Front. Comput. Sci., 2017, 11(5): 746–761
DOI 10.1007/s11704-016-6159-1

REVIEW ARTICLE

A survey of neural network accelerators

Zhen LI[1], Yuqing WANG[2], Tian ZHI[1], Tianshi CHEN (✉)[1]

1 State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China

2 School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China

Abstract Machine-learning techniques have recently been proved to be successful in various domains, especially in emerging commercial applications. As a set of machine-learning techniques, artificial neural networks (ANNs), requiring considerable amount of computation and memory, are one of the most popular algorithms and have been applied in a broad range of applications such as speech recognition, face identification, natural language processing, ect. Conventionally, as a straightforward way, conventional CPUs and GPUs are energy-inefficient due to their excessive effort for flexibility. According to the aforementioned situation, in recent years, many researchers have proposed a number of neural network accelerators to achieve high performance and low power consumption. Thus, the main purpose of this literature is to briefly review recent related works, as well as the DianNao-family accelerators. In summary, this review can serve as a reference for hardware researchers in the area of neural networks.

Keywords neural networks, accelerators, FPGAs, ASICs, DianNao series

1 Introduction

Designing a system with the ability to acquire their knowledge from raw input data to make it perform as if it thinks, is the domain called machine learning. The machine learning techniques allow computers to operate in complicated, real-world environments and make subjective decisions.

Received March 18, 2016; accepted September 18, 2016

E-mail: chentianshi@ict.ac.cn

The brain is a very fantastic artwork by the nature. It is able to live up to various amazing features: perception, action, learning, and cognition with little power and space. Inspired by mammalian central nervous systems, artificial neural networks (ANNs) are a family of models which imitate the connectivity of brain to solve certain problems, and they represent the world as a nested hierarchy of concepts which are capable of learning a greater amount of composition of either functions or concepts than traditional machine learning techniques.

The first conceptual model of artificial neural networks was developed by a neuroscientist, Warren S. McCulloch, and a logician, Walter Pitts in 1943 [1]. In their work, a neuron, the basic element in neural networks, is a cell that receives inputs, then processes inputs, and finally generates outputs. A neural network is a hierarchy computational model constructed by tens to millions of neurons.

Around seven decades have passed since the first neural network model, the MP model, were introduced by McCulloch and Pitts [1]. A great many researchers contributed to the development of neural networks. In 1949, Donald Hebb [2] proposed the first learning strategy for neural networks based on the synaptic physiology model. In 1958, Frank Rosenblatt [3] created the perceptron for pattern recognition and proved that a perceptron can come to convergence with supervised learning policy. The backpropagation [4], a classical supervised learning algorithm by Paul Werbos in 1975, pushed the steps forward neural networks. The state-of-art neural networks, such as deep learning [5, 6] and recurrent neural networks [7–9] (RNNs), attracted more researchers to the study of neural networks for their excellent performance.

A neural network is not simply a complex system with massive neurons, but a complex adaptive nonlinear information processing system which combines numerous processing units with a series of characteristics such as self-adapting, self-organizing and real-time learning [10]. It aims at estimating or approximating functions with massive inputs and uncertainties. During the past years, neural networks were widely used in various domains, such as medical diagnosis [11, 12], system identification and control (unmanned driving [13], trajectory prediction [14], natural resources management [15]), financial applications (like markets forecast [16, 17], decision system [18, 19] and commodity recommendation [20]). Neural networks have demonstrated their potential in fault tolerance [21–23], hypercomputation [24, 25], counter-intuitive to uninterpretable problems [26] properties compared to other applications and aroused great interest of investigating community.

Original neural networks were conducted on a CPU/GPU. However, with the exploding scale of networks, single CPU/GPU seemed to be insufficient for a network. The community turned to produce more powerful processors or to construct computing system with multiple processors.

Meanwhile, with the gate oxide scaling in nanometer semiconductor process, the gate oxide leakage is becoming a non-ignorable part of the overall chip power consumption. The channel doping concentration should keep increasing as the Dennard scaling [27] assumed to enable the voltage scaling, which in turn would cause the junction leakage to increase. Therefore, the power density of chip will ascend during device scaling, meaning that the Dennard scales to be broken down. The increase of power density makes it difficult for all the transistors to simultaneously switch in full frequency and the nominal voltage while the chip is in safe operating temperature range.

Furthermore, Esmaeilzadeh et al. [28] explored how much more performance can a multi-core path obtain with PARSEC benchmarks, ITRS scaling projections and empirical results from parallel workloads. Their study demonstrated that due to the power and parallelism constraints, at 8 nm, the amount of Dark Silicon (the amount of components of an integrated circuit that should be powered-off under given thermal design power constraint) may reach up to 50%–80%. Meanwhile, system would merely achieve best-case average speedup of 7.9 times in contrast to 32 times predicted by moore's law between 2011 and 2024.

The Dark Silicon phenomenon [28] has cast a shadow on using the general processor design to accelerate neural network algorithms. Hence, a broad consensus is to design specialized efficient co-processors to work along with general-purpose processors, which becomes an effective approach to improving system performance, i.e., combination of general-purpose processors and accelerators.

The rest of this paper is structured as follows. First, Section 2 briefly introduces a typical group of ANNs, state-of-art CNNs and DNNs , including the theory and implementation on general-purpose processors. Then, Section 3 catches a glimpse of efforts for supporting and accelerating conventional neural networks in the past decades. Section 4 reviews the DianNao-family accelerators. Finally, conclusions and discussions on the future work are made in Section 5.

2 Background

In this section, we firstly introduce the basic concepts of CNNs and DNNs, and then discuss both the achievements and limits on accelerating ANNs via conventional general-purpose processors (CPUs and GPUs).

2.1 Brief introduction to CNNs and DNNs

As two popular neural networks, convolutional neural networks (CNNs) and deep neural networks (DNNs) are members of multi-layer perceptrons (MLPs) family and generally contain four main layers: convolutional layer for characteristic identification of input data, pooling layer for dimensionally reducing input feature map, normalization layer for competing among neurons and their neighbors, and classifier layer at the end of the layer sequence for correlation between extracted features of last step and output categories of the whole network.

• **Convolutional layer** A convolutional layer uses a number of filters to extract characteristics from the input data. Each filter, named a kernel, consists of $K_x \times K_y$ coefficients which as a subset of the layer synaptic weights. Different filters cover different area of input feature maps by strides of S_x and S_y (x-and-y-direction respectively), and produce output neurons forming output feature maps (i.e., input feature maps of next layer). An output neuron at position (x, y) of the output feature map f_o is formally computed as follows:

$$Out_{a,b}^{f_o} = f\left(\sum_{f_i \in A_{f_o}} \left(\beta^{f_i,f_o} + \sum_{i=0}^{K_x-1}\sum_{j=0}^{K_y-1} \omega_{f_i,a,b,i,j}^{f_i,f_o} \times In_{aS_x+i,bS_y+j}^{f_i}\right)\right),$$

where $f(*)$ denotes a non-linear active function such as *sigmoid* and *tanh*, A_{f_o} indicates the input feature maps used to produce output feature map f_o, β^{f_i,f_o} and ω^{f_i,f_o} present the bias and kernel values between input feature map f_i and out-

put feature map f_o.

Neurons in a DNN convolutional layer have private synaptic weights (refer to the formula above), while in CNNs, neurons in the same output feature map share a common kernel so that the synaptic weight in the formula should be replaced with a simpler one: $\omega_{i,j}^{f_i,f_o}$. Due to this distinction, synaptic weights (i.e., parameters) in a DNN convolutional layer are drastically more (about $\frac{N_x}{S_x} \times \frac{N_y}{S_y}$ times) than those in a CNN layer with the same dimension.

• **Pooling layer** To aggregate information and reduce feature map dimensions, a pooling layer simply selects the maximum or computes the average from certain input neurons partitioned by non-overlapping pooling windows with size of $K_x \times K_y$. The formula of the maximum pooling output is:

$$Out_{a,b}^{f_o} = \max_{0 \leqslant i \leqslant K_x, 0 \leqslant j \leqslant K_y} In_{aK_x+i,bK_y+j}^{f_i},$$

where f_o equals f_i due to the one to one mapping relationship between input and output feature maps. If average pooling is computed, the maximum operator shown above should be changed into average operator. Moreover, a pooling layer has no parameter to learn because of its special operation, and is previously suggested to perform a non-linear active function but not now [29, 30].

• **Normalization layer** Normalization layers attempt to simulate lateral inhibition of biological neurons by performing competition among neurons in different input feature maps and at the same location. The effects of this competition and biological lateral inhibition are postulated to be similar by Krizhevsky et al. [31]. There are two normalization layer types, i.e., local contrast normalization (LCN) [29] and local response normalization (LRN) [31]. We only introduce the LRN as follows:

$$Out_{a,b}^{f_o} = In_{a,b}^{f_i} / \left(k + \alpha \times \sum_{j=\max(0,f_i-M/2)}^{\min(f_i-1,f_i+M/2)} \left(In_{a,b}^{j} \right)^2 \right)^{\beta},$$

where α, β and k are specified parameters and M is the number of neighbors beside the input feature map f_i.

• **Classifier layer** One or more classifier layers are generally appended at the end of the layer sequence consisted of interleaved previous three layer types. Each pair of input and output neurons in a typical classifier layer has a connection carrying an independent synaptic weight. In spite of the dramatic reduction of neurons in this layer (because of the effect of previous pooling layers), full connectivity still brings a great quantity of synaptic weights which consumes a high percentage of all parameters in the neuron network. Formally, the output neuron can be computed with:

$$Out^{n_o} = f\left(\beta^{n_o} + \sum_{n_i=0}^{N_i-1} \omega^{n_i,n_o} \times In^{n_i} \right),$$

where $f(*)$ is $\max(0, x)$ for ReLU [31] or an active function, β^{n_o} is the bias value owned by output neuron n_o, ω^{n_i,n_o} is the synaptic weight.

• **Inference and training** Practitioners generally use feedforward procedure for inference and backpropagation procedure for training. In a feedforward procedure, the first layer of the network reads a fixed-size input like an image, and passes its temporary result to the second layer. Then the data flow continues to go from one layer to its next until the last layer which produces a fixed-size output like scores of a set of selections. Conventionally, neurons in the middle layer (i.e., not the first or the last one) are called hidden units.

In backpropagation procedure, stochastic gradient descent (SGD) is mostly used. It consists of feeding several training sets to the input layer and fetching the outputs, computing the errors and gradients, averaging the gradients, and updating the weights accordingly. The computation of gradients is nothing more than computing the derivatives of objective function with respect to the network parameters such as weights and biases which can be calculated gradually from the output layer to the input layer with the help of chain rule. Before the objective function achieves the minimum point, the process is repeated for enough times over the whole training set.

2.2 Discussion on general-purpose processors

Conventionally, training and prediction phases of neural networks are performed on general-purpose processors like CPU [32, 33] and GPU [34–36]. There are also many kinds of tools which provide relatively complete programming environment for neural networks on CPU/GPU. Caffe [37] is one well-known platform providing an implementation of convolutional neural networks in CPU and GPU as well as the training algorithm. Cuda-convnet [38] is another tool which is able to parallelly process both feedforward pass (FP) and back propagation (BP) phase of CNNs on multiple CPUs or GPUs.

It is reasonable for researchers to seek better realization of neural networks. Vanhoucke et al. [33] achieved x4 speedup of neural network on CPUs over an aggressively optimized floating-point baseline at no cost in accuracy through some methodologies, including unrolling the loops and accumulating in parallel via SIMD instructions (SSE), aligning data to

gain better memory locality and consequently leverage SIMD instructions, quantizing the majority of the network down to 8 bits to shrink the total memory footprint, and evaluating the last layer in a lazy manner to reduce the visitation of the parameter space.

Generally, large clusters are more likely to be applied to meet the performance requirement of applications in large-scale networks. Google trained a deep network with billions of parameters using a framework named DistBelief [39] made up of tens of thousands of CPU cores. The mode was trained three days using a set of ten million 200×200 images downloaded from the Internet and achieved 15.8% accuracy of recognizing 22,000 objects on ImageNet database [40].

Compared with another common general-processor, namely GPU, CPU clusters waste a lot of resources on communication between cores even if they are candidates for sparse or networks beyond memory volume of GPUs. GPUs employ the fundamentally different SPMD (single program, multiple data) architectures and are specialized for intense, highly parallel computations. Considering these properties of these algorithms, GPUs exceed CPUs in CNNs and DNNs [41] applications with a high arithmetic density, that is, where the memory transactions is small compared to the arithmetic operations.

GPUs have not only more generality, but higher performance than CPUs. Oh and Jung [42] successfully accelerated ANNs via GPU by means of transferring dot product into matrix multiplication in 2004. Scherer et al. [35] implemented large-scale CNNs with CUDA framework. They assigned each training pattern to one block, each thread for one weight, and computed multiple feature maps or error signals in reverse to coalesce the memory accesses of different threads into one memory transaction that in which way dramatically improves the memory utilization. To reuse the input data or error signals as many times as possible they utilized the shared memory as a circular buffer to hold a small region of the source feature map.

Further, researches are conducted to fully explore how faster GPUs are able to accelerate machine learning algorithms over CPUs. Coates et al. [43] demonstrated that GPUs are capable of gaining over 90× speedup. Furthermore, several state-of-the-art GPUs cases from Teodoro et al. [44], Ciresan et al. [36], Liu et al. [45], and Chen et al. [46] respectively achieved 15−49×, 10−60×, 17.74× on average, 58.82× on average performance enhancement over SIMD CPUs for machine learning applications.

However, CPUs and GPUs are still inefficient in dealing with the deep learning. These general-purpose processors firstly take the universality into account. Lots of logics are used on universal computing, which in turn fairly limits the efficiency of core.

From perspective of arithmetical unit, general-purpose cores only support fundamental calculations considering flexibility. Complicated arithmetic operations are accomplished by a series of fundamental calculations, hence bringing about numerous dataflow between registers and memory and energy consumption. Chen el al. [46] observed that GPU is much more efficient than CPU on LRN layers, since it supports a dedicated exponential instruction but a CPU does not.

From a memory aspect, general-purpose processors in von Neumann architecture adopt hierarchical storage architecture. The cache capacity is usually smaller than other dedicated fabrics. The limited on-chip RAM (e.g., 1.5MB on NVIDIA K20M, no more than totally 50MB in Intel E7-8880L) compared to requirement of the state-of-art neural networks with 1-billion parameters [47], even with 10-billion parameters [48] inevitably leads to frequent long-latency access to external off-chip storage. In addition, the cache replacement between cache and external memory would further lower the cores' power efficiency.

3 Recent efforts in neural network accelerators

In this section, we present an overview of different platforms designed for accelerating neural networks during the past decades. All of these achievements can be roughly sorted into three categories, field-programmable gate array (FPGA) based realization, application specific integrated circuit (ASIC), and other implementations.

3.1 Field-programmable gate array (FPGA)

Field-programmable gate arrays (FPGAs) provideing large amount of logic resources are candidates for accelerating compute-intensive applications like CNNs which share the synapses in a convolution kernel. The programmability and reconfigurable characteristics of FPGAs allow a custom design to be evaluated in a short time, and thus shorten development period and save development expenses for a design. Researchers take FPGAs as candidates for neural network accelerators even if it is lower performance-energy efficient than ASICs.

Farabet et al. [49] implemented a RISC-like programmable ConvNet Processor which computes partial sums of different outputs with an identical input in parallel via multiple DSPs

and shifts inputs to accomplish convolutions of the receptive for an output on Xilinx Virtex-4 SX35 platform. The system realizes only one 2-D convolver due to resource constraint and does not fully take advantage of the intrinsic parallelism of CNNs for it convolutes the inputs in serial.

Later work by Farabet et al. [50] presented a scalable dataflow hardware architecture using a Xilinx Virtex 6 FPGA platform and providing ×100 speedup for real-world applications. The system contains multiple computational tiles. In a computational tile, there integrates multiple one-dimensional convolver, multiply-add accumulation (MAC), forming a 2-D convolution operator. Input-to-output blocks formed by cascading convolver and other operations elements with programmable wires which is adequate in FPGA are connected to a global bus through a routing multiplexer.

The nn-X by Gokhale et al. [51] is a scalable, low-power coprocessor for accelerating deep neural networks which is able to achieve a performance of 227 Gops/s theoretically and 200 Gops/s in practical applications while consuming 8 watts for the entire platform in which 4 watts for the core design. The core design prototyped on the Xilinx ZC706 platform comprises of two ARM Cortex-A9 CPUs as the host processor to parse a deep network then translate it to corresponding instructions and send data to the coprocessor, nn-X coprocessor formed by 8 collections (pipelined convolution engine, pooling module and non-linear operator) and AXI memory interconnect allowing four DMA channel to DDR3 memory.

Maashri et al. [52] developed a neuromorphic system for universal recognition. This system is based on HMAX, a bio-inspired neural network model for vision processing. The neuromorphic system includes certain multi-FPGA systems connected with Intel Xeon processors. It could attain a speedup of 7.6× and 4.3× over the CPU and GPU, and an energy reduction of 12.8× and 9.7× over the CPU and GPU platform respectively.

For CNN, the systolic architecture has a number of advantages [53] and disadvantages [54] such as: significant reuse of input data; simple processing element design; well-tuned pipeline for concurrency; relatively high performance to deal with computation intensive problems; simple data flow and regular control; expandability and modularization; insufficient flexibility in aspect of different CNN scales; memory bandwidth constrains. SCoNN [55] is a systolic implementation for inference phase of CNN. Parallel convolution processes are realized by multiple 2D array processing elements and one sliding window. To further save energy and memory bandwidth, SCoNN directly stores intermediate results into a compacted RAM and then uses them as input data of the next layer. These designs largely reduced memory accesses compared to [56, 57]. SCoNN was implemented on FPGA with max frequency of 132 MHz. Sankaradas et al. [58] proposed a coprocessor for CNN which is composed of systolic parallel convolution primitives and a controller dedicated to data movement. It has two key characteristics: one is the off-chip large-bandwidth memory banks serving for the coprocessor, the other is low data precision and packed data words in each memory operation. The coprocessor was mapped to FPGA and needed to work with a host together.

Cardells-Tormo et al. [59] presented three FPGA-based architectures for two dimensional convolution operations in image processing. These architectures need relatively less on-chip memory consumption so that they could be implemented in a low-cost FPGA device. In addition, a criterion about the maximum throughput per area is proposed to show the architecture efficiency.

Ordoñez-Cardenas et al. [60] used a low-cost Xilinx Spartan-3E FPGA to implement multi-layer perceptron (MLP) neural networks and the corresponding learning algorithms. A modular scheme is designed to make the system flexible to adapt to a specific application, and the pipeline architecture improves the system performance.

As the data size of CNN scales to larger and larger, external storages are used to store them. Thus the bandwidth requirement are exacerbated when the CNN goes into deeper, so researchers seek to find designs that take fully reuse of data in CNN.

Peemen et al. [61] presented a CNN accelerator based on FPGAs to reduce external memory bandwidth requirement. A flexible memory organization is designed to support data access and optimize data locality of CNN. Thus, this memory-centric accelerator could minimize on-chip memory requirement and maximize data utilization, so that it avoids unnecessary footprint and energy consumption.

Zhang et al. [62] implemented a 61.62 GFLOPS CNN accelerator on a VC707 FPGA board under 100MHz working frequency. Loop tiling and transformation techniques are used to quantitatively analyze the computation and bandwidth requirements of convolutional layers and then a unified configuration is explored with the help of roofline model.

Suda et al. [63] further developed Zhang's study, rather than focusing on the acceleration of convolutional layers, they presented a systematic design space exploration methodology to maximize the performance of an OpenCL-based FPGA accelerator for the entire CNN, considering the FPGA resource constraints.

Qiu et al. [64] proposed a FPGA based CNN acceler-

ator which employed SVD to the weight matrix of the full-connected layers to reduce memory access and adopted dynamic-precision data quantization method to reduce power and logic consumption. They also schemed the data of convolutional layers and that of full-connected layers differently based on the finding that the former is computational-centric and the latter is memory-centric. The design implemented on Xilinx Zynq ZC706 board achieved an average performance of 137.0 GOP/s for CNN under 150 MHz.

Even though FPGAs seem to be more suitable for CNNs, there exists attention on other networks. Rice et al. [65] reported a FPGA-based implementation of a hierarchical Bayesian network model based on the theoretical framework developed by George and Hawkins [66]. Results indicated that their hardware provides an average throughout gain of 75 times overs software fully running on Cray XD1. Another work by Kim et al. from Stanford [67] introduced a prototype of building a highly efficient Deep Belief Nets research machine on a single FPGA and achieved a speedup of 25–30 over an optimized software implementation on a temporal high-end CPU.

3.2 Application specific integrated circuit (ASIC)

The application specific integrated circuits (ASICs) are a group of custom circuits which bring designers the greatest degree of freedom to realize the design, and they count largely on the designers' determination on the scope of functions an ASIC support. To meet the specific needs of consumers, ASICs in batch production and universal integrated circuits used to be smaller volume, lower power consumption, reliability improved, higher performance, security enhanced and lower cost.

Convolution operation which is intrinsic suitable for parallel computing accounts for most of computation in networks such as CNNs. Early research for custom ANN accelerators mainly focuses on implementations of convolution operations as fast as possible to meet the requirements for real time processing.

Lee and Aggarwal [68] introduced a parallel 2-D convolution structure, a mesh connected array processor consisting of equal number of simple processing elements to pixel counts in the image, which could be arbitrary 2-D even 3-D convolution kernel size. The L64240 Multi-bit Filter by Stearns et al. [69] is a filter processor with 8×8 taps which is operational at a speed up to 20MHz. However, the L64240 lacks internal line delays and thus requires a second variable length video shift register chip. Kamp et al. [70] provided a fully integrated two-dimensional (2D) filter macrocell which offers a programmable 7×7 kernel only for vertically symmetric coefficient masks, operates at 20MHz pel-frequency and also contains internal line buffer of up to 512 pels per line. Hecht et al. [71] presented a defect tolerant systolic array implementation of the 2D convolution algorithm with maximum kernel size up to 256 taps and the ability to convolve one input signal with different masks in parallel which contains on-chip delay lines, special processing of frames borders and support of adaptive filtering. A very compact bit-level super-systolic array for 2D convolution which needs only 1-bit ports each input and output sequence rather than n-bit ports was proposed by Lee et al. [72], and generated two third area decrease without any time loss.

Similar attempt in other aspects like spiking neurosynaptic core by Merolla et al. [73] integrating 16×16 digital integrate-and-fire neurons grids and a 1024×256 bit SRAM crossbar memory for synapses in 4.2 mm^2 of silicon using IBM 45nm SOI process fits well with neural network algorithms, a restricted Boltzmann machine for digit recognition for instance. The core [73] uses an asynchronous event-driven scheme, request-acknowledge handshakes communication, and avoids high latency memory accesses by integrating synapses storage with neurons to lower total power. Besides, this hardware breaks the neural updates into two phases to keep functionally equivalence to software models and provide more flexibility for programmers.

Restricted by network parameters (convolutional kernel size, stride of kernel, etc), as well as the high memory bandwidth requirements, systolic architectures seem not to exhibit sufficient efficiency and flexibility in supporting different settings of NNs. Arithmetic components in NN accelerators are usually hungry for input data to perform computation at full speed. Feeding arithmetic components of adequate data is becoming more serious in NN accelerators based on the fact that the rise rate of bandwidth cannot catch up pace with that of transistor frequency.

The hardware accelerated convolutional neural networks by Farabet et al. [34] is a scalable hardware architecture designed for large-scale CNNs, which consists of a flexible Control Unit, usually a general CPU, to control the configuration bus, numerous independent processing tiles connected through routers to perform typical macroscopic operators of CNNs, and a Multi-port DMA Streaming Engine specifically allowing simultaneously multiple operations on images.

Kim et al. [74] introduced a 201.4 GOPS real-time multi-object recognition processor with power dissipation of 496 mW at 1.2V. The processor mimics operations of human

visual system and applies bio-inspired neural networks and fuzzy logic circuits to the neural perception engine with a three-stage pipeline (visual perception, descriptor generation and object decision).

Pham et al. [75] adapted the NeuFlow architecture, primitively designed for FPGA, to the IBM 45nm SOI process in Multi-Corner/Multi-Mode implementation for sake of design closure and power. The implemented chip dissipates 0.6W energy on average running at a system clock of 400MHZ and 1V core voltage, and achieves power efficiency of 490 GOPS/W in contrast to 14.7 GOPs/W of FPGA and 1.8 GOPs/W of GPU.

Esmaeilzadeh et al. [76] developed the Parrot transformation, an operation that replaces a region of imperative code with a well-trained neuron network running on an accelerator tightly coupled to the processor pipeline. With the help of the accelerator called neural processing unit (NPU), Parrot transformation achieves an average speedup of 2.3× and energy reduction of 3.0× on a set of different applications.

Esmaeilzadeh et al. [77] developed a neural network stream processing core (NnSP). The PE-array NnSP treats the computation of diverse applications as data streams with the help of compiler and FIFO-based caching scheme. Although it was developed for embedded systems, a 64-PE NnSP can perform at most 51.2 giga 32-bit fixed point operations per second with clock frequency of 400MHz and standard cell library of 0.18μm.

Qadeer et al. [78] proposed the convolution engine (CE), which can balance efficiency and flexibility in convolution-like data-flow processing. To be more energy efficient, CE improves utilization of data by exploring algorithm characteristics, reducing data transfers and operating as much as possible during each memory access.

There are also designs oriented to accelerate deep convolutional neural networks in more power-economy manners since it has shown state-of-the-art accuracy on many computer vision tasks. Sim et al. [79] from Korea Advanced Institute of Science and Technology presented a Deep CNN processor with 1.42 TOPS/W energy efficiency. To reduce the power consumption, they applied a dual-range multiply-accumulate (DRMAC) block for low-power convolution operations. They [79] also performed the convolution operations in a tiled-manner with the size of a tile equal to the on-chip storage and compressed kernel data to reduce off-chip memory expense and bandwidth requirement. Eyeriss by Chen et al. [80] which possesses a novel organization of computation units, memory hierarchy and on-chip network to exploit data reuse for different CNN shapes achieved 34.7fps on the five convolutional layers in AlexNet under 200MHz operating clock in 65nm CMOS. Park et al. [81] proposed a deep-learning-oriented processor, which achieved 411.3 GOPS peak performance with frequency of 200MHz and peak power of 213.1mW in 65nm CMOS, with 3 main features: a learning engine with a fine tuned pipeline, an inference engine with a systolic engine array, and a random number generator.

Besides accelerating neural networks in a economical manner, there is striving towards leveraging the intrinsic error resilience of neural networks in hardware accelerators. Hashmi et al. [82] proposed a biologically-inspired computation model and revealed the fault-tolerant intrinsic of relative cortical networks. It it a key principle for the model using a stuck-at scheme to preserve function results without requiring anything when hardware fault occurs. Temam [83] proposed and synthesized a hardware neural networks accelerator with tolerance to multiple defects based on the fact that ANNs are intrinsically tolerant to transient or permanent errors, which can potentially implement the computational kernels of some of the emerging high-performance tasks using alternative ANN-based algorithms, and, like custom circuits, which can achieve more than two orders of magnitude better energy efficiency than general-purpose cores. Du el at. [84] put forward an inexact neural network accelerator which achieved 43.91%–62.49% energy savings in 65nm technology.

3.3 Others

This section mainly exhibits several works in different models ranging from multi-core system, API, resistance array, and memristor. These studies are difficult to be completely classified into above types.

Neuro Turbo [85], a MIMD (multiple instruction and multiple data) type parallel ANN cccelerator, which is constructed of four general purpose 24 bits floating point digital signal processor (DSP) MB86220 and 4 dual port memories (DPM) gains 20 times performance promotion to the SUN4/260.

Khan et al. [86] presented the SpiNNaker system, a many-chip supercomputer. SpiNNaker is a chip which contains 20+ ARM9 processors and supports highly scalable neuron networks simulations. The goal [86] is to simulate a billion-neuron networks by a million-core system in real time, which means the system would run the same speed as a biological neuron collection with a similar size. A SpiNNaker system uses a packet-switching routing network which is asyn-

chronous and highly efficient to connect the processing cores, in order to be capable of the tremendous communication among neurons.

FACETS system [87] by the European research project FACETS is a wafer-scale integration of artificial neural network containing about 16k inputs and 60 million synapses in a single 20cm wafer with a power-density of 1.6W/cm^2. The system interconnects chips directly on the wafer rather than cuts into dies to simplify the printed-circuit boards design and packaging process.

A high abstraction-level software API by Chakradhar et al. [88] automatically analyzes the workload for each layer of the CNNs, finding the optimal mix of different types of parallelism under the scheduled off-chip bandwidth constraint, automatically translates the network specification written into a parallel microprogram and dynamically configures hardware resources during the runtime to match the precise complex control and data flows in the CNN workload as projected prior using a host processor.

Liu et al. proposed a neuromorphic computing accelerator named RENO [89] which leverages the efficient mixed-signal computation capability of the memristor crossbar arrays to speedup the executions of ANNs, including Brian-State-in-a-Box training scheme [90, 91], MLP [89], spiking network [92], and auto-associative memory [89]. Simulations show that the RENO achieves on average 178.4× performance speedup and 184.2× energy savings in MLP implementation than general-purpose processor, and respectively 27.06× and 25.23× that of high-accurate auto-associative memory.

4 The DianNao series accelerators

With the wide use of neural networks in various real-time scenes, it has been a hot spot seeking for schemes to speed up them. Conventional machine-learning accelerating methodologies mainly focused on efficiently implementing the computational part of the algorithms on general-purpose processors. However, restricted by the limited power and other constrains, accommodating machine-learning by way of other platforms is becoming more and more popular, and ASIC accelerators occupy an important part.

In this section, we review a series of neural network accelerators called DianNao series. The DianNao family has four members, DianNao, PuDianNao, DaDianNao, and ShiDianNao. Table 1 lists more detailed information about them.

4.1 DianNao

DianNao [93] is the first member in the DianNao family. This high-throughout coprocessor is capable of performing 452 GOP/s (multiplications and additions which are the predominant issues in neural networks) with a area consumption of 3.02mm^2 and 485mW energy dissipation at 65nm TSMC GP technology. The experiment results on several representative neural network benchmarks demonstrated that DianNao obtained over 100× performance bonus over a common CPU core with 1/30–1/5 power and area expense to that of CPU meaning that its energy efficiency comes up to three orders of magnitude. Although GPU competes DianNao in speed, its required power and area is about 100 times of DianNao.

DianNao mainly focuses on the accelerator on memory usage to fully feed massive arithmetic elements of data. The key challenge is the tradeoff between minimal memory transfers and higher performance for neural networks, which is determined by architecture parameters such as number of arithmetic units, memory policy, and the structure and amount of on-chip rams.

Since there are numerous parameters, it is impossible to estimate the entire chip simply using simulators. Chen et al. [93] proposed a heuristic modeling method based on the properties of neural network and prior works to deal with this trouble. The resulting core achieved perfect balance between computation and memory hierarchy, and dramatically improved the power efficiency for neural network.

Figure 1 draws the architecture of DianNao: the control logic (CP) controls an input buffer (NBin)and another buffer (SB) feed input neurons and synaptic weights to a computational block called the Neural Functional Unit (NFU), then an output buffer receives the output neurons from the NFU. A memory interface is applied to route the data stream of three buffers mentioned above to/from the memory.

The staggered 3-stage pipelined NFU [93] undertakes the computations of neural networks. Different layer types are

Table 1 Information of DianNao series accelerators

Designation	Technology/nm	Performance/GOP·s^{-1}	Power/W	Area/mm^2	Target application
DianNao	65	452	0.485	3.02	Neural networks
DaDianNao	28	5,584	15.97	67.73	Neural networks
PuDianNao	65	1,056	0.596	3.51	Machine learning
ShiDianNao	65	194	0.320	4.86	Convolutional neural networks

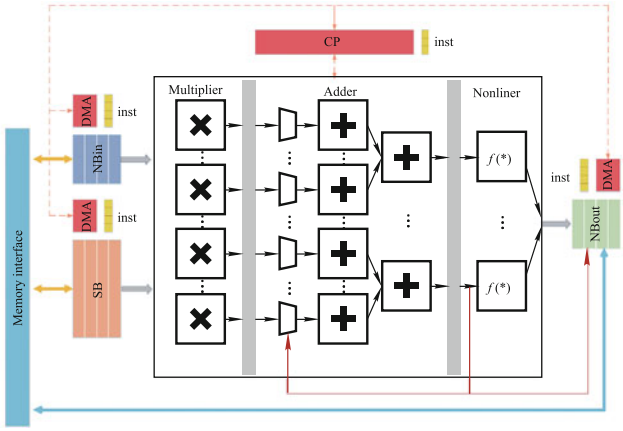


Fig. 1 Block diagram of DianNao

decomposed in either two or three stages on DianNao and pipelined under the control of CP. Multiple outputs are computed simultaneously to exploit data locality and reduce the number of access to buffers.

Even if the data has been loaded from memory on chip buffers, the energy overhead of data transferring is yet very high. Steve Keckler, the chief scientist of the Nvidia, has once concluded that the power consumption of transporting a 64-bit data for a distance of 20mm is several times of 64-bit float-point multiplication at 40nm technology node. Therefore, it is far from enough to merely lower down the power dissipation of computation to achieve higher efficiency, and optimizing data transferring policy is of equal importance.

The storage of DianNao was split into three blocks [93]: the input buffer (NBin), the output buffer (NBout) and the synapse buffer (SB). There are several benefits of splitting on-chip ram structures. Firstly, SRAMs can be tailored to the appropriate read/write width instead of identical width. Since the size of synapse weights is around an order of magnitude larger than input neurons and output neurons, this dedicated structure allows the best energy and time for read request. Secondly, Split storage and prior information of locality in neural networks allows DianNao to avoid data conflicts which would occurs in a cache and wastes time and energy to make up. Thirdly, DianNao is capable of rotating the NBin buffer working as a circle buffer to temporally reuse input neurons. Considering all the benefits above, DianNao reduces the data transfer transection to around 1/30–1/10 compared to that of CPU/GPU.

4.2 DaDianNao

In recent years, the deep neural network has received surprising achievements in the field of pattern recognition. However, the deep neural network is used to possess more hidden layers than custom neural networks, and thus the same situation to the exploding synapse weights. For instance, Baidu implemented a large-scale deep neural network with up to 20 billion synapses. Rapid neural network scale growth rate has brought challenges to all kinds of neural computational processors. It is difficult to meet the requirement of such large-scale networks using single or few cores. It is considerate for designers to extend uni-core accelerator to multi-core processor.

DaDianNao [46] is a multicore member of the DianNao family. On the basis of DianNao, DaDianNao further expanded the scale of the processor to 16 cores, shown as Fig. 2, and more on-chip memory to access high sustained machine-learning performance. In 28nm process, DaDianNao runs at a clock frequency of 606MHZ while the area overhead is 67.7mm^2 and energy consumption is around 16W.

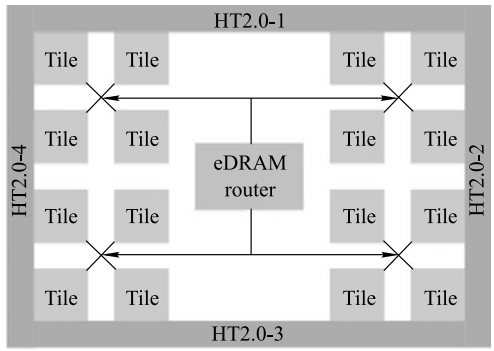


Fig. 2 Tile-based organization of DaDianNao (16 tiles connect to two central eDRAM banks via a fat-tree. There is also a router to manage data flow between HT2.0 interface and the eDRAMs)

DaDianNao is an accelerator aimed at broader support for machine-learning algorithms, not only the inference phase but also the training algorithms, even the weights pre-training phase (Restricted Boltzmann Machine, RBM). Compared to the Nvidia K20M GPU baseline, DaDianNao achieves a speedup of 450.65× over a GPU, and reduce the energy by 150.31× on average for a 64-chip system, and respectively 216.72× and 150× for a single chip with 16 nodes.

DaDianNao is organised in a tile-based [46] design, as shown in Fig. 2, to avoid logic/data congestion. All the computations of output neurons are split into 16 (equal to the number of tiles) segments corresponding to each tile. As shown in Fig. 3, each tile processes 16 input neurons of 16 output neurons at same time, i.e., 256 parallel operations in a single chip.

Similar to DianNao, DaDianNao adopts the split storage architecture. However, DaDianNao uses massive distributed eDRAMs [46] to locate all synapses close to computational

operators, serving as SB. Two reasons may be responsible for this scheme: on one side, there are much more synapse weights than neurons for MLP and convolutional layers with private kernels, it is of course more reasonable to move neurons other than synapses; on the other side, storing all the weights near computational units provides low-energy/low-latency data supply.

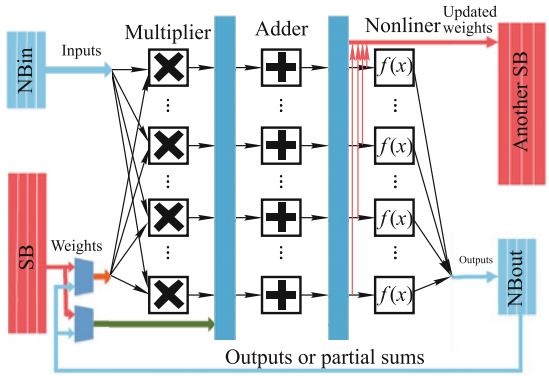


Fig. 3 NFU architecture of DaDianNao

A high internal bandwidth fat tree [46] is adopted to broadcast the identical input neurons values to each tile, and to collect different output neurons values from each tile. On the end of fat tree, there are two eDRAMs playing the similar roles as NBin and NBout in DianNao. Input neurons from an central eDRAM are broadcast to all tiles to compute multiple different outputs which are collected to another central eDRAM. To communicate with peer chips, the central eDRAMs are connected to HyperTransport (HT) 2.0 interface.

4.3 PuDianNao

Although neural network algorithm has already been an effective one in certain areas like pattern recognition, users would sometimes prefer other classical algorithms for better performance and higher accuracy because an algorithm will not always be the best in different domains. According to this observation, PuDianNao [45] is proposed, which accommodates various representative machine learning algorithms, i.e., *k*-nearest neighbors, naive bayes, *k*-means, linear regression, vector machine, deep neural network and classification tree. The architecture design of PuDianNao mainly needs to realize two difficult parts: execution unit and memory hierarchy, which are corresponding to computation and structure characteristics of machine learning respectively.

The main components of PuDianNao [45] are functional units, data buffers, a control module, an instruction buffer, as well as a DMA, as illustrated in Fig. 4. The starting point of functional unit design is to effectively perform the most frequent fundamental operations in machine learning techniques. A functional unit is divided into a machine learning functional unit (MLU) used for supporting several basic computational operations and a arithmetic logic unit (ALU) to complement MLU.

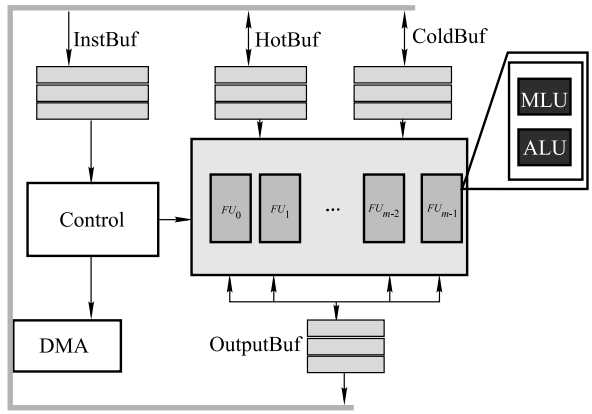


Fig. 4 Structure of PuDianNao

The MLU consists of six pipeline stages [45], as shown in Fig. 5. Red and blue lines mean the stage directly fetches input data from data buffers. Gray arrows indicate that the input data is the outputs of previous stage. A stage may be bypassed when it is unuseful. The counter stage accelerates counting operations by performing bitwise-AND or comparing input data and then accumulating the result. Counting operations are often used in classification tree and naive bayes. In the Adder stage, the common vector addition in machine learning is performed. The Multiplier stage operates vector multiplication and can receive input data from either previous stage or data buffers. The Adder tree stage sums multiplication results of pervious stage and the summation can be accumulated by Acc stage if needed. The two stages realize the dot product operation together with the Multiplier stage. At last, the Misc stage is responsible for *k*-sorter and linear interpolation operations, which are respectively used to find the smallest *k* values from Acc stage outputs and calculate approximate results of non-linear functions. Moreover, arithmetic units with staggered data bits are implemented across those six stages to save footprint and power consumption without obvious accuracy loss.

The ALU supports the miscellaneous operations in machine learning besides the computational primitives supported by MLU. Executing such operations on chips will further reduce bandwidth requirement, so a lightweight ALU with an adder, a divider, a multiplier and converters between 16-bit and 32-bit float is added into the functional unit.

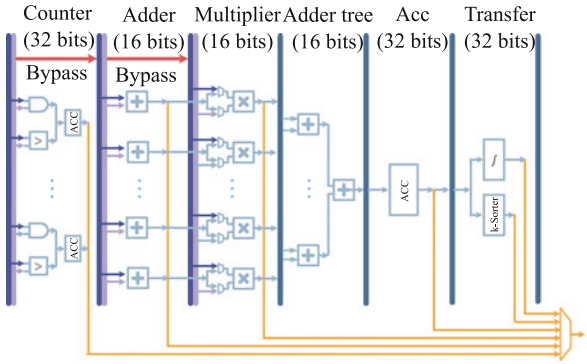


Fig. 5 Pipelined implementation of MLU in PuDianNao

The destination of memory hierarchy design is to maximize on-chip data utilization by exploring features of memory access, so that PuDianNao would require less memory bandwidth. Because tiling can utilize the locality of machine learning techniques effectively, the on-chip data buffers are divided into three separate components [45]: 8KB HotBuf, 16KB ColdBuf and 8KB OutputBuf. HotBuf is for the input data with short reuse distance, ColdBuf contrarily stores the long-reuse-distance input data and OutputBuf is for temporary or output data. This design brings two benefits. First, it accommodates the class number of clustered average reuse distances of variables in machine learning techniques, i.e., two or three classes. Second, it can eliminate the extra bandwidth overhead caused by different read width of data. Therefore, the buffer architecture successfully makes memory bandwidth avoid being critical bottleneck.

PuDianNao attains 1GHz frequency, peak performance of 1.056 trillion operations (addition, multiplication, etc.) per second, power consumption of 0.596W and area of 3.51mm^2 in 65nm CMOS. The average performance of those previous seven algorithms performed by PuDianNao is approximately equivalent to that of general-purpose GPU, but PuDianNao only requires about one percent power consumption of the latter.

4.4 ShiDianNao

ShiDianNao [54] is a full custom circuit aiming at processing image oriented convolutional neural networks in real time. The primary intention is implementing an energy-efficient visual recognition accelerator which is able to be embedded with any sensor and process realtime images. To achieve this goal, ShiDianNao not only maps an entire CNN with a SRAM to eliminate DRAM accesses of synapse weights but also gets input images directly from the CMOS or CCD sensors to further minimize data movements. On the whole, it achieves about 4700× and 60× energy efficiency than the GPU and DianNao.

The ShiDianNao accelerator includes four main components [54], see Fig. 6: a synapse buffer (SB), two buffers used to store input/output neurons (NBin, NBout), a neural functional unit (NFU) as well as an arithmetic logic unit (ALU) used to compute output neurons, a buffer and a decoder for instruction storage and decoding (IB). Specially, NFU is responsible for neural primitives (additions, multiplications and comparisons) and ALU dedicates to computing activation functions.

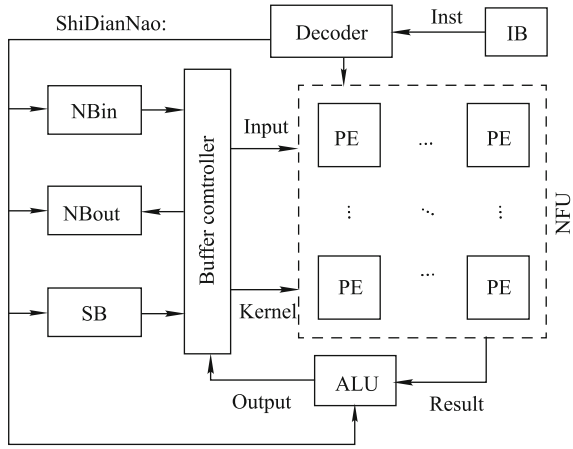


Fig. 6 Architecture of ShiDianNao

An NFU [54] consists of arrays of processing elements (PE), as illustrated in Fig. 6. Depending on the 2D nature of sliding window and limited kernel size in convolutional layer, processing elements have such characteristics: each represents a neuron, arranged in 2D mesh topology. It can also transmit data stored in FIFO to neighbours. As an addition to NFU, a small ALU is implemented to support 16-bit fix point operations such as division, and linear interpolation for active functions (e.g., sigmoid, tanh).

Here we cite the example provided in [54] to illustrate how the accelerator calculates a convolutional layer. Assume the size of PE array is 2×2 ,the kernel and step size of convolutional layer are 3×3 and 1×1 respectively, see Fig. 7. During the process of computing a feature map, every PE dedicates to a single output neuron and then moves to a new one after finishing computing. In the first computing cycle Cycle 0, all four PEs ($PE_{0,0}$, $PE_{0,1}$, $PE_{1,0}$, $PE_{1,1}$) respectively fetches the same kernel value $k_{0,0}$ from SB and their first input neuron ($x_{0,0}$, $x_{0,1}$, $x_{1,0}$, $x_{1,1}$) from NBin. Then the PEs multiply the input neuron and $k_{0,0}$, store the temporary result in registers and input neuron in FIFOs. In Cycles 1 and 2, $PE_{1,0}$ and $PE_{1,1}$ read their input neurons ($x_{2,0}$, $x_{2,1}$ in Cycle 1 and $x_{3,0}$, $x_{3,1}$ in Cycle 2) from NBin, $PE_{0,0}$ and $PE_{0,1}$ read their

input neurons ($x_{1,0}$, $x_{1,1}$ in Cycle 1 and $x_{2,0}$, $x_{2,1}$ in Cycle 2) from the FIFO of their right-hand PE ($PE_{1,0}$ and $PE_{1,1}$). At the same time, kernel values $k_{1,0}$ and $k_{2,0}$ are broadcasted and input neurons are stored into FIFOs. The multiplier results are accumulated with the previous temporary results. From Cycle 3 to Cycle 8, everything goes the same except that each PE reads required input neurons from its bottom-hand PE. Kernel values $k_{0,1} - k_{2,1}$ and $k_{0,2} - k_{2,2}$ are sequentially fetched from SB and broadcasted to all PEs, multiplication and accumulation are still preformed like before. $PE_{i,0}$ $(i = 0, 1)$ fetches input neurons $x_{i,1} - x_{i+2,1}$, $x_{i,2} - x_{i+2,2}$ from $PE_{i,1}$, and $PE_{i,1}$ $(i = 0, 1)$ reads required neurons $x_{i,2} - x_{i+2,2}$, $x_{i,3} - x_{i+2,3}$ from SB. Thus, each PE has finished their calculation and then sent the accumulation to the ALU. Output neurons $y_{0,0}$, $y_{1,0}$, $y_{0,1}$ and $y_{1,1}$ have been generated.

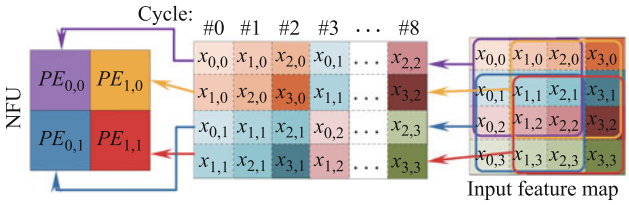


Fig. 7 The mapping between algorithm (convolutional layer) and hardware

The mapping principles of other neural layer types are similar to this example. From the description above, one can note that the inter-PE data propagation dramatically reduce memory bandwidth requirement between NFU and on-chip data buffers compared to the calculation process without such propagation. Due to the reduction of data movement, energy consumption also becomes less. Experimentally, the accelerator is capable of performing 194 GOP/s in a footprint of 4.86mm^2 in 65nm process while consuming 320.10mW at a clock frequency of 1GHZ. It achieves about respectively 50×, 30×, and 1.87× speedup than a mainstream CPU, a high-end GPU, and DianNao [93].

5 Conclusion and future work

With the advent of the big data era, the Internet daily produces huge amounts of data needed to be dealt with deep learning. More complicated neural networks are applied to solve various problems. Therefore, how to speedup the inference and training procedures of neural networks is becoming an emergent issue. In this article, we discuss the features of neural networks, and introduce some recent efforts of accelerating machine learning techniques.

Through the efforts of numerous researchers, significant progresses have been made in accelerating neural networks in economical ways (i.e., high performance while small footprint and low energy consumption). However, there are still many problems and challenges to be addressed. We believe that the following aspects are worth considering in the future work:

- **Exploiting better architectures** Obviously, seeking for architectures to better fit the neural networks is mainstream of accelerating them. This approach ranges from reducing main memory accesses, exploiting higher parallelism, and even structures to further reduce the transistor count.
- **Generalization** Narrow-domain accelerators are fairly fragile facing rapid changes of data characteristics and application scenarios. Coprocessors that make full use of the intercommunity of different kinds of neural networks, even may cause harm to the efficiency for certain algorithms which are no doubt of great importance.
- **CMOS process** Although limited by power and area constrains [28, 94], the expense for per performance gain would become higher and higher. Advanced manufacturing technology is still an obviously important way to promote the frequency of a processor, and thus promote the performance.
- **Efficient multi-node interconnection topology and protocols** Since a single chip has ultimately limited computational capacity, multi-node solutions could deal with large scale neuron networks better. More powerful interconnection topology for neural network applications automatically comes to be the inevitable choice.
- **Framework** Corresponding to the architecture and topology, new program frameworks (e.g., a custom programming language, a corresponding compiler, even integrated development environment) make it easier and more efficient to make full use of the accelerators.
- **Looking forward to next high-technology wave** Technological revolutions are usually accompanied by leaps in different domains. Hardware bio-inspired spiking neural network, quantum computer, and manufacturing process using new medium like the memristor are promising candidates.

Acknowledgements This work was partially supported by the National Natural Science Foundation of China (Grants Nos. 61100163, 61133004, 61222204, 61221062, 61303158, 61432016, 61472396, and 61473275), the National High Technology Research and Development Program (863 Program) of China (2012AA012202), the Strategic Priority Research Program of the CAS (XDA06010403), the Internatioanal Collaboration Key Program

of the CAS (171111KYSB20130002), and the 10,000 talent program.

References

1. McCulloch W S, Pitts W. A logical calculus of ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 1943, 5(4): 115–133
2. Hebb D O. The Organization of Behavior: A Neuropsychological Theory. London: Psychology Press, 2005
3. Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological Review, 1958, 65(6): 386
4. Werbos P. Beyond regression: new tools for prediction and analysis in the behavioral sciences. Dissertation for the Doctoral Degree. Cambridge, MA: Harvard University, 1974.
5. Hinton G E, Osindero S, Teh Y. A fast learning algorithm for deep belief nets. Neural Computation, 2006, 18(7): 1527–1554
6. Bengio Y. Learning deep architectures for AI. Foundations and Trends in Machine Learning, 2009, 2(1): 1–127
7. Williams R J, Zipser D. A learning algorithm for continually running fully recurrent neural networks. Neural Computation, 1989, 1(2): 270–280
8. Back A, Tsoi A. FIR and IIR synapses, a new neural network architecture for time series modeling. Neural Computation, 1991, 3(3): 375–385
9. Frasconi P, Gori M, Soda G. Local feedback multilayered networks. Neural Computation, 1992, 4(1): 120–130
10. Ding S F, Li H, Su C Y , Yu J Z, Jin F X. Evolutionary artificial neural networks: a review. Artificial Intelligence Review, 2013, 39(3): 251–260
11. Alneamy J S M, Alnaish R A H. Heart disease diagnosis utilizing hybrid fuzzy wavelet neural network and teaching learning based optimization algorithm. Advances in Artificial Neural Systems, 2014
12. Pereira L A M, Rodrigues D, Ribeiro P B, Papa J P, Weber S A T. Social-spider optimization-based artificial neural networks training and its applications for parkinson's disease identification. In: Proceedings of the 27th IEEE International Symposium on Computer-Based Medical Systems. 2014, 14–17
13. Harmon F G, Frank A A, Joshi S S. The control of a parallel hybrid-electric propulsion system for a small unmanned aerial vehicle using a cmac neural network. Neural Networks the Official Journal of the International Neural Network Society, 2005, 18(5-6): 772—780
14. Zissis D, Xidias E K, Lekkas D. A cloud based architecture capable of perceiving and predicting multiple vessel behaviour. Applied Soft Computing, 2015, 35: 652–661
15. Mishra A K, Desai V R. Drought forecasting using feed-forward recursive neural network. Ecological Modelling, 2006, 198(1-2): 127–138
16. Azoff E M. Neural Network Time Series Forecasting of Financial Markets. New York: John Wiley & Sons, Inc., 1994
17. Kaastra I, Boyd M. Designing a neural network for forecasting financial and economic time series. Neurocomputing, 1996, 10(3): 215–236
18. Tam K Y. Neural network models and the prediction of bank bankruptcy. Omega, 1991, 19(5): 429–445
19. West D, Dellana S, Qian J X. Neural network ensemble strategies for financial decision applications. Computers & Operations Research, 2005, 32(10): 2543–2559
20. Pokrajac D, Obradovic Z. A neural network-based method for site-specific fertilization recommendation. In: Proceedings of ASAE Annual Meeting. 2001
21. Protzel P W , Palumbo D L , Arras M K. Performance and fault-tolerance of neural networks for optimization. IEEE Transactions on Neural Networks, 1993, 4(4): 600–614
22. Chandra P, Singh Y. Fault tolerance of feedforward artificial neural networks- a framework of study. In: Proceedings of the International Joint Conference on Neural Networks. 2003, 489–494
23. Dias F M, Antunes A. Fault tolerance of artificial neural networks: an open discussion for a global model. International Journal of Circuits, Systems and Signal Processing, 2008, 329–333
24. Siegelmann H T, Sontag E D. Analog computation via neural networks. Theoretical Computer Science, 1994, 131(2): 331–360
25. Siegelmann H. Neural Networks and Analog Computation: Beyond the Turing Limit. Springer Science & Business Media, 2012
26. Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R. Intriguing properties of neural networks. 2013, arXiv preprint arXiv:1312.6199
27. Dennard R H, Rideout V L, Bassous E, LeBlanc A R. Design of ion-implanted mosfet's with very small physical dimensions. IEEE Journal of Solid-State Circuits, 1974, 9(5): 256–268
28. Esmaeilzadeh H, Blem E, Amant R S, Sankaralingam K, Burger D. Dark silicon and the end of multicore scaling. In: Proceedings of the 38th Annual International Symposium on Computer Architecture. 2011, 365–376
29. Jarrett K, Kavukcuoglu K, Ranzato M A, LeCun Y. What is the best multi-stage architecture for object recognition? In: Proceedings of the 12th IEEE International Conference on Computer Vision. 2009, 2146–2153
30. LeCun Y, Kavukcuoglu K, Farabet C. Convolutional networks and applications in vision. In: Proceedings of IEEE International Symposium on Circuits and Systems. 2010, 253–256
31. Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks. In: Proceedings of the Neural Information Processing Systems Conference. 2012, 1097–1105
32. Chakradhar S, Sankaradas M, Jakkula V, Cadambi S. A dynamically configurable coprocessor for convolutional neural networks. In: Proceedings of the 37th Annual International Symposium on Computer Architecture. 2010, 247–257
33. Vanhoucke V, Senior A, Mao M Z. Improving the speed of neural networks on cpus. In: Proceedings of Deep Learning and Unsupervised Feature Learning NIPS Workshop. 2011
34. Farabet C, Martini B, Akselrod P, Talay S. Hardware accelerated convolutional neural networks for synthetic vision systems. In: Proceedings of IEEE International Symposium on Circuits and Systems. 2010, 257–260
35. Scherer D, Schulz H, Behnke S. Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors. In: Proceed-

ings of International Conference on Artificial Neural Networks. 2010, 82–91

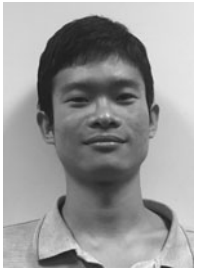
36. Ciresan D C, Meier U, Masci J, Gambardella L M, Schmidhuber J. Flexible, high performance convolutional neural networks for image classification. In: Proceedings of International Joint Conference on Artificial Intelligence. 2011
37. Jia Y Q, Shelhamer E, Donahue J, Karayev S, Long J H, Girshick R, Guadarrama S, Darrell T. Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM International Conference on Multimedia. 2014, 675–678
38. Krizhevsky A. One weird trick for parallelizing convolutional neural networks. 2014, arXiv preprint arXiv:1404.5997
39. Dean J, Corrado G, Monga R, Chen K, Devin M, Mao M, Senior A, Tucker P, Yang K, Le Q V, Ng A Y. Large scale distributed deep networks. In: Proceedings of the Neural Information Processing Systems Conference. 2012, 1223–1231
40. Deng J, Dong W, Socher R, Li L J, Li K, Li F F. Imagenet: a large-scale hierarchical image database. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. 2009, 248–255
41. Ciresan D, Meier U, Schmidhuber J. Multi-column deep neural networks for image classification. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. 2012, 3642–3649
42. Oh K S, Jung K. GPU implementation of neural networks. Pattern Recognition, 2004, 37(6): 1311–1314
43. Coates A, Baumstarck P, Le Q, Ng A Y. Scalable learning for object detection with gpu hardware. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. 2009, 4287–4293
44. Teodoro G, Sachetto R, Sertel O, Gurcan M N, Meira W, Catalyurek U, Ferreira R. Coordinating the use of GPU and CPU for improving performance of compute intensive applications. In: Proceedings of IEEE International Conference on Cluster Computing and Workshops. 2009, 1–10
45. Liu D F, Chen T S, Liu S L, Zhou J H, Zhou S Y, Teman O, Feng X B, Zhou X H, Chen Y J. PuDianNao: a polyvalent machine learning accelerator. In: Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems. 2015, 369–381
46. Chen Y J, Luo T, Liu S L, Zhang S J, He L Q, Wang J, Li L, Chen T S, Xu Z W, Sun N H, Temam O. DaDianNao: a machine-learning supercomputer. In: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture. 2014, 609–622
47. Le Q V, Ranzato M A, Monga R, Devin M, Chen K, Corrado G S, Dean J, Ng A Y. Building high-level features using large scale unsupervised learning. In: Proceedings of the International Conference on Machine Learning. 2011
48. Coates A, Huval B, Wang T, Wu D, Catanzaro B, Ng A Y. Deep learning with COTS HPC systems. In: Proceedings of the 30th International Conference on Machine Learning. 2013, 1337–1345
49. Farabet C, Poulet C, Han J F, LeCun Y. CNP: an FPGA-based processor for convolutional networks. In: Proceedings of IEEE International Conference on Field Programmable Logic and Applications. 2009, 32–37
50. Farabet C, Martini B, Corda B, Akselrod P, Culurciello E, LeCun Y. Neuflow: a runtime reconfigurable dataflow processor for vision. In: Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. 2011, 109–116
51. Gokhale V, Jin J, Dundar A, Martini B, Culurciello E. A 240 G-ops/s mobile coprocessor for deep neural networks. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition Workshops. 2014, 682–687
52. Maashri A A, Debole M, Cotter M, Chandramoorthy N, Xiao Y, Narayanan V, Chakrabarti C. Accelerating neuromorphic vision algorithms for recognition. In: Proceedings of the 49th Annual Design Automation Conference. 2012, 579–584
53. Kung H T. Why systolic architectures? IEEE Computer, 1982, 15(1): 37–46
54. Du Z D, Fasthuber R, Chen T S, Ienne P, Li l, Luo T, Feng X B, Chen Y J, Temam O. ShiDianNao: shifting vision processing closer to the sensor. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture. 2015, 92–104
55. Dawwd S A. The multi 2D systolic design and implementation of convolutional neural networks. In: Proceedings of the 20th IEEE International Conference on Electronics, Circuits, and Systems. 2013, 221–224
56. Draper B A, Beveridge J R, Bohm A P W, Ross C, Chawathe M. Accelerated image processing on FPGAs. IEEE Transactions on Image Processing, 2003, 12(12): 1543–1551
57. Dawwd S A, Mahmood B S. A reconfigurable interconnected filter for face recognition based on convolution neural network. In: Proceedings of the 4th International Conference on Design and Test Workshop. 2009, 1–6
58. Sankaradas M, Jakkula V, Cadambi S, Chakradhar S, Durdanovic I, Cosatto E, Graf H P. A massively parallel coprocessor for convolutional neural networks. In: Proceedings of the 20th IEEE International Conference on Application-specific Systems, Architectures and Processors. 2009, 53–60
59. Cardells-Tormo F, Molinet P L. Area-efficient 2-D shift-variant convolvers for FPGA-based digital image processing. In: Proceedings of IEEE Workshop on Signal Processing Systems Design and Implementation. 2005, 209–213
60. Ordo nez-Cardenas E, Romero-Troncoso R D J. MLP neural network and on-line backpropagation learning implementation in a low-cost FPGA. In: Proceedings of the 18th ACM Great Lakes symposium on VLSI. 2008, 333–338
61. Peemen M, Setio A A, Mesman B, Corporaal H. Memory-centric accelerator design for convolutional neural networks. In: Proceedings of the 31st IEEE International Conference on Computer Design. 2013, 13–19
62. Zhang C, Li P, Sun G Y, Guan Y J, Xiao B J, Cong J S. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2015, 161–170
63. Suda N, Chandra V, Dasika G, Mohanty A, Ma Y F, Vrudhula S, Seo J, Cao Y. Throughput-optimized opencl-based FPGA accelerator for large-scale convolutional neural networks. In: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2016, 16–25

64. Qiu J T, Wang J, Yao S, Guo K Y, Li B X, Zhou E, Yu J C, Tang T Q, Xu N Y, Song S, Wang Y, Yang H Z. Going deeper with embedded FPGA platform for convolutional neural network. In: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2016, 26–35
65. Rice K L, Taha T M, Vutsinas C N. Scaling analysis of a neocortex inspired cognitive model on the Cray XD1. The Journal of Supercomputing, 2009, 47(1): 21–43
66. George D, Hawkins J. A hierarchical bayesian model of invariant pattern recognition in the visual cortex. In: Proceedings of IEEE International Joint Conference on Neural Networks. 2005, 1812–1817
67. Kim S K, McAfee L C, McMahon P L, Olukotun K. A highly scalable restricted boltzmann machine FPGA implementation. In: Proceedings of IEEE International Conference on Field Programmable Logic and Applications. 2009, 367–372
68. Lee S Y, Aggarwal J K. Parallel 2-D convolution on a mesh connected array processor. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1987, (4): 590–594
69. Stearns C C, Luthi D A, Ruetz P A, Ang P H. A reconfigurable 64-tap transversal filter. In: Proceedings of the IEEE Custom Integrated Circuits Conference. 1988
70. Kamp W, Künemund R, Söldner H, Hofer R. Programmable 2D linear filter for video applications. IEEE Journal of Solid-State Circuits, 1990, 25(3): 735–740
71. Hecht V, Ronner K. An advanced programmable 2D-convolution chip for, real time image processing. In: Proceedings of IEEE International Sympoisum on Circuits and Systems. 1991, 1897–1900
72. Lee J J, Song G Y. Super-systolic array for 2D convolution. In: Proceedings of IEEE Region 10 Conference. 2006, 1–4
73. Merolla P, Arthur J, Akopyan F, Imam N, Manohar R, Modha D S. A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. In: Proceedings of IEEE Custom Integrated Circuits Conference. 2011, 1–4
74. Kim J Y, Kim M, Lee S J, Oh J, Kim K, Yoo H J. A 201.4 GOPS 496 mW real-time multi-object recognition processor with bio-inspired neural perception engine. IEEE Journal of Solid-State Circuits, 2010, 45(1): 32–45
75. Pham P H, Jelaca D, Farabet C, Martini B, LeCun Y, Culurciello E. Neuflow: dataflow vision processing system-on-a-chip. In: Proceedings of the 55th IEEE International Midwest Symposium on Circuits and Systems. 2012, 1044–1047
76. Esmaeilzadeh H, Sampson A, Ceze L, Burger D. Neural acceleration for general-purpose approximate programs. In: Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture. 2012, 449–460
77. Esmaeilzadeh H, Saeedi P, Araabi B N, Lucas C, Fakhraie S M. Neural network stream processing core (NnSP) for embedded systems. In: Proceedings of IEEE International Symposium on Circuits and Systems. 2006
78. Qadeer W, Hameed R, Shacham O, Venkatesan P, Kozyrakis C, Horowitz M A. Convolution engine: balancing efficiency & flexibility in specialized computing. In: Proceedings of the 40th Annual International Symposium on Computer Architecture. 2013, 24–35
79. Sim J, Park J S, Kim M, Bae D, Choi Y, Kim L S. 14.6 a 1.42 tops/w deep convolutional neural network recognition processor for intelligent IoE systems. In: Proceedings of IEEE International Solid-State Circuits Conference. 2016, 264–265
80. Chen Y H, Krishna T, Emer J, Sze V. 14.5 eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. In: Proceedings of IEEE International Solid-State Circuits Conference. 2016, 262–263
81. Park S, Bong K, Shin D, Lee J, Choi S, Yoo H J. 4.6 A1. 93TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications. In: Proceedings of IEEE International Solid-State Circuits Conference. 2015, 1–3
82. Hashmi A, Berry H, Temam O, Lipasti M. Automatic abstraction and fault tolerance in cortical microachitectures. In: Proceedings of the 38th Annual International Symposium on Computer Architecture. 2011, 1–10
83. Temam O. A defect-tolerant accelerator for emerging high-performance applications. ACM SIGARCH Computer Architecture News, 2012, 40(3): 356–367
84. Du Z D, Lingamneni A, Chen Y J, Palem K, Temam O, Wu C Y. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In: Proceedings of the 19th Asia and South Pacific Design Automation Conference. 2014, 201–206
85. Iwata A, Yoshida Y, Matsuda S, Sato Y, Suzumura N. An artificial neural network accelerator using general purpose 24 bit floating point digital signal processors. In: Proceedings of the International Joint Conference on Neural Networks. 1989, 171–175
86. Khan M M, Lester D R, Plana L A, Rast A, Jin X, Painkras E, Furber S B. SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor. In: Proceedings of IEEE International Joint Conference on Neural Networks. 2008, 2849–2856
87. Schemmel J, Fieres J, Meier K. Wafer-scale integration of analog neural networks. In: Proceedings of IEEE International Joint Conference on Neural Networks. 2008, 431–438
88. Chakradhar S, Sankaradas M, Jakkula V, Cadambi S. A dynamically configurable coprocessor for convolutional neural networks. In: Proceedings of the 37th Annual International Symposium on Computer Architecture. 2010, 247–257
89. Liu X X, Mao M J, Liu B Y, Li H, Chen Y R, Li B X, Wang Y, Jiang H, Barnell M, Wu Q, Yang J H. RENO: a high-efficient reconfigurable neuromorphic computing accelerator design. In: Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference. 2015, 1–6
90. Hu M, Li H, Chen Y R, Wu Q, Rose G S. Bsb training scheme implementation on memristor-based circuit. In: Proceedings of IEEE Symposium on Computational Intelligence for Security and Defense Applications. 2013, 80–87
91. Hu M, Li H, Wu Q, Rose G. Hardware realization of neuromorphic BSB model with memristor crossbar network. In: Proceedings of IEEE Design Automation Conference. 2012, 554–559
92. Afifi A, Ayatollahi A, Raissi F. Implementation of biologically plausible spiking neural network models on the memristor crossbar-based CMOS/nano circuits. In: Proceedings of European Conference on Circuit Theory and Design. 2009, 563–566
93. Chen T S, Du Z D, Sun N H, Wang J, Wu C Y, Chen Y J, Temam O.

DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. ACM SIGPLAN Notices, 2014, 49(4): 269–284

94. Muller M. Dark silicon and the Internet. In: Proceedings of EE Times "Designing with ARM" Virtual Conference. 2010, 285–288

Zhen Li received the BS degree in physics from Special Class for the Gifted Young, University of Science and Technology of China, China in 2014. He is currently a PhD candidate in Institute of Computing Technology, Chinese Academy of Sciences, China. His research interests include approximate computing, neuron networks accelerators and computer architecture.



Yuqing Wang received the BS degree in computer science from School of Computer Science and Technology, University of Science and Technology of China (USTC), China in 2015. He is currently a PhD candidate in USTC. His major research interests include neural network accelerators and optimization of corresponding compiler.

Tian Zhi received the BS degree of engineering from Zhejiang University, China, and the PhD degree in computer science from the Institute of Microelectronics of the Chinese Academy of Sciences, China in 2009 and 2014, respectively. Her current research interests include computer architecture, reconfigurable computing, and hardware neural networks.

Tianshi Chen received the BS degree in mathematics from Special Class for the Gifted Young and the PhD degree in computer science from School of Computer Science and Technology, University of Science and Technology of China, China in 2005 and 2010, respectively. He is currently a researcher in Institute of Computing Technology, Chinese Academy of Sciences. His research interests lie in computer architecture and computational intelligence. He is an awardee of the NSFC Excellent Young Scholars Program in 2015.