ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2018/12(10)-1645-13
doi: 10.3778/j.issn.1673-9418.1711010

E-mail: fcst@vip.163.com
http://www.ceaj.org
Tel: +86-10-89056056

面向训练阶段的神经网络性能分析

李景军+,张 宸,曹 强
华中科技大学 武汉光电国家研究中心,武汉 430074

Analyzing Performance of Neural Networks in Training Phase

LI Jingjun+, ZHANG Chen, CAO Qiang
Key Laboratory of Information Storage System, Ministry of Education of China. Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, China
+ Corresponding author: E-mail: jingjunli@hust.edu.cn

LI Jingjun, ZHANG Chen, CAO Qiang. Analyzing performance of neural networks in training phase. Journal of Frontiers of Computer Science and Technology, 2018, 12(10): 1645-1657.

Abstract: Recently, the neural networks have increasingly delopyed in many fields. However, as complexity of neural networks grows, graphics processing units (GPUs) begin to be applied in deep learning. Though GPUs have exhibited excellent performance on accelerating matrix multiplication, the real computing resources and memory resources of GPUs have not been fully utilized in the compute-intensive neural network training phase due to the complexity and diversity of network models. This paper focuses on doing an experimental and fine-grained performance analysis for deep neural network models. First, it divides the training phase into six stages in the sight of data flow and measures the latency of each stage. And then, it presents a quantitative analysis for GPU compute efficiency and resource utilization in each layer from point of views of GPU-accelerated libraries, neural network models, and batch size. Finally, weights and feature maps of each layer are given quantitatively to reveal the GPU memory utilization. These experiments and analysis show that (1) The compute efficiency of cuDNN in convolution layers is 2 times than cuBLAS. (2) The resource utilization of convolution layers is 50% higher than full-connected layers. (3) The GPU memory utilization in different layers are varied, and the overall utilization is not high, no more than 20% of the total memory space.

Key words: network models; graphics processing unit (GPU); resource utilization; compute efficiency; data flow; GPU-accelerated library

Received 2017-09, Accepted 2017-11.
CNKI网络出版: 2017-11-28, http://kns.cnki.net/kcms/detail/11.5602.TP.20171128.0900.012.html

摘 要:最近,神经网络被广泛应用到许多领域。然而,随着神经网络模型越来越复杂,图形处理单元(graphics processing unit,GPU)被应用到深度学习中。GPU在加速矩阵计算方面展现出了卓越的性能,但是多样和复杂的神经网络模型导致网络训练阶段GPU的计算资源和显存并没有充分利用。对神经网络训练阶段进行细粒度的性能分析。首先从数据流的角度把训练过程分解为6个阶段,并测试每个阶段的延时;然后从GPU加速库、神经网络模型和批次三方面量化分析每一层的GPU计算效率和资源利用率;最后分析每层的参数和特征图的显存占用情况。实验发现:(1)cuDNN库卷积的计算效率是cuBLAS库的2倍。(2)卷积层的资源利用率比全连接层高50%。(3)不同层的显存利用率差异很大,整体利用率不高,最大不超过显存的20%。

关键词:网络模型;图形处理单元(GPU);资源利用率;计算效率;数据流;GPU加速库

文献标志码:A **中图分类号:**TP183;TP302

1 引言

随着深度学习技术的发展,卷积神经网络(convolutional neural network,CNN)被广泛应用到人脸识别、物体检测等领域。另外,处理器计算能力的增强,不仅缩短了模型训练的时间,而且也使得CNN技术得到了进一步的研究和开发。例如,Google公司开发的FaceNet网络模型人脸识别精度可以达到99.63%[1]。微软开发的Optasia,在大城市交通摄像机的关联查询方面,也表现出很高的精度和性能[2]。

虽然CNN取得了很高的精度,但是复杂的网络模型给处理器带来了极大的挑战。例如,为了达到57.1%的top-1精度和80.2%的top-5精度,AlexNet[3]需要迭代358 000次[4]。VGG-16[5]拥有13 800万个权值参数,一次迭代需要155亿次浮点操作[6]。对于大型网络模型(例如ResNet[7]),常常由几十或者几百层组成,模型具有更多的参数,迭代计算需要更多的浮点操作。由于CPU采用复杂的控制逻辑和分支预测,利用CPU训练大型神经网络模型时间很长,完成一次训练往往需要几天甚至十几天。

为了加速训练过程,FPGA、GPU和一些专用加速器[8-13]被相继开发,通过优化并行计算、流水线或近数据处理等方式,加快神经网络的执行速度。GPU具有强大的并行处理能力,因此深度学习工作者大多使用GPU训练神经网络模型。Song等人[14]通过对任务进行分类,动态调度GPU的内部资源,在mobile GPU上实现了不同任务在延时、能耗和精度方面的权衡。Rhu等人[15]同时利用显存和内存存储特征图和参数,实现了GPU训练更大的网络模型。此外,cuDNN[16]、cuBLAS[17]等GPU加速库和一些并行算法的出现,从算法上优化了矩阵乘法,缩短了矩阵运算的时间。

利用GPU神经网络训练速度得到了很大的提升,但是GPU的计算资源和显存仍然未被充分利用[18]。训练复杂的网络模型时,如何高效地利用GPU,仍然有待进一步研究。与之前的工作相比,本文注重于探究和分析在训练CNN过程中GPU资源是否被充分利用,以及GPU的计算效率是否达到最高。量化了GPU加速库、神经网络模型和批次对GPU计算效率和资源利用率的影响。另外,统计了神经网络训练过程中每层的参数和特征图的显存占用量,为后续利用GPU训练大规模网络模型的研究铺平道路。

总之,本文的主要贡献如下:

(1)把神经网络的训练过程分解为6个阶段,通过细粒度的实验,给出了每个阶段的延时,明确地指出图像预处理和反向传播的矩阵乘法是最耗时的操作。

(2)对比了cuDNN和cuBLAS库加速CNN训练的差异。卷积层cuDNN的计算效率和资源利用率分别是cuBLAS的2倍和1.7倍,但是在全连接层两者差距不大。

(3)探究了网络模型和批次对GPU计算效率和资源利用率的影响。对于不同网络模型,卷积层的计算效率和资源利用率都远远高于全连接层。另外,较大的批次虽然会带来计算效率的提升,但是资源利用率不会升高。

(4)统计了不同网络模型每层的显存使用情况,

为以后研究训练超出GPU显存的网络模型奠定基础。

本文组织结构如下：第2章简述相关工作；第3章介绍CNN的背景、数据流以及实验方法；第4章针对CNN数据流，对每个阶段进行细粒度的量化分析；第5章展示并分析实验结果；第6章总结全文。

2 相关工作

先前大部分工作研究和设计了神经网络加速器，从硬件来加速网络训练过程[19-23]。本文重点从数据流的角度阐述神经网络训练过程，并量化了训练过程中GPU的计算效率和资源利用率，为之后GPU或者神经网络加速器的研究奠定基础。

神经网络的计算非常耗时，因此有大量的工作对此优化[24-26]，但是图像预处理耗时也比较长，需要进一步优化。尽管使用快速存储设备可以分摊这一开销，但预处理的问题并没有得到解决。随着计算能力的提升，图像预处理会拖慢神经网络的训练。

另外，还有一些探究网络模型性能的工作。Shi等人[27]在不同的硬件平台上对多种网络模型进行了评估。通过比较各种模型的运行时间，衡量网络模型对硬件平台的敏感程度。虽然本文也是通过时间延时衡量性能，但是主要在单个GPU平台，评估GPU计算效率和资源利用率，分析不同网络模型性能差异的根本原因。Li等人[28]对比了cuda-convnet和cuDNN下网络模型的吞吐量，然后针对内存布局重点讨论了矩阵乘法带来的吞吐量差异。从块和寄存器分配等角度探究GPU资源利用情况，更注重于充分开发GPU的计算潜力。

3 背景

本章首先对CNN进行简要介绍，阐述CNN数据流的每个步骤，然后针对训练网络模型的过程设计实验方法。

3.1 CNN简介

常见的卷积神经网络主要由3种类型的层组成：卷积层(conv)、池化层(pool)和全连接层(fc)。为了防止过拟合，提高网络的准确率，研究人员会添加额外的层(norm、dropout等)。CNN就是由这些层按照一定顺序组合而成，网络的模型参数和权值都保存在这些层中。图1展示了一个简单的CNN网络模型。

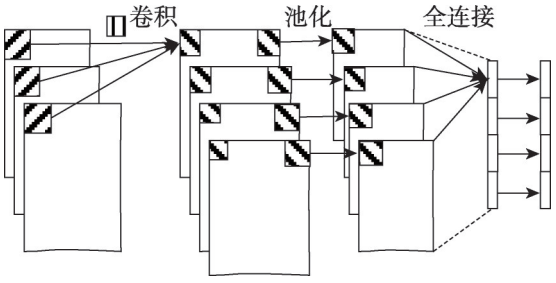

Fig.1 Convolutional neural network

图1 卷积神经网络

卷积层，卷积层是利用一系列卷积核与输入数据进行卷积操作。在*NCHW*内存布局中，卷积计算如式(1)所示：

$$Y(Ni,Co,Hi,Wi)=\sum_{ci=0}^{Ci-1}\sum_{kh=0}^{KH-1}\sum_{kw=0}^{KW-1}K(Co,ci,kh,kw)*X(Ni,Ci,Hi+kh,Wi+kw) \quad (1)$$

其中，K是卷积核；X和Y分别是卷积层的输入和输出；Ni是批次大小；Ci和Co分别是输入和输出通道数；Hi和Wi分别是输入特征图的高和宽；KH和KW代表卷积核的大小。总之，卷积层通过卷积核从输入的特征图中提取各种局部特征，比如边、角等。

池化层，也叫下采样层，普遍的实现方式有两种：最大池化和均值池化。最大池化是对邻域内的特征点求最大值，均值池化是对邻域内的特征点求平均值。以均值池化为例，计算如式(2)所示：

$$Y(Ni,Ci,Hi,Wi)=\frac{1}{KH*KW}*\sum_{kh=0}^{KH-1}\sum_{kw=0}^{KW-1}X(Ni,Ci,Hi*KH+kh,Wi*KW+kw) \quad (2)$$

池化层对卷积层提取的局部特征进行下采样，抽象为低分辨率的特征图。

全连接层，又叫作内积层。全连接层将之前层学到的特征图映射到样本空间。全连接层可以看作卷积核为KH=1，KW=1的卷积层。其中，H和W分别为上一层结果的高度和宽度。

激活层，执行一个非线性函数，例如sigmoid函数$\frac{1}{1+e^{-x}}$或relu函数$\max(0,x)$。

Softmax层，一般在网络模型的最后，用来计算分

类对象的概率。它的输入来自全连接层。Softmax层的计算如式(3)所示：

$$Yi=\frac{e^{Xi-Xk}}{\sum_{i=0}^{Co-1}e^{Xi-Xk}} \tag{3}$$

其中，Xk 是上层输出的最大预测值。

3.2 前向传播和反向传播

训练神经网络是学习神经网络权值参数 W 的过程，该过程需要进行多次迭代（比如AlexNet迭代450 000次），每次迭代又可以细分为前向传播（Forward）和反向传播（Backward），如图2所示。

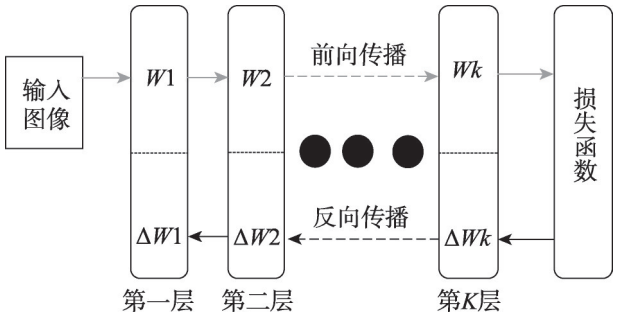

Fig.2 Forward and backward propagation of CNN

图2 卷积神经网络的前向和反向传播

在前向传播中，CNN网络模型中的层按照从前至后顺序依次执行，第 L 层的输出是第 $L+1$ 层的输入。在前向传播的末尾，一个损失函数 J 被定义，用于衡量网络模型输出的预测结果 Yi 与真实标签 Ti 的差异，得到推断误差，如式(4)所示：

$$J=\frac{1}{2}\sum_{i=0}^{Ni}(Yi-Ti)^2 \tag{4}$$

在反向传播过程中，利用链式法则和随机梯度下降算法[29-31]，由后至前计算权值梯度 ΔW 。计算如式(5)、式(6)所示：

$$\Delta w=-\eta\nabla J(w) \tag{5}$$

$$\frac{\partial J}{\partial w}=\frac{\partial}{\partial w}\frac{1}{2}\sum_{i=0}^{Ni}(Yi-Ti)^2 \tag{6}$$

最后，使用权值的梯度 ΔW 更新每层对应的权值：

$$W=W+\Delta W \tag{7}$$

3.3 CNN数据流

图3展示了一个典型的CNN数据流。对于训练阶段，神经网络的迭代过程可以细分为6个阶段：

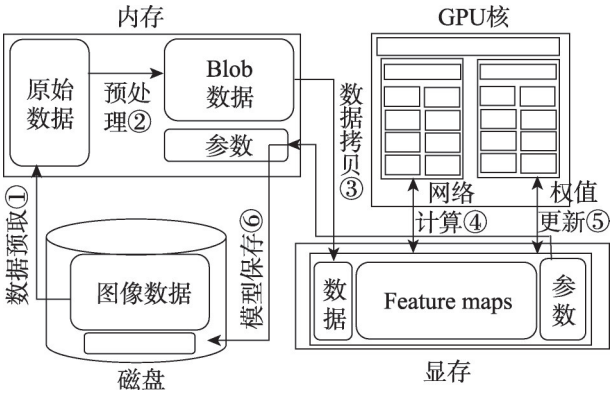

Fig.3 Data flow of CNN in computer systems

图3 计算系统的CNN数据流

数据预取①：首先，预取线程从存储设备（比如硬盘、SSD）读取图像数据和对应的标签到内存。为了提高图像读取速度，图像数据可以以数据库（LMDB、LevelDB）或者HDF5格式保存在存储设备。

图像预处理②：对图像进行均值消减、裁剪、缩放等操作，变换到网络规定的输入大小（AlexNet网络模型是3×227×227）。

数据拷贝③：将批次大小张图像数据从内存异步拷贝到显存，供GPU计算使用。

网络计算④：从显存中读取图像数据、权值等，执行每层的计算（包括前向传播和反向传播），并在反向传播过程中计算每层的权值梯度 ΔW 。

权值更新⑤：利用权值梯度 ΔW ，通过 $W=W+\Delta W$ 更新显存中的权值，并准备下次迭代，迭代直至训练结束。

模型保存⑥：在训练过程中，为了避免进程异常或者系统崩溃丢失训练的网络模型，常常需要使用快照技术在一定数量的迭代次数后（例如10 000次），将网络模型保存到存储设备。

3.4 方法评估

本文的目的是探索神经网络训练阶段GPU的计算效率和资源利用率，并分析影响GPU性能的根本原因。使用延时时间作为评价性能的指标。首先对神经网络的一次迭代过程进行量化分析，从数据流的角度详细地给出迭代中各个阶段的执行时间。然后，从GPU加速库、网络模型和批次三方面讨论对GPU计算效率和资源利用率的影响。最后统计了网

络模型中每层的权值数据和特征图对显存的使用情况。Caffe[32-33]提供了一个开发友好的平台，并且在训练神经网络方面展现出了良好的性能，因此选用Caffe作为本文的实验平台。

硬件和软件平台：表1详细地展示了实验中使用的CPU和GPU的规格信息。本文使用的软件平台如表2所示。cuDNN是专为神经网络设计的GPU加速库，对卷积层、池化层、正则化层和激活层进行大量优化。与cuDNN不同，cuBLAS是针对GPU设计的线性代数运算库，能够利用GPU加速计算密集型操作。

Table 1 Hardware configurations

表1 硬件配置

Item	Value
CPU Type	Intel® Xeon® CPU E5-2683 v3
CPU Cores	14Cores, 2.00 GHz
Graphic Card	NVIDIA Geforce GTX1080
Architecture	Pascal
CUDA Cores	2 560
Base Clock	1 607 MHz
Compute Capability	6.1
Graphic Memory	8 GB GDDR5X
Memory Bandwidth	320 GB/s
Main Memory	128 GB

Table 2 Software configurations

表2 软件配置

Software	Major version
OS	Ubuntu 16.04
Caffe	Berkeley Vision 1.0
cuDNN	V5.1
CUDA	8.0
OpenCV	3.0

网络模型和负载：在实验中，本文选择MNIST[34]和ImageNet[35]数据集作为不同网络模型的负载。Ima-geNet数据集作为AlexNet[3]网络和VGG-16[5]网络的输入，MNIST数据集作为LeNet[36]网络的输入。ImageNet（240 GB，LMDB）包含128万张训练图像和50 000张测试图像，是一个用于训练视觉类网络模型的典型大数据集。数据集中每一张图像都用从0到999范围内的一个数字标记，每一个数字代表自然界中一个具体的物种。另一方面，MNIST（59 MB，LMDB）数据集通常用于手写数字的分类，有60 000张训练图像和10 000张测试图像。MNIST中的所有图像都是28×28像素的灰度图，每张图像用0到9中的一个数字标记，每一个数字代表一种手写数字。

4 数据流分析

首先从数据流的角度，量化地给出一次迭代中各个阶段的执行时间，包括数据预取、图像预处理、数据拷贝、网络计算和权值更新。然后针对网络计算阶段，统计每层的执行时间，并根据每层主要函数的执行时间，分析网络计算的瓶颈。

图4列出了在磁盘阵列（4×HDD，RAID 0）和单个硬盘上训练AlexNet网络模型，批次为256时各阶段的执行时间和一次迭代总时间。其中权值更新耗时较短，图像预处理（transform）和网络计算（forward和backward）耗时较长，数据预取时间与存储设备有关。

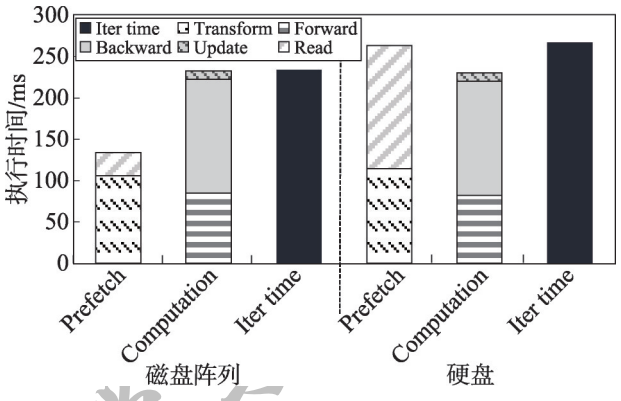

Fig.4 Alexnet's per stage computation latency over disk array and HDD

图4 Alexnet磁盘阵列和硬盘上各个阶段计算延迟

图像预处理：依次从内存中读取256张图像，对每张图像进行均值消减、剪裁、缩放等操作。由于数据集有限，网络模型需要迭代几十万次，为了防止之后的迭代过程使用与本次迭代相同的图像数据作为输入，图像预处理引入了随机函数，在一定程度上随机控制图像转换操作，防止过拟合。图像转换操作虽然简单，但是要对每个像素进行操作，CPU处理256张图像转换需要大约110 ms。

数据拷贝:转换后的图像数据仍然保存在内存中,实验中,150 MB数据从内存拷贝到显存需要13 ms。如果每次迭代前都需要串行地执行数据预取、图像预处理和数据拷贝,GPU大部分时间空闲。因此如图3所示,使用单独的预取线程进行数据预取,同时数据拷贝以异步的方式执行,这样不仅可以高效利用内存带宽,而且分摊了数据预取阶段的时间花销。

网络计算:网络计算阶段耗时223 ms,其中,前向传播和反向传播分别占40%和60%。

另外对于磁盘阵列,迭代时间主要取决于网络计算(computation)的时间。而对于单个磁盘,数据读取时间是磁盘阵列的5.4倍,较慢的读取速度,大大增加了预取时间。数据层需要等待预取线程的图像数据,因此迭代时间决定于数据预取的时间。磁盘阵列的读取速度能够满足网络计算的需求,因此接下来重点分析网络计算阶段。

为了进一步细粒度探究网络计算阶段的真实执行情况,使用Nvidia Visual Profiler统计AlexNet网络模型一次迭代中每层的函数执行情况,并按照延时时间从大到小排序,表3展示了延时最高的15个函数。在所有耗时较高的函数中,卷积层的矩阵乘法运算占87%,尤其是前两个卷积层反向传播中矩阵乘法运算(bconv1和bconv2),占整个迭代时间的50%。因此,细粒度地分析卷积层矩阵操作的延时,量化评估GPU的计算效率和资源利用情况,揭示影响GPU性能的根本原因,显得格外重要。此外,池化层和全连接层反向传播中的延时也不容忽视。

以AlexNet网络模型为例,量化地给出了每一层计算操作的时间,为之后的实验分析提供了基础。

5 实验分析

针对网络计算阶段,首先对比AlexNet网络在不同GPU加速库下的GPU计算效率和资源利用率。然后细粒度地分析在大型、中型和小型网络模型下GPU的性能差异。此外,还量化地揭示了在训练网络模型过程中,每层的特征图和权值数据的显存占用情况。

Table 3 Top-15 latency of major functions in layers

表3 层中主要函数时间延时的前15名

层	主要函数	执行时间/ms
bconv1	maxwell_scudnn_128×32_stridedB_splitK_interior_nn	46.063 562
bconv2	maxwell_scudnn_128×64_stridedB_small_nn	22.123 730
bconv3	maxwell_scudnn_128×128_stridedB_splitK_small_nn	21.883 249
bconv2	maxwell_scudnn_128×64_stridedB_small_nn	21.649 680
bconv2	maxwell_scudnn_128×128_stridedB_splitK_small_nn	18.747 877
conv1	maxwell_scudnn_128×32_interior_nn	18.660 997
bconv2	maxwell_scudnn_128×128_stridedB_splitK_small_nn	17.383 040
bconv3	maxwell_scudnn_wino grad_128×128_tile228n_nt	16.728 253
fc6	maxwell_sgemm_128×128_raggedMn_tn	15.994 523
conv3	maxwell_scudnn_winograd_128×128_tile228n_nt	15.649 017
bpool1	void caffe::MaxPoolBackward	14.962 007
conv2	maxwell_scudnn_128×128_small_nn	14.097 587
conv2	maxwell_scudnn_128×128_small_nn	13.859 347
bconv4	maxwell_scudnn_128×64_stridedB_splitK_small_nn	11.563 883
bconv1	void calc_bias_diff	10.450 503

5.1 cuDNN与cuBLAS的对比分析

统计了在cuDNN和cuBLAS上训练神经网络模型的计算延时,包括前向传播、反向传播和梯度更新,如图5所示。cuBLAS库的前向传播和反向传播

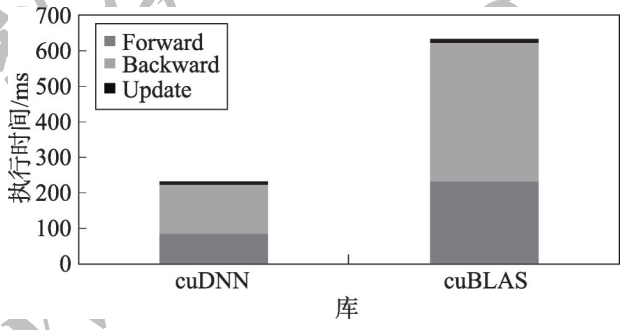

Fig.5 Latency for computation phase over cuDNN and cuBLAS

图5 cuDNN和cuBLAS在计算阶段的延时

过程比cuDNN慢了65%。由于权值更新阶段的时间延时不超过4.4%，省略这部分的分析。

图6展示了cuDNN和cuBLAS库每层执行时间对比（前向传播和反向传播）。从图中可以看出，在相同的存储设备和预取延时条件下，cuDNN计算更快，尤其是conv层和norm层，cuBLAS的平均延时分别是cuDNN的4倍和3.25倍。从整体来看，norm层、relu层和pool层耗时较短，重点分析conv层。

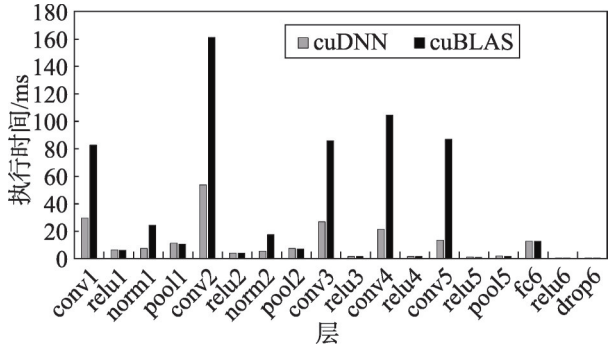

Fig.6 Per layer computation latency over cuDNN and cuBLAS

图6 cuDNN和cuBLAS每层的延时

为了深入地探究cuDNN和cuBLAS的计算效率，量化分析计算性能，用实际吞吐量与峰值吞吐量的比值衡量GPU的计算效率：

$$CE=\frac{throughput}{GPUpeakThroughput}=\frac{flops/t}{2\times GPUfreq\times Cores} \quad (8)$$

其中，*flops*是卷积层单精度浮点计算次数；t是卷积层的执行时间；*Cores*是GPU的CUDA核数量。

图7展示了本文的实验结果，卷积层cuDNN的计算效率远高于cuBLAS，是cuBLAS的2倍。虽然cuBLAS在全连接层的计算效率较高，但是与cuDNN差别不大。前面提到，卷积层cuBLAS的总延时是cuDNN的4倍，但卷积层的计算效率仅仅为cuDNN的一半，这是因为卷积层cuBLAS每次处理一张图像，循环执行256次（批次256）。cuDNN一次处理256张图像，减少了CUDA函数调用次数。另外，处理过程中的一些异步操作（im2col[16]）也会在一定程度上减少耗时。

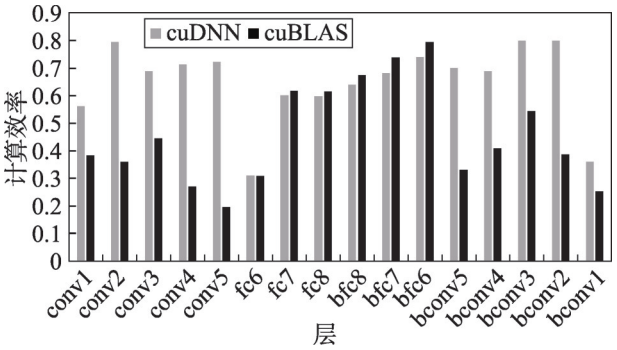

Fig.7 Per layer compute efficiency over cuDNN and cuBLAS

图7 cuDNN和cuBLAS每层的计算效率

表4列出了卷积层和全连接层网络计算过程中的矩阵信息。可以看出，对于相同大小的结果矩阵，cuDNN和cuBLAS实际执行的子矩阵大小也有很大差别。这一方面是由硬件条件决定的（例如显存大小、寄存器数量等），另一方面也与GPU内部块和线程的划分有关。

为了进一步探究GPU内部计算资源的利用情

Table 4 Detail information of CNN dominated kernels

表4 CNN主要核函数的详细信息

层	Result matrix		Submatrix		Blocksize		Registers/Thread		Grid size	
	cuDNN	cuBLAS	cuDNN	cuBLAS	cuDNN	cuBLAS	cuDNN	cuBLAS	cuDNN	cuBLAS
conv1	94×774 400	94×3 025	128×32	128×128	128	256	81	116	18 150	24
conv2	128×186 624	128×729	128×128	128×64	256	128	128	120	1 458	36
conv3	384×43 264	384×169	128×128	128×64	256	128	128	120	6 144	60
conv4	192×43 264	192×169	128×128	128×64	256	128	128	120	3 072	24
conv5	128×43 264	128×169	128×128	128×64	256	128	128	120	2 048	16
fc6	256×4 096	256×4 096	128×128	128×128	256	256	116	116	64	64
fc7	256×4 096	256×4 096	128×128	128×128	256	256	116	116	64	64
fc8	256×1 000	256×1 000	128×64	128×64	128	128	120	120	256	256

况，探究GPU计算效率低的原因，定义矩阵乘法计算过程中实际划分的线程块的数量：

$$Gridsize=\left\lceil\frac{M}{m}\right\rceil\times\left\lceil\frac{N}{n}\right\rceil \tag{9}$$

其中，$M\times N$ 是结果矩阵的大小；$m\times n$ 是CUDA划分后实际执行的子矩阵的大小，这样结果矩阵就可以通过执行若干个子矩阵操作计算。

由于GPU寄存器资源有限，而矩阵乘法是寄存器密集型的操作，每个线程至少使用81个寄存器，因此从寄存器利用的角度衡量最大可用的线程块数：

$$maxBlocks=SMs\times\left\lfloor\frac{R}{blocksize\times r}\right\rfloor \tag{10}$$

如果 *Gridsize* 小于 *maxBlocks*，则说明GPU资源未充分利用。如果 *Gridsize* 大于 *maxBlocks*，则GPU资源也存在利用率不高的情形，为了量化GPU资源利用率，使用 RU 衡量GPU的资源利用效率：

$$RU=\frac{Gridsize}{cycles\times maxBlocks}=\frac{Gridsize}{\left\lceil\frac{Gridsize}{maxBlocks}\right\rceil\times maxBlocks} \tag{11}$$

其中，*cycles* 是GPU执行完所有 *Gridsize* 需要的时钟周期。在反向传播过程中，既有权值梯度的计算，又有梯度映射的计算。表3中显示，对于大多数层，权值梯度 ΔW 的计算延时高于梯度映射 ΔY 的计算延时。尽管conv2层梯度映射延时较高，但是与权值梯度延时差距不大。因此，重点分析反向传播中权值梯度的计算过程。图8展示了本文的实验结果。

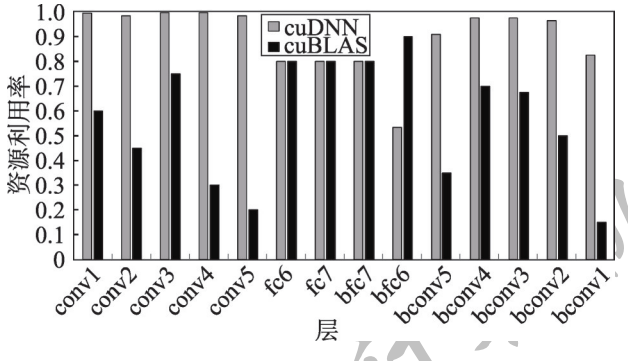

Fig.8 Resource utilization of main layers over cuDNN and cuBLAS

图8 cuDNN和cuBLAS在关键层的资源利用率

对于cuDNN，卷积层的资源利用率最高，基本都在95%以上，全连接层在80%左右。而对于cuBLAS，卷积层的资源利用率平均只有47%，全连接层的资源利用率达到最高。这是因为卷积层cuBLAS每次处理一张图像，数据量较少，实际分配的 *Gridsize* 要小于 *maxBlocks*，导致GPU的计算资源并没有充分利用。而cuDNN是将256张图像的处理过程融合成一次大矩阵操作，训练过程较大的批次，使得结果矩阵很大，从而 *Gridsize* 远大于 *maxBlocks*，实现了较高的资源利用率和计算效率。另外对于全连接层，cuDNN和cuBLAS都是将一批图像的计算看作一次矩阵操作，因此两者的资源利用率相近。

bfc6层cuBLAS资源利用率远高于cuDNN，是因为cuBLAS使用了较大的 *blocksize*，一定程度上使得 *Gridsize* 大于 *maxBlocks*。

总之，由于cuDNN和cuBLAS矩阵操作方面处理方式的不同，使得它们在计算效率和资源利用率上差异很大。在训练神经网络模型时，由于较大的批次和数据量，使得cuDNN的性能要高于cuBLAS。因此，根据网络模型结果矩阵的大小，合理地分配设置子矩阵大小、块大小和线程数，会增加GPU的计算效率和资源利用率，加快训练过程。

5.2 网络模型

不同网络模型使用的卷积核和图像大小不同，间接地导致GPU资源分配不均匀，从而影响训练性能。因此，在不同模型下，对比了GPU的计算效率和资源利用率。图9、图10分别展示了3种网络模型LeNet、AlexNet和VGG-16的GPU计算效率和资源利用率。

首先无论是大型、中型还是小型网络模型，卷积层的计算效率都高于全连接层，是全连接层的1.94倍。这是因为卷积层的矩阵大小远远大于全连接层，GPU计算效率更高。另外VGG-16中卷积层矩阵更大，计算效率最高，其次是AlexNet和LeNet。而全连接层的计算效率AlexNet最高，其次是VGG-16和LeNet。由于较小的网络模型和简单的计算操作，LeNet的卷积层和全连接层的计算效率都不高。

从资源利用率角度看，较大的矩阵规模和复杂的矩阵运算使VGG-16和AlexNet在卷积层的资源利用率几乎达到了100%。AlexNet网络在全连接层的

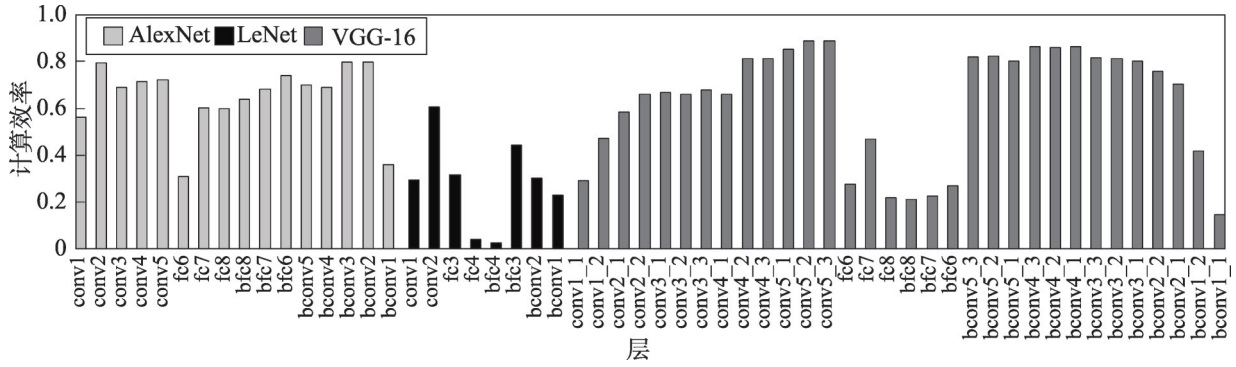

Fig.9 Compute efficiency of GPU in main layers over AlexNet, LeNet and VGG-16

图9 AlexNet、LeNet和VGG-16在关键层GPU的计算效率

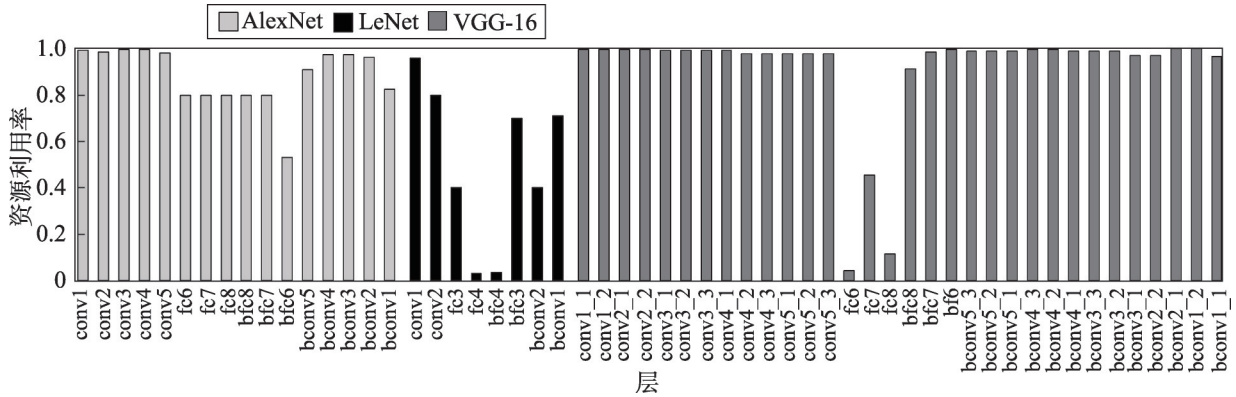

Fig.10 Resource utilization of main layers over AlexNet, LeNet and VGG-16

图10 AlexNet、LeNet和VGG-16在关键层的资源利用率

资源利用率达到了80%,VGG-16和LeNet都低于50%。这是因为AlexNet的全连接层分配了较大的*blocksize*和*r*,较小的*maxBlocks*使得*Gridsize*是*maxBlocks*的1.5倍。而LeNet和VGG-16网络模型,实际分配的*Gridsize*不到*maxBlocks*的一半,导致了极低的资源利用率。

VGG-16的全连接层反向传播(bfc6、bfc7和bfc8)的资源利用率远高于前向传播(fc6、fc7和fc8)。由5.1节的分析可知,对于大部分卷积层和全连接层,反向传播计算参数梯度的时间长于计算输入的梯度,因此,主要讨论参数梯度计算阶段的资源利用情况。VGG-16的全连接层在参数梯度计算中拥有比输入梯度计算更大的*batchsize*和*r*,另外VGG-16复杂的网络模型和较多的参数,也进一步加大了*Gridsize*和*maxBlocks*之间的差值,增大了资源利用率。

总之,通过图9和图10,卷积层的资源利用率和计算效率较高,并且相比于小型网络(LeNet),中型网络(AlexNet)和大型网络(VGG-16)的计算效率达到了70%。全连接层的资源利用率和计算效率普遍较低,对于全连接层较多的网络,增大全连接层的计算效率显得格外重要。

5.3 批次大小

通过5.1节和5.2节可知,分配的*blocksize*大小、寄存器的数量以及使用的*Gridsize*大小会对计算效率和资源利用率产生很大影响。前面已经量化地讨论了GPU加速库和网络模型带来的性能差异。但是训练阶段批次大小也会影响结果矩阵的大小,进而通过*Gridsize*影响计算效率和资源利用率。

图11和图12分别给出了AlexNet网络模型在批次为128、256和512时的计算效率和资源利用率。可以看出,批次越大计算效率越高。批次为512的计算效率比256和128分别高出11%和41%。因此,适

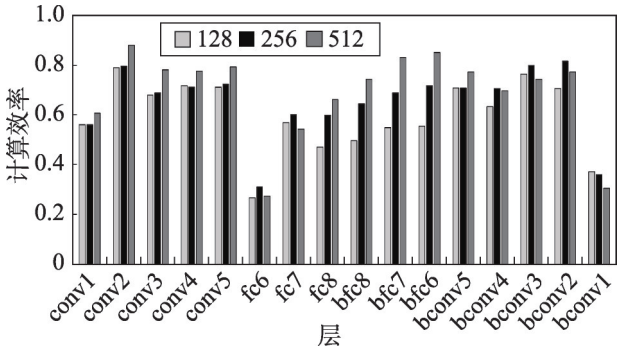

Fig.11 Compute efficiency of main layers over different batchsizes

图11 不同批次下关键层的计算效率

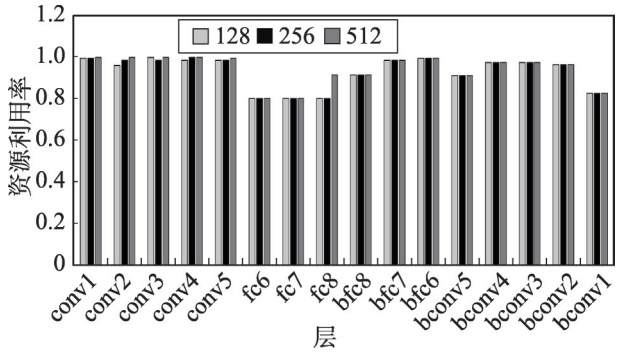

Fig.12 Resource utilization of main layers over different batchsizes

图12 不同批次下关键层的资源利用率

当增加批次会增加GPU的计算效率。

从资源利用率角度看，不同批次并没有太大差别且都大于80%。这是因为在前向传播中，尽管批次的差异会造成*Gridsize*差别很大（与批次大小成正比），但是不同批次的*blocksize*和*r*完全相同，因此资源利用率基本相同。fc8层512批次的GPU资源利用率较高的原因是完成所有*Gridsize*需要的时钟周期*cycles*不严格地与*Gridsize*成正比。另外，在反向传播中没有出现这种现象，这是因为在反向传播中参数梯度的计算，参数的数量不会随着批次改变。总之，计算效率会在一定程度上随着批次的增大而提高，但是批次对资源利用率的影响极小，基本可以忽略。因此计算效率和资源利用率并不是严格正相关。

5.4 显存利用

图13展示了不同网络模型的显存占用情况。可以看出，随着人们对模型计算精度需求的提升，网络模型不断增大，显存使用越来越大。在当前的框架中，为了提升网络的训练过程速度，所有的数据包括权值、特征图和预取图像都存放在显存，通过这种方式减少GPU对内存的访问，减低内存与显存数据拷贝的时间开销。但是当网络模型所需显存大于显存的最大容量时，模型的训练过程将无法执行。另外，尽管总的显存使用量很高，但是层级的显存利用率不到20%。在分析GPU计算效率和资源利用率的同时，很有必要关注并量化地分析训练过程中显存的使用情况。

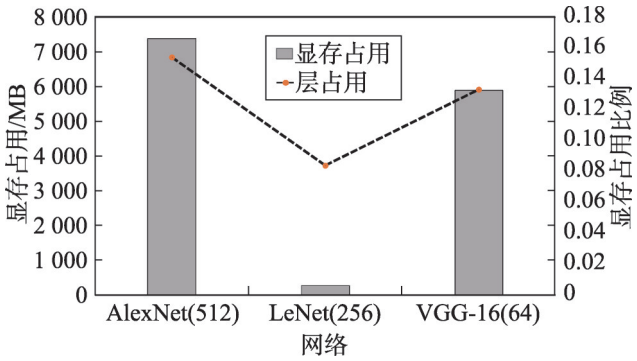

Fig.13 GPU memory allocation size and max layer-wise usage

图13 不同模型的显存分配和层级的最大显存使用

为了探究不同网络模型下显存的使用情况，量化显存利用率，对LeNet、AlexNet和VGG网络每层的显存占用情况进行了详细的分析，结果如图14~图16所示。

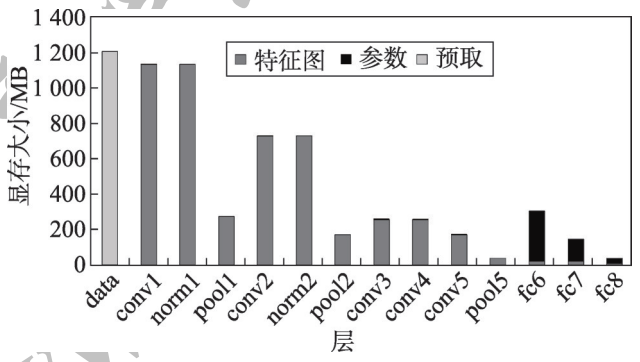

Fig.14 GPU memory allocation size of each layer in AlexNet(512)

图14 AlexNet(512)网络每层显存占用情况

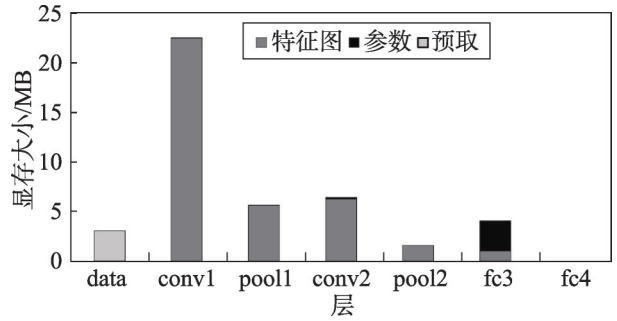

Fig.15 GPU memory allocation size of each layer in LeNet(256)

图15 LeNet(256)网络每层显存占用情况

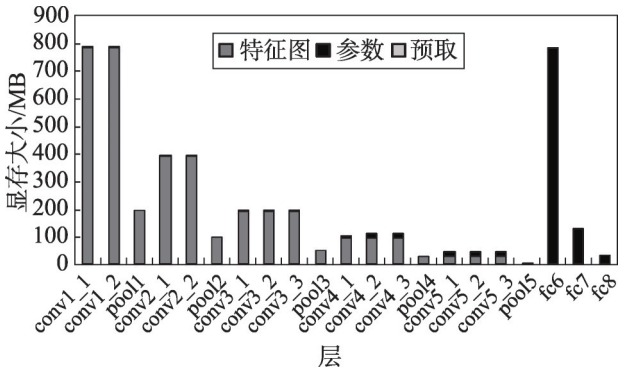

Fig.16 GPU memory allocation size of each layer in VGG-16(64)

图16 VGG-16(64)网络每层显存占用情况

前两个卷积层和第一个全连接层显存占用较高,因为前两个卷积层使用大量卷积核在原始图像上进行特征提取(比如AlexNet的conv1层有96个卷积核),生成大量特征图。全连接层将特征图映射到样本空间,参数数据量远大于特征图量。Relu层和drop层没有显存分配,因为这些层在卷积层上进行原地更新。

虽然训练阶段显存占用量很大,但是每层的显存占用较低,因此根据每层的数据量对显存进行合理的分配将会提高GPU训练大型网络的能力。

6 总结

本文针对神经网络的训练阶段进行了细粒度的分析,量化地给出了数据预取、数据拷贝、网络计算等阶段的时间延时以及每层的时间延时。对于网络计算阶段,探讨了GPU加速库、网络模型和批次对GPU计算效率和资源利用率的影响。另外,给出了不同网络模型每层的参数、特征图等的显存占用量,指出了通过调度每层显存分配能实现训练大规模神经网络。

References:

[1] Schroff F, Kalenichenko D, Philbin J. Facenet: a unified embedding for face recognition and clustering[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, Jun 8-10, 2015. Washington: IEEE Computer Society, 2015: 815-823.

[2] Lu Yao, Chowdhery A, Kandula S. Optasia: a relational platform for efficient large-scale video analytics[C]//Proceedings of the 7th ACM Symposium on Cloud Computing, Santa Clara, Oct 5-7, 2016. New York: ACM, 2016: 57-70.

[3] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Proceedings of the 26th Annual Conference on Neural Information Processing Systems, Lake Tahoe, Dec 3-6, 2012. Red Hook: Curran Associates, 2012: 1097-1105.

[4] Shelhamer E. Caffe models bvlc_alexnet[EB/OL]. (2017-04-21) [2017-07-21]. https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet.

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

[6] Emer J, Sze V, Chen Y H. Benchmarking metrics for DNN hardware[EB/OL]. (2016) [2017-07-22]. http://www.rle.mit.edu/eems/wp-content/uploads/2016/11/Tutorial-on-DNN-7-of-9-Benchmarking-Metrics-for-DNN-Hardware.pdf.

[7] He Kaiming, Zhang Xiangyu, Ren Shaoqing, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, Jun 27-30, 2016. Washington: IEEE Computer Society, 2016: 770-778.

[8] Alwani M, Chen Han, Ferdman M, et al. Fused-layer CNN accelerators[C]//Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture, Taipei, China, Oct 15-19, 2016. Washington: IEEE Computer Society, 2016: 22.

[9] Shen Yongming, Ferdman M, Milder P. Maximizing CNN accelerator efficiency through resource partitioning[C]//Pro-

ceedings of the 44th Annual International Symposium on Computer Architecture, Toronto, Jun 24-28, 2017. New York: ACM, 2017: 535-547.

[10] Shen Yongming, Ferdman M, Milder P. Overcoming resource underutilization in spatial CNN accelerators[C]//Proceedings of the International Conference on Field Programmable Logic and Applications, Lausanne, Aug 29-Sep 2, 2016. Piscataway: IEEE, 2016: 1-4.

[11] Song Lili, Wang Ying, Han Yinhe, et al. C-Brain: a deep learning accelerator that tames the diversity of CNNs through adaptive data-level parallelization[C]//Proceedings of the 53rd Annual Design Automation Conference, Austin, Jun 5-9, 2016. Piscataway: IEEE, 2016: 123.

[12] Zhang Yan, Shalabi Y H, Jain R, et al. FPGA vs. GPU for sparse matrix vector multiply[C]//Proceedings of the International Conference on Field-Programmable Technology, Monterey, Dec 9-11, 2009. Piscataway: IEEE, 2009: 255-262.

[13] Zhuo Ling, Prasanna V K. Sparse matrix-vector multiplication on FPGAs[C]//Proceedings of the ACM/SIGDA 13th International Symposium on Field Programmable Gate Arrays, Monterey, Feb 20-22, 2005. New York: ACM, 2005: 63-74.

[14] Song Mingcong, Hu Yang, Chen Huixiang, et al. Towards pervasive and user satisfactory CNN across GPU microarchitectures[C]//Proceedings of the International Symposium on High Performance Computer Architecture, Austin, Feb 4-8, 2017. Washington: IEEE Computer Society, 2017: 1-12.

[15] Rhu M, Gimelshein N, Clemons J, et al. vDNN: virtualized deep neural networks for scalable, memory-efficient neural network design[C]//Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture, Taipei, China, Oct 15-19, 2016. Washington: IEEE Computer Society, 2016: 18.

[16] Chetlur S, Woolley C, Vandermersch P, et al. cuDNN: efficient primitives for deep learning[J]. arXiv: 1410.0759, 2014.

[17] Vorotnikova D G, Golovashkin D L. CUBLAS-aided long vector algorithms[J]. Journal of Mathematical Modelling and Algorithms in Operations Research, 2014, 13(4): 425-431.

[18] Awan A A, Hamidouche K, Hashmi J M, et al. S-Caffe: co-designing MPI runtimes and caffe for scalable deep learning on modern GPU clusters[C]//Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Austin, Feb 4-8, 2017. New York: ACM, 2017: 193-205.

[19] Chen Y H, Emer J S, Sze V. Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks[C]//Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture, Seoul, Jun 18-22, 2016. Washington: IEEE Computer Society, 2016: 367-379.

[20] Bojnordi M N, Ipek E. Memristive boltzmann machine: a hardware accelerator for combinatorial optimization and deep learning[C]//Proceedings of the IEEE International Symposium on High Performance Computer Architecture, Barcelona, Mar 12-16, 2016. Washington: IEEE Computer Society, 2016: 1-13.

[21] Han Song, Liu Xingyu, Mao Huizi, et al. EIE: efficient inference engine on compressed deep neural network[J]. ACM SIGARCH Computer Architecture News, 2016, 44(3): 243-254.

[22] Mahajan D, Park J, Amaro E, et al. Tabla: a unified template-based framework for accelerating statistical machine learning[C]//Proceedings of the IEEE International Symposium on High Performance Computer Architecture, Barcelona, Mar 12-16, 2016. Washington: IEEE Computer Society, 2016: 14-26.

[23] Shafiee A, Nag A, Muralimanohar N, et al. ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars[J]. ACM SIGARCH Computer Architecture News, 2016, 44(3): 14-26.

[24] Chen Yunji, Luo Tao, Liu Shaoli, et al. Dadiannao: a machine-learning supercomputer[C]//Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, Dec 13-17, 2014. Washington: IEEE Computer Society, 2014: 609-622.

[25] Chen Tianshi, Du Zidong, Sun Ninghui, et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning[J]. ACM SIGPLAN Notices, 2014, 49(4): 269-284.

[26] Liu Daofu, Chen Tianshi, Liu Shaoli, et al. PuDianNao: a polyvalent machine learning accelerator[J]. ACM SIGARCH Computer Architecture News, 2015, 43(1): 369-381.

[27] Shi Shaohuai, Wang Qiang, Xu Pengfei, et al. Benchmarking state-of-the-art deep learning software tools[C]//Proceedings of the 7th International Conference on Cloud

Computing and Big Data, Macau, China, Nov 16-18, 2016. Washington: IEEE Computer Society, 2016: 99-104.

[28] Li Chao, Yang Yi, Feng Min, et al. Optimizing memory efficiency for deep convolutional neural networks on GPUs [C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, Nov 13-18, 2016. Washington: IEEE Computer Society, 2016: 633-644.

[29] Jin Jing, Lai Siyan, Hu Su, et al. GPUSGD: a GPU-accelerated stochastic gradient descent algorithm for matrix factorization[J]. Concurrency & Computation Practice & Experience, 2016, 28(14): 3844-3865.

[30] Konečný J, Liu Jie, Richtárik P, et al. Mini-batch semi-stochastic gradient descent in the proximal setting[J]. IEEE Journal of Selected Topics in Signal Processing, 2016, 10 (2): 242-255.

[31] Wang Mengdi, Fang E X, Liu Han. Stochastic compositional gradient descent: algorithms for minimizing compositions of expected-value functions[J]. Mathematical Programming, 2017, 161(1/2): 419-449.

[32] The Berkeley Vision and Learning Center (BVLC). BVLC caffe[EB/OL]. (2017-05-19) [2017-07-19]. https: //github.com/BVLC/caffe.

[33] The Berkeley Artificial Intelligence Research (BAIR) Lab. Deep learning framework: caffe[EB/OL]. [2017-07-19]. http://caffe.berkeleyvision.org/.

[34] Lecun Y, Cortes C, Burges C J C. The MNIST database of handwritten digits[EB/OL]. http://yann.lecun.com/exdb/mnist/.

[35] Russakovsky O, Deng Jia, Su Hao, et al. Imagenet large scale visual recognition challenge[J]. International Journal of Computer Vision, 2015, 115(3): 211-252.

[36] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

LI Jingjun was born in 1994. He is an M.S. candidate at Huazhong University of Science and Technology. His research interests include computer architecture and machine learning.

李景军(1994—),男,华中科技大学硕士研究生,主要研究领域为系统结构,机器学习。

ZHANG Chen was born in 1993. He is a Ph.D. candidate at Huazhong University of Science and Technology. His research interests include deep learning, computer architecture and machine learning.

张宸(1993—),男,华中科技大学博士研究生,主要研究领域为深度学习,系统结构,机器学习。

CAO Qiang was born in 1975. He is a professor at Huazhong University of Science and Technology. His research interests include computer architecture, machine learning, large scale storage system and its design and optimization, etc.

曹强(1975—),男,华中科技大学教授,主要研究领域为系统结构,机器学习,大规模存储系统及其设计与优化等。