在可微神经网络与加速器协同探索中实现硬约束

Deokki Hong[1], Kanghyun Choi[1], Hye Yoon Lee[1],Joonsang Yu[2],Noseong Park[1],Youngsok Kim[1],和 Jinho Lee[1*1]延世大学计算学院， [2]NAVER CLOVA AI Lab CLOVA ImageVision[1] {dk.hong, kanghyun.choi, hylee817, noseong, youngsok, leejinho}@yonsei.ac.kr [2] joonsang.yu@navercorp.com

arXiv:2301.09312v1 [cs.LG] 23 Jan 2023

摘要

协同探索最优神经网络架构及其硬件加速器是一种日益受到关注的方法，能够解决计算成本问题，尤其适用于低规格系统。通常采用可微分神经架构搜索的思路来处理庞大的协同探索空间。然而，尽管可微协同探索具有出色的搜索效率，却面临一项关键挑战：无法系统地满足帧率等硬约束。为解决可微协同探索中的硬约束问题，我们提出 HDX，它能够在不损害全局设计目标的前提下搜索满足硬约束的解。通过沿给定硬约束的方向调整梯度，可以获得满足约束的高质量解。

1 引言

大多数深度神经网络 （DNN）研究的主要关注点是应用性能（即准确率）。然而，这也导致网络规模迅速增长，使其执行需要大量计算资源。近年来，出现了许多缓解资源问题的方案，大多属于以下两类之一：网络侧优化 和硬件侧优化。网络侧优化是指改进神经网络的架构，在保持准确率相近的同时减少计算量 [28, 31]。硬件侧优化通常涉及使用经过优化的硬件（也称为加速器）来提高 DNN 的执行效率 [5, 14]。遗憾的是，一侧的努力往往会削弱另一侧所带来的收益。例如，MobileNet [28] 中使用的逐通道可分离卷积操作，其主要优势源自其结构：运算仅使用单个通道。然而，谷歌的 TPU [14], 这一著名加速器主要利用通道级并行来提升速度。因此，MobileNet 在 TPU 上的执行时间表现不佳 [10]。

因此，协同探索硬件加速器与网络架构 [1, 6, 9, 11, 19, 22, 30] 是兼顾准确率和硬件指标（如延迟、能耗及硅芯片面积）的自然方向。为应对协同探索中庞大的搜索空间，可微协同探索 方法 [6, 9, 19]被视为很有前景的方法，因为与基于强化学习的方法相比，它们能够更快地探索搜索空间 [1, 11, 20, 22, 30]。

遗憾的是，可微协同探索有一个严重缺陷：无法处理许多现实场景中的关键硬约束。例如，一项重要约束是目标检测系统 [26] 必须满足摄像头的帧率要求（例如，每秒 30 帧）。此外，使用电量有限的电池运行的移动子系统通常有功耗预算。由于可微协同探索方法依赖单一损失函数，因此往往无法满足这些约束，只能盲目地反复进行超参数调优和重新探索。

为解决这一问题，我们提出了*HDX* （**硬**约束**可**微**探**索），实现神经网络架构与硬件加速器的硬约束可微协同探索。我们提出的核心概念是一种梯度操控方法，可确保解不会偏离满足约束的方向。除了全局损失的梯度外，我们还计算硬件约束的梯度，并在存在约束违反时用它来操控全局损失的梯度，从而满足以下条件：i）两者的点积为正（即指向相近的方向）；ii）该方向能够缓解约束违反。

据我们所知，这是首项在可微协同探索问题中考虑硬约束的工作。我们进行了大量评估，以证明 HDX 能够：1）即使在约束严格的情况下，也满足硬件约束；2）在不牺牲质量的前提下搜索解。

我们的贡献总结如下。

- 我们提出了 HDX，这是一种面向网络和加速器的硬约束可微协同探索方法，旨在无需反复试错即可找到有效解。
- 我们提出使用梯度操控，逐步将解推向满足约束的区域。
- 我们对 HDX 进行了大量评估，以展示其满足约束的能力和效率。

2 背景与相关工作 2.1 神经架构搜索

神经架构搜索（NAS）是指自动化神经网络设计过程的技术。为降低其高昂的成本（数千 GPU 小时），人们提出了可微神经架构搜索 [21]，作为一种高效的替代方案。它将这一问题转化为训练超网，从而将耗时缩短几个数量级。

无论如何，仅针对网络性能进行优化是不够的，因为这类方法没有考虑硬件效率。一些近期研究 [4] 通过添加损失项来考虑硬件成本，以解决这一问题；另一些研究则尝试使用简单的延迟模型 [2, 23]，缩小网络规模以满足延迟约束。然而，这些方法只考虑网络设计，无法用于协同探索，因为模型没有反映网络准确率与硬件结构之间的关系。

* 通讯作者。

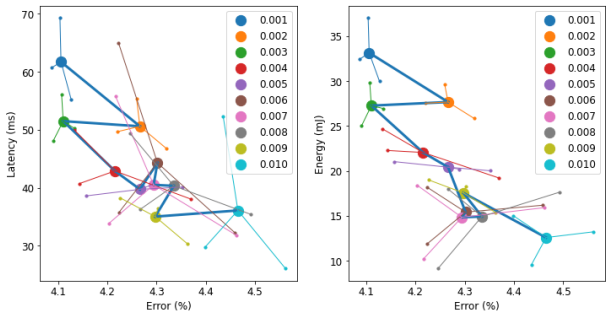


图1：动机实验。在每个图中，我们将 λ_{Cost} 0的取值从 0.001 扫描到 0.010。显然，轨迹并非严格线性地趋向 λ_{cost}，且存在较大波动。

2.2 DNN-加速器协同探索

早期的协同探索研究采用强化学习的变体或进化算法，以利用其简洁性 [1, 11, 13, 20, 22, 30]。每个候选网络都需要经过训练以供评估，同时还要分析加速器设计的硬件效率。这些指标会生成奖励，供智能体用来生成下一个候选解。然而，这些方法都继承了基于强化学习的神经架构搜索方法的相同问题：评估每个候选解都需要耗费大量资源进行训练。更糟的是，与仅搜索网络相比，协同探索所需的网络和硬件搜索空间更大。

鉴于此，人们采用可微方法进行协同探索 [19]。Auto-NBA [9] 使用可微加速器搜索引擎构建联合搜索流程，而 DANCE [6]则训练辅助神经网络以进行硬件搜索和成本评估。然而，以上方法都未能妥善解决硬约束问题。本文提出一种全面的方法，用于处理可微协同探索中的硬约束。

3 动机实验

在可微协同探索中处理硬约束，最直接、最简单的方法是调整硬件成本的相对权重。例如，以下是可微协同探索中使用的损失函数 [6, 9]。

$$\mathcal{L}oss = \mathcal{L}oss_{CE} + \lambda_{Cost} Cost_{HW}, \tag{1}$$

该方法旨在同时优化准确率和硬件成本，并通过 λ_{Cost} 平衡这两个项[1]。增大 λ_{Cost}，可以间接指示搜索过程更多地考虑硬件指标。然而，施加更大的惩罚并不会直接降低受约束指标的值。图1展示了将式（1）中的 λ_{Cost} 从 0.001 改为 0.010 对 CIFAR-10 数据集的延迟/能耗和分类错误率的影响。每种设置均进行了三次搜索，并用相同颜色绘制，较大的点表示平均值。尽管可以观察到一些取决于 λ_{Cost} 的趋势，但轨迹在方向和方差上的不一致性更为显著。

考虑这样一种场景：设计者希望设计一对神经网络与加速器架构，使其延迟低于某个约束（例如 33.3 毫秒），并使用传统的协同探索方法。设计者会尝试从某个初始 λ_{Cost} 开始搜索，并在多次搜索过程中不断调整其值。然而，λ_{Cost} 与延迟之间存在这种不一致，使得很难获得合适的解，更不用说多次搜索所耗费的巨大时间成本。尽管解决硬约束协同探索问题困难重重，设计有效的策略仍然十分必要。

[1]这与典型机器学习公式中的超参数不同，后者的两个项共同服务于单一目标。

4 硬约束协同探索

4.1 问题定义

硬约束可微协同探索的数学公式如下：

$$\begin{aligned} &\arg\min_{\alpha,\beta}(\mathcal{L}oss_{NAS}(w^*, net(\alpha)) + \lambda_{Cost} Cost_{HW}(eval(\alpha,\beta))), \\ &\text{s.t. } w^* = \arg\min_{w}(\mathcal{L}oss_{NAS}(w, net(\alpha))), \quad t \le T, \end{aligned} \tag{2}$$

其中，t 表示当前受约束指标的值，例如延迟或能耗；T为目标值（例如，延迟为 33.3 ms）。α 和 β 分别表示网络架构参数和硬件加速器配置。w 是神经架构搜索超网的权重，net（α）是当前选定的主导网络架构。$eval$（α，β）表示针对 α 和 β评估的硬件指标。协同探索的目标采用两个不同的评估指标来表示：神经架构损失（$Loss_{NAS}$）和由用户定义的硬件成本（$Cost_{HW}$）。

4.2 可微协同探索框架

尽管我们的主要贡献是实现了硬约束，但由于硬约束与协同探索密切相关，我们也对协同探索框架进行了说明。图 2 展示了所提方法的整体架构，该架构与现有方法 [6, 9] 类似。图 2 (a) 为网络搜索模块。该模块通过从超网中选择一条路径来搜索网络架构。随后，网络结构被送入评估器模块。

评估器网络 $eval()$ 是可微协同探索的关键，它使梯度能够流入超网，同时考虑硬件加速器之间的关系。它由两个子网络组成：硬件生成器 $gen()$ 和估计器 $est()$。硬件生成器以神经网络架构参数为输入，并据此输出最优硬件实现（β，见公式 2）。在协同探索过程中，硬件生成器会接受联合训练，因此生成器不依赖于特定的成本函数，并且能够适应约束。估计器网络以生成器的输出和网络为输入，输出硬件相关指标。它根据网络和加速器搜索空间进行预训练。预训练估计器时，使用 MAESTRO [18],、Timeloop [25], 和 Accelergy [29] 等传统（不可微）成本估算框架作为真实值。预训练后，探索过程中会冻结估计器，仅用于推断给定

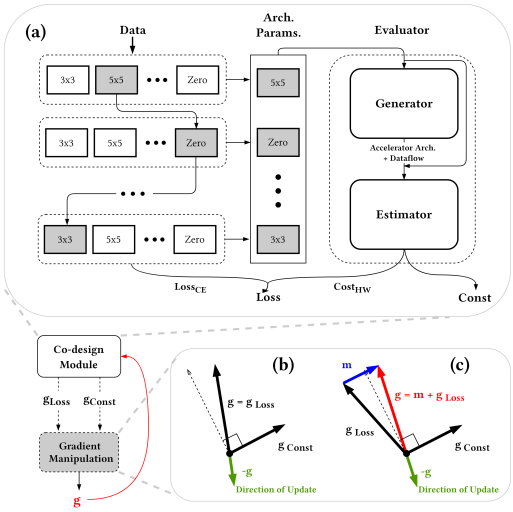


图 2：HDX 的整体结构。

网络架构的硬件成本。基于此，我们将公式 2 转换如下：

$$\arg\min(Loss_{NAS}(w^*, net(\alpha)) + \lambda_{Cost} Cost_{HW}(est(\alpha, gen(v^*, \alpha)))), \alpha$$

$$\text{s.t. } w^* = \arg\min_{w}(\mathcal{L}oss_{NAS}(w, net(\alpha))),$$
$$v^* = \arg\min_{v}(Cost_{HW}(est(\alpha, gen(v, \alpha)))), \quad (3)$$

其中，v 是硬件生成器的权重 .

4.3 通过梯度操控启用硬约束

除了可微协同探索方法外，我们还提出了梯度操控这一新颖理念，将其作为解决硬约束问题的有效方案。直接操控梯度是一种常用于实现多个目标的策略，例如持续学习 [27] 或微分方程 [15]。本文提出了一种将梯度操控应用于协同探索问题的方案，以满足硬约束。

图 2(b) 和 (c) 展示了我们梯度操控方法的高层抽象。其核心思想是人为生成一种作用力，将梯度推向与约束一致的方向。应用该方法计算新梯度 g 的条件定义如下：

$$g = \begin{cases} g_{\mathcal{L}oss} & \text{, if } t \le T \\ & \text{or } t > T \wedge g_{\mathcal{L}oss} \cdot g_{Const} \ge 0, \\ m_\alpha + g_{\mathcal{L}oss} & \text{, otherwise} \end{cases} \quad (4)$$

$$g_{Const} = \frac{\partial \max(t - T, 0)}{\partial \alpha}. \quad (5)$$

在上述公式中，$g_{\mathcal{L}oss}$ 是全局损失函数定义的原始梯度，其定义如下：

$$\mathcal{L}oss = \mathcal{L}oss_{NAS} + \lambda_{Cost} \cdot Cost_{HW}, \quad (6)$$

如式 3 所示，g_{Const} 是约束损失的梯度，我们将其定义为：C $onst = max(t - T, 0)$。注意，t 是 α 的函数，因此可以通过反向传播求得关于 α 的梯度。在理想情况下，$t \le T$，约束已经满足，因此我们无需采取任何操作。不幸的是，如果约束仍未满足，我们计算两个梯度的点积，以确定它们方向的一致性。如果 $g_{\mathcal{L}oss} \cdot g_{Const} \ge 0$（即两个梯度之间的夹角小于 90°），这意味着梯度下降更新有助于满足约束。因此，我们将其视为方向一致，并原样使用相同的 $g_{\mathcal{L}oss}$。图 2 (b) 描述了这种情况。然而，如果二者方向不一致，如图 2 (c) 所示（即 $g_{\mathcal{L}oss} \cdot g_{Const} < 0$），我们会强制梯度沿 m_α 改变方向，该方向由 $(m_\alpha + g_{\mathcal{L}oss}) \cdot g_{Const} \ge 0$ 得到，以确保梯度下降后目标成本降低。它可以重新表示为 $m_\alpha \quad g \quad onst + g \quad oss \quad g \quad onst = \delta \quad \delta \ge 0$ $\cdot C\mathcal{L} \cdot$ C，其中是一个较小的值，用于确保逐步朝满足约束的方向移动。

为了更新 α 和 w，我们求解相对于 α 的最优 m_α，其中 α 是网络架构的参数。为了尽量减小 m_α 对 $g_{\mathcal{L}oss}$ 的影响，我们采用基于伪逆的解，该解已知能够使 $||m_\alpha||_2^2$ 的大小最小化，如下所示：

$$m_\alpha^* = \frac{-(g_{\mathcal{L}oss} \cdot g_{Const}) + \delta}{||g_{Const}||_2^2} g_{Const}. \quad (7)$$

为了控制拉力的大小，我们在 δ 上使用一个较小的乘数因子 $p > 0$。使用 p 更新 δ 的策略如下：对于 δ，存在一个初始值 δ_0。如果目标指标未能满足约束，则将 δ 乘以 $1+p$ 以增强拉力（$\delta' = (1+p)\delta$）。如果约束得到满足，则将 δ 重置为其初始值（$\delta' = \delta_0$）。

请注意，我们还使用梯度下降来训练硬件生成器的权重 v。因此，我们以相同的方式计算 m_v^*，但在更新生成器时，使用 $g_{Cost_{HW}}$ 代替 $g_{\mathcal{L}oss}$。

尽管单个约束已经是一个颇具挑战性的目标，我们的方法还可以进一步推广，以适应多个约束。现在，梯度仅在不满足各个约束的方向上进行修正。我们给出一种更通用的表述：

$$g = \begin{cases} g_{\mathcal{L}oss} & \text{, if } \bigwedge_{i=1}^{n}(t_i \le T_i) \\ & \text{or } \bigvee_{i=1}^{n}(t_i > T_i) \wedge g_{\mathcal{L}oss} \cdot g_{Const} \ge 0, \\ m_\alpha + g_{\mathcal{L}oss} & \text{, otherwise} \end{cases} \quad (8)$$

$$g_{Const} = \frac{\partial \sum_{i=1}^{n} \max(t_i - T_i, 0)}{\partial \alpha}. \quad (9)$$

4.4 实现细节

硬件成本函数。在这项工作中，我们选择推理延迟、能耗和芯片面积作为常用的硬件指标。综合考虑这些指标，一种常用的成本函数是将它们相乘（即 EDP、EDAP），如 [6, 9] 所示。然而，我们发现能耗通常更容易优化，因此使用这种成本函数会不公平地偏向能耗导向设计。因此，我们使用如下的平衡加权和作为成本函数。

$$Cost_{HW} = C_E Energy + C_L Latency + C_A Area. \quad (10)$$

估计器与生成器网络。接下来 [6],，我们使用带有残差连接的五层多层感知机（MLP）来分别构建估计器和生成器。为了训练估计器，我们首先从搜索空间中随机采样 10.8M 个网络-加速器配对（占总搜索空间的 2.95e−9 %），构建数据集，并使用 Timeloop [25]和 Accelergy [29]对其硬件指标进行评估。利用该数据集，估计器

训练周期为 200，批次大小为 256。使用学习率为 1e-4 的 Adam优化器更新权重。估计器在所有指标上的准确率均超过 99%，足以作为协同探索的引擎。生成器采用随机初始化，并与神经架构搜索超网联合训练。当硬约束产生的操纵梯度反向传播时，生成器会学习为给定的神经网络架构创建符合约束的加速器。

搜索空间。我们使用 ProxylessNAS [4] 作为神经架构搜索的骨干网络，并通过路径采样训练 α。它由多种 MBConv操作配置组成，卷积核大小为 {3, 5, 7}，扩展比为 {3, 6}。CIFAR-10 [16] 和 ImageNet [17] 数据集的网络层数分别为 18 层和 21 层。然而，我们的方法与神经架构搜索的实现正交，因此可以灵活地选择任意可微神经架构搜索算法，例如 DARTS [21]或 OFA [3]。

我们采用 Eyeriss [5]作为加速器的骨干架构。它由一个二维处理单元（PE）阵列组成，每个处理单元都连接有一个乘累加（MAC）单元和一个寄存器文件。因此，硬件加速器设计空间包括处理单元阵列大小（从 12×8 到 20×24）以及每个处理单元的寄存器文件大小（从 16B 到 256B）。此外，搜索空间还包括数据流：类似于 [14], 的权重驻留（WS）、类似于 [8] 的输出驻留（OS），以及类似于 [5]的行驻留（RS）。

5 实验

5.1 实验环境

我们在 HDX 上使用 CIFAR-10 [16] 和 ImageNet [17] 数据集进行了实验。对于报告的所有硬件指标（延迟、能耗和芯片面积），我们使用 Timeloop [25] 和 Accelergy [29] 对设计硬件进行直接评估，而非采用估计器输出的数值，以避免学习模型可能产生的误差。网络准确率通过训练最终网络架构进行评估；我们从头开始训练该架构 300 个周期，批量大小为 64。我们使用带有 Nesterov动量 [24], 的 SGD优化器，并采用余弦学习率调度，初始学习率为 0.008；权重衰减项和动量分别为 1e-3 和 0.9。两个数据集的训练数据均采用 AutoAugment [7], 进行增强。

5.2 与现有方法的比较

表1列出了与 HDX 进行比较的现有可微协同探索方法。我们比较了以下方法：

• **神经架构搜索 → 硬件**：仅进行普通的可微神经架构搜索 [4]，随后使用 Timeloop 进行穷举硬件搜索 [25]。
• **Auto-NBA [9]**：一种可微协同探索方法，通过梯度下降直接搜索硬件参数，其中硬件与 DNN 之间的关系由查找表表示。

•**DANCE [6]**：一种不含硬约束的最先进可微协同探索方法，其中生成器和估计器均建模为神经网络。

• **DANCE + 软约束**：为表示一种利用现有解考虑约束的合理方法，我们添加了软约束项 $\lambda_{Soft} \cdot max(t/T - 1, 0)$，如 [12] 所示。• **HDX**：所提出的方法采用 $p = 1e{-}2$。

方法	Hard 约束	神经网络-硬件 关系	在 60 帧/秒约束下搜索		
			#搜索次数	成本*	平均误差（%）
神经架构搜索 [4] → 硬件搜索	✗	✗	5.0	10.9 小时	7.26
Auto-NBA [9]	✗	✓	6.8	10.2小时	5.67
DANCE [6]	✗	✓	6.6	12.2小时	5.32
DANCE +软约束 [12]	✗	✓	4.9	9.1h	5.36
HDX（提出的方法）	✓	✓	**1**	**2.0h**	**4.65**

*GPU小时

表1：可微协同探索的比较

如表1所示，HDX 是唯一一种通过可微协同探索考虑硬约束的方法。

为了进行定量比较，我们设计了一种算法，可在不使用硬约束的情况下通过协同探索寻找受约束解。由于受约束指标（例如延迟）过低往往意味着解在其他指标（例如准确率和能耗）方面效率不佳，因此我们将解的指标达到目标约束的50%~100%作为标准。下面简要介绍一种类似二分搜索的算法：

（1）选择一个会间接影响受约束指标（例如延迟）的控制参数（λ_{Soft}和 λ_{Cost}）。（2）使用默认控制参数值进行搜索。（3）每一步将控制参数加倍并进行搜索，直到指标满足约束。（4）如果指标低于目标值的50%，则以二分搜索的方式缩小控制参数。

请注意，由于每次搜索的结果差异很大，且存在非线性关系（如第3节所示），因此无法通过二分搜索精确求解该问题。在实际实现中，我们加入了更多功能来处理边界情况，并避免进入错误的参数区域（这可能由每次搜索的结果差异导致）。我们运行该算法100次并报告平均结果。我们还报告平均误差，作为解的质量的替代指标。

如表1所示，所有基线方法都需要多次搜索才能找到满足约束的解。对于 DANCE，平均需要搜索 6.6 次、耗费 12.2 GPU小时；即使采用软约束，也需要搜索 4.9 次、耗费 9.1 GPU小时。另一方面，HDX 只需搜索一次即可找到满足约束的解。此外，基线方法的搜索次数很大程度上取决于控制参数默认值的选择。很难猜出合适的初始参数，而且猜错可能导致更高的重复成本。这说明 HDX 的必要性。

5.3 协同探索结果

图 3 展示了多种技术在 CIFAR-10 数据集上的协同探索实验结果。在实验中，我们为延迟设置了两个不同的约束：16.6 ms（60 帧/秒）和 33.3ms（30 帧/秒）。对于所有协同探索方法，我们通过将 λ_{Cost} 从 0.001 变到 0.005，获得五个解，以便公平比较各方法得到的不同设计点。对于 $NAS \rightarrow HW$ 我们获得了取值范围各异的 10 个解作为参考，因为直接应用 λ_{Cost} 不可行。对于 DANCE 和 Auto-NBA，黑色标记表示无约束结果，彩色标记表示采用相应颜色的软约束所获得的结果。

在所有实验中，我们使用了 $C_E = 2.9$、$C_L = 6.2$ 和 $C_A = 1.0$ 10，（来自公式），使各项指标的差异尺度大致相同，从而进行公平比较。对于仅神经架构搜索方法，$Cost_{HW}$ 仅用于硬件设计阶段。

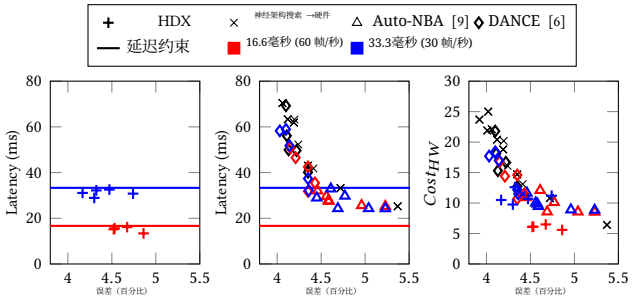


图 3：协同探索结果。（左）和（中）表示延迟，（右）表示硬件成本。彩色标记表示具有相同颜色约束的方法。

图 3（左）和（中）展示了误差与延迟之间的关系。彩色水平条表示我们采用的两个延迟目标。可以很容易看出，无论 λ_{Cost} 的值如何，HDX 找到的所有解都满足给定的硬约束。此外，所有解的延迟都略低于约束值，表明这些解并未针对受约束指标（延迟）过度优化。DANCE [6] 和 Auto-NBA [9] 能够利用硬件指标与准确率之间的权衡，但无法控制是否满足约束。即使加入软约束项，它们也大多未能找到满足约束的解。乍看之下，Auto-NBA 似乎稍微更擅长满足约束，但这是因为其基线方法更倾向于硬件高效解，而非高准确率解，并非因为它具备满足约束的能力；这一点可由如下事实证明：不存在准确率高或延迟低于 16.6 毫秒的解。

5.4 HDX 找到的解的质量

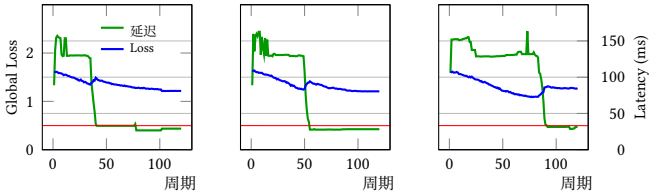
在本小节中，我们展示 HDX 能够：1）处理来自全部三项指标（延迟、能耗和芯片面积）的约束；2）处理多个约束；3）获得整体质量良好的解。图 3（右）同时绘制了 $Cost_{HW}$ 和误差，从而可以根据帕累托最优性评估解的质量。由于图 3（左）和（中）忽略了其他指标，因此必须将 $Cost_{HW}$ 放在一起比较，才能确保公平。

从图中可以清楚看出，HDX 得到的解的质量优于 NAS→HW 方法，且不逊于现有的协同探索方法。事实上，在帕累托最优性方面，严格约束（16.6ms）下得到的解甚至优于现有解。

表 2：展示解的质量的结果

索引	约束	延迟（毫秒）	E（毫焦）	面积（毫米²）	误差（百分比）	$Cost_{HW}$	Loss
A	锚点	69.23	37.00	2.53	4.10 ± 0.16	21.84	0.632
	延迟	**43.99**	21.79	2.10	4.20 ± 0.07	13.87	0.624
	能耗	51.98	**29.18**	2.53	4.38 ± 0.17	17.44	0.630
	芯片面积	64.00	34.82	**2.53**	4.05 ± 0.06	20.56	0.629
	All	**63.72**	**12.09**	**1.86**	4.12 ± 0.18	13.29	0.623
B	锚点	49.65	27.53	2.53	4.22 ± 0.06	16.67	0.638
	延迟	**48.02**	27.33	2.53	4.27 ± 0.09	16.41	0.644
	能耗	95.02	**24.45**	1.89	4.05 ± 0.10	20.76	0.648
	芯片面积	54.74	29.81	**2.53**	4.11 ± 0.13	17.96	0.645
	All	**41.32**	**8.59**	**1.86**	4.35 ± 0.05	9.50	0.629

*粗体彩色数字表示它们受到相同颜色的非粗体数字所对应的约束。



(a) p=1e-2 **(b)** p=7e-3 **(c)** p=4e-3

图4：HDX 对 p 的敏感性。红线表示 33.3 ms 的延迟约束。

为了进一步研究 HDX 找到的解的质量，我们又进行了一组实验。我们选取了 DANCE 方法找到的若干解作为“锚点”解，并列于表 2 中。基于这些解，我们选择其中一项或全部三项硬件指标作为硬约束，并使用 HDX 进行协同探索。由于可以保证此类解确实存在，因此一种好的方法应当能够找到满足约束、且质量至少相近的解。与第5.3节一样，我们考察的 8 种情况均成功找到了有效解。此外，如最右列所示，所有解的全局损失值都与锚点解相近。

5.5 ImageNet 数据集的结果

表3展示了在 125 毫秒约束下，ImageNet 数据集的协同探索结果 [17],。如表中所示，HDX 总能在约束范围内成功找到一个解，而其他方法往往无法满足约束。此外，Top-1 错误率和全局损失表明，与 DANCE 或其变体相比，HDX 找到的解的质量完全没有受到影响。

5.6 拉动幅度 I 的敏感性研究I

n HDX, the hyperparameter controlling the pull 超参数

拉取幅度。图4展示了在延迟约束（33.3毫秒）的探索过程中，全局损失和延迟如何变化，其中 p 分别取 1e-2、7e-3 和 4e-3。

无论 p 的取值如何，约束值曲线都呈现出相似的趋势。起初，全局损失主要得到优化，而延迟保持稳定。在这一阶段，拉动幅度 δ（见公式 7）仍在增大，尚不足以带来实质性变化。到某个时刻，δ变得足够强，开始推动解向降低延迟的方向移动。当延迟满足约束后，全局损失开始下降，同时延迟保持不变。最终解在全局损失和延迟方面没有显著差异，这表明 HDX 对超参数 p不敏感。

表3：ImageNet 实验结果

方法	是否约束?	延迟（毫秒）	误差（百分比）	硬件成本	Loss
神经架构搜索→硬件	✗	242.92	24.84	46.29	1.99
	✗	135.39	28.83	24.26	2.17
DANCE	✗	165.98	25.46	29.37	2.04
	✓	121.58	25.28	25.92	2.09
DANCE+软约束	✗	188.69	25.09	33.14	1.99
	✓	105.65	26.37	25.58	2.08
HDX（提出的方法）	✓	92.06	25.01	24.48	1.98
	✓	112.11	25.20	22.63	2.00

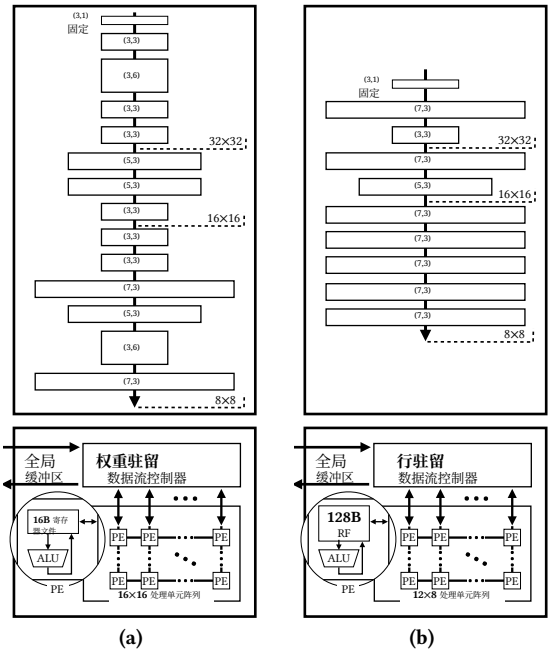


图 5：针对 60 帧/秒和 30 帧/秒约束搜索得到的网络和加速器。(*m*,*n*)表示使用 *m*×*m* 卷积核和 *n* 扩展比的 MBConv块。

5.7 对搜索所得解的分析

图 5 展示了针对 60 帧/秒（a）和 30 帧/秒（b）约束搜索得到的网络和加速器。对于找到的设计组合（a），其设计包含相对较小的卷积核、更多层，以及性能强大的加速器。为了满足严格的约束，同时保持准确率，该网络采用了较小的卷积核，主要为 3×3。使用较小的卷积核可使乘法次数呈二次方减少。因此，减小卷积核尺寸并增加层数，是降低推理延迟的良好选择。从加速器设计来看，其处理单元阵列相对较大（16×16），从而实现较低的延迟。它采用权重驻留（WS）数据流，这种数据流以低延迟著称。此外，网络中还有一些具有较高通道扩展比的卷积核。WS 利用通道并行性实现快速执行，因此比搜索得到的网络更具优势。

另一方面，在针对 30 帧/秒的设计（b）中，设计最终选定了一个在满足约束的同时能够优化能耗的解。该设计在网络中使用更大的卷积核，并在加速器中采用行驻留（RS）数据流。众所周知，RS 具有良好的能效 [5],，并能利用卷积核和激活值空间维度上的并行性。因此，较大的卷积核有利于行驻留数据流。为降低能耗，该设计采用较少的处理单元（12×8）、较大的寄存器文件以节省片外访问能耗，并减少网络中的层数。

6 结论

本文提出了 HDX，这是一种硬约束可微协同探索方法。通过有条件地应用梯度操控，使解朝着满足约束的方向移动，可以可靠地满足硬约束，并获得高质量的解。我们相信，这项提议将显著助力基于 DNN 的系统开发。

致谢

本研究得到了韩国国家研究基金会（NRF）由韩国政府（MSIT）资助的项目（2022R1C1C1008131、2022R1C1C1011307），以及韩国政府（MSIT）资助的信息通信技术规划评估院（IITP）项目（2020-0-01361，人工智能研究生院项目（延世大学））的支持。

参考文献

[1] M. S. Abdelfattah 等。2020。《两全其美：卷积神经网络及其硬件加速器的自动化机器学习协同设计》。载于 DAC。[2] M. Berman 等。2020。《AOWS：具有延迟约束的自适应最优网络宽度搜索》。载于 CVPR。[3] H. Cai 等。2019。《Once-for-All：训练一个网络并将其专门化以实现高效部署》。载于 ICLR。[4] H. Cai、L. Zhu 和 S. Han。2019。《ProxylessNAS：面向目标任务和硬件的直接神经架构搜索》。载于 ICLR。[5] Y.-H. Chen、T. Krishna、J. S. Emer 和 V. Sze。2016。《Eyeriss：一种节能的深度卷积神经网络可重构加速器》。IEEE JSSC（2016）。[6] K. Choi 等。2021。《DANCE：可微分加速器/网络协同探索》。载于 DAC。[7] E. D. Cubuk 等。2019。《AutoAugment：从数据中学习数据增强策略》。载于 CVPR。[8] Z. Du 等。2015。《ShiDianNao：让视觉处理更靠近传感器》。载于 ISCA。[9] Y. Fu 等。2021。《Auto-NBA：在网络、位宽和加速器的联合空间中进行高效有效的搜索》。载于 ICML。[10] S. Gupta 和 B. Akin。2020。《使用自动化机器学习进行加速器感知神经网络设计》。arXiv 预印本 arXiv:2003.02838（2020）。[11] C. Hao 等。2019。《FPGA/DNN协同设计：面向边缘端物联网智能的高效设计方法》。载于 DAC。[12] Y. Hu、X. Wu 和 R. He。2020。《TF-NAS：重新思考延迟约束可微分神经架构搜索的三种搜索自由度》。载于 ECCV。[13] W. Jiang 等。2019。《准确率与效率：通过感知 FPGA 实现的神经架构搜索兼得二者》。载于 DAC。[14] N. P. Jouppi 等。2017。《张量处理单元在数据中心中的性能分析》。载于 ISCA。[15] J. Kim 等。2021。《DPM：一种用于外推的物理信息神经网络新型训练方法》。载于 AAAI。[16] A. Krizhevsky、G. Hinton 等。《从微小图像中学习多层特征》。http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf [17] A. Krizhevsky、I. Sutskever 和 G. E. Hinton。2012。《使用深度卷积神经网络进行 ImageNet分类》。载于 NeurIPS。[18] H. Kwon 等。2020。《MAESTRO：一种以数据为中心的方法，用于理解深度神经网络映射的复用、性能和硬件成本》。IEEE Micro（2020）。[19] Y. Li 等。2020。《EDD：面向嵌入式人工智能解决方案的高效可微分 DNN 架构与实现协同搜索》。载于 DAC。[20] Y. Lin、M. Yang 和 S. Han。2021。《NAAS：神经加速器架构搜索》。载于 DAC。[21] H. Liu、K. Simonyan 和 Y. Yang。2019。《DARTS：可微分架构搜索》。载于 ICLR。[22] Q. Lu 等。2019。《资源受限硬件平台上的神经架构搜索》。载于 ICCAD。[23] N. Nayman、Y. Aflalo、A. Noy 和 L. Zelnik-Manor。2021。《HardCoRe-NAS：硬约束可微分神经架构搜索》。arXiv 预印本 arXiv:2102.11646（2021）。[24] Y. E. Nesterov。1983。《一种求解凸规划问题的方法及其收敛速度 $O(1/k^2)$》。苏联科学院通报（1983）。[25] A. Parashar 等。2019。《Timeloop：一种系统化的深度神经网络加速器评估方法》。载于 ISPASS。[26] J. Redmon、S. Divvala、R. Girshick 和 A. Farhadi。2016。《You Only Look Once：统一的实时目标检测》。载于 CVPR。[27] G. Saha、I. Garg 和 K. Roy。2021。《用于持续学习的梯度投影记忆》。载于 ICLR。[28] M. Sandler 等。2018。《MobileNetv2：倒置残差与线性瓶颈》。载于 CVPR。[29] Y. N. Wu、J. S. Emer 和 V. Sze。2019。《Accelergy：一种面向加速器设计的架构级能耗估算方法》。载于 ICCAD。[30] L. Yang 等。2020。《面向多项任务的神经架构与异构ASIC加速器设计协同探索》。载于 DAC。[31] S. Zhou 等。2016。《DoReFa-Net：使用低位宽梯度训练低位宽卷积神经网络》。arXiv 预印本 arXiv:1606.06160（2016）。