神经网络加速器及其软件编程系统

李 威
中国科学院计算技术研究所

关键词：神经网络加速器 软件编程系统

引言

近年来，随着神经网络算法研究不断取得重大进展，其在图像识别、目标检测、语音识别和自然语言处理等方面均得到了广泛的应用。然而，随着神经网络层数的不断增加，算法所需的计算量也呈指数级上升，这对传统的处理器架构提出了巨大的挑战。面对这个问题，研究人员先后研发了多种不同架构的神经网络加速器，本文旨在回顾并分析神经网络加速器的发展历程以及对应的软件编程系统。

神经网络模型通常具有三种典型的层结构，分别是卷积层、池化层以及全连接层。从功能性方面来看，卷积层通常用于特征抽取，通过卷积核在输入特征图上滑动进行乘法和加法运算，输出特征图。池化层通常位于卷积层之后，用于减小信息的空间维度，以避免过拟合，并提高神经网络的容错性。通过挖掘神经网络结构中的数据复用模式和神经网络层的运算模式，并针对关键层和运算进行加速优化，可以有效提升整个模型的处理速度。

基于以上基本思想，在过去的几十年中有众多研究人员在硬件架构和编程系统两方面进行了不懈探索，提出了多种神经网络加速器架构。在硬件架构方面，研究人员提出了并行化、硬件神经元时分复用、深度学习指令集等思路来设计专门的神经网络加速器，以减少运算时间，提高准确率。

编程系统是连接用户和神经网络加速器硬件之间的桥梁，其通过构建优化的软件架构，简化神经网络算法在神经网络加速器的部署过程，力图使端到端的性能达到最优。

神经网络加速器

2012 年，Chen 等人提出硬件神经网络加速器在 RMS（Recognition-Mining-Synthesis，识别－发掘－综合）应用领域将具有广阔的应用前景 [1]。随着暗硅（dark silicon）时代的到来，在合理功耗范围内芯片已经无法通过简单的增加处理器核数来提高性能。因此，针对特定应用需求的加速器将成为未来的发展趋势。研究结果表明，硬件神经网络加速器可以胜任诸多对算力性能要求较高的应用场景。

基于神经网络算法的结构特征和运算机制，研究人员提出了多种不同的硬件架构，下面将简述其中的一些代表性成果。

通用处理芯片

以通用处理器 CPU、图形处理器 GPU 实现深度学习应用的这条技术路线在工业界有较为广泛的应用。谷歌、脸书、百度、阿里巴巴、腾讯等公司纷纷构建大规模的 CPU/GPU 计算集群来并行加速神经网络算法，其中大多数计算集群的计算节点采用 CPU 或者 CPU+GPU 的异构模式进行。为了保证计算性能，这需要付出极大的功耗代价。例如，谷歌在 2012 年利用 1.6 万个 CPU 核的集群经过数天训练了一个 10 亿参数的神经网络模型，并于 2013 年用 GPU 集群将其扩展至 110 亿参数。而在实现用于围棋对弈的 AlphaGo 系统时，谷歌采用了超过 1000 颗 CPU 和 200 颗

GPU。这种集群的功耗动辄数万瓦，具有较高的使用门槛。

CPU 和 GPU 的提供商也在积极开发新版本的产品，以提供更好的性能和更高的效率。例如，CPU 芯片提供商英特尔在 2020 年发布了第三代英特尔至强可扩展处理器 Ice Lake，该处理器能够支持图像分类、推荐引擎、语音识别和语言建模等应用的推理和训练；GPU 芯片提供商英伟达在 2020 年推出了首款基于安培（Ampere）架构的 GPU——NVIDIA A100 以及对应的智能计算系统 NVIDIA DGX A100。

现场可编程门阵列

现场可编程门阵列（FPGA）具有可编程性，允许用户在短时间内对设计进行原型评估，以此缩短开发周期，节省设计的开发费用。因此，很多研究人员基于 FPGA 平台进行了神经网络加速器的研究和实践。

Farabet 等人提出了基于 FPGA 的卷积神经网络加速器 [2]，它利用 FPGA 中多个嵌入式数字信号处理单元并行地计算具有相同输入不同输出的局部和，然后平移输入（窗口）来完成相应输出的卷积。

由于二维卷积广泛应用于图像处理等场景中，Cardells-Tormo 等人提出了基于 FPGA 的二维卷积运算结构 [3]。这种结构仅需要相对较少的片上内存开销，因此可以用低成本的 FPGA 实现。

随着神经网络算法计算复杂度的提升，功耗问题日益严峻。Maashri 等人提出了一个用于视觉识别的神经网络系统 [4]，通过在 FPGA 平台上构建 HMAX 计算模型，来实现目标识别、人脸识别及行为识别。相比于 CPU 和 GPU，它能够分别实现 7.6 倍和 4.3 倍的性能加速，且能效分别为它们的 12.8 倍和 7.7 倍。

使用 FPGA 实现神经网络算法的加速已经被证明有较好的性能，但 FPGA 有限的带宽和片上内存限制了大规模神经网络算法的高性能实现。Qiu 等人 [5] 分析了卷积神经网络在 FPGA 上的实现过程，发现卷积层占用了大量的计算资源，而全连接层则占用了大量内存资源。他们据此提出了一种动态精度数据量化方法以及一个通用卷积器来改进带宽和资源利用率，最终基于 Xilinx Zynq ZC706 FPGA 实现了性能达 137.0 GOP/s 的卷积神经网络运算。

DianNao 系列神经网络加速器

中科院计算所从 2008 年开始从事人工智能和体系结构的交叉研究。2013 年，计算所和法国国家信息与自动化研究所（INRIA）的合作者设计了国际上首个深度学习加速器架构 DianNao[6]。该架构采用启发式模型方法，实现了计算和存储之间的平衡，其能效相比 CPU 取得了两个数量级以上的提升。DianNao 架构如图 1 所示，其中的核心部件是神经功能单元（NFU），它利用三级流水线结构，将不同类型的层分解为 2 个或 3 个阶段，并在控制逻辑（CP）的控制下进行流水线运算。这种运算逻辑设计使得 DianNao 可以同时对多个输出神经元进行计算，从而灵活、高效地处理规模高达上百层、包含千万神经元和上亿突触的各种深度神经网络模型。DianNao 架构还对数据传输策略进行了优化，它将片上存储划分为 3 部分：输入缓冲区（NBin）、输出缓冲区（NBout）和突触缓冲区（SB）。这种存储方案可以为 SRAM 设计合适的读写带宽，并可避免在缓冲区发生数据冲突。在运算过程中，输入缓冲区可以处于循环缓存状态，实现输入神经元的复用。以上设计使得 DianNao 的数据传输带宽可以相比 CPU 和 GPU 有显著降低。

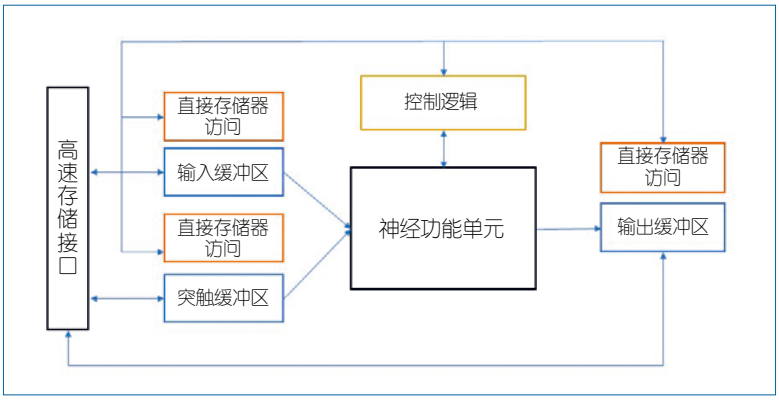

图 1　DianNao 架构示意图

基于DianNao架构，中科院计算所又进一步和法国INRIA的合作者设计了国际上首个多核深度学习加速器结构DaDianNao[7, 8]，它将加速器核心的规模扩大到了16个，同时具备推理和训练功能。DaDianNao采用分块的加速器架构，每个运算块（tile）可以同时计算16个输出节点，单个芯片可以同时进行256个并行运算。DaDianNao采用分布式的eDRAM（增强动态随机存取存储器）使得所有突触单元均靠近运算器，采用高带宽的胖树（fat tree）结构向每个块广播相同的输入数据，并收集输出值，这些输出值被集中到一个中心eDRAM中。以上设计使得DaDianNao可以有效实现低功耗与低延时的数据传输。

PuDianNao[9]是中科院计算所设计的国际上首个机器学习处理器架构，它可以支持多种机器学习算法，包括线性回归、朴素贝叶斯、*k*-近邻、*k*-均值、分类树、支持向量机和深度神经网络等。PuDianNao架构的主要组成部分包括功能单元、数据缓存、控制模块、指令缓存和直接存储器访问（DMA），其中功能单元包括机器学习功能单元（MLU）和算法逻辑单元（ALU）。

针对实时图像处理任务的需求，中科院计算所设计了ShiDianNao[10]架构，该架构可将整个卷积神经网络模型储存在片上存储中，并将其放置在图像传感器旁边，从而彻底消除了对片外存储的访问。它可以被嵌入到传感器中，直接获取输入图像，进行完整的卷积神经网络映射，从而实现实时的图像处理。

中科院计算所团队提出的Cambricon-X[11]架构针对神经网络模型的稀疏性和不规则性进行设计，可以有效提升处理性能。基于稀疏神经网络的计算模式和访存特征，Cambricon-X设计了专门的神经元处理器（PE）和索引模块（IM）来选择需要运算的神经元，将不需要计算的数据剔除，从而在有限的带宽下实现高性能和低功耗的神经网络处理。

2019年，中科院计算所团队提出了分形神经网络加速器架构Cambricon-F[12]，这是一种采用了层次同性原理的专用并行体系结构，相同任务负载在不同规模的分形神经网络加速器上可以分别自动展开、执行。不同规模的Cambricon-F加速器可以共享一套指令集和编程接口，因此可以显著地提高编程便利性和效率。

软件编程系统

神经网络加速器为提升神经网络算法的实现性能提供了基础架构，而神经网络应用的实际执行必须有合适的软件编程系统配合，才能充分发挥神经网络加速器的硬件优势。在实际应用场景中，无论所采用的硬件平台是云端服务器、移动端设备，还是嵌入式设备，其中神经网络加速器的运行都离不开编程系统。编程系统的设计直接决定着前端开发的敏捷性，并对测试和调试过程中的编程友好性产生影响。此外，编程系统的可移植性也是实际应用中的另一个重要方面，决定了应用是否可以方便地部署和移植到目标平台上。用户希望模型在移植到新平台上时只需要尽可能少的调试，就可以保持初始神经网络模型的正确性和准确性。在理想情况下，移植后的程序仍然可以对加速器的硬件性能进行充分的利用[13]。

编程系统的设计主要可以分为两个部分，分别是神经网络编程方法和神经网络模型编译及优化。用户在特定的加速器硬件上进行开发时，编程方法是编程界面的第一层级，用户直接使用结构化描述来进行神经网络模型设计。网络模型的编译过程则是将对神经网络模型不同层次的描述翻译成加速器可以执行的一系列机器语言。由于不同加速器可能使用不同的指令集架构，它们的编译方法也各不相同。对于特定的硬件架构，可以有针对性地进行编程工具的优化，以充分发挥硬件优势。

编程方法

编程方法是神经网络模型设计过程中需要首先考虑的问题之一。当前神经网络算法仍然在快速发展，神经网络模型的规模和复杂程度也在不断增加。这些都对神经网络编程方法提出了更高的要求。当前主要有两种编程方法：一种是利用深度学习编程框架，另一种是使用编程语言直接进行编程，整个编程系统结构如图2所示。

深度学习编程框架 目前学术界和工业界已有多个通用的深度学习编程框架，例如TensorFlow[14]、PyTorch[15]、Caffe[16]、MXNet[17]等。这些编程框架使

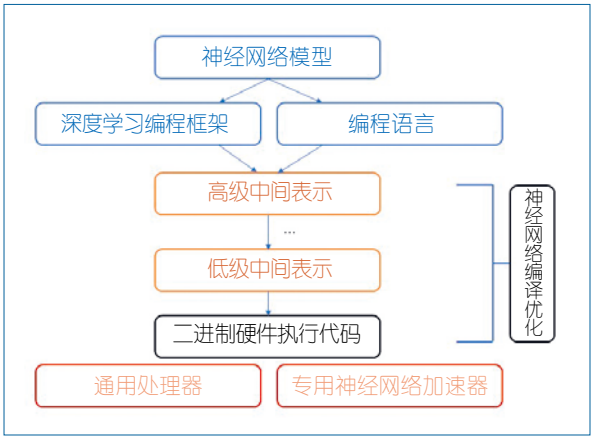

图 2　编程系统示意图

得神经网络的表示更加简洁。部分编程框架是基于计算图表示，另一些则是基于层表示。通过使用注册和封装在编程框架中的算子，用户可以方便地描述神经网络模型的结构。

编程语言　编程语言包括通用编程语言和领域专用语言。面向人工智能领域，传统通用编程语言难以同时满足高开发效率、高性能和高可移植性的需求，而智能编程语言则是同时满足上述三大需求的重要技术途径[18]。作为连接编程框架和神经网络加速器的桥梁，智能编程语言既是实现编程框架算子的基础，也是神经网络加速器高效编程的核心用户入口。针对神经网络的结构特点，智能编程语言除了提供智能计算核心操作（算子）级别的高层操作语义，其抽象层次也在不断提高，向高层次和专用化的方向发展；与此同时，为了满足较高的实现性能，智能编程语言也为用户提供了足够丰富的硬件细节；而对于不同的神经网络加速器平台，用户还可以针对特定的硬件结构特点来设计专用编程语言，用于编译和优化。

编译及优化

神经网络模型的编译及优化是神经网络加速器编程系统的核心，也是连接应用和底层硬件的桥梁。神经网络模型编译就是基于输入的神经网络模型描述，得到可在神经网络加速器上运行的指令的过程。在编译过程中，代码可以根据模型结构特征、访存特征和加速器架构进行优化，从而使得应用算法可以充分利用硬件资源，获得更高的运算速率和更低的功耗。

神经网络模型编译　编译器的输入是对神经网络模型的描述，通常情况下，神经网络模型描述是采用计算图的形式。编译器首先将计算图转变成中间表示（IR），以方便优化和代码生成。编译器中间表示可以包含不同的层次，例如高级中间表示或低级中间表示等。编译器可对不同层次的中间表示进行优化，生成二进制硬件执行代码。研究人员提出了一种基于张量的中间表示方法 DLIR[19]，其内置的张量表示可以直接映射到神经网络加速器硬件上，从而产生更高效的硬件代码。DLIR 还包含了编译器和运行时，编译器可以将输入的计算图用 DLIR 的架构进行描述，再进行优化和代码生成。谷歌提出了 XLA 编译器[20]用于编译 TensorFlow 生成的计算图。XLA 接收的输入是 TensorFlow 计算图，然后将其转化为中间表示 HLO，在对 HLO 进行优化之后生成代码。它可以和不同种类的神经网络加速器后端相匹配，例如 CPU、GPU 和自定义的神经网络加速器等。TVM[21] 是一个端到端的编译器，它可以支持深度学习任务部署在不同的编程框架以及不同的神经网络加速器架构上。TVM 使用 Relay IR 描述计算图，并提供了多种调度原语，如循环顺序交换、分块计算，等等。用户或编译器可以通过这些调度原语实现对不同操作的优化。TVM 也可以生成不同后端的代码，实现对不同硬件架构的支持。

计算图优化　计算图的优化和代码生成是决定神经网络加速器性能是否得到充分发挥的关键环节之一。优化内容主要包括对计算图的化简、数据布局优化、数据转移优化以及与硬件架构相结合的运算并行性优化等。过去的编译优化主要是通过手动编写汇编指令实现的，这种方法虽然可以起到较好的效果，但需要用户付出巨大的时间成本。随着神经网络复杂性的增加，手动编写汇编代码的方法已经无法满足需求，因此编译器需要实现优化操作的自动化或半自动化。当前，几乎所有的神经网络编译器都内置了自动优化功能。部分编译器提供了通用优化方法，例如 XLA 提供了两种不同的优化，

分别是目标相关优化和目标无关优化。目标无关优化主要基于计算图的结构信息进行优化，与硬件信息无关，方法包括代数化简、常数折叠、公共子表达式消元和层融合等；目标相关优化需要结合硬件信息，因此对于不同的硬件架构，优化方法也有所不同。TVM 提供了不同层级的优化，优化过程可以划分为 3 个步骤：（1）计算图优化，TVM 通过将多个算子整合到一起实现算子融合，并进行数据布局优化；（2）算子层面的优化和代码生成，包括张量化、时间及空间优化、嵌套并行、延迟隐藏等；（3）TVM 通过硬件感知优化原语针对硬件架构进行优化，最终生成适用于神经网络后端的底层优化程序。

未来发展方向

尽管目前研究人员已经在神经网络加速器方面取得了诸多重要进展，但是在加速器性能及能效提升、编程系统优化等方面仍然面临较多挑战。我们认为如下 5 个方面将是未来神经网络加速器和编程系统研究的可行方向 [13]：

1. **运算性能优化**　神经网络在嵌入式设备上的应用将是未来的趋势。可以通过网络剪枝及低精度量化运算等方法，在允许的精度损失范围内减少神经网络的参数数量和运算量，缩减整个神经网络的规模，以更好地实现其在嵌入式设备上的部署。

2. **访存性能优化**　当前已经有一些优化访存性能的方法，例如剪枝和压缩等，然而数据存取的速度仍然无法和运算速度相匹配，这是当前神经网络加速器设计过程中面临的一个关键问题。

3. **运算单元功耗和面积优化**　在神经网络加速器中，乘法器是运算单元中面积和功耗占用最多的部件。我们可以通过优化数据组织形式，如存内计算等技术，减少数据交换过程对硬件资源的占用，从而降低加速器的功耗。

4. **开发兼容性更强的模块化编程框架**　编程框架在神经网络应用开发过程中起着重要的作用，它可以使开发过程更加便利和高效。未来的编程框架应当是模块化的，这样用户就只需要专注于神经网络算法，而无须过多考虑硬件架构。此外不同编程框架之间应当互相兼容，以进一步提高编程效率。

5. **面向硬件的自动优化**　编译优化对于提升神经网络加速器硬件性能非常关键。当前，硬件层面的优化需要由用户来进行，而未来可能会出现表示硬件特征的通用方法。编译器可以通过这种方法来实现面向不同硬件架构的自动优化，从而大大降低用户的开发成本。

结论

当前，神经网络加速器不仅在学术研究中受到重视，在实际的工业生产中也得到了广泛的应用。随着人工智能应用越来越广，神经网络算法也在不断演进，复杂多变的应用场景、多种多样的算法和庞大的数据量对神经网络加速器和编程系统提出了越来越高的要求。

随着加速器算力的提升，硬件不再是制约人工智能应用发展的瓶颈。与此同时，如何将算法部署到不同的软件和硬件平台上，并充分利用平台优势仍然是一个巨大的挑战。面对这个挑战，将加速器系统当作一个整体来考虑是很有必要的。我们认为，未来的发展方向应当是软件和硬件研究深度结合，共同进行技术迭代，从而促进神经网络加速器领域的发展。 ■

李　威
中科院计算所副研究员。主要研究方向为高性能芯片设计技术、大规模集成电路设计、FPGA 优化结构研究等。
liwei2017@ict.ac.cn

参考文献

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

[2] Farabet C, Poulet C, Han J Y, et al. CNP: An FPGA-based processor for convolutional networks. [C]// International Conference on Field Programmable Logic and Applications, 2009: 32-37.

[3] Cardells-Tormo F, Molinet P, Sempere-Agullo J, et al. Area-efficient 2D shift-variant convolvers for FPGA-based digital image processing [J]. International Conference on Field Programmable Logic and Applications, 2005: 578-581.

[4] Maashri A A, Debole M, Cotter M, et al. Accelerating neuromorphic vision algorithms for recognition. [C]// Proc of the 49th Design Automation Conf. ACM, 2012: 579-584.

[5] Qiu J, Wang J, Yao S, et al. Going deeper with embedded FPGA platform for convolutional neural network. [C]// Proc of the 24th ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. ACM,2016: 26-35.

[6] Chen T, Du Z, Sun N, et al. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. [C]//Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating systems (ASPLOS). ACM, 2014: 269-284.

[7] Chen Y, Luo T, Liu S, et al. DaDianNao: A machine-learning supercomputer. [C]// Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE Computer Society, 2014: 609-622.

[8] 韩栋 , 周圣元 , 支天等 . 智能芯片的评述和展望 [J]. 计算机研究与发展 , 2019, 56(1): 7-22.

[9] Liu D, Chen T, Liu S, et al. PuDianNao: A polyvalent machine learning accelerator. [C]//ACM Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). 2015: 369-381.

[10] Du Z, Fasthuber R, Chen T, et al. ShiDianNao: Shifting vision processing closer to the sensor. [C]//ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA). 2015: 92-104.

[11] Zhang S, Du Z, Zhang L, et al. Cambricon-X: An accelerator for sparse neural networks[C]//49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). 2016: 1-12.

[12] Zhao Y, Du Z, Guo Q, et al. Cambricon-F: machine learning computers with fractal von neumann architecture. [C]//Proceedings of the 46th International Symposium on Computer Architecture (ISCA). 2019: 788-801.

[13] Song J, Wang X, Zhao Z, et al. A survey of neural network accelerator with software development environments[J]. J. Semicond., 2020, 41(2), 021403.

[14] Abadi M, Barham P, Chen J, et al. Tensorflow: A system for large-scale machine learning[C]// Proceedings of the 12th USENIX symposium on operating systems design and implementation (OSDI). 2016: 265-283.

[15] Ketkar N. Deep learning with python[M/OL]. Apress, Berkeley, CA, 2017. DOI: https://doi.org/10.1007/978-1-4842-2766-4.

[16] Jia Y, Shelhamer E, Donahue J, et al. Caffe: Convolutional architecture for fast feature embedding[C]// Proceedings of the 22nd ACM International Conference on Multimedia. ACM, 2014: 675-678.

[17] Chen T, Li M, Li Y, et al. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems[J]. arXiv preprint arXiv:1512.01274, 2015.

[18] 陈云霁 , 李玲 , 李威等 . 智能计算系统 [M]. 机械工业出版社 , 2020.

[19] Lan H, Du Z. DLIR: an intermediate representation for deep learning processors. [C]// IFIP International Conference on Network and Parallel Computing, 2018, 169-173.

[20] Fischer K, Saba E. Automatic full compilation of Julia programs and ML models to cloud TPUs. arXiv: 1810.09868, 2018.

[21] Chen T, Moreau T, Jiang Z, et al. TVM: An automated end-to-end optimizing compiler for deep learning. [C]//13th USENIX Symposium on Operating Systems Design and Implementation (OSDI). 2018:578-594.