REVIEWS

Journal of Semiconductors
(2020) 41, 021403
doi: 10.1088/1674-4926/41/2/021403

A survey of neural network accelerator with software development environments

Jin Song[1, 2, 3], Xuemeng Wang[3, 4], Zhipeng Zhao[3, 4], Wei Li[1], and Tian Zhi[1, †]

[1]SKL of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[2]University of Chinese Academy of Sciences, Beijing 100049, China
[3]Cambricon Tech. Ltd, Beijing 100191, China
[4]University of Science and Technology of China, Hefei 230026, China

Abstract: Recent years, the deep learning algorithm has been widely deployed from cloud servers to terminal units. And researchers proposed various neural network accelerators and software development environments. In this article, we have reviewed the representative neural network accelerators. As an entirety, the corresponding software stack must consider the hardware architecture of the specific accelerator to enhance the end-to-end performance. And we summarize the programming environments of neural network accelerators and optimizations in software stack. Finally, we comment the future trend of neural network accelerator and programming environments.

Key words: neural network accelerator; compiling optimization; programming environments

Citation: J Song, X M Wang, Z P Zhao, W Li, and T Zhi, A survey of neural network accelerator with software development environments[J]. *J. Semicond.*, 2020, 41(2), 021403. http://doi.org/10.1088/1674-4926/41/2/021403

1. Introduction

Recent years, artificial intelligence (AI) developed at an explosive speed. With the great progress made by the neural network (NN) algorithms, the application scenarios also widely spread, including image processing[1, 2], object detection[3, 4], speech recognition[5, 6] and natural language processing[7, 8] and many other fields. Furthermore, the dimensionality of NNs goes deeper[9] and the amount of computations increases exponentially, which brings severe challenge to traditional computing architectures. Therefore, different types of NN accelerator have emerged continuously. This article aims to review the development of NN accelerators and their programming environments

Inspired by biological NNs, artificial NNs are proposed to solve artificial intelligent problems. Initially, McCulloch and Pitts proposed the concept, that a single neuron, the basic element in NN, receives inputs, processes and generates outputs[10]. In 1958, Rosenblatt created the perceptron for pattern recognition[11]. With the supervised learning policy, the perceptron is proved to be convergent. After that, the backpropagation algorithm and multilayer perceptron[12] are proposed and push forward the development of the NN research.

Recently, with the continuous decline of process nodes, the deep learning[13] concept is proposed by Hinton in 2006. Afterwards such hierarchy computing systems can perform a fair accuracy in some AI tasks. Since AlexNet[14] achieved 15.3% top-5 error rate on ILSVRC-2012 database, more and more deep learning methods began to show advantages in computer vision fields.

However, a deeper NN model may not perform better than a shallower model. Besides the increase of computation, the gradient vanishment also affects the training effect. By adjusting the structure of NN model, the training effect, convergence speed and model accuracy can be increased. He *et al.* developed residual blocks[15]. The inception structure has improved after several versions through engineering experience and experiment[16, 17].

And with the efforts of the researchers, there is not only the structure of the NN model has improved, but the computation pattern of the NN layer also evolved. Yu and Koltun[18] proposed dilated convolution to solve multi-scale recognition problems in test database. Mamalet and Garcia[19] introduced various strategies to simplify filters that used as feature extractors learnt in CNNs, so as to modify the hypothesis space and speed up processing. Howard *et al.* presented an effective NN model for mobile devices and embedded visual applications, called MobileNets[20]. Its architecture is streamline-based which takes depth-wise separable convolutions to construct lightweight deep neural network (DNN) model. These fixed combination structures of NN models and optimization of algorithm (Fig. 1), could also inspire the development of NN accelerator systems, from bottom hardware to above software.

In the network structure design, in addition to the sequential execution of the direct connection of the NN layer, there are also ring topology. The recurrent neural network (RNN) is a class of NN model with recurrent connections. And due to the ring topology and internal state of the cyclic structure, it has significance on processing and predicting sequential data by overcoming many limitations of input and output data in traditional NN algorithms.

But after many layers of RNNs, the gradient tends to vanish in most cases. Long-short time memory (LSTM) network is

Correspondence to: T Zhi, zhitian@ict.ac.cn
Received 23 OCTOBER 2019; Revised 18 DECEMBER 2019.

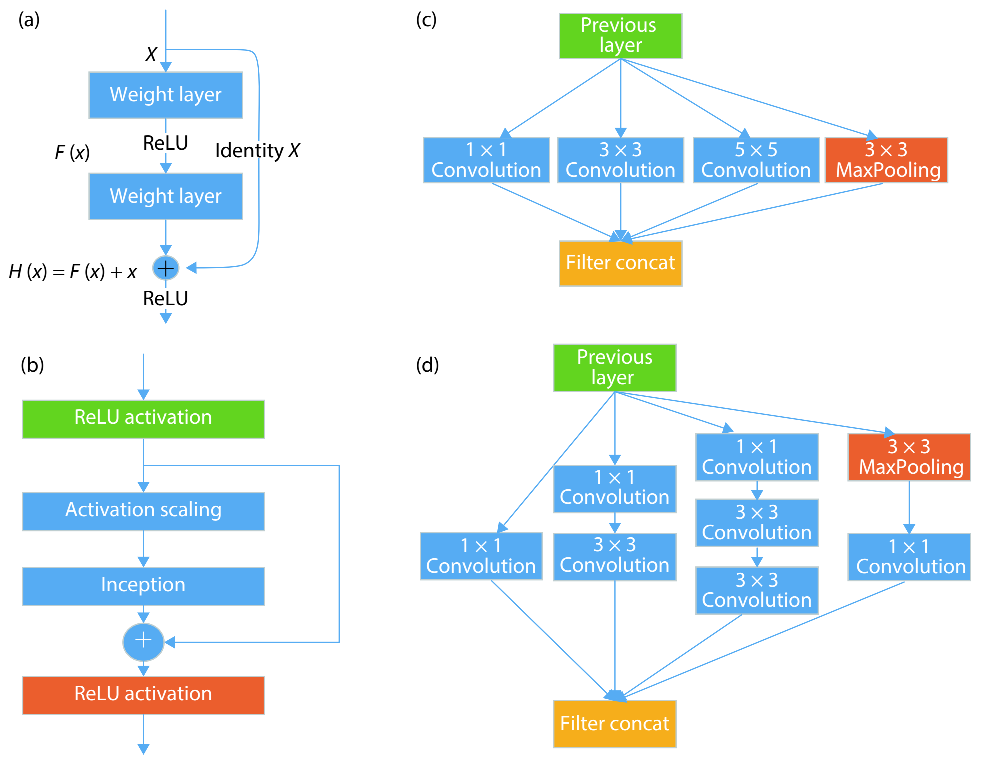

Fig. 1. (Color online) Classical CNN model architectures. There are four fixed combination of layers in the figure. Among them, (a) stands for residual net in ResNet series networks, (b) expresses Inception-ResNet combination structure, (c) represents naïve inception structure, and (d) shows an upgraded version of inception with dimension reduction feature.

a widely used recurrent structure network architecture in practical applications, which was proposed by Hochreiter[21] to improve the problems existing in the practical application of RNN and realize the long-term preservation of information. LSTM structure has three gates, input gate, forget gate and output gate, to control state and output at different time. LSTM combines short-term memory with long-term memory through a gate structure to alleviate the problem of gradient vanishment. Another popular variant of LSTM unit is a simplified structure called gated recurrent unit (GRU) proposed by Cho *et al.*[22]. GRU only has two gates, namely update gate and reset gate. It gets rid of cell state and uses hidden state to transmit information. Vaswani *et al.* proposed a new simple network architecture, the transformer, based solely on self-attention mechanisms, dispensing with recurrence and convolutions entirely[23].

In addition, transformer can increase to a very deep depth, fully exploit the characteristics of DNN model, require significantly less time to train, and improve the accuracy of the model. In 2018, Devlin *et al.* proposed bidirectional encoder representations from transformer (BERT)[24]. BERT can pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers, and the BERT can be fine-tuned with just one additional output layer to create state-of-the-art models for many other tasks without modifying substantial task-specific architecture. BERT outperforms previous methods because it is the first unsupervised, deeply bidirectional system for pre-training in NLP. While its model size is too large that it is still a challenge in software platform to train from random value of weight initialization.

From a functional point of view, the convolution layers are usually placed at the front of the NN model to extract features. The number of channels is increased by sliding the filters over the input data, doing multiplication and addition. The pooling layer is usually following after convolution layers to reduce spatial dimension information, to avoid over-fitting, and improve the fault tolerance of the NN model. By exploiting data reuse pattern and calculation order of the NN layer in the network structure, a corresponding optimization method can be designed in hardware accelerator or software algorithm. And from a computation point of view, most part of computation in a NN model is occupied by the multiplications and additions in convolution layers and in fully-connected layers. Naturally, accelerating these types of layers is a key point to reduce the execution time of the whole network.

From the above methods, in the past few decades, lots of researchers have proposed many NN accelerator architectures, and put efforts from algorithm to hardware. From algorithm view, researchers use methods such as sparseness[25], pruning[26], quantization bit width[27], matrix decomposition[28], and entropy coding[29] to compress data and reduce computation to accelerate NN computation. In terms of hardware, researchers propose parallelization[30], self-organizing feature maps[31] and other methods[32–34] to design neural-network-specific accelerators, reduce execution time, and improve computational efficiency. Section 2 reviews several existing NN accelerators and programmable hardware design.

From a point of software view, programming system of NN accelerators includes the programming method of NNs, compilation and optimization. The design of the NN accelerator software stack is a bridge between programmers and the underlying hardware. Many deep learning frameworks have put efforts to simplify the deployment of NN algorithms on NN accelerators and maximize the performance end-to-end. Some frameworks are versatile and some are designed for NN accelerators. Section 3 reviews the detailed design of software stack of NN accelerator.

Section 4 comments the challenges and future trends of the development and implementation of the NN accelerator whole programming system. At the end, we summarize in Section 5.

2. Neural network accelerator and programmable hardware design

2.1. Neural network accelerator

In 2012, Chen *et al.*[35] demonstrated that hardware NN accelerators can have potential broad applications by developing and evaluating software NN implementations of several recognition, mining, and synthesis (RMS) applications from the PARSEC suite. As dark silicon age has already come, chips can no longer rely on simply increasing the operational core counts to improve performance without surpassing a reasonable power budget. So, accelerators targeting an application or an application domain seems quite promising. And the results show that a hardware NN accelerators are indeed compatible with many of the emerging high-performance workloads.

Researchers have proposed diverse accelerator schemes by utilizing the characteristics of the computing patterns in NN algorithms.

Some accelerators propose lower-bit precision computation and sparse representation. In 2009, Farabet *et al.* proposed an FPGA-based convolutional NN accelerator CNP[36] that uses 18-bit vertex data to implement almost all convolutional network operations. The implementation takes full advantage of multiple hardware multiply-accumulate units on the FPGA. And a software compiler is also implemented to take the description of trained CNN model and compile it into a sequence of instructions. This CNP system can be used for low-power and lightweight embedded vision systems. In 2016, Zhang *et al.* proposed Cambricon-X[37], which exploits the sparsity and irregularity of sparse NN models for efficiency. According to the computing mode and memory access characteristics of sparse NNs, Cambricon-X designs dedicated neuron processing elements (PE) and indexing module (IM) to select the neurons that need to be computed, and then achieves high performance and energy efficient NN acceleration under the limited bandwidth requirements. In 2019, a more recent study showed a new quantization method with mixed data structure and bit-shifting broadcast accelerator structure BSHIFT, which reduces the storage requirement of NNs models from 32 to 5 bits without affecting their accuracy[38].

Some accelerators design pipeline structure for NN. In 2016, Shafiee *et al.* designed a pipelined architecture, define new data encoding techniques and many supporting digital components, exploring the balance between memristors, ADCs, and eDRAMs[39]. In 2017, Chen *et al.* optimizes for the energy efficiency of the entire system. The accelerator chip and off-chip DRAM, focus on the adaptive dataflow for various CNN shapes by reconfiguring the architecture, which can minimize the on-chip data and memory[40]. Google released the first tensor processor units (TPU) that has been used in the Google data center for two years[41]. It uses a dedicated matrix unit to perform matrix multiply and convolution, an activation unit to perform nonlinear functions and a programmable DMA controller to transfer data. TPU leverages advantage in MACs and on-chip memory. On specific TensorFlow framework, it runs 15 times as fast as the K80 GPU, and 29 times in performance per Watt. In the same year, the second version of TPU was introduced. The calculation of the provided floating-point operations reaches 180 TFLOPS, which is 30x and 15x higher than the conventional CPU and GPU, respectively.

DNN models are computationally and memory intensive, and their efficiency and scalability have been severely restricted by the limited memory bandwidth. Near-data processing is an effective way of addressing the above issue. Since 2014, Chen *et al.* have proposed the DianNao series of ASIC deep learning accelerators[42], which can accelerate machine learning algorithms including CNN and DNN. DianNao[43] accelerates the inference process of deep learning with a special emphasis on the impact of memory in design, performance and energy. It focuses on the optimization of memory reading, and uses fragmentation technology and data locality. It is capable of performing 452 GOPS in a small footprint of 3.02 mm^2 and 485 mW. DaDianNao[44] is a machine learning supercomputer architecture proposed for the efficient processing of large-scale NNs. It includes multiple identical chips connected with a mesh interconnection network. Each chip contains 16 tiles and has a neural functional unit and 4 eDRAM banks. By keep the whole model within the chips simultaneously, it can eliminate main memory accesses. In the visual recognition scenario with CNNs at mobiles or embedded devices, it has strict power and area limit. To improve the overall throughput of the accelerator, the proposed ShiDianNao[45] store the whole CNN model within on-chip storage. It is placed next to the image sensor and completely eliminates the system's off-chip memory access. PuDianNao[46] is a polyvalent ASIC accelerator for ML scenarios at different scales, supporting seven representative machine learning techniques. In Ref. [47], Du *et al.* combined inexact computing with NN accelerators and describe the benefits and associated costs expressed by increased error, proving that using inexact multipliers in NNs is feasible.

Some accelerators use reconfigurable architectures, considering programmability and flexibility. The concept of reconfiguration was first put forward by Professor Gerald Estrion in his article in 1960[48]. He defined that a computer can be composed of a main processor and a set of reconfigurable hardware. And those hardware structures can be configurated by the main processor to adapt to specific tasks. Thus, reconfigurable prototype systems are developed. In 1999, Dehon *et al.* further defined the reconfigurable processor, the task-to-chip spatial mapping can be customized and realized to a great extent after the chip was manufactured[49]. It was also a new choice to apply the FPGAs to computing, combining the advantages of traditional software and hardware computing, which called reconfigurable computing architecture. This architecture can program the hardware and reconstruct the circuit structure, so that the computation of the device can meet the immediate requirements of a certain application, and can be reused in different time domains. After that, as artificial NNs develop rapidly and the amount of information has increased dramatically, reconfigurable accelerators start to appear to solve a series of problems that follow.

In 2012, Cadambi *et al.* presented a massively parallel, energy efficient programmable accelerator that can execute multiple learning and classification algorithms[50]. And different MapReduce accelerators can be reconfigured dynamically according to the applications requirements. In 2017, Ansari *et al.* proposed a reconfigurable accelerator that uses basic processing elements as building blocks of its computational engine and can be extended to be a network-agnostic architecture that supports various networks[51]. In 2017, a versatile reconfigurable accelerator for binary/ternary DNNs was presented by Ando *et al.*[52]. It featured a massively parallel in-memory processing architecture and stores varieties of binary/ternary DNNs with a maximum of 13 layers, 4.2 K neurons, and 0.8 M synapses on chip, improving the energy efficiency dramatically. Lee *et al.* proposed a unified DNN accelerator in 2018, which is a unified neural processing unit supporting convolutional layers, recurrent layers and fully connected layers with fully-variable weight bit-precision from 1 to 16 bits[53]. In 2019, You and Wu presented an input row based sparse convolution neural network (CNN) accelerator on FPGAs and a weight merging method to balance the computation load on different PUs, which performs sparse CNN computing efficiently and maximize the overall computation efficiency[54].

In the aspect of programmability of NN, there are more research work. Liu *et al.* put forward a new instruction set architecture for NN accelerators, called Cambricon[55], which achieved higher code density over vector and matrix instructions. Since the programming productivity and software stack development, becomes an important reason instead of performance and power efficiency that hinders the application of machine learning computers. In 2019, Zhao *et al.* proposed Cambricon-F[56], a series of homogeneous, sequential, multi-layer, layer-similar, ML computers with the same ISA. A Cambricon-F machine has a fractal von Neumann architecture and its components are managed iteratively. Cambricon-F instances with different scales can share the same software stack on the common ISA, so that it can significantly improve the programming productivity.

2.2. Accelerator hardware design summarization

Generally, NN accelerators have been implemented on various hardware platforms, which can be mainly divided into three categories.

The first is general purpose hardware platform, such as GPU, CPU, DSP and other processors belong to this type. They are based on Von Neumann structure that takes arithmetic logic units (ALU) as its computing core in general, and follow the workflow of fetching, decoding and executing instructions. Due to its versatility, the CPU needs to deal with various application scenarios, which may include complex types of branch jumps and interrupts. So that the control logic and cache hit ratio are the key factors that affect instruction throughput. Specialized optimization within a specific domain is an option. GPU tremendously reduces the space of control logic and cache, and adds a large number of single instruction multiple data (SIMD) computing unit, which greatly improves the parallelism of processor computing, making it suitable for large-scale, similar-type and repetitive computing applications. But its power consumption is high. General Purpose Processor with small buffer capacity and only supporting basic operations, and the complex arithmetic operations are composed of a series of basic operations. Thus, frequent data exchange between registers and memory, also between on-chip cache and off-chip storage is required, which not only reduces performance but also increases energy consumption of NN. Researchers began to design special accelerators for NN algorithms.

The second category is the application-specific integrated circuit (ASIC). ASIC is a special processor designed for specific applications, which has the advantages of small size, low power consumption, fast calculation speed and high reliability. ASIC adopts hardware circuit paths for fixed type computing tasks, so ASIC can achieve very high energy-efficiency ratios at very low power consumption generally (down to milliwatts). Therefore, it is a good choice in the scenario where the NN algorithm and application requirements are relatively fixed. However, ASIC has low flexibility, and its fixed hardware structure makes it lack of scalability. As long as the application requirements change slightly or NN algorithm begins to evolve, the whole hardware circuit needs to be redesigned. In addition, ASIC requires a long development cycle and the cost is high.

The third kind is based on reconfigurable devices, including field-programmable gate array (FPGA) and coarse-grained reconfigurable array (CGRA). FPGA can provide a large amount of computing and storage resources for computing-intensive applications (such as CNN, DNN, etc.). The programmable and reconfigurable features of this class of processors allow users to customize the processor structure according to their needs, and can complete the design evaluation in a very short time, thus shortening the development cycle. Because the FPGA sacrifices too much chip area and computing speed, CGRA is proposed. CGRA integrates the computing part into configurable processing elements (PE), and changes the link between PE and memory by configuring information, thereby realizing the dynamic configuration of the hardware structure. Because CGRA solidifies the internal hardware circuitry of PE and reduces the additional cost of its interconnect configuration, it can be closer to the ASIC in terms of energy efficiency, and the power consumption can be controlled at the milliwatt level.

Combined with the analysis above, we can get the comparison of different hardware acceleration platforms as Table 1 displays. In summary, the reconfigurable devices represented by FPGA have achieved a compromise in flexibility and performance between general hardware platform and ASIC.

3. Software design and optimization of NN accelerator

This section mainly reviews the NN accelerator programming environments. We summarize the methods to improve NN programming performance, which are mainly based on the characteristics of NN algorithms and the architectures of NN accelerators.

3.1. Overview

In the NN accelerator design, it is not enough to consider the hardware features of the memory access and parallel of neural network computing, but also the entire programming system. The computing performance and energy effi-

Table 1. Comparison of different hardware acceleration platforms.

Features	GPU/CPU	ASIC	FPGA
Speed	Slow	Medium	Fast
Chip area	Big	Small	Medium
Parallelism	Low	High	Medium
Cost	Low	High	Medium
Power consumption	High	Low	Low
Development cycle	Short	Long	Short
Flexibility	Medium	Low	High

ciency of the hardware platform is only the premise of speeding up the neural network algorithm. The execution of neural network application also needs the cooperation of the software stack to enhance hardware efficiency. In actual application scenarios, regardless of the cloud servers, the IP in the mobile devices, or the cameras in embedded environment, the use of the neural network accelerator in any scenario is inseparable from the programming system. The hardware-based optimization in the software stack, directly determines the overall system workload and performance of the application. On the other hand, when the user deploys the application on the NN accelerator (especially the accelerator in ASIC form), the software stack and development environment must be adapted to the particularity of the hardware architecture. So, the design of the whole programming system directly determines the agility of front-end development, and influences the friendliness of the testing and debugging process. The portability of NN accelerator programming system is an important factor of the application, which can transplant or deploy to the target platform. Developers prefer not to re-debug or refine-tuned the network after the transplantation. And with the original NN model, the correctness and accuracy of the output should not be affected at all. In the ideal situation, the program after porting could fully utilize the acceleration performance of the accelerator.

The design of the programming system is mainly divided into two parts, the NN programming and the NN model compilation and optimization. The programming method of NN is the first level of interface that the programmer uses to develop on the specific accelerator hardware. The structured description is directly proposed by user to describe the NN model. The compilation of the network model is to translate the different levels of representation of the NN model to a series of machine code of the specific accelerator. Generally, the computational graph will be converted into assembly instructions. Since different accelerators may use different instruction set architectures (ISA), the compilation method is inexhaustible for different accelerators. So specialized optimizations can be made for the specific hardware architectures to maximize the benefits of hardware.

3.2. Programming of neural network

The programming of NNs is one of the first issues to be considered in NN model design. At present, the NN algorithm is still developing rapidly, and the scale and complexity of the NN model are increasing. All of these lead higher requirements on NN programming. Currently there are two methods of programming, one is using the NN frameworks (Fig. 2) and the other one is directly implemented in high-level programming language by programmers.

Use a neural network framework. There are many general neural network frameworks have been proposed, such as TensorFlow[57], Caffe[58], MXNet[59], etc. These NN frameworks simplify the representation of neural network. Some frameworks are based on data representation, and some are based on layer representation. By using the representation registered and encapsulated in the framework, the user can conveniently describe the NN model structure. Using the existing NN framework has a low learning cost and powerful portability. But programmers only can use the layers and operations predefined in the framework, which reduces flexibility.

Use a high-level programming language. Using high-level programming languages includes the general-purpose programming languages such as C++, and the domain-specific languages (DSL). When describe the NN model with a common programming language, high-performance libraries provided by NN accelerator developers could be helpful. These libraries provide functions of NN algorithms implementation, such as layer-based convolution operations, data-based matrix multiplication, etc. And the libraries directly optimize the execution of operations and calculations of the specific hardware, so that developers can get a better performance. Using NN accelerator libraries in NN programming or compiling, can also reduce programming difficulty and reduce coupling in software stack. Domain-specific languages including Latte[60], Swift for TensorFlow, are dedicated for NN model description and NN inference and training process. For instance, with the differential operators feature, it is easier to deal with the NN training process. When using DSL, the optimization of the computational graph is provided at the language level. Different NN accelerator developers may also develop accelerator-specific programming languages that are closely tied to the hardware's features to compile and optimize. The specific compiler is usually used as a back-end of the whole NN programming to generate the machine code of the NN accelerator.

3.3. Compilation and optimization of NN accelerator

The compilation and optimization of NN models are the core of the NN accelerator programming system, and also the bridge connecting the software application and the underlying hardware. Compilation of the NN model is to generate the instructions running on the NN accelerator based on the input NN model description. During compiling, the code can be optimized according to the characteristics of the model structure, the memory access pattern, and the architecture of the accelerator. Therefore, the software application can efficiently utilize hardware resources to achieve better performance and less power consumption.

Compilation of the NN model. The input of the compiler of the neural network accelerator is the description of the NN model, using the methods described in Section 3.2. Generally, the abovementioned description of a NN model is computational graph form. The compiler converts the computational graph into intermediate representations (IR) which are convenient to optimize and code generation. There may be several levels of the IR in the compiler, such as high-level intermediate representation and low-level intermediate representation, etc. The compiler will optimize the intermediate representation at different levels and generate binary hardware execution code. DLIR[61] is a tensor-based intermediate repres-

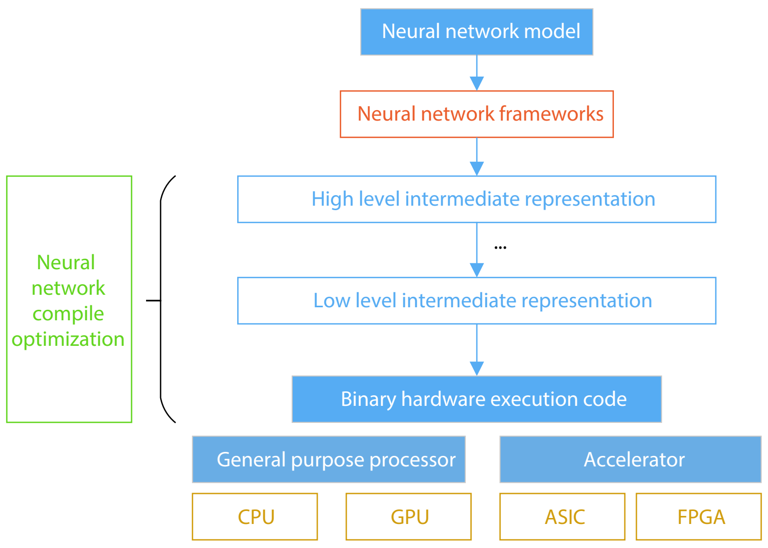
Neural network model
Neural network frameworks
High level intermediate representation
...
Low level intermediate representation
Binary hardware execution code
General purpose processor
Accelerator
CPU
GPU
ASIC
FPGA
Neural network compile optimization

Fig. 2. (Color online) Programming system hierarchy diagram.

entation proposed by Lan. The built-in tensor intrinsic of DLIR can directly be mapped to hardware primitives, which provide the ability to generate more efficient code for neural network accelerators. It also comes with a compiler and runtime. The compiler can convert the input computational graph described using a framework to DLIR, and then take optimization and code generation. Du proposed Zhuque[62], a development kit focusing on data layout for Cambricon-X[44] which is a NN accelerator. It contains methods for the NN description, compiling and optimizing of network models, and implementation of memory access operation and graph computing. The software stack of Nvidia's open source deep learning accelerator NVDLA also adopts this design idea. After the compiler obtains the network model generated by the framework such as Caffe, the software stack performs parsing and optimization inside the compiler, and then generates files for the backend. XLA[63] is a compiler for optimizing computational graphs generated by Tensorflow. It can match different NN accelerator backends such as CPU, GPU, ASIC accelerators, etc. XLA inputs Tensorflow computational graphs, and then converts them into internal custom intermediate representations HLO (high level optimizer). Code will be generated after the HLO IR is optimized. But HLO IR describes the computational graph in a high level and cannot represent the operations such as data moving between main memory and on-chip memory, so it cannot fully exploit hardware performance when using neural network accelerators based on ASIC. TVM[64] is a deep learning compiler framework proposed by Chen *et al.* which proposes a unified intermediate representation. As a bridge between increasingly deep learning frontends and hardware backends, TVM can parse computational graphs of various frontend frameworks. It leverages Halide[65] IR to present computation loops and provide several optimization levels. After optimizing, the low-level loop program can be used in various scenarios such as accelerator backend, LLVM framework, CUDA, OpenCL, etc.

Computational graph optimization. The optimization of computational graphs and code generation is a crucial part to fully play the performance of NN accelerators. Optimization mainly includes the effective simplification of the network structure represented by the computational graph, and the optimization of data layout, data transfer, and computational parallelism in combination with hardware characteristics. Previous compilation optimizations were mainly implemented by writing assemble instructions manually. Although this method can achieve almost the best effect case-by-case, programmers need to put huge time and effort on it. As the complexity of NN algorithms increases, the inefficiency of handwritten code cannot meet the demands. Therefore, the compilation needs to perform optimization of operations automatically or semi-automatically. Most of the current neural network compilers have built-in automatic optimizers. The accelerator developers also propose various compiler optimizations to meet the characteristics of the accelerators. Song[66] proposed a series of optimization methods for NN accelerator, including layer-based high-level optimization and low-level optimization within layers. The optimization performs intra layer unrolling and pipelining to the computational graph, including fine-grained and coarse-grained two levels. Layer integration based on computational graph, expanding pipelining stages of layers are also mentioned. There are also some frameworks that provide general optimization methods. XLA provides two optimization levels (Fig. 3), target-independent optimizations and target-dependent optimizations. Target-independent optimizations are mainly based on the structural information of the entire computational graph and no hardware information is not involved. The optimization methods include algebraic simplification, constant folding, common subexpression elimination, and layers fusion, etc. Target-dependent optimizations are optimized using hardware architecture information, which means that optimization methods are different for different architectures. The TVM compiler stack also provides multiple levels of optimization (Fig. 3). The optimization in TVM is divided into three steps. The first is the optimization of the computational graph. Operator fusion in TVM combines multiple operators into one kernel. Data layout will be optimized according to the structure of the computational graph. The second is operator-level optimization and code generation, including the optimization of tensor expression and schedule space, nested parallelism with cooperation, explicit memory latency hiding, etc. Finally, TVM performs hardware-specific optimization using hardware-aware optimization primitives, and get the optimized low-level loop program for NN backends.

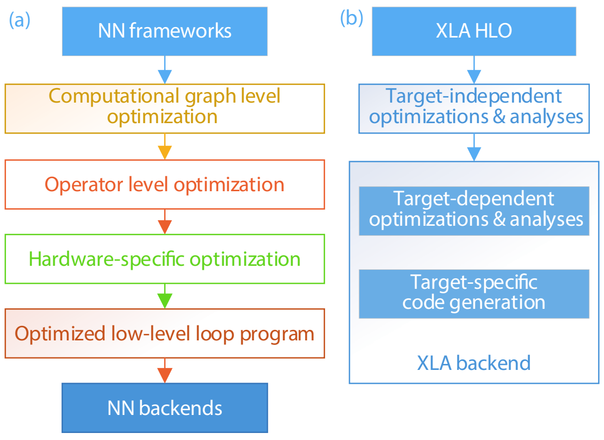
(a)
NN frameworks
Computational graph level optimization
Operator level optimization
Hardware-specific optimization
Optimized low-level loop program
NN backends
(b)
XLA HLO
Target-independent optimizations & analyses
Target-dependent optimizations & analyses
Target-specific code generation
XLA backend

Fig. 3. (Color online) (a) TVM and (b) XLA compiling optimization stack overview diagram.

4. Future development trend

We have already reviewed some NN accelerator schemes in this paper. Some of them are implemented on the general-purpose chip platform, including supporting low-precision computing, supporting more NN frameworks, and designing for accelerated convolution operation. The implementation includes improved arithmetic storage structure, optimized data flow and design specific NN instruction set architecture, data level parallelism and data prefetching technology implemented on reconfigurable platforms. We also introduced the programming system of NN accelerators. With the concerted efforts of NN frameworks and compilers, developers can deploy and debug their NN algorithms efficiently and conveniently on accelerator hardware. While these designs have made significant advances in NN acceleration, there are still many challenges. In our view, the following five aspects are feasible directions for future research of NN accelerators and programming system.

(1) **Optimizing computational performance.** Application of NN in embedded equipment is a future trend, In the aspect of arithmetic, we can reduce the number of parameters and the amount of calculation of the neural network within the allowable range of precision loss by pruning the network, quantifying and low-precision calculation, so that the neural network of smaller scale can be deployed to embedded devices.

(2) **Optimizing memory access performance.** At present, pruning, compression and other technologies have appeared to optimize memory access performance, but the storage speed cannot keep up with the calculation speed is still a difficult problem in the current neural network accelerator design.

(3) **Optimizing power consumption and chip area.** In accelerator, multipliers are the units that consume most area and power in computational units. Therefore, further exploration of data organization forms can reduce the use of hardware resources, reduce the power consumption of accelerator, and reduce data exchange. Moreover, some effective techniques such as approximate computing, pruning and compression can also improve performance and power consumption.

(4) **Developing more versatile and modular programming framework.** It is evident that programming framework plays an important role in the development of NN applications, which can help researchers develop conveniently and efficiently. In the future the programming framework should be modular that the application developers will only focus on neural network algorithms rather than the optimization method and hardware architecture. And different programming frameworks should be compatible with each other to improve programming efficiency.

(5) **Hardware-oriented automatic optimization.** The compilation optimization is a crucial part to fully play the performance of NN accelerators. Nowadays, the optimization for hardware must be implemented by the hardware platform developers. In the future, there may be a general method to represent the characteristics of the hardware. With this method the compiler will do hardware-oriented optimization automatically, which will significantly alleviate the burden on hardware developers.

With the development of NN algorithm, more NN operators need to be developed on the accelerator. In addition to optimizing the computation and memory access delay in hardware, the friendliness of programming and the efficiency of the library of NN accelerator should also be taken into account. So, the developers can spend less energy on the programming on the specific accelerator details and iterate faster. The automatic compile optimization of NN accelerator is a research direction. And the whole programming system may also consider of a heterogeneous computing platform for the performance end-to-end.

5. Conclusion

Nowadays, the NN accelerator has not only gained extensive attention in academic research, but has also been widely deployed in industrial applications. But as the applications of AI algorithms are becoming ubiquitous, the NN algorithm is also evolving. The variability of application scenarios, the diversity of algorithms and the huge amount of data put forward higher requirements for NN accelerators and their programming systems.

We sketch out the NN algorithms and NN accelerators. With the accelerator performance getting faster, hardware is no longer the bottleneck in the AI application. Meanwhile, implementing these algorithms on different software and hardware platforms and implementing them efficiently is still a huge challenge. It leads to the necessity and importance of acceleration as a whole entity. Then we review the latest development of from software aspects, including the implementation methods and compile optimization of the existing programming system for NN accelerators. We also comment the future development trend of NN accelerator, which direction must be the combination of software and hardware iteration, stimulating development.

Acknowledgments

This work is partially supported by the National Key Research and Development Program of China (under Grant 2017YFB1003101, 2018AAA0103300, 2017YFA0700900, 2017YFA0700902, 2017YFA0700901), the National Natural Science Foundation of China (under Grant 61732007, 61432016, 61532016, 61672491, 61602441, 61602446, 61732002, 61702478, and 61732020), Beijing Natural Science Foundation (JQ18013), National Science and Technology Major Project (2018ZX01031102), the Transformation and Transfer of Scientific and Technological Achievements of Chinese Academy of Sciences (KFJ-HGZX-013), Key Research Projects in Frontier Science of Chinese Academy of Sciences (QYZDB-SSW-JSC001), Strategic Priority Research Program of Chinese Academy of Science (XDB32050200, XDC01020000), Standardization Research Project of Chinese Academy of Sciences (BZ201800001), Beijing Academy of Artificial Intelligence (BAAI) and Beijing Nova Program of Science and Technology (Z191100001119093).

References

[1] Huang W, Jing Z. Multi-focus image fusion using pulse coupled neural network. Pattern Recogn Lett, 2007, 28(9), 1123

[2] Paik J K, Katsaggelos A K. Image restoration using a modified hopfield network. IEEE Trans Image Process, 1992, 1(1), 49

[3] Li X, Zhao L, Wei L, et al. DeepSaliency: multi-task deep neural network model for salient object detection. IEEE Trans Image Process, 2016, 25, 3919

[4] Zhu Y, Urtasun R, Salakhutdinov R, et al. segDeepM: exploiting segmentation and context in deep neural networks for object detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, 4703

[5] Graves A, Mohamed A R, Hinton G. Speech recognition with deep recurrent neural networks. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, 6645

[6] Abdelhamid O, Mohamed A, Jiang H, et al. Convolutional neural networks for speech recognition. IEEE/ACM Trans Audio Speech Language Process, 2014, 22(10), 1533

[7] Collobert R, Weston J. A unified architecture for natural language processing. International Conference on Machine Learning, 2008

[8] Sarikaya R, Hinton G E, Deoras A. Application of deep belief networks for natural language understanding. IEEE/ACM Trans Audio, Speech, Language Process, 2014, 22(4), 778

[9] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv: 1409.1556, 2014

[10] McCulloch W S, Pitts W. A logical calculus of ideas immanent in nervous activity. Bull Math Biophys, 1943, 5(4), 115

[11] Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain. Psycholog Rev, 1958, 65(6), 386

[12] Werbos P. Beyond regression: new tools for prediction and analysis in the behavioral sciences. Dissertation for the Doctoral Degree, Harvard University, 1974

[13] Hinton G E, Osindero S, Teh Y. A fast learning algorithm for deep belief nets. Neur Comput, 2006, 18(7), 1527

[14] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolu-tional neural networks. Advances in Neural Information Processing Systems, 2012, 1097

[15] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, 770

[16] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, 1

[17] Szegedy C, Ioffe S, Vanhoucke V, et al. Inception-v4, Inception-ResNet and the impact of residual connections on learning. National Conference on Artificial Intelligence, 2016, 4278

[18] Yu F, Koltun V. Multi-scale context aggregation by dilated convolutions. arXiv: 1511.07122, 2015

[19] Mamalet F, Garcia C. Simplifying convnets for fast learning. international conference on artificial neural networks. International Conference on Artificial Neural Networks, 2012, 58

[20] Howard A G, Zhu M, Chen B, et al. MobileNets: efficient convolutional neural networks for mobile vision applications. arXiv: 1704.04861, 2017

[21] Cho K, Van Merrienboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv: 1406.1078, 2014

[22] Hochreiter S, Schmidhuber J. Long short-term memory. Neur Comput, 1997, 9(8), 1735

[23] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. Advances in Neural Information Processing Systems, 2017, 5998

[24] Devlin J, Chang M W, Lee K, et al. BERT: pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv: 1810.04805, 2018

[25] Parashar A, Rhu M, Mukkara A, et al. SCNN: An accelerator for compressed-sparse convolutional neural networks. 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), 2017

[26] Han S, Mao H, Dally W J. Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv: 1510.00149, 2015

[27] Lin D D, Talathi S S, Annapureddy V S. Fixed point quantization of deep convolutional networks. International Conference on Machine Learning, 2016, 2849

[28] Xue J, Li J, Yu D, et al. Singular value decomposition based low-footprint speaker adaptation and personalization for deep neural network. IEEE International Conference on Acoustics, 2014

[29] Park E, Ahn J, Yoo S. Weighted-entropy-based quantization for deep neural networks. IEEE Conference on Computer Vision & Pattern Recognition, 2017

[30] Song L, Wang Y, Han Y, et al. C-Brain: A deep learning accelerator that tames the diversity of CNNs through adaptive data-level parallelization. Design Automation Conference, 2016

[31] Kuo R J, An Y L, Wang H S, et al. Integration of self-organizing feature maps neural network and genetic K-means algorithm for market segmentation. Expert Syst Appl, 2006, 30(2), 313

[32] Roska T, Bártfai G, Szolgay P, et al. A digital multiprocessor hardware accelerator board for cellular neural networks: CNN-HAC. Int J Circuit Theory Appl, 1992, 20(5), 589

[33] Gokhale V, Zaidy A, Chang A X M, et al. Snowflake: a model agnostic accelerator for deep convolutional neural networks. arXiv preprint arXiv: 1708.02579, 2017

[34] Page A, Jafari A, Shea C, et al. SPARCNet: a hardware accelerator for efficient deployment of sparse convolutional networks. ACM J Emerg Technolog Comput Syst, 2017, 13(3), 1

[35] Chen T, Chen Y, Duranton M, et al. BenchNN: On the broad potential application scope of hardware neural network accelerators. 2012 IEEE International Symposium on Workload Characterization (IISWC), 2012, 36

[36] Farabet C, Poulet C, Han J Y, et al. CNP: An FPGA-based processor for convolutional networks. International Conference on Field Programmable Logic and Applications, 2009

[37] Zhang S, Du Z, Zhang L, et al. Cambricon-X: An accelerator for sparse neural networks. The 49th Annual IEEE/ACM International Symposium on Microarchitecture, 2016, 20

[38] Yu Y, Zhi T, Zhou X, et al. BSHIFT: a low cost deep neural networks accelerator. Int J Paral Program, 2019, 47, 360

[39] Shafiee A, Nag A, Muralimanohar N, et al. ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016

[40] Chen Y H, Krishna T, Emer J S, et al. Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE J Solid-State Circuits, 2017, 52(1), 127

[41] Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit. 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), 2017, 1

[42] Chen Y, Chen T, Xu Z, et al. DianNao family: energy-efficient hard-

ware accelerators for machine learning. Commun ACM, 2016, 59(11), 105
[43] Chen T, Du Z, Sun N, et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, 2014
[44] Chen Y, Luo T, Liu S, et al. Dadiannao: A machine-learning supercomputer. Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014, 609
[45] Du Z, Fasthuber R, Chen T, et al. ShiDianNao:shifting vision processing closer to the sensor. ACM/IEEE International Symposium on Computer Architecture, 2015
[46] Liu D, Chen T, Liu S, et al. Pudiannao: A polyvalent machine learning accelerator. ACM SIGARCH Comput Architect News, 2015, 43(1), 369
[47] Du Z, Palem K, Lingamneni A, et al. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), 2014, 201
[48] Estrin G. Organization of computer systems: the fixed plus variable structure computer. Western Joint IRE-AIEE-ACM Computer Conference, 1960, 33
[49] Dehon A, Wawrzynek J. Reconfigurable computing: what, why, and implications for design automation. Proceedings 1999 Design Automation Conferenc, 1999
[50] Majumdar A, Cadambi S, Becchi M, et al. A massively parallel, energy efficient programmable accelerator for learning and classification. ACM Trans Architect Code Optim, 2012, 9(1), 1
[51] Ansari A, Gunnam K, Ogunfunmi T, et al. An efficient reconfigurable hardware accelerator for convolutional neural networks. 2017 51st Asilomar Conference on Signals, Systems, and Computers, 2017, 1337
[52] Ando K, Ueyoshi K, Orimo K, et al. BRein memory: a single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W. IEEE J Solid-State Circuits, 2017, 53(4), 983
[53] Lee J, Kim C, Kang S H, et al. UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision. International Solid-State Circuits Conference, 2018, 218
[54] You W, Wu C. A reconfigurable accelerator for sparse convolutional neural networks. Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2019, 119
[55] Liu S, Du Z, Tao J, et al. Cambricon: An instruction set architecture for neural networks. ACM SIGARCH Comput Architect News, 2016, 44(3), 393
[56] Zhao Y, Du Z, Guo Q, et al. Cambricon-F: machine learning computers with fractal von neumann architecture. International Symposium on Computer Architecture, 2019, 788
[57] Abadi M, Barham P, Chen J, et al. Tensorflow: A system for large-scale machine learning. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, 265
[58] Jia Y, Shelhamer E, Donahue J, et al. Caffe: Convolutional architecture for fast feature embedding. Proceedings of the 22nd ACM International Conference on Multimedia, 2014, 675
[59] Chen T, Li M, Li Y, et al. MXNet: a flexible and efficient machine learning library for heterogeneous distributed systems. arXiv: 1512.01274, 2015
[60] Truong L, Barik R, Totoni E, et al. Latte: a language, compiler, and runtime for elegant and efficient deep neural networks. ACM SIGPLAN Notices, 2016, 51, 209
[61] Lan H, Du Z. DLIR: an intermediate representation for deep learning processors. IFIP International Conference on Network and Parallel Computing, 2018, 169
[62] Du W, Wu L, Chen X, et al. ZhuQue: a neural network programming model based on labeled data layout. International Symposium on Advanced Parallel Processing Technologies, 2019, 27
[63] Fischer K, Saba E. Automatic full compilation of Julia programs and ML models to cloud TPUs. arXiv: 1810.09868, 2018
[64] Chen T, Moreau T, Jiang Z, et al. TVM: an automated end-to-end optimizing compiler for deep learning. 13th USENIX Symposium on Operating Systems Design and Implementation, 2018, 578
[65] Mendis C, Bosboom J, Wu K, et al. Helium: lifting high-performance stencil kernels from stripped ×86 binaries to halide DSL code. Program Language Des Implem, 2015, 50(6), 391
[66] Song J, Zhuang Y, Chen X, et al. Compiling optimization for neural network accelerators. International Symposium on Advanced Parallel Processing Technologies, 2019, 15