawrzynek, K. Asanovi´c, B. Kingsbury, J. Beck, D. Johnson, and N. Morgan, “Spert-II: A vector microprocessor syste”, IEEE Computer, 29(3):79–86, March 1996.

[39] M. Duranto, “Image processing by neural networks”, IEEE Micro, 16(5):12–19, October 1996.

[40] Fernando Morgado Dias, Ana Antunes, Alexandre Manuel Mota, “Commercial Hardware for Artificial Neural Networks: A Survey”, IFAC Proceedings Volumes, Vol. 36, Issue 12, pp.189-196, 2003.

[41] Jung-Wook Cho and Soo-Young Lee, “Active Noise Cancelling using Analog NeuroChip with On-Chip Learning Capability”, NIPS Proceedings, 1998.

[42] Mark Holler, Simon Tam, Hernan Castro, Ronald Benson, “An Electrically Trainable Artificial Neural Network (ETANN) with 10240 "Floating Gate" Synapses”, Neural Networks, 1989, IJCNN., International Joint Conference, 1989.

[43] Takeshi Kamio, Haruyasu Adachi, Hiroshi Ninomiya, Hideki Asai, “A Design Method of DWT Analog Neuro Chip for VLSI Implementation”, Instrumentation and Measurement Technology Conference, 1997. IMTC/97. Proceedings. Sensing, Processing, Networking., IEEE, 1997.

[44] Daiki Masumoto, Hiroki Ichiki, Hideki Yoshizawa, Hideki Kato, Kazuo Asakawa, “An Analog Neurochip and Its Applications to Multilayered Artificial Neural Networks”, TOC, vol. 74, issue 9, pp. 92-103, 1991.

[45] Wikichip, “ETANN - Intel”. [Online] Available: https://en.wikichip.org/wiki/intel/etann, Accessed on: Oct. 19, 2017.

[46] Eduard Sackinger, Bernhard E. Boser, Lawrence D. Jackel, “A Neurocomputer Board Based on the ANNA Neural Network Chip”, Advances in Neural Information Processing Systems 4 (NIPS 1994), pp. 773-780.

[47] Alan F. Murray et. al., “Pulse Stream VLSI Neural Networks”, IEEE Macro, Vol. 14, Issue 3, June 1994, p. 29-39.

[48] Karl Freund, (2017, March 3), “A machine learning landscape: where AMD, Intel, Nvidia, Qualcomm and Xilinx AI engines live”, Forbes. [Online]. Available
: https://www.forbes.com/sites/moorinsights/2017/03/03/a-machine-learning-landscape-where-amd-intel-nvidia-qualcomm-and-xilinx-ai-engines-live/#4436108a742f.

[49] Gaurav Nakhare, (2017, July 31), “Hardware options for machine/deep learning”, MS&E 238 Blog. [Online]. Available: https://mse238blog.stanford.edu/2017/07/gnakhare/hardware-options-for-machinedeep-learning/.

[50] Cade Metz, (2016, October 26), “How AI is shaking up the chip market”. [Online]. Available: https://www.wired.com/2016/10/ai-changing-market-computer-chips/.

[51] “Intel Xeon Phi Processors”. [Online] Available:
https://www.intel.com/content/www/us/en/products/processors/xeon-phi/xeon-phi-processors.html, Accessed on: Dec. 19, 2017.

[52] Nvidia, “Why GPUs?”. [Online] Available: http://www.fmslib.com/mkt/gpus.html, Accessed on: Dec. 20, 2017.

[53] Kevin Krewell, (2009, December 16), “What’s the difference between a CPU and a GPU?”. Nvivdia [Online]. Available: https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/.

[54] William Dally, (2015, July 12), “High performance hardware for machine learning”, NIPS Tutorial. [Online]. Available: https://media.nips.cc/Conferences/2015/tutorialslides/Dally-NIPS-Tutorial-2015.pdf.

[55] Nvidia, “Why GPUs?”. [Online] Available: http://www.fmslib.com/mkt/gpus.html, Accessed on: Dec. 20, 2017.

[56] Nvidia NVLink high-speed interconnect”, Nvidia. [Online]. Available: http://www.nvidia.com/object/nvlink.html. Accessed on: Sept. 29, 2017.

[57] Nvidia, “Tegra Processors”. [Online] Available: http://www.nvidia.com/object/tegra-x1-processor.html, Accessed on: Dec. 20, 2017.

[58] Nuno Edgar Nunes Fernandes, (2017, April 3), “FPGA chips will be the hardware future for deep learning and AI”, Wordpress. [Online]. Available: https://theintelligenceofinformation.wordpress.com/2017/04/03/fpga-chips-will-be-the-hardware-future-for-deep-leaning-and-ai/.

[59] Nvidia, “Nvidia Introduces Nexus, The Industry’s First Integrated GPU/CPU Environment for Developers Working with Microsoft Visual Studio”. [Online] Available: http://www.nvidia.com/object/pr_nexus_093009.html.

[60] Kishore Kothapalli et. al., “CPU and/or GPU: Revisiting the GPU Vs. CPU Myth”. [Online] Available: https://arxiv.org/pdf/1303.2171.pdf.

[61] William J., (2017, July 24), “Machine Learning on Intel FPGAs”, Intel. [Online]. Available: https://software.intel.com/en-us/articles/machine-learning-on-intel-fpgas.

[62] Utku Aydonat, Shane O’Connell, Davor Capalija, Andrew C. Ling, Gordon R. Chiu, “An OpenCL Deep Learning Accelerator on Arria 10”, 2017.

[63] Suhap Sahin, Yasar Becerikli, Suleyman Yazici, “Neural Network Implementation in Hardware Using FPGAs”, Neural Network Implementation in Hardware Using FPGAs. In: King I., Wang J., Chan LW., Wang D. (eds) Neural Information Processing. ICONIP 2006. Lecture Notes in Computer Science, vol. 4234, Springer, Berlin, Heidelberg.

[64] Cox, C.E. and E. Blanz, “GangLion - a fast field-programmable gate array implementation of a connectionist classifier”, IEEE Journal of Solid-State Circuits, 1992. 28(3): pp. 288-299.

[65] Pedro Ferreira, Pedro Ribeiro, Ana Antunes, Fernando Morgado Dias, “Artificial Neural Networks Processor - a Hardware Implementation using a FPGA”, Becker J., Platzner M., Vernalde S. (eds) Field Programmable Logic and Application. FPL 2004. Lecture Notes in Computer Science, vol. 3203, Springer, Berlin, Heidelberg.

[66] Andrei Dinu, Marcian N. Cirstea, and Silvia E. Cirstea, “Direct Neural-Network Hardware-Implementation Algorithm”, IEEE Transactions on Industrial Electronics (vol. 57, Issue: 5, May 2010).

[67] Seul Jung, Sung su Kim, “Hardware Implementation of a Real-Time Neural Network Controller with a DSP and an FPGA for Nonlinear Systems”, IEEE Transactions on Industrial Electronics, vol. 54, No. 1, February 2007.

[68] Intel FPGA and SoC, “Arria 10”. [Online] Available: https://www.altera.com/products/fpga/arria-series/arria-10/overview.html.

[69] Intel FPGA and SoC, “Stratix 10”. [Online] Available: https://www.altera.com/products/fpga/stratix-series/stratix-10/overview.html.

[70] Eriko Nurvitadhi et. al., “Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC”, IEEE International Conference on Field-Programmable Technology, 7-9 Dec., 2016.

[71] Nvidia, “Jetson Automotive Development Platform”. [Online] Available: http://www.nvidia.in/object/jetson-pro-automotive-development-platform-in.html.

[72] Nvidia, “Nvidia Drive PX”. [Online] Available: https://www.nvidia.com/en-us/self-driving-cars/drive-px/.

[73] Nicole Hemsoth (2017, April 5), “First In-depth Look at Google’s TPU Architecture”. [Online] Available: https://www.nextplatform.com/2017/04/05/first-depth-look-googles-tpu-architecture/.

[74] Intel Nervana, [Online] Available: https://www.intelnervana.com/.

[75] J . Zhang, Z. Wang, N. Verma, “A machine-learning classifier implemented in a standard 6T SRAM array,”, Sym. on VLSI, 2016.

[76] Z. Wang, R. Schapire, N. Verma, “Error-adaptive classifier boosting (EACB): Exploiting data-driven training for highly fault-tolerant hardware,”, ICASSP, 2014.

[77] B. Murmann, D. Bankman, E. Chai, D. Miyashita, L. Yang, “Mixed-signal circuits for embedded machine-learning applications”, Signals, Systems and Computers, 49th Asilomar Conference, 2015.

[78] Pai-Yu Chen, Deepak Kadetotad, Zihan Xu, Abinash Mohanty, Binbin Lin, Jieping Ye, Sarma Vrudhula, Jae-sun Seo, Yu Cao, Shimeng Yu, “Technology-design co-optimization of resistive cross-point array for accelerating learning algorithms on chip”, Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015.

[79] Junjie Lu, Steven Young, Itamar Arel, Jeremy Holleman, “A 1 TOPS/W Analog Deep Machine-Learning Engine with Floating-Gate Storage in 0.13 μm CMOS”, IEEE Journal of Solid-State Circuits (Volume: 50, Issue: 1, Jan. 2015).

[80] Y. Chen and et al., “DaDianNao: A Machine-Learning Supercomputer,”, MICRO, 2014.

[81] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, V. Srikumar, “ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars,”, ISCA, 2016.

[82] P. Chi, S. Li, Z. Qi, P. Gu, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, “PRIME: A Novel Processing-In-Memory Architecture for Neural Network Computation in ReRAM-based Main Memory,”, ISCA, 2016.

[83] Takashi Morie and Yoshihito Amemiya, “An All-Analog Expandable Neural Network LSI with On-Chip Backpropagation Learning”, IEEE Journal of Solid-State Circuits, Vol. 29, No. 9, September, 1994.

[84] Arindam Basu, SunShuo, HongmingZhou, MengHiotLim, Guang-BinHuang, “Silicon spiking neurons for hardware implementation of extreme learning machines”, Neurocomputing, 102, pp.125–134, 2013.

[85] Jae-sun Seo et al, “A 45nm CMOS Neuromorphic Chip with a Scalable Architecture for Learning in Networks of Spiking Neurons”, Custom Integrated Circuits Conference (CICC), 2011 IEEE.

[86] Yu-Hsin Chen, Joel Emer, Vivienne Sze, “Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks”, Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium, 2016, ISSN: 1063-6897.

[87] Joe Osborne, (2016, Aug. 22), “Google’s Tensor Processing Unit Explained: This is What the Future of Computing Looks Like”. Techradar [Online] Available: http://www.techradar.com/news/computing-components/processors/google-s-tensor-processing-unit-explained-this-is-what-the-future-of-computing-looks-like-1326915.

[88] Kaz Sato, (2017, May 12), “An In-depth Look at Google’s First Tensor Processing Unit (TPU)”, Google Cloud Platform. [Online] Available: https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu.

[89] Google AI, “Cloud TPUs”. [Online] Available: https://ai.google/tools/cloud-tpus/.

AUTHOR

Pooja Jawandhiya was born in Nagpur, India on May 2, 1995. She received the Bachelor of Engineering degree in Electronics and Telecommunication from University of Mumbai in June, 2017. Currently, she is a student in Nanyang Technological University, Singapore and is pursuing Master of Science (Electronics) from the School of Electrical and Electronic Engineering.