arXiv:2304.04125v1 [cs.LG] 8 Apr 2023

面向近似硬件执行的神经网络训练

Tianmu Li 加利福尼亚大学洛杉矶分校，美国洛杉矶

Shurui Li 加利福尼亚大学洛杉矶分校，美国洛杉矶

Puneet Gupta 加利福尼亚大学洛杉矶分校，美国洛杉矶

摘要

近似计算方法在深度学习领域展现出巨大潜力。由于硬件成本较低，这些方法尤其适用于受功耗预算限制的电池供电设备上的推理任务。然而，由于缺乏针对训练方法的研究，近似计算尚未充分发挥其潜力。在这项工作中，我们探讨了适用于近似硬件的训练方法。我们展示了训练方法需要针对近似硬件进行专门设计，并提出了可将训练过程加速高达 18 倍的方法。

关键词

机器学习,神经网络, 近似计算, 随机计算, 模拟计算, 存内计算, 光子加速器

ACM 引用格式：

TianmuLi、ShuruiLi 和 Puneet Gupta。2022。面向近似硬件执行的神经网络训练。载于*tinyML Research Symposium* 论文集（*tinyML ResearchSymposium'22*）。ACM，纽约州纽约市，美国，7 页。

1 引言

近年来，使用神经网络的机器学习日益普及。神经网络的规模和复杂性不断增加，以提高准确率。由于模型复杂性不断提升，人们提出了包括近似算术、随机计算和模拟计算在内的近似计算方法，以牺牲部分计算准确率来换取更好的性能。然而，近似计算方法引入的误差会影响推理准确率，使得为浮点或定点计算训练的模型难以使用。此前很少有研究关注近似硬件所需的训练技术。因此，近似计算一直局限于简单模型和数据集，或者不得不牺牲性能来维持准确率。

在本工作中，我们提出了使用近似计算改进模型训练的方法。我们的贡献包括：

- 使用激活函数在反向传播中近似计算误差。
- 使用误差注入，最多可将每次训练迭代的耗时缩短 36.6 倍。
- 将误差注入与精确建模相结合，以保持模型精度。
- 将端到端训练时间最多缩短18倍，使得原本无法在单个消费级 GPU 上训练的模型成为可能。

允许出于个人或课堂用途免费制作本作品的部分或全部内容的数字副本或纸质副本，前提是不得出于营利或商业目的制作或分发副本，且副本首页必须附有本声明和完整引文。本作品中第三方组件的版权必须得到尊重。其他用途请联系所有者/作者。*tinyML Research Symposium'22，2022*年*3*月，加利福尼亚州圣何塞© 2022 版权归所有者/作者所有。

2 动机

在本节中，我们将讨论带误差计算对神经网络的益处，并说明开展专用训练的必要性。

2.1 神经网络的不精确计算

尽管有多种方法可以通过不精确计算提升计算性能，但我们将研究重点放在以下三种类型上：

（1）随机计算（SC）。随机计算使用随机化比特流表示数字，并支持通过单个门实现加法和乘法。随机计算较小的面积开销能够实现更高的计算密度和更高的内存效率，与传统定点加速器相比，能效最高可提升38.7倍 [17]。

（2）近似算术。近似算术专注于降低高成本计算的开销，通常采用近似乘法的形式，同时不降低输入位宽和计算位宽。通过从精确乘法器中移除组件，可以减少面积和功耗，但代价是误差增加[9]。通过引入不同程度的不精确性，近似计算能够实现多种精度和性能水平；在极端情况下，可将乘法器的功耗和面积降低4倍以上 [14]。

（3）模拟计算。人们对模拟神经网络加速器的兴趣日益浓厚，其中包括存内处理（PIM）[1,11] 和片上光子学 [12, 18]。模拟加速器的主要优势在于，与数字加速器相比，它们计算点积所需的功耗显著更低，从而使整体计算更加节能；与现代数字加速器相比，其能效甚至可以提高10倍以上 [18]。模拟加速器通常由多个阵列组成，每个阵列计算一个点积。模拟-数字转换器（ADC）用于将点积结果从模拟域转换回数字域。

上述大多数方法都可以基于权重剪枝和权重/激活量化等传统模型压缩方法，进一步提升执行效率。

2.2 专项训练的必要性

神经网络通常假设训练过程中不存在误差，而一旦计算中引入误差，就无法可靠地运行。随机计算本质上具有随机性，而使用逐位加法器进行的随机计算累加无法实现精确累加 [17]。

近似乘法在设计上会在乘法运算中引入误差，对于某些特定输入组合，误差可能非常大 [9]。受阵列规模限制，许多情况下，模拟加速器无法在单个阵列中完成整个卷积运算，而是需要计算部分和 [7]。随后，部分和会由模数转换器量化，再在数字域中进一步累加，因为模数转换器的精度和范围有限。然而，在常规训练中，部分和不会被量化；如果 ADC 位宽较低，就可能引入显著误差，导致学得的权重不适用于实际硬件。训练过程中必须考虑所有这些影响。如表4所示，与在训练期间对计算进行恰当建模相比，直接将针对定点计算预训练的模型用于近似计算，会使准确率下降 8–57 个百分点。

如表1所示，与精确乘法和加法相比，在前向传播中准确地模拟近似计算的成本很高。在实际应用中，运行时间差异甚至可能更大，因为浮点 conv2d/linear层可以使用cuDNN等库中的优化内核 [4],，而用于模拟近似计算方法的函数则需要额外编写代码（即实现自定义内核）。因此，**有必要采用专门的方法来减少近似硬件的训练时间**。

表1：相对乘法和加法成本。以FP32 乘法和加法作为基准。C++ 中的运算次数用作成本值。

方法	乘法	加法
浮点	0.5（融合）	0.5(融合)
随机计算（32 位）	64(展开) 2(已打包)	64(展开) 2（已打包）
近似乘法	86	1
模拟计算	1	1（信道内） 9(通道之间)

2.3 近似硬件训练方面的前人工作。

由于在训练过程中对计算误差进行建模存在困难，先前的研究未能充分发挥不精确计算方法的优势。对于模拟计算，已有少数研究探讨了存内处理中的部分和（或模数转换器）量化 [7, 22] 以及光子加速器中的量化 [8, 15]。然而，这些研究的主要目标是利用其提出的量化方法优化精度，而且通常聚焦于更简单的数据集。

s

以及模型，从而使运行时间相对可控。这些工作并未解决运行时间问题，而这正是我们工作的主要关注点。

以往的研究也尝试在训练期间准确地模拟计算误差，但高昂的仿真成本阻碍了更复杂模型的训练 [13]（对于随机计算），或将这种方法限制在适合映射到商用训练硬件（通常为 GPU）的计算上 [5]（对于近似乘法）。其他研究则试图降低近似误差，从而使针对定点计算训练的模型能够直接用于推理。对于随机计算，以定点加法替代随机计算加法，以降低误差 [19]。对于近似乘法，仅在对模型准确率影响不大的位置使用近似计算 [20, 21]。由于近似计算依赖误差来提升性能，降低误差就要以牺牲性能收益为代价。

3 方法论

我们的训练改进包括三个组成部分：用于非线性计算的激活近似、训练期间结合微调的误差注入，以及梯度检查点。本节将详细讨论这三个组成部分。由于较大数据集存在运行时间限制，本节大部分内容将使用较小的 CIFAR-10 数据集。该设置限制了我们方法的性能收益，也使运行时间结果由于不同运行之间存在较大差异而变得不太可靠，但我们将在第4节中展示完整的性能收益。我们重点关注第2节中提到的三类近似计算方法，以展示我们方法的优势：

• 随机计算。我们使用线性反馈移位寄存器生成比特流，使用与 [17] 中类似的设置，采用与门进行乘法运算，采用或门进行加法运算。我们使用 32 位分离单极性比特流（总计 64 位）。
• 近似乘法器。我们使用 EvoApproxLib [14] 中的近似乘法器。具体而言，我们采用 mul7u_09Y 配置。它是一个 7 位无符号乘法器，属于平均相对误差的帕累托最优集合，适用于神经网络；结合符号位后，其输入精度为 8 位。与精确的 7 位乘法器相比，该近似乘法器的误差和功耗降低了 4 倍。• 模拟计算。为了提出更通用的论点，我们设置模拟加速器的有效阵列大小，使每个卷积通道的部分和都由模数转换器进行量化。对于 Resnet-tiny（为 TinyML 应用而缩小的 Resnet-18，见 [2]），该大小为 9；对于 Resnet-18 [6],，以及 TinyConv（一个用于 [10] 的四层 CNN ），该大小为 25。由于模拟加速器通常只支持正输入和正权重，因此我们在训练设置中使用分离单极性累加来处理负权重，这会导致 $2\times$ 计算。我们假设本文所有评估均使用 4 位模数转换器。我们选择相对较低的 ADC位宽，因为模数转换器通常是模拟加速器中的功耗瓶颈；因此，只要准确率影响不显著，低位宽模数转换器总是更受青睐。所有情况下，输入和权重的位宽均设置为 8 位，以集中考察部分和量化的影响。

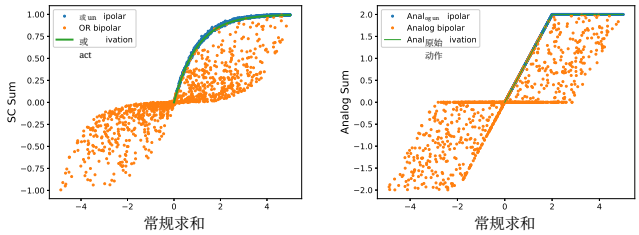
尽管还有其他近似硬件实现方式，但由于上述三种方法涵盖了多种近似计算形式，我们将研究范围限定在这三种方法上。我们的方法只需进行极少修改，即可应用于其他近似硬件配置。

3.1 ApproximationProxy 激活

不精确计算的一个问题是乘累加运算中引入的非线性，这与ReLU等非线性激活函数是分开的。我们以随机计算和模拟计算为例。近似乘法器不存在非线性问题，因为误差仅在乘法过程中引入。对于随机计算，当两个输入值 a 和 b 不- 相关时，OR 加法器平均执行 $a+b-ab$。对于模拟计算

表 2：使用激活函数带来的准确率提升。

设置	TinyConv	Resnet-tiny
随机计算		
无激活函数	41.64%	10.00%
使用激活函数	72.18%	79.76%
模拟计算（4 位）		
无激活函数	74.85%	69.21%
使用激活函数	79.20%	77.23%



(a) (b)

图 1：单极和双极（a）随机计算与（b）模拟计算的激活建模行为。双极版本通过对两个单极（正、负）输入进行相减来实现完整取值范围。对于模拟计算，本例将模数转换器饱和建模为在 2 处截断，且未考虑其他因素（例如累加规模）。

计算时，部分和与输出结果需要进行截断和量化。在反向传播中，精确建模计算中的不完美之处可能代价高昂。例如，先前提到的 OR 加法器在反向传播期间需要跟踪加法器中几乎所有的输入（$\frac{\partial}{\partial a_i}\mathrm{OR}(a_j) = \hat{1}_{j\neq i}(1-a_j)$），而精确加法在反向传播中则简单得多（偏导数=1）。除了建模成本高之外，如果不加以考虑，非线性还可能阻碍收敛。使用激活函数作为反向传播期间的代理，最初由[17] 提出，用于模拟随机计算 OR 累加的效果。与常规激活函数不同，新增的激活代理仅在训练期间的反向传播中使用，在推理期间不会使用。如果正确实现激活函数，其开销可以忽略不计。如表 2 所示，使用激活函数对非线性进行建模是必要的。TinyConv 是 [10], 中使用的四层 CNN，Resnet-tiny 则是为 TinyML 应用缩小后的 Resnet-18，见 [2]。模型在前向传播中对随机计算和模拟计算进行精确建模后训练。对于模拟计算，未使用激活函数进行训练时，准确率明显较低。对于随机计算，若不使用激活函数，训练则完全无法收敛。

使用激活函数要求计算具有（大致）结合律。以先前使用的 OR 累加和模拟计算为例。由于两者都处理单极性（仅正值）输入，计算被拆分为正、负两部分。每部分内部的累加具有结合律，但两部分相减则不具有结合律。图 1 展示了这一现象。虽然激活函数可以近似单极性 OR 累加，但在考虑减法时，无法用单个激活来模拟整个累加过程。因此，在反向传播中，累加需要拆分为正、负两部分，这样每部分都可以通过使用激活函数进行精确累加来建模。在模拟计算设置中也能看到类似效果。由于正、负部分各自会饱和，单个激活函数同样不足以应对。表 3 列出了随机计算和模拟计算中使用的激活函数。

表 3：用于随机计算和模拟计算的激活函数。x 是激活前某层的输出。x_{pos} 是正权重的输出，x_{neg} 是负权重的输出。由于 ReLU 激活，假设输入为非负值。

方法	激活函数
随机计算	随机计算_激活(x) =$(1-e^{-x_{pos}})-(1-e^{-x_{neg}})$
模拟计算	模拟_激活(x) =HardTanh(x_{pos}) –HardTanh(x_{neg})

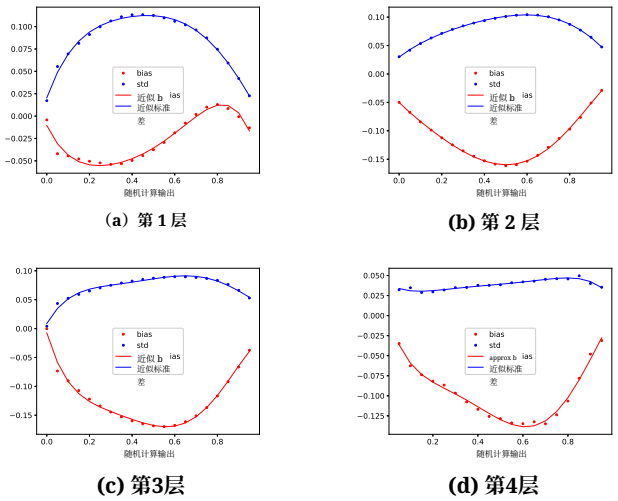
3.2 误差注入

尽管解决了反向传播的问题，激活函数方法仍无法完全替代前向传播。训练模型以进行非浮点计算，通常需要在前向传播中准确地对计算过程建模，无论是低精度定点计算（包括二值化等极端精度情况 [16]），还是近似计算 [5]。对于定点计算，在前向传播中对计算过程建模的成本相对较低。只需使用逐元素伪量化算子，再接一个常规卷积/线性算子即可。近似计算方法则并非如此。随机计算需要在硬件中模拟随机流生成和按位乘法。第 2 节所述的近似乘法器由数百行位操作代码组成。模拟计算则需要在计算过程中模拟有限的硬件阵列尺寸和模数转换器精度。表 1 展示了模拟近似计算的高昂成本。在所有情况下，模拟计算都代价高昂，需要额外编写代码，即使拥有充足的编程资源，也无法彻底解决这一问题。

表 4：对近似计算进行建模的准确率影响。“采用模型”会在前向传播中准确地模拟近似计算方法。

	TinyConv		Resnet-tiny	
方法	仅推理	采用模型	仅推理	使用模型
随机计算	14.87%	72.18%	48.57%	79.76%
近似乘法	71.88%	83.35%	76.95%	85.25%
模拟计算（4b）	58.43%	78.94%	47.15%	77.23%

虽然在前向传播中对近似计算进行精确建模的成本很高，但如表4所示，跳过建模会降低准确率。尽管第3.1节中定义的激活方法足以用于反向传播，但不足以用于前向传播。为克服这一局限性，我们提出将精确建模替换为两种与常规训练相结合的误差注入方法。



（a）第 1 层 (b) 第 2 层

(c) 第3层 (d) 第4层

图2：流计算与常规计算输出之间的差异+激活。

类型*1*。第一类误差注入将激活函数与精确建模之间的差异建模为某层输出值的函数，我们将其用于随机计算和近似乘法。图 2 展示了使用随机计算时 TinyConv四层的误差平均值和变化情况。与简单激活函数 $y = 1 - e^{-x}$ 所预期的值相比，实际层输出存在显著差异。图中的方差（表示为"标准差"）意味着仅使用激活函数无法完全捕获计算细节。鉴于激活函数在设计上只是一种近似，它无法捕获不精确计算的所有特征。另一方面，非零平均误差意味着激活函数在平均意义上无法准确表示目标计算。激活函数是在特定假设下推导得到的。对于OR累积而言，假设是累积规模 n较大，且所有输入值都较小并且数值相近。这些假设在真实模型中并不总是成立，而各层平均误差之间的差异意味着不可能为整个模型使用单一激活函数。对于深度模型，激活函数的误差会不断累积，导致模型在训练后无法使用。为解决这一问题，我们提出在训练期间向激活函数中添加校正项。

以图 2 所示的误差曲线为例。图中绘制了误差的均值和方差随激活函数输出的变化，两者看起来都是平滑函数。根据这一观察，我们在原始激活函数的基础上，将均值建模为激活后输出值的函数，并将方差建模为一个正态分布随机误差，其方差取决于激活后的值。两条曲线都拟合为多项式函数，并且每一层使用不同的函数，每个训练周期校准 5 次。虽然还可以进一步降低校准频率，但这一频率已能充分摊薄校准成本。

类型*2*。第二种误差注入用于训练模拟加速器。该方法与前文提到的用于随机计算和近似乘法的误差注入方法类似，但存在细微差异。在这种情况下，对输出激活的整体量化误差（各个部分和量化误差之和）进行建模并注入。我们首先对一个批次执行精确的前向传播，在累加之前对每个通道的部分和进行量化。将精确结果与由普通 Conv2d函数分别生成的结果（不对部分和进行量化）相减，以获得实际的整体量化误差。然后计算量化误差的均值和方差，并将其存储用于每个卷积层。我们不按照每个激活或每个滤波器的粒度获取统计数据，而是仅针对整个层计算一个均值和方差。选择这种方式有两个原因：1）经验结果表明，使用单个均值和方差的准确性优于以其他粒度计算均值和方差；2）节省内存，因为每层只需存储两个值。我们每10个批次校准一次误差统计数据，在准确性和运行时间之间取得了良好的平衡。对于大多数不需要校准的批次，我们在前向传播中仅计算普通Conv2d（不对部分和进行量化）。然后，根据均值和方差设置为上一次校准批次所得数值的正态分布，生成随机误差，以模拟量化误差。模拟的量化误差会直接加到Conv2d 的输出上。这样，对于非校准批次，运行时间应与普通卷积几乎相同，因为与卷积运算相比，生成量化误差所需的时间显著更少。

如表5所示，与完全不进行建模相比，使用误差注入替代精确建模可以提高准确率。然而，仅使用误差注入进行训练仍不足以完全弥合差距，尤其是对于ADC位宽较低的模拟计算。稍后我们将展示，通过微调步骤可以消除准确率差距。如表7所示，与训练期间使用精确模型相比，使用误差注入可将运行时间最多提升6.7倍。在大多数情况下，误差注入的运行时间与常规训练（"Without Model"列）相当。模拟计算中误差注入运行时间略长，是因为其校准频率更高。在相对更大的网络和数据集上（ImageNet上的Resnet-18），误差注入对模拟计算带来的运行时间提升可能更大（见第4节）。

3.3 微调

虽然第3.2节提到的误差注入无法达到使用精确建模进行训练时的准确率，但它可以减少达到相同准确率所需的精确建模微调量。误差注入的目的是模拟训练期间的误差，从而提高模型对误差的鲁棒性。然而，由于注入的误差是随机生成的，而实际误差取决于输入，因此误差注入无法达到精确建模所能达到的准确率。尽管如此，误差注入能增强模型的鲁棒性，因此只需少量精确建模，模型权重就能快速收敛到最优

表 5：错误注入训练对准确率的影响。

	TinyConv				Resnet-tiny			
方法	仅推理	使用模型	误差注入	微调	仅推理	使用模型	误差注入	微调
随机计算	14.87%	72.18%	64.01%	73.09%	48.57%	79.76%	76.71%	81.29%
近似乘法	71.88%	83.35%	79.06%	83.05%	76.95%	85.25%	81.06%	84.85%
模拟计算（4b）	58.43%	78.94%	62.02%	77.95%	47.15%	77.23%	71.10%	76.05%

点。图3a比较了在 CIFAR-10 上训练随机计算时，误差注入与精确建模相结合的收敛情况。对于随机计算，将误差注入与5轮微调相结合，就足以达到整个训练过程中始终使用精确模型时的相同准确率。相比之下，不进行误差注入时收敛效果较差，即使经过20轮微调，模型也无法收敛到相同的准确率。

图 3b 比较了同一模型在使用近似乘法器时的收敛情况。由于使用近似乘法器时准确率差距较小，因此即使不进行误差注入，经过 5 轮微调后也能正常收敛。不过，误差注入可将轮次进一步减少至 2 轮。

对于采用 4 位部分和量化的模拟计算，使用准确模型进行训练时，基于预训练的 INT8 权重，在 CIFAR-10 数据集上通常经过 10-15 轮次即可收敛。没有必要使用准确模型或误差注入从头开始训练。模拟计算训练的经验结果表明，只使用四分之一轮次即可达到与完整轮次微调相同的准确率。因此，微调仅使用最后四分之一轮次，带来的运行时开销极小。与随机计算或近似乘法不同，微调无法完全弥补低 ADC 位宽造成的准确率差距，但与准确模型相比，准确率下降通常较小（约 1%）。图 3c 比较了模拟计算中的收敛情况：采用校准和微调时，误差注入可以达到与准确模型相近的准确率。不使用误差注入时，即使保持与误差注入方案相同的校准和微调设置，训练也无法收敛到误差注入方案所达到的准确率。

总体而言，在误差注入的基础上进行微调，可以弥补单独使用误差注入造成的准确率损失。如表5所示，微调后的准确率（“微调”列）最多比完全使用精确模型进行训练低1个百分点。

3.4 梯度检查点

第 Sec 3.2 节中描述的误差注入会在训练期间为计算图增加额外步骤，从而增加内存需求，并限制较大模型的批次大小。为降低内存消耗，我们采用了 [3] 中的梯度检查点设置。由于新增函数都是逐点函数，计算强度较低（< 20 OPS/内存访问），因此即使不受内存限制，对所有新增计算进行检查点处理对运行时间的影响也很小。表6比较了在 ImageNet 数据集上运行 Resnet-18 时，启用和不启用检查点处理的运行时间与内存消耗。检查点处理使训练能够使用更大的批次大小，从而提高 GPU 效率；在本例中，每轮所需运行时间减少了 22%。

4 结果

本节将展示我们的方法在更复杂工作负载上的优势。我们使用 ImageNet 数据集上的 Resnet-18 训练来说明这些优势。模型在单张 RTX 3090 上采用混合精度进行训练。我们以 PyTorch 1.12 作为基线框架，并将额外算子实现为 CUDA C ++ 扩展，其中包括对随机计算的精确建模、近似乘法和模拟计算。尽管我们尽力优化用于近似计算建模的内核，但无法保证这些实现已达到最优。为缩短训练时间，模型从 PyTorch 提供的检查点开始训练，该检查点使用 fp32。因此，三种近似计算设置所需的收敛轮次各不相同。表8列出了详细的训练设置。

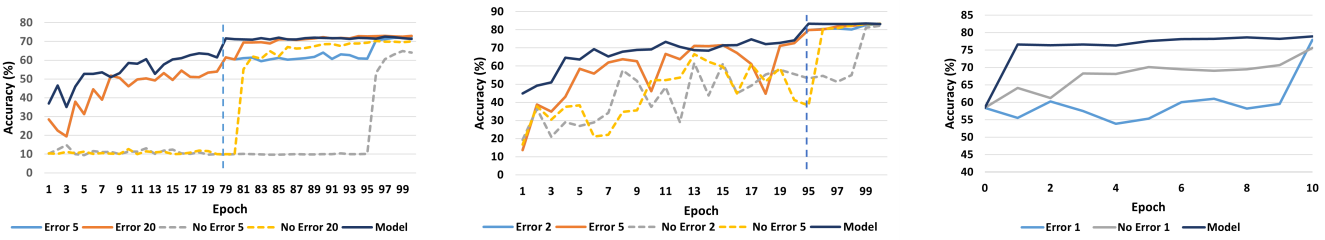
表9展示了三种设置下达到的准确率。虽然目前没有可供比较的既有准确率结果，但这些结果呈现出与第3节讨论的较小模型相同的趋势。近似乘法和模拟计算的训练耗时过长，无法直接用于训练；我们将在后文展示这一点。

表6：使用和不使用梯度检查点时，随机计算的训练运行时间和内存需求。我们采用可实现的最大批次大小，且该大小为2的幂。

设置	内存（MB）	批量大小	运行时间（秒/轮）
使用检查点	19840	256	1326
不使用检查点	12766	128	1692

表7：错误注入训练对运行时间的影响。表中显示的是每个周期所需的时间（秒）。运行时间是在 RTX 3090 上以 TF32 精度和批量大小=256测得的。由于需要拆分第3.1节所述的正负计算，随机计算和近似乘法的速度较慢。微调的运行时间与使用精确模型时相同（“使用模型”列）。

方法	无模型	使用模型误差注入	
	TinyConv		
随机计算	3.86	9.50	3.90
近似乘法	3.86	28.3	4.20
模拟计算（4b）	2.13	3.91	2.73
	Resnet-tiny		
随机计算	8.51	13.3	8.65
近似乘法	8.13	38.4	9.11
模拟计算（4b）	4.88	8.51	6.53



(a) (b) (c)

图3：使用（a）随机计算、（b）近似乘法和（c）模拟计算时，注入误差与不注入误差的 TinyConv 的收敛行为。随机计算和近似乘法从头开始训练100轮次，并展示前20轮次和最后几轮次的结果。模拟计算使用 INT-8 预训练权重训练10轮次，因为无需从头开始训练。"Error X"表示在注入误差的情况下进行训练，并使用精确模型进行 X 轮次的微调。"No Error X"表示在不注入误差的情况下进行训练，并进行 X 轮次的微调。"Model"表示在整个训练过程中都采用精确建模。

表10展示了所提方法结合使用时的性能优势。由于以下因素，此处展示的改进低估了所提方法带来的收益：

- "无改进"列假设反向传播过程中使用了第3.1节中提到的激活函数。如果不使用激活函数，则还需要对反向传播进行精确建模。
- 每轮训练后都会对模型进行验证。无论是否进行改进，验证方式都相同，因此如果降低验证频率，性能差距将会更加显著。

尽管存在这些局限，我们的方法仍将端到端训练时间缩短了 2.4 倍至 18.2 倍。对于随机计算和近似乘法，其性能收益大致与表7所示的运行时间差异相符，因为即使对于较小的模型，运行时间瓶颈也在于精确建模。近似乘法尤其受益于误差注入：与精确建模相比，迭代时间缩短了 36.6 倍。对于模拟计算，ImageNet 上的运行时间差异大于 CIFAR-10 上的差异，因为在 CIFAR-10 上，TinyConv 和 Resnet-tiny 都无法充分利用 GPU。

5 结论

在这项工作中，我们提出了几种方法，以提升在近似硬件上进行推理的训练性能。通过使用激活建模、结合微调的误差注入以及梯度检查点技术，我们在各种近似硬件上实现了收敛。我们的方法使训练大型模型成为可能。

表8：训练所用轮次。准确率较高的方法所需的训练轮次较少。

方法	误差注入	微调
随机计算	30	5
近似乘法	34	1
模拟计算（4b）	14	1

表 9：Resnet-18 在 ImageNet 上的 Top-1 准确率。

方法	未改进	改进后
随机计算	52.45%	54.29%
近似乘法	N/A	67.81%
模拟计算 (4b)	N/A	63.40%

表10：端到端运行时间改进。时间以收敛所需的小时数衡量。"无改进"结果是对没有这些改进便无法训练的模型进行的估算。

方法	无改进	改进后
随机计算	42.8	17.6
近似乘法	592	32.4
模拟计算（4b）	55.3	12.1

在单个消费级 GPU 上，对 ImageNet 数据集运行 Resnet-18 等模型。

参考文献

[1] Shaahin Angizi、Zhezhi He、Adnan Siraj Rakin和 DeliangFan。2018年。CMP-PIM：一种基于比较器的高能效存内处理神经网络加速器。在第*55*届年度设计自动化会议论文集（加利福尼亚州旧金山）（*DAC '18*）。Associationfor Computing Machinery，美国纽约州纽约，文章 105，共 6 页。https://doi.org/10.1145/3195970.3196009[2] Colby Banbury、Vijay Janapa Reddi、Peter Torelli、Jeremy Holleman、Nat Jeffries、Csaba Kiraly、Pietro Montino、David Kanter、Sebastian Ahmed、Danilo Pau 等。2021 年。Mlperftiny基准测试。*arXiv* 预印本 *arXiv:2106.07597*（2021）。[3] Tianqi Chen、Bing Xu、Chiyuan Zhang 和 CarlosGuestrin。2016 年。使用亚线性内存成本训练深度网络。*CoRR* abs/1604.06174（2016）。arXiv:1604.06174http://arxiv.org/abs/1604.06174[4] Sharan Chetlur、Cliff Woolley、Philippe Vandermersch、Jonathan Cohen、John Tran、Bryan Catanzaro 和 EvanShelhamer。2014 年。cuDNN：深度学习的高效原语。*CoRR* abs/1410.0759（2014）。arXiv:1410.0759 http://arxiv.org/abs/ 1410 0759。Jing Gong、Hassaan Saadat、Hasindu Gamaarachchi、Haris Javaid、Xi- aobo Sharon Hu 和 SriParameswaran。2022年。ApproxTrain：用于 DNN 训练和推理的近似乘法器快速模拟。https://doi.org/ 10 48550 2209 04161。/ARXIV。[5] Kaiming He、Xiangyu Zhang、ShaoqingRen 和 Jian Sun。2016 年。用于图像识别的深度残差学习。在 *IEEE*计算机视觉与模式识别会议论文集。770–778。[6]

[7] Sitao Huang、Aayush Ankit、Plinio Silveira、Rodrigo Antunes、Sai Rahul Chala- malasetti、Izzat El Hajj、Dong Eun Kim、Glaucimar Aguiar、Pedro Bruel、Sergey Serebryakov、Cong Xu、Can Li、Paolo Faraboschi、John Paul Strachan、Deming Chen、Kaushik Roy、Wen-mei Hwu 和 Dejan Milojicic。2021。用于基于 ReRAM 的 DNN 推理加速器的混合精度量化。载于*2021*年第*26*届亚洲及南太平洋设计自动化会议（*ASP-DAC*）。372–377。[8] M. Kirtas、A. Oikonomou、N. Passalis、G. Mourgias-Alexandris、M. Moralis- Pegios、N. Pleros 和 A. Tefas。2022。面向低精度光子神经网络的量化感知训练。神经网络 155（2022），561–573。https://doi.org/ 10 1016 2022 09 015 。/j.neunet. . . Parag Kulkarni、Puneet [9] Gupta 和 Milos Ercegovac。2011。通过设计不足的乘法器架构以精度换取功耗。载于*2011*年第*24*届国际 *VLSI* 设计会议。 346 351 10 1109 2011 51 –。 [10] 2018 https://doi.org/。/VLSID. . Liangzhen Lai、Naveen Suda 和 Vikas Chandra。。 CMSIS-NN：适用于 Arm Cortex-M CPU 的高效神经网络内核。（2018），1–10。 http://arxiv.org/abs/1801.06601 arXiv：1801.06601。[11] Sukhan Lee、Shin-haeng Kang、Jaehoon Lee、Hyeonsu Kim、Eojin Lee、Seungwoo Seo、Hosang Yoon、Seungwon Lee、Kyounghwan Lim、Hyunsung Shin 等。2021。基于商用 DRAM 技术的存内处理硬件架构与软件栈：工业产品。载于*2021*年*ACM/IEEE*第*48*届计算机体系结构年度国际研讨会（*ISCA*）。IEEE，43–56。[12] Shurui Li、Hangbo Yang、Chee Wei Wong、Volker J Sorger 和 Puneet Gupta。2022。PhotoFourier：基于光子联合变换相关器的神经网络加速器。*arXiv* 预印本 *arXiv:2211.05276*（2022）。[13] Tianmu Li、Wojciech Romaszkan、Sudhakar Pamarti 和 Puneet Gupta。2021。GEO：面向神经网络的生成与执行优化随机计算加速器。载于*2021*年欧洲设计、自动化与测试会议暨展览（*DATE*）。1–6。[14] Vojtech Mrazek、Radek Hrbacek、Zdenek Vasicek 和 Lukas Sekanina。2017。EvoApprox8b：用于电路设计和近似方法基准测试的近似加法器与乘法器库。载于*2017*年欧洲设计、自动化与测试会议暨展览（*DATE*）。258–261。 https://doi.org/10.23919/DATE.2017.7926993

[15] A.Oikonomou,M.Kirtas,N.Passalis,G.Mourgias-Alexandris,M.Moralis-Pegios, N.Pleros,and A. Tefas. 2022. 面向光子神经网络的稳健量化感知训练方法。在神经网络的工程应用，Lazaros Iliadis、Chrisina Jayne、Anastasios Tefas和EliasPi menidis（编）。施普普林格国际出版公司，查姆，427–438。[16] Mohammad Rastegari, Vicente Ordonez,Joseph Redmon,andAliFarhadi. 2016.XNOR-Net：使用二值卷积神经网络进行ImageNet分类。*CoRR* abs/1603.05279 (2016)。 arXiv:1603.05279http://arxiv.org/abs/1603.05279[17] Wojciech Romaszkan, Tianmu Li, Tristan Melton, SudhakarPamarti, andPuneet Gupta.2020. ACOUSTIC：通过或单极跳跃随机计算加速卷积神经网络。在*2020*年欧洲设计、自动化与测试会议暨展览（*DATE*）。768–773。https://doi.org/ 10 23919 48585 2020 9116289 。/DATE。Kyle Shiflett,Avinash [18] Karanth,Razvan Bunescu, andAhmedLouri.2021. Albireo：通过硅光子学实现卷积神经网络的节能加速。在*2021*年第*48*届*ACM/IEEE*计算机体系结构年度国际研讨会（*ISCA*）。IEEE，860–873。[19] HyeonukSimand JongeunLee. 2017. 一种应用于深度卷积神经网络的新型随机计算乘法器。在*2017*年第*54*届*ACM/EDAC/IEEE*设计自动化会议（*DAC*）。1–6。https://doi.org/ 10 1145 3061639 3062290 。/ 。 [20] Swagath Venkataramani, Ashish Ranjan, KaushikRoy, and AnandRaghunathan. 2014. AxNN：使用近似计算的节能神经形态系统。在*2014*年*IEEE/ACM*低功耗电子与设计国际研讨会（*ISLPED*）。 27 32 10 1145 2627369 2627613 – 。https://doi.org/ 。/。 [21] 2015 Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, andQiang Xu. 。ApproxANN：用于人工神经网络的近似计算框架。在*2015*年欧洲设计、自动化与测试会议暨展览（*DATE*）。701–706。[22] Wenqiang Zhang, Xiaochen Peng, Huaqiang Wu, Bin Gao,Hu He, YouhuiZhang, ShimengYu, andHeQian. 2019. RRAM神经处理单元的设计指南：器件-电路-算法分析。在*2019*年第*56*届*ACM/IEEE*设计自动化会议（*DAC*）。1–6。