doi:10.3772/j.issn.1002-0470.2023.03.006

基于多段插值拟合的深度神经网络非线性层加速方法①

黄一凡②* 张 欣** 支 天** 张 蕊③** 张曦珊** 周学海*

(*中国科学技术大学计算机科学与技术学院 合肥 230026)

(**中国科学院计算技术研究所 北京 100190)

摘 要 针对传统量化算法无法应用于非线性运算层的问题,本文提出了一种基于多段插值拟合的非线性层加速方法,利用插值表存储插值函数的参数,通过查表来计算得到非线性层的输出结果。使用本方法,可以在对非线性层进行有效加速的同时实现拟合误差可控。此外在硬件部署时,仅需要基础硬件指令支持,在边缘端和服务器都可以部署。实验结果表明,使用本文提出的多段插值方法拟合多种非线性层,可以取得平均 1.44 倍的加速效果。这种非线性层可以方便快捷地部署在图像分类、自然语言处理和机器翻译等多种任务模型上,并且每个模型对拟合精度有不同需求的情况下,均可以保证推理和训练精度损失小于 0.5%。

关键词 深度神经网络(DNN);量化;非线性层加速;多段插值拟合

0 引 言

近年来,深度神经网络(deep neural networks, DNN)取得了突破性进展,已经在计算机视觉、商业、金融和医疗保健等多个领域得到广泛应用[1-2],收获巨大成功。为了处理更加复杂的任务,达到更高精度,深度神经网络的规模日趋庞大,耗费的计算资源和存储资源越来越多,推理和训练的开销变得难以负荷[3]。此外,深度神经网络存在过度参数化问题[4],许多参数是冗余的,它们对有效信息的传递贡献极小。因此,深度神经网络的压缩优化引起了研究人员的广泛关注。

量化是一种广泛应用的模型压缩方法,可以有效加速深度神经网络推理和训练的过程。主流的量化方法是在保证模型精度的前提下,用低位宽的定点数代替浮点数,对模型进行简化。模型量化可以减少计算和存储开销[5-7],从而能够将深度神经网络部署在性能功耗受限的边缘端设备上。

传统的量化方法适用于在推理和训练时对卷积层、全连接层等线性层进行加速,但无法处理包含非线性运算的层,如 Sigmoid 层、Softmax 层和 BatchNorm 层等。这是由于量化非线性层时,如果使用定点数计算开方、超越函数等非线性运算会导致误差过大;如果使用低位宽的浮点数,由于表示范围有限容易出现溢出。在最近兴起的基于 Transformer 的相关模型中,Softmax 等非线性层占总计算时间的 40%,加速非线性层可以使模型性能获得较大提升[8]。一些工作利用泰勒展开等高阶展开方法,将非线性运算替换成乘积、移位等适合量化的计算[9-11]后再进行量化。但这种方法需要对不同的非线性层设计专用的高阶展开公式,通用性差。因此,如何在推理和训练时对非线性层进行加速是一个亟待解决的问题。

① 国家重点研发计划(2020AAA0103802),国家自然科学基金(61925208,61906179,62102399,U20A20227),中国科学院战略性先导科技专项(XDB32050200)和中国科学院稳定支持基础研究领域青年团队计划(YSBR-029)资助项目。

② 男,1997 年生,硕士生;研究方向:计算机体系结构;E-mail:hyf15@mail.ustc.edu.cn。

③ 通信作者,E-mail:zhangrui@ict.ac.cn。

(收稿日期:2021-12-16)

在训练过程中对非线性层进行量化尤为困难,一些推理的量化方法也不再适用。比如推理时,可以将 BatchNorm 层融入前面的卷积层,再对卷积层进行量化[12]。但训练时反向传播过程需要更新 BatchNorm 层的均值和方差,无法将 BatchNorm 层融进前面的卷积层。随着深度神经网络参数量呈爆炸式增长,它的训练成本也变得极其高昂。降低超大模型的训练开销已成为一个重要的难题。尽管现有的量化方法可以实现在训练过程中对线性层的有效加速,但对于非线性层加速的研究仍不完善。因此,非线性层的加速对于深度神经网络训练过程的进一步加速具有重要意义。

针对这一问题,本文提出了一种基于多段插值拟合的深度神经网络非线性层加速方法。将非线性函数划分成多段,每段用不同的线性函数拟合,这些预定义好的线性函数的参数存放在插值表中,通过查表找到输入对应的线性函数,计算得到非线性层的输出结果。可以通过增加划分的插值段数来减小拟合误差,相应的查表开销也会增加,利用浮点数值格式的特点,将不同插值段数的查表开销降到 $O(1)$。

本文的主要贡献如下。

(1)提出一种利用多段插值来拟合各种非线性函数的方法。在对不同非线性层进行有效加速的同时实现拟合误差可控,此外在硬件部署时,仅需要基础的算术逻辑指令支持,可以应用在边缘端和服务器等不同硬件平台上。

(2)基于 Pytorch 框架实现了插值算子,可以拟合各种非线性层,实现相同的前传、反传功能。相较于原生非线性层,插值算子可达平均 1.44 倍的加速比。

(3)在图像分类、自然语言处理和机器翻译等多个任务模型上测试,使用插值算子代替非线性层。每个模型对拟合精度有不同需求的情况下,均可以保证推理和训练精度无损。

1 相关工作

本节将介绍量化推理、量化训练的主流方法,及这些方法加速非线性层的局限性。并介绍非线性层加速的相关工作。

1.1 推理和训练的量化

量化推理可以加速神经网络的推理过程。主流的量化推理方法可以根据是否需要重训练分为 2 大类。训练后量化(post training quantization,PTQ)是最简单的量化推理方法。它直接对浮点预训练好的模型量化,无需重训练,可以应用在任何模型上。如 Banner 等人[13]将卷积神经网络(convolutional neural networks,CNN)量化到 4 位。Choukroun 等人[14]将不同网络量化到 4 位。一些工作解决了训练后量化导致的精度损失问题,如 Nagel 等人[15]成功把 MobileNetV2 量化到 8 位。Fong 等人[16]提出了分段线性量化。微调(fine-tuning)则通过重训练来减小量化误差对推理精度的影响,在保证精度的前提下可以把模型量化到更低位宽。Jacob 等人[12]提出了量化感知训练(quantization aware training,QAT);Rastegari 等人[17]提出了三值量化网络 XNOR-Net。上述方法在推理时对线性层进行量化,较少对非线性层的加速进行讨论。

量化训练着眼于加速神经网络的训练过程。一些工作将网络的权重、激活值、梯度量化成定点数,Zhou 等人[18]提出了 Dorefa-Net。Wu 等人[10]提出了 WAGE 全定点量化。一些工作通过训练时动态调整量化位宽,降低量化误差,Zhang 等人[19]提出了自适应调整量化位宽方法。Fu 等人[20]启发式地决定每一层的量化位宽。一些工作着眼于数值格式的改进。Johnson[21]设计了全新的 8 位数据格式和硬件实现。Wang 等人[22]提出一种 8 位浮点格式。Sun 等人[23]提出前传反传使用不同指数位宽的 8 位混合浮点格式。一些新的数据格式,如 TF32 和 Bfloat16[24],可以应用在 NVIDIA 图形处理器(graphics processing unit, GPU)等硬件上,取得显著的性能收益。由于训练相较于推理更不稳定,上述方法通常让非线性层保持全精度运算,没有考虑非线性层的加速。

1.2 非线性层加速

以卷积神经网络为代表的深度神经网络,卷积层、全连接层占计算量的绝大部分,这些线性层满足 $L(S\boldsymbol{q}) = S \cdot L(\boldsymbol{q})$,这意味着对量化输入 $\boldsymbol{q}$ 进行线性运算之后,乘以扩放因子 S,与用浮点输入 $S\boldsymbol{q}$ 计算得到的结果是一致的[11]。但非线性层不满足这个

性质,使用量化输入 q 计算,会导致极大的误差。而在最近兴起的 Transformer 模型中,Softmax 等非线性层计算是一笔不可忽略的开销。为了进一步加速推理和训练,非线性层的加速尤为重要。

可以在软件层面对非线性层进行加速。一些量化方法通过精妙设计来代替非线性运算,从而可以对非线性层进行量化加速。如 Courbariau 和 Bengio[9]提出的二值神经网络(binary neural network, BNN)用移位运算代替 BatchNorm 层和 ADAM 优化器。Wu 等人[10]提出的 WAGE 量化,通过合适的权值初始化来代替 BatchNorm 层,Softmax 函数、交叉熵函数则用最小平方差代替。Kim 等人[11]提出的全量化 BERT 在推理过程中,用高阶插值拟合函数代替 GELU、Softmax 等非线性函数。这些方法只适用于特定模型,通用性差,且不适用于训练过程。有些工作通过使用新的数值格式来加速非线性层,如 NVIDIA 的 Apex 混合精度框架可以支持任意模型的训练。但低位宽的浮点格式容易导致溢出,难以应用在非线性层上。

一些工作从硬件层面对非线性层进行加速。针对非线性函数的专用硬件设计,可以加速其运算效率,如基于坐标旋转数字计算(coordinate rotation digital computer, CORDIC)算法[25],设计专用计算器来加速三角函数、双曲线函数、指数和对数等运算。由于需要专有硬件支持,加速大规模非线性运算时,硬件成本过于高昂。一些深度学习加速器利用神经网络的鲁棒性,通过查表等方式实现了非精确的非线性函数计算,如 Chen 等人[26]在 2014 年提出的 DianNao 加速器,使用插值表计算非线性的激活函数。NVIDIA 提出的深度学习加速器 NVDLA[27],其中采用了查找表(look-up table,LUT)的硬件设计。Lai 等人[28]基于 ARM(advanced RISC machines)的边缘端中央处理器(central processing unit, CPU) Cortex-M,提出了通过查表实现定点 Sigmoid 和 Tanh 的方法。这类方法使用较低的插值段数,并且插值段数固定,适用于推理过程对非线性层的加速,而训练过程对非线性层精度更敏感,它们无法保证网络训练收敛。

深度神经网络包含多种非线性层,不同模型对非线性层有不同的精度要求。如何设计一种通用的非线性层加速方法,适用于不同非线性层,能满足不同模型推理训练过程的精度要求,可以部署在常用的硬件平台上并取得加速效果,是一个需要解决的问题。基于此,本文提出了一种基于插值的非线性层加速策略,适用于不同的非线性函数和深度神经网络,拟合的误差可控。可以通用地加速各种非线性层,并保证模型推理和训练的精度。

2 插值设计

本节提出了一种基于分段插值的非线性层实现方式。首先分析了非线性层的优化方向,确保能在实际硬件上取得加速效果。然后给出了插值拟合的理论误差分析,说明拟合误差是可控的。接着提出了具体的多段插值拟合方法设计,和基于高阶展开的拟合方法对比,验证本文提出的插值设计是更优的。

2.1 非线性层优化方向

将浮点数量化成定点数的量化方法,利用定点运算在硬件上执行效率优于浮点运算,可以取得加速效果。区别于线性层只需要乘加计算,非线性层的运算更为复杂。为了在硬件平台取得实际加速效果,本节将分析非线性层的执行过程,提出可行的优化方向。

非线性层包含开方、超越函数等非线性函数。不同的处理器通过不同方式实现这些非线性函数,对其进行硬件或者软件优化,并保证函数计算结果的误差符合 IEEE 754 标准[29]。最后将具体实现封装成函数库,供高层语言调用。根据需要的非线性运算次数区分,一些层的非线性计算次数与输入规模无关,如 BatchNorm 层,仅在计算方差时需要有限次开方、求倒数操作。另一部分的计算次数正比于输入规模,如 Tanh、Sigmoid 等激活层,每个输入都需要若干次非线性计算。随着输入规模的增长,后一类非线性层的优化就变成了如何加速大规模的非线性运算。

大规模的非线性计算面临着硬件上的挑战。主流的深度神经网络加速器架构可以分成 2 类,一类是单指令多线程架构(single instruction multi thread, SIMT),如 NVIDIA GPU[30];另一类是单指令多数据(single instruction multi data,SIMD)架构,如 DianNao

加速器[26]。SIMT 架构有众多核心,单个核心可以通过浮点运算单元(float point unit,FPU)完成非线性计算;SIMD 架构则是单核心由多个浮点运算器件来支持多数据并行的非线性计算。无论是 SIMT 架构或是 SIMD 架构,在所有浮点运算器上部署非线性运算加速硬件,成本高昂。因此深度学习加速器一般通过简单硬件指令的组合实现非线性运算。

可以从软硬件 2 个角度进行大规模非线性计算的优化。软件层面上,由于深度神经网络的鲁棒性,可以采用精度更低、方便计算的函数来拟合非线性函数。而且这类非线性层绝大部分是激活层,对误差的容忍性高。硬件层面则是考虑如何提高计算的并行度。深度学习加速器有众多简单的运算器,对于单输入的非线性计算,由于不涉及归并等操作,可以用简单指令实现非线性计算,利用已有的运算器并行加速。

为了用简单指令实现非线性计算,需要一个合适的拟合函数。由于多项式只需要进行乘法、加法操作,无需复杂的硬件支持,因此被广泛采用。Stewart[31]总结了利用多项式来拟合函数的方法,本文选择用多段插值函数进行拟合的方式。

2.2 拟合误差分析

本节将分析多项式拟合的理论误差上界。根据拉格朗日插值定义,对于待拟合的函数 f,选择 $n+1$ 个数据点 $\{(x_0,y_0),(x_1,y_1),\cdots,(x_n,y_n)\}$,能找到一个经过这些点的 n 阶多项式,见表达式(1)。

$$L(x)=\sum_{i=0}^{n} y_i\, l_i(x), l_i(x)=\prod_{0\leqslant j\leqslant n, j\neq i}\frac{x-x_j}{x_i-x_j} \tag{1}$$

可以给出这个多项式对原函数的截断误差估计,见表达式(2)。$x\in[a,b]$,$\xi\in(a,b)$且依赖于 x。

$$|f(x)-L(x)|=\left|\frac{f^{n+1}(\xi)}{(n+1)!}(x-x_0)\cdots(x-x_n)\right| \tag{2}$$

当 $x\in[a,b]$时,已知高阶导数上限式(3),可以给出插值多项式 $L(x)$的截断误差上限式(4)。

$$\max_{a\leqslant x\leqslant b}|f^{n+1}(x)|=M_{n+1} \tag{3}$$

$$|f(x)-L(x)|\leqslant\frac{M_{n+1}}{(n+1)!}(x-x_0)\cdots(x-x_n) \tag{4}$$

容易看出,随着多项式的阶增加,截断误差逐渐趋近于 0。当 x 所在的区间$[a,b]$足够小,$L(x)$与$f(x)$的误差也无穷趋近于 0。

从而可以通过一个高阶函数,或者多段低阶函数拟合原函数,并将误差限制在给定范围内。如果需要更高的拟合精度,可以通过增加更多的拟合点,提高多项式函数的阶数;或者将拟合区间划分成更多段,使用更多的低阶拟合函数。

2.3 插值具体设计

本文提出的插值设计利用多段线性函数来拟合非线性函数。将非线性函数划分成多段,每段用不同的线性函数拟合,这些预定义好的线性函数的参数存放在插值表中,通过查表找到输入对应的线性函数,计算得到非线性层的输出结果。

首先是插值段的划分。待拟合的函数 f 定义域记为$[a,b]$。当$x_{\min}$、$x_{\max}$或$f(x_{\min})$、$f(x_{\max})$趋于无穷大时,无法在整个定义域上进行插值拟合。所以截取一个长度有限的插值区间$[a,b]$,它是 f 定义域的一个子集,在这个区间进行插值拟合。将其划分成 n 段,每一段的左右端点记为x_{i-1}、x_i($x_0=a$,$x_n=b$),每段插值区间长度为x_i-x_{i-1},第 i 个插值函数的表达式为式(5)。特别地,当$x_i\in(-\infty,a)$或$x_i\in(a,\infty)$时,分别使用区间$[x_0,x_1]$和$[x_{n-1},x_n]$的插值函数来计算$f(x_i)$。

$$L_i(x)=k_ix+b_i=\frac{f(x_i)-f(x_{i-1})}{x_i-x_{i-1}}x+\frac{x_{i-1}f(x_i)-x_if(x_{i-1})}{x_i-x_{i-1}} \tag{5}$$

这些预定义的插值函数,其斜率k_i、截距b_i、插值区间$[x_{i-1},x_i]$存放在表中,如图 1 所示。非线性层运算时,加载预定义的表,对于每一个输入 x,通过查表找到其所属区间,计算插值函数$L_i(x)=k_ix+b_i$作为输出。

差值表			
下标	区间范围	截距	斜率
1	[x0,x1]	b1	k1
2	[x1,x2]	b2	k2
3	[x2,x3]	b3	k3
…	…	…	…
n	[x(n-1),xn]	bn	kn

图 1 插值表

增加插值段数可以减小拟合误差。随着划分区间的增加，查找x_i所属的区间开销较大，最坏情况下，查表开销为$O(n)$。本文基于浮点数值格式，设计了一种特殊的插值区间分段方式，将查表开销降为$O(1)$。首先对浮点格式进行分析。32 位浮点输入x_i的表示格式如图 2 所示。其中，符号占 1 位，0 代表正数，1 代表负数。指数占 8 位，用二进制表示的指数值换算成十进制，再减去 127 可以得到真实指数值。二进制尾数可以写成 1. xxxx 的形式，小数点左侧恒为 1，23 位尾数位仅保存小数点之后的部分。尾数共有 24 位有效数字。分别记这 3 个值为$sign_x$、exp_x、$frac_x$，则$x_i = sign_x \cdot 2^{exp_x} \cdot frac_x$。容易看出，当符号、指数部分相同时，比较 2 个浮点数的大小仅需要比较尾数部分。

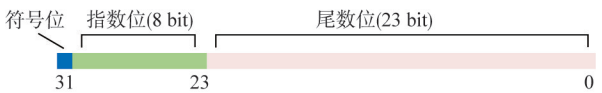

图 2　32 位浮点数格式

限定区间总长度为 2 的整次幂，即$b = a + 2^m$。将插值区间均匀划分成2^n段，每一段的长度为2^{m-k}。输入$x_i \in [a,b]$时，设$\acute{x} = x_i - a$，$\acute{x}$是一个不小于 0 的数。若x_i落在第 k 个区间$[x_{k-1}, x_k]$，$\acute{x}$相应地落在区间$[(k-1)\cdot 2^{m-n}, k\cdot 2^{m-n}]$。

注意到浮点数$2.0^m = +1\cdot 2^m \cdot 1$，浮点表示中，尾数位全为 0，补上隐藏的有效位，如图 3 所示。

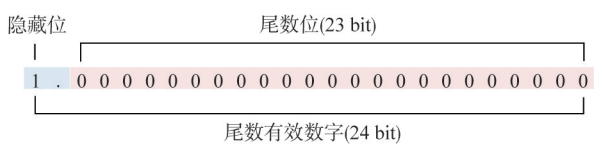

图 3　二进制尾数表示

浮点数$2.0^{m-n} = +1\cdot 2^{m-n}\cdot 1$，将指数位对齐到$2^m$，则$2.0^{m-n} = +1\cdot 2^m \cdot 2^{-n}$，尾数的二进制表示如图 4 所示。尾数中被隐藏的 1 移到了小数点后第 n 位。基于这种对齐后的表示，$k\cdot 2.0^{m-n}$的二进制尾数，小数点后前 n 位，正好是整数 k 的二进制表示。

记$\acute{x} = sign_{\acute{x}} \cdot 2^{exp_{\acute{x}}} \cdot frac_{\acute{x}}$，$sign_{\acute{x}} = \pm 1$。将指数对齐到$2^m$，即$\acute{x} = sign_{\acute{x}} \cdot 2^m \cdot (2^{exp_{\acute{x}}-m} \cdot frac_{\acute{x}})$。尾数相应地乘以$2^{exp_{\acute{x}}-m}$，对于二进制表示的尾数，等价于

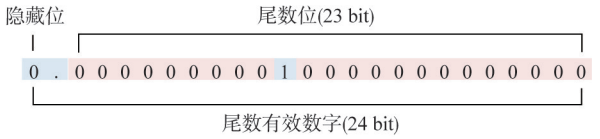

图 4　对齐后的二进制尾数

左移$exp_{\acute{x}} - m$ 位。将$\acute{x}$的二进制尾数做如图 4 的处理，$\acute{x}$落在区间$[(k-1)\cdot 2^{m-n}, k\cdot 2^{m-n}]$时，其二进制尾数，小数点右侧前 n 位恰好是整数 k 的二进制表示。从而可以将这 n 位作为索引，查找$\acute{x}$所属的插值区间，相应地也能找到x_i所属的插值区间。

上述过程的伪代码如算法 1 所示。

算法 1　基于浮点数值格式的查找插值函数算法

1. 已知浮点输入 x，区间长度2^m，插值段数2^n，区间左右端点 a，b，斜率数组 $pk[\,]$和截距数组 $pb[\,]$
2. if $x \leqslant a$ then
3. return $x \times pk[0] + pb[0]$
4. elif $x \geqslant b$ then
5. return $x \times pk[n-1] + pb[n-1]$
6. else
7. $x_{new} = x - a$
8. 不改变x_{new}的二进制表示，将x_{new}重解释成一个 32 位的定点数x_{int}
9. $x_{exp} = ((x_{int} \,\&\, 0x7F800000) >> 23) - 127$
10. $x_{mant} = (x_{int} \,\&\, 0x007FFFFF) \mid 0x00800000$
11. $shift = m - x_{exp}$
12. $index = ((x_{mant} >> shift) \,\&\, 0x007FFFFF) >> (23-n)$
13. return $x \times pk[index] + pb[index]$
14. end if

基于这种特殊设计，可以将任意 $x \in [a,b]$，通过有限次定点加法运算和位运算，得到 x 所在的插值区间，时间复杂度为 $O(1)$。对于 x 落在区间$[a,b]$外的情况，只需要 2 次比较操作，就可以判断需要使用哪一个插值函数。这种简单设计可以进一步用定制硬件加速，可以很方便地部署在流水线上。

2.4　高阶展开与多段插值对比

根据 2.2 节的分析，为了减小误差，可以将原函数展开成更高阶的拟合函数，或者将插值区间划分得更密。本文选择了第 2 种做法，可以从性能开销和内存开销对 2 种方法进行对比。

对于 n 阶拟合函数$L_n(x)=\sum_{i=0}^{n}C_i x^i$,改写成表达式(6),可以通过 n 次乘加运算实现。可以看出,计算开销正比于阶数 n,对于不同非线性函数、不同精度要求,需要展开成不同阶的拟合函数。相比之下,2.3 节提出的插值设计,计算所需的浮点操作次数,与拟合函数、插值精度无关。

$$L_n(x)=(\cdots((C_n x+C_{n-1})x+C_{n-2})\cdots)x+C_0 \tag{6}$$

对于高阶拟合函数,硬件无法在一次计算中完成 n 次连乘,需要保存中间结果。Sigmoid 等激活层,每一个输入都需要非线性运算,当输入规模较大,超过内存限制时,使用高阶拟合函数,有一半的存储空间浪费在保存中间结果上,还会影响计算效率。如果采用分段插值的做法,每个非线性层预先将区间表和插值表加载到内存中,m 段插值函数需要 $2+2^{m+1}$个浮点参数,实际应用中 m 小于 12,相较于输入的规模几乎可以忽略。容易看出,多段插值的内存开销远低于高阶拟合函数。

对于大规模非线性运算,内存无法装载所有输入,如果使用高阶展开拟合,计算过程中一半的内存空间浪费在保存中间结果,并导致计算效率降低。此外,不同的高阶展开函数计算过程不一致,不利于流水线加速。相比之下,分段函数更适合作为拟合函数。

3 拟合误差

3.1 拟合误差公式

为了更精确地衡量拟合函数和原函数的误差,在拟合区间$[a,b]$,定义L^2距离和L^∞距离作为拟合误差,表达式分别为式(7)和式(8)。其中 y'_i 和y_i分别为输入x_i的拟合函数值、真实函数值,n 为采样点个数。当均匀分布在拟合区间$[a,b]$的采样点足够多时,L^2距离和L^∞距离可以反映真实拟合误差的大小。

$$L^2\text{距离}=\sqrt{\sum_{i=0}^{n}(y'_i-y_i)^2} \tag{7}$$

$$L^\infty\text{距离}=\max(\sqrt{(y'_i-y_i)^2}) \tag{8}$$

3.2 非线性层拟合

本节选择 4 种代表性的非线性层:Softmax、Sigmoid、Tanh、GELU 进行插值拟合。

Softmax 的计算公式如式(9)所示。它需要对每一个输入x_i计算指数结果e^{x_i},将所有指数结果求和后取倒数,乘在每个e^{x_i}上。随着输入的增大,exp 函数值呈爆炸式增长。为了避免数值过大,通常将输入减去最大值。

$$\text{Softmax}(x_i)=\frac{e^{x_i-x_{\max}}}{\sum_{j=1}^{n}e^{x_j-x_{\max}}}=\frac{e^{x_i}}{\sum_{j=1}^{n}e^{x_j}} \tag{9}$$

Sigmoid 的计算公式如式(10)所示。与 Softmax 不同,Sigmoid 没有不同输入之间的归并操作。其函数图像呈 S 型,在 0 附近变化大,随着输入趋于无穷,输出趋于稳定。

$$\text{Sigmoid}(x_i)=\frac{1}{1+e^{-x_i}} \tag{10}$$

Tanh 的计算公式如式(11)所示。它是一个双曲正切函数,函数图像与 Sigmoid 非常相似,都是呈 S 型,区别是 Tanh 函数关于原点中心对称。

$$\text{Tanh}(x_i)=\frac{e^{x_i}-e^{-x_i}}{e^{x_i}+e^{-x_i}} \tag{11}$$

GELU 的计算公式如式(12)所示,近似公式如式(13)。它可以视作一个更为平滑的 RELU 函数,常用于 Transformer 模型中,如 Bert[32] 和 GPT-3[33]。

$$\text{GELU}(x_i)=x_i\cdot\Phi(x)=x\cdot\frac{1}{2}\cdot\left[1+\text{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \tag{12}$$

$$0.5\cdot x\cdot\left(1+\tanh\left[\sqrt{\frac{2}{\pi}}(x+0.044\,715\cdot x^3)\right]\right) \tag{13}$$

由 2.2 节的分析可知,插值段数越多,拟合误差越小。为了更直观地说明,分别绘制这 4 类非线性函数的 8 段和 32 段插值的函数图像,如图 5~8。可以看出,随着插值段数的增加,分段插值函数图像与原始函数图像愈发贴近,说明拟合误差越小。此外,拟合函数的L^2距离和L^∞距离在不同区间量级差异大。以 GELU 函数为例,在区间$[-0.25,0.375]$的拟合误差比区间$[-5.0,-4.25]$大 500 倍。插值段数每增加 4 倍,拟合函数的最大L^2距离和L^∞距离减少一个数量级。

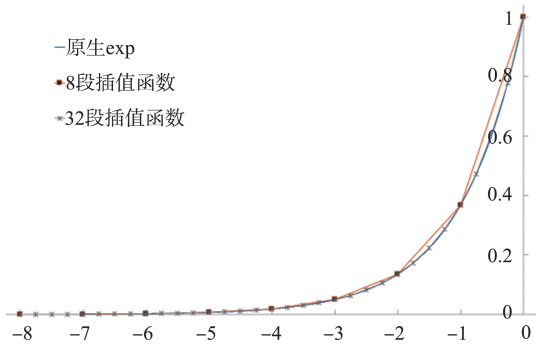

图 5　Exp 和拟合函数图像

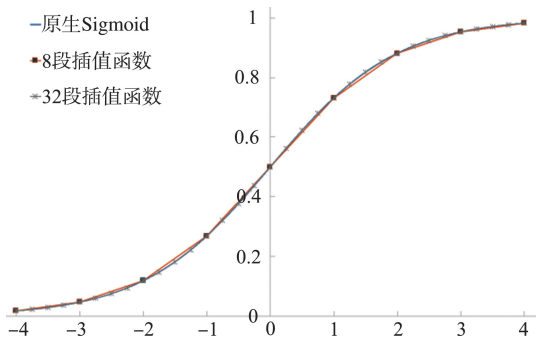

图 6　Sigmoid 和拟合函数图像

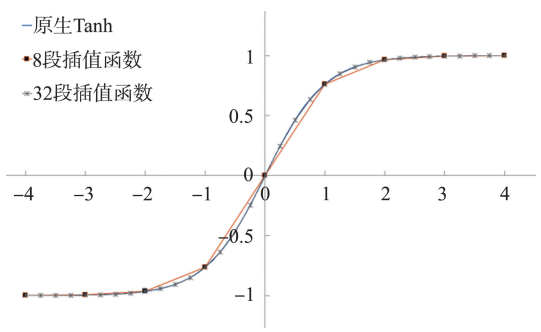

图 7　Tanh 和拟合函数图像

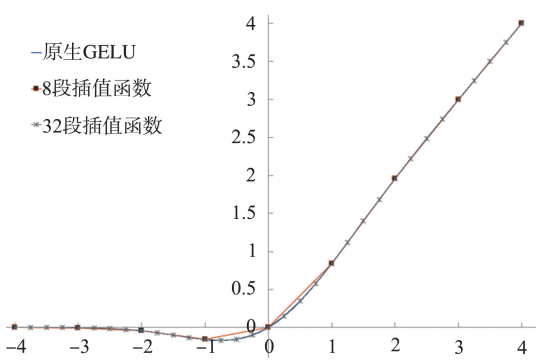

图 8　GELU 和拟合函数图像

4　实验和结果

本节介绍实验所用的实验平台和神经网络模型，评估多段插值拟合方法的性能收益，以及对模型推理训练的精度影响。

4.1　实验配置

实验平台：本文的实验平台选择 NVIDIA V100，采用 NVIDIA Volta 架构，内存为 32 GB，单精度算力为 14 TFLOPS。驱动版本为 460.32.03，CUDA 版本为 10.2。

实验模型：本文的实验模型见表 1。RegNetY、Vision Transformer 和 Swin Transformer 用于图像分类任务，GNMT 和 Transformer-Fairseq 用于机器翻译任务，Bert 用于自然语言处理任务。这些模型基于 github 的开源项目，使用 Pytorch-1.6.0 框架搭建。本文使用 Pytorch 框架的 CUDA 拓展算子功能，实现了 2.3 节的插值设计，在 GPU 上训练模型时，可以替代原生非线性层。

表 1　实验模型

模型	非线性层	数据集
RegNetY-400MF	Sigmoid	ImageNet
Transformer-Fairseq	Softmax	WMT16
GNMT-NGC	Tanh、Sigmoid	WMT16
Bert-Base-Uncased	GELU	Squad 1.1
Vision Transformer	GELU	ImageNet
Swin Transformer	GELU	ImageNet

4.2　性能对比

实验目的：对比非线性函数和插值函数的性能，验证插值函数的加速效果。

实验设计：非线性层执行时，通过调用底层数学库的非线性函数完成计算。以 Pytorch 1.6.0 框架在 NVIDIA GPU 的执行为例。第 3 节中 4 类非线性层在 Pytorch 框架对应的算子，以及在 GPU 上调用的非线性函数如表 2 所示。实际性能测试中，将表 2 中的非线性函数和相应的插值函数分别封装成 kernel 函数，随机生成 5 000 000 个测试数

据,利用 nvprof 工具分别统计这些 kernel 函数的总执行时间,多次执行取平均值。exp 的插值区间取[-16.0,0],Tanh 的插值区间取[-4.0,4.0],Sigmoid 的插值区间取[-8.0,8.0],GELU 的插值区间取[-5.0,3.0]。插值段数分别取 64、256、1024 和 4096。

表 2 非线性层

非线性层	Pytorch 算子	非线性函数
Softmax	torch. nn. Softmax	std::exp
Tanh	torch. nn. Tanh	std::Tanh
Sigmoid	torch. nn. Sigmoid	1 /(1 + std::exp)
GELU	torch. nn. GELU	::normcdff

表 3 性能测试

非线性函数	执行时间 1 原生函数	执行时间 2 64 段插值	执行时间 3 256 段插值	执行时间 4 1024 段插值	执行时间 5 4096 段插值	加速比
exp	549.13 ms	484.70 ms	484.70 ms	484.72 ms	484.77 ms	1.13x
Tanh	574.43 ms	484.71 ms	484.72 ms	484.74 ms	484.83 ms	1.19x
Sigmoid	767.46 ms	484.70 ms	484.77 ms	484.74 ms	484.82 ms	1.58x
GELU	900.52 ms	484.64 ms	484.64 ms	484.64 ms	484.63 ms	1.86x

实验结果:性能测试如表 3 所示,分别测试了原生非线性函数的性能,以及不同插值段数下插值函数的执行时间。原生非线性函数的性能为执行时间 1。统计不同插值段数下插值 kernel 函数的总执行时间,分别记为执行时间 2、3、4、5。可以看出,对同种拟合函数,不同插值精度下执行时间相差在0.1 ms 以内,加速比近似相同,区别在于加载表的开销,以及存表的内存开销。插值函数在拟合不同非线性函数时,计算过程相同,所以执行时间也近似相同,差别在 0.1 ms 以内,区别在于加载的表内容不同。

可以看出,插值函数相较于这 4 种非线性函数,能取得平均 1.44 倍的加速效果。计算过程最为复杂的 GELU 函数,插值函数可以达到最高 1.86 倍的加速比,这也是符合预期的。

进一步分析插值方法对模型加速效果。表 1 中的 6 类模型,分别统计表 2 中这 4 类非线性层在前传、反传计算总占比,以及使用插值方法计算非线性函数,模型整体计算时间减少的最大比例。实验结果如表 4。

根据表 4 的结果可以看出,在传统 CNN 模型中非线性运算占比极低,但在 Transformer、LSTM 模型中,非线性计算有不可忽略的占比。加速非线性层可以有效地提高模型性能,尤其是训练过程。以 Swin Transformer 为例,8 卡训练需要 4 d 时间,使用插值方法计算非线性函数,最大可以节省 7.56 h。

表 4 模型加速效果

模型	非线性层计算占比	优化效果
Bert-Base-Uncased	前传:31.30%	前传:6.3%
	反传:35.28%	反传:7.1%
Swin Transformer	前传:13.95%	反传:5.8%
	反传:16.47%	前传:7.1%
Vision Transformer	前传:9.55%	前传:2.7%
	反传:11.27%	反传:3.2%
GNMT-NGC	前传:11.45%	前传:2.7%
	反传:13.84%	反传:3.6%
Transformer-Fairseq	前传:6.02%	前传:0.7%
	反传:7.28%	反传:0.9%
RegNetY-400MF	前传:0.44%	前传:0.2%

4.3 推理和训练精度

实验目的:验证多段插值拟合对不同模型的推理训练精度影响。

实验设计:表 1 的 6 个实验模型,基于 Pytoch-1.6.0框架,进行 8 卡的分布式训练(Bert 使用 4 卡),使用默认的训练超参。Pytorch 框架设置固定 seed,保证训练结果可复现。每个原生模型训练的最终精度作为 baseline。再将模型中的非线性层替换成插值算子,不同函数的插值段数均为 4096。插值推理过程加载原生训练得到的权重,使用插值算子进行推理,得到插值推理精度,如表 5 所示。插值训练过程使用插值算子,在其余条件与原生训练保持一致的情况下

重新训练,得到插值训练精度,如表 6 所示。

实验结果:使用插值算子与原生模型的推理精度差别在 0.2% 以内,对推理精度的影响极小。对拟合更敏感的训练过程,插值训练与原生训练的精度差别在 0.5% 以内,RegNetY-400MF 和 Bert-Base-Uncased 模型达到比原生训练更高的精度。实际应用中,将非线性层替换为插值实现,不会影响推理和训练的精度。

表 5 推理精度

模型	评价指标	原生精度/%	插值精度/%
RegNetY-400MF	Acc@1	84.09	84.06
	Acc@5	91.85	91.85
Transformer-Fairseq	BLEU	28.00	27.92
GNMT-NGC	BLEU	24.04	23.85
Bert-Base-Uncased	Exact	81.67	81.67
	F1	88.91	88.91
Vision Transformer	Acc@1	83.77	83.60
Swin Transformer	Acc@1	81.21	81.13
	Acc@5	95.52	95.45

表 6 训练精度

模型	评价指标	原生精度/%	插值精度/%
RegNetY-400MF	Acc@1	84.09	84.20
	Acc@5	91.85	92.00
Transformer-Fairseq	BLEU	28.00	27.79
GNMT-NGC	BLEU	24.04	23.62
Bert-Base-Uncased	Exact	81.67	81.68
	F1	88.91	88.92
Vision Transformer	Acc@1	83.77	83.50
Swin Transformer	Acc@1	81.21	81.09
	Acc@5	95.52	95.33

由于训练对拟合误差更加敏感,为了验证插值段数对最终训练精度的影响,分别对 RegNetY-400MF、Vision Transformer 和 Bert-Base-Uncased 3 个模型,使用不同插值精度训练。插值段数分别为 64、256、1024、4096。3 个模型插值训练的精度见表 7、8、9。

可以看出,RegNetY-400MF 对 Sigmoid 函数的拟合精度并不敏感,4 种插值段数都可以保证最终训练精度。在插值段数为 256 和 1024 时,插值训练的精度高于原生训练精度。这是因为拟合误差可以视作是训练过程的噪声,噪声较小时,可以提高模型的训练效果,增强模型的泛化性,从而提高模型在测试集上的精度。

Vision Transformer 对 GELU 函数的拟合精度较为敏感,插值段数为 16、64、256、1024 时,插值训练的精度无法回到原生训练精度。对于这一类模型,需要使用更高精度的插值函数来拟合非线性函数。

表 7 RegNetY-400MF 插值训练精度

插值段数	评价指标	原生精度/%	插值精度/%	Diff
64	Acc@1	84.09	83.92	-0.17
	Acc@5	91.85	91.77	-0.08
256	Acc@1	84.09	84.36	0.27
	Acc@5	91.85	91.92	0.07
1024	Acc@1	84.09	84.04	-0.05
	Acc@5	91.85	91.82	-0.03
4096	Acc@1	84.09	84.20	0.11
	Acc@5	91.85	92.00	0.15

表 8 Vision Transformer 插值训练精度

插值段数	评价指标	原生精度/%	插值精度/%	Diff
16	Acc@1	83.77	77.69	-6.08
64	Acc@1	83.77	77.13	-6.64
256	Acc@1	83.77	80.21	-3.56
1024	Acc@1	83.77	82.15	-1.62
4096	Acc@1	83.77	83.5	-0.27

表 9 Bert-Base-Uncased 插值训练精度

插值段数	评价指标	原生精度/%	插值精度/%	Diff
64	Exact	81.67	81.32	-0.35
	F1	88.91	88.66	-0.25
256	Exact	81.67	81.61	-0.06
	F1	88.91	88.78	-0.13
1024	Exact	81.67	81.43	-0.24
	F1	88.91	88.72	-0.19
4096	Exact	81.67	81.68	0.01
	F1	88.91	88.92	0.01

Bert-Base-Uncased 和 Vision Transformer 都使用 GELU 激活函数,但 Bert 对 GELU 函数的拟合精度并不敏感,4 种插值精度都能达到原生训练精度。

可以看出,本文提出的插值拟合方式适用于对非线性层精度有不同要求的模型,可以保证推

理训练精度,不同拟合精度的非线性层都能取得性能收益。

4.4 加速方法横向对比

本节将与相关工作中的加速方法进行横向对比。首先与 I-Bert[11] 的非线性层拟合方法进行对比。它用 i-GELU 来拟合 GELU 层,将 GELU 量化到 INT32。见表达式(14),其中 $a=-0.2888$,$b=-1.769$。

$$\text{i-GELU}(x)=x\cdot\frac{1}{2}\left[1+L\left(\frac{x}{\sqrt{2}}\right)\right] \tag{14}$$

$$L(x)=\text{sign}(x)[a(\text{clip}(|x|,\max=-b)+b^2)+1] \tag{15}$$

与 i-GELU 的推理精度对比见表 10。本文提出的插值方法,用 4096 段插值拟合 GELU 函数,相比使用 Float32 格式和 INT32 格式的 i-GELU 拟合 GELU,在 Bert-Base-Uncased 模型上推理精度更优。

表 10 Bert-Base-Uncased 推理精度

	原生精度/%	推理精度/%	Diff
i-GELU: Float32	Exact: 81.67	Exact: 81.63	-0.04
i-GELU:	F1: 88.91	F1: 88.86	-0.05
INT32	Exact: 81.67	Exact: 81.54	-0.13
本文插值方法:	F1: 88.91	F1: 88.80	-0.11
4096 段	Exact: 81.67	Exact: 81.67	+0.00
	F1: 88.91	F1: 88.91	+0.00

与 i-GELU 的推理速度对比见表 11。本文使用插值拟合非线性层,相较于使用 Float32 的 i-GELU(x),加速比为 2.46 倍;相较于使用 INT32 的 i-GELU(x),加速比为 1.97 倍。使用插值拟合非线性层的推理性能更优。但 I-Bert 同时量化了线性层和非线性层,从而可以较好地改善模型的推理速度。本文对非线性层的计算过程进行了优化,可以与线性层的量化方法结合,进一步提高推理速度。

表 11 GELU 性能测试

	原生 GELU 用时/ms	执行时间/ms	加速比
i-GELU:Float32	900.52	1190.86	0.76
i-GELU:INT32	900.52	953.82	0.94
本文插值方法	900.52	484.63	1.86

与 DianNao 和 NVDLA 的插值方法进行对比。这 2 种方法为模型推理过程设计,使用了较低的插值段数,并且插值段数固定。DianNao 使用 16 段插值,NVDLA 使用 256 段插值。在 Vision Transformer 的训练过程中,使用 2 种方法的插值段数,最终训练精度为 77.69%(-6.08%)、80.21%(-3.56%)。可以看到,均无法保证 Vision Transformer 收敛到原生训练精度。本文使用 4096 段,最终训练精度为 83.50%(-0.27%),可以保证其收敛到原生训练精度。

容易看出,非线性层的计算误差可能导致模型训练精度受损,需要更高精度的插值拟合。现有 DianNao、NVDLA 的插值方法中,查表操作复杂度与插值段数直接相关,n 段插值的查表开销为 $O(\log n)$,因此其加速效果会随着表项的增大而下降。本文提出的插值方法,可以将查表开销降为 $O(1)$。

在 GPU 上对 DianNao 和 NVDLA 的插值方法进行模拟(普通插值方法),通过二分查找完成查表操作,进一步对比了在不同插值段下,不同插值方法的加速比,详见表 12。可以看到,相较于 DianNao(16 段),本文插值方法的加速比为 0.404 倍,相较于 NVDLA(256 段),加速比为 1.370 倍。插值段数为 4096 时,本文插值方法的加速比为 9.645 倍。可以看到,在高插值段数下,本文的插值方法加速效果更优,更适用于训练过程。

表 12 与原生 GELU 的加速比

插值段数	16	64	256	1024	4096
普通插值加速比	4.559	2.794	1.356	0.516	0.193
算法 1 加速比	1.858	1.858	1.858	1.858	1.858
算法 1 相对加速比	0.404	0.665	1.370	3.602	9.645

4.5 非均匀插值

2.3 节提出的插值设计均匀划分插值区间,在不同插值区间的拟合误差差异大。以 GELU 函数为例,在区间[-2.0,1.0]的拟合误差比区间[2.0,3.0]大一个数量级。控制不同区间的拟合误差接近,从而可以使用更少的插值函数,减小内存开销。

可以设计一种非均匀插值方法。将总插值区间 $[a,b]$ 划分成几个长度不等的子插值区间 $[a_0,b_0]$,

…,$[a_s,b_s]$($a_0=a,b_s=b$),选择合适的划分方式,使得不同子区间的拟合误差接近。每个子插值区间再利用 2.3 节提出的插值设计,都均匀划分成2^m段。查找输入x_i的插值函数时,先利用二分查找找到所属的子插值区间,再基于算法 1 计算得到结果。

以对 GELU 插值为例,该设计的性能如表 13 所示。分别对应均匀插值的 64 段和 4096 段。可以看到,当拟合误差越小时,非均匀插值的内存占用明显低于均匀插值,但执行时间显著高于均匀插值。在 Bert 上使用非均匀插值训练,可以达到原生训练的精度。考虑到插值表大小相较于输入规模可以忽略,均匀插值更适合作为非线性层的拟合方法。

表 13 非均匀插值函数性能测试

L^2距离	L^∞距离	执行时间/ms	插值段数	内存占用/Bytes
0.0051	0.0069	1666.63	48	384
1.4×10^{-6}	1.9×10^{-6}	1814.27	96	768

5 结 论

许多量化策略对线性层进行优化加速,缺乏针对非线性层的分析和探究。本文提出了一种基于插值的非线性层加速策略,利用多段线性插值来拟合各种非线性函数,通过调整插值段数,保证拟合误差可控,可以满足不同任务需求。采用特殊的插值表设计,查表、计算开销低,仅需要基础的硬件指令支持,方便部署。实验结果表明,本文提出的插值设计相较于原生非线性层,可以取得最高 1.86 倍的加速比。在图像分类、自然语言处理和机器翻译等任务模型上,不同任务对拟合精度有不同需求的情况下,都可以保证推理和训练精度无损,具有较好的通用性。

未来将考虑对本文提出的插值设计进行硬件优化,可以设计定制硬件来加快插值非线性层的计算,并利用流水线实现进一步的加速。除此之外,线性插值是量化友好的,可以考虑使用定点数来计算插值,从而实现全定点量化模型。

参考文献

[1] CAPRA M, BUSSOLINO B, MARCHISIO A, et al. Hardware and software optimizations for accelerating deep neural networks: survey of current trends, challenges, and the road ahead[J]. IEEE Access, 2020, 8: 225134-225180.

[2] 尹宝才, 王文通, 王立春. 深度学习研究综述[J]. 北京工业大学学报, 2015, 41(1): 48-59.

[3] SHAFIQUE M, THEOCHARIDES T, BOUGANIS C S, et al. An overview of next-generation architectures for machine learning: roadmap, opportunities and challenges in the IoT ERA[C]//2018 Design, Automation & Test in Europe Conference & Exhibition. Dresden: IEEE, 2018: 827-832.

[4] DENIL M, SHAKIBI B, DINH L, et al. Predicting parameters in deep learning[C]//Proceedings of the 26th International Conference on Neural Information Processing Systems. Red Hook: Curran Associates Inc, 2013: 2148-2156.

[5] CHENG Y, WANG D, ZHOU P, et al. A survey of model compression and acceleration for deep neural networks [EB/OL]. (2017-10-30)[2021-12-16]. https://arxiv.org/pdf/1710.09282v2.pdf.

[6] 曾焕强, 胡浩麟, 林向伟, 等. 深度神经网络压缩与加速综述[J]. 信号处理, 2022, 38(1): 183-194.

[7] 高晗, 田育龙, 许封元, 等. 深度学习模型压缩与加速综述[J]. 软件学报, 2021, 32(1): 68-92.

[8] STEVENS J R, VENKATESAN R, DAI S, et al. Softermax: hardware/software co-design of an efficient softmax for transformers[C]//The 58th ACM/IEEE Design Automation Conference. San Francisco: IEEE, 2021: 469-474.

[9] COURBARIAUX M, BENGIO Y. BinaryNet: training deep neural networks with weights and activations constrained to +1 or -1[EB/OL]. (2016-02-09)[2021-12-16]. https://arxiv.org/pdf/1602.02830v1.pdf.

[10] WU S, LI G, CHEN F, et al. Training and inference with integers in deep neural networks[EB/OL]. (2018-02-13)[2021-12-16]. https://arxiv.org/pdf/1802.04680.pdf.

[11] KIM S, GHOLAMI A, YAO Z, et al. I-BERT: integer-only BERT quantization[EB/OL]. (2021-01-08)[2021-12-16]. https://arxiv.org/pdf/2101.01321.pdf.

[12] JACOB B, KLIGYS S, CHEN B, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference[C]//2018 IEEE Conference on Computer Vision and Pattern Recognition. Salt Lake City: IEEE, 2018: 2704-2713.

[13] BANNER R, NAHSHAN Y, SOUDRY D. Post training 4-bit quantization of convolutional networks for rapid-deployment[C]//Annual Conference on Neural Information Processing Systems. Vancouver: NeurIPS, 2019: 7948-7956.

[14] CHOUKROUN Y, KRAVCHIK E, YANG F, et al. Low-bit quantization of neural networks for efficient inference [C]//2019 IEEE/CVF International Conference on Computer Vision Workshops. Seoul: IEEE, 2019: 3009-3018.

[15] NAGEL M, VAN BAALEN M, BLANKEVOORT T, et al. Data-free quantization through weight equalization and bias correction[C]//2019 IEEE/CVF International Conference on Computer Vision. Seoul: IEEE, 2019: 1325-

1334.
[16] FANG J, SHAFIEE A, ABDEL-AZIZ H, et al. Post-training piecewise linear quantization for deep neural networks[C]// The 16th European Conference on Computer Vision. Glasgow: ECCV, 2020: 69-86.
[17] RASTEGARI M, ORDONEZ V, REDMON J, et al. XNOR-Net: ImageNet classification using binary convolutional neural networks[C]// The 14th European Conference on Computer Vision. Amsterdam: ECCV, 2016: 525-542.
[18] ZHOU S, NI Z, ZHOU X, et al. DoReFa-Net: training low bitwidth convolutional neural networks with low bitwidth gradients[EB/OL]. (2018-02-02)[2021-12-16]. https://arxiv.org/pdf/1606.06160.pdf.
[19] ZHANG X, LIU S, ZHANG R, et al. Adaptive precision training: quantify back propagation in neural networks with fixed-point numbers[EB/OL]. (2019-11-01)[2021-12-16]. https://arxiv.org/abs/1911.00361.
[20] FU Y, YOU H, ZHAO Y, et al. FracTrain: fractionally squeezing bit savings both temporally and spatially for efficient DNN training[EB/OL]. (2020-12-24)[2021-12-16]. https://arxiv.org/pdf/2012.13113.pdf.
[21] JOHNSON J. Rethinking floating point for deep learning[EB/OL]. (2018-11-01)[2021-12-16]. https://arxiv.org/pdf/1811.01721.pdf.
[22] WANG N, CHOI J, BRAND D, et al. Training deep neural networks with 8-bit floating point numbers[C]// Annual Conference on Neural Information Processing Systems. Montréal: NeurIPS, 2018: 7686-7695.
[23] SUN X, CHOI J, CHEN C Y, et al. Hybrid 8-bit floating point(HFP8) training and inference for deep neural networks[C]// Annual Conference on Neural Information Processing Systems. Vancouver: NeurIPS, 2019: 4901-4910.
[24] KALAMKAR D D, MUDIGERE D, MELLEMPUDI N, et al. A study of BFLOAT16 for deep learning training[EB/OL]. (2019-01-13)[2021-12-16]. https://arxiv.org/pdf/1905.12322v3.pdf.
[25] VOLDER J E. The CORDIC trigonometric computing technique[J]. IRE Transactions on Electronic Computers, 1959, 8(3): 330-334.
[26] CHEN T, DU Z, SUN N, et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning[C]// Architectural Support for Programming Languages and Operating Systems. Salt Lake City: ASPLOS, 2014: 269-284.
[27] ZHOU G, ZHOU J, LIN H. Research on NVIDIA deep learning accelerator[C]// 2018 12th IEEE International Conference on Anti-Counterfeiting, Security, And Identification(ASID). Xiamen: IEEE, 2018: 192-195.
[28] LAI L, SUDA N, CHANDRA V. CMSIS-NN: efficient neural network kernels for arm cortex-m CPUs[EB/OL]. (2018-01-19)[2021-12-16]. https://arxiv.org/pdf/1801.06601.pdf.
[29] KAHAN W. IEEE standard 754 for binary floating-point arithmetic[J]. Lecture Notes on the Status of IEEE, 1996, 754: 11.
[30] WITTENBRINK C M, KILGARIFF E, PRABHU A. Fermi GF100 GPU architecture[J]. IEEE Micro, 2011, 31(2): 50-59.
[31] STEWART G W. After notes on numerical analysis[M]. SIAM: Philadelphia, 1996: 135-153.
[32] DEVLIN J, CHANG M W, LEE K, et al. BERT: pre-training of deep bidirectional transformers for language understanding[C]// Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Minneapolis: NAACL-HLTMN, 2019: 4171-4186.
[33] BROWN T B, MANN B, RYDER N, et al. Language models are few-shot learners[EB/OL]. (2020-07-22)[2021-12-16]. https://arxiv.org/pdf/2005.14165v4.pdf.

A method based on multi-segment interpolation fitting to accelerate nonlinear layers in deep neural networks

HUANG Yifan* ZHANG Xin** ZHI Tian** ZHANG Rui** ZHANG Xishan** ZHOU Xuehai*
(* School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026)
(** Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

Abstract

Aiming at the problems that the classic quantization methods can not be used for layers with nonlinear computation, a method based on multi-segment interpolation fitting for nonlinear layer acceleration is proposed. It uses interpolation tables to store parameters of interpolating function, looks up these tables to compute output of nonlinear layers. This method can efficiently accelerate nonlinear layers and its fitting error is controllable. It only needs basic hardware instruction support, and it can be applied on edge devices and server. The experimental results show that interpolated nonlinear layers can speed up the computation by 1.44 times on average. Those interpolating nonlinear layers can be applied conveniently on tasks such as image classification, natural language processing, and sentence translation models, even though different models requires various fitting precision, inference and training process can achieve promised accuracy.

Key words: deep neural network(DNN), quantization, nonlinear layer acceleration, multi-segment interpolation fitting