

**AIoT课程报告**

|  |  |
| --- | --- |
| 指导教师： | 王玲 |
| 组长姓名： | 陈思达 |
| 学号： | 2021112558 |
| 报告日期： | 2024/12/25 |

**题目：边缘设备深度学习加速**

边缘设备深度学习加速的意义和方法

深度学习技术与边缘计算技术推动了万物互联的智能物联网发展，改变了人们的生活生产方式。而物联网的昆虫纲特性决定了智慧应用的多样化，如何对强异构的低能边端按照多样化的需求进行赋能.是当前学术及工业界亟待解决的问题。在边缘环境下，受限于物理尺寸和能源供给，边缘设备的计算资源往往非常有限，而深度学习模型对算力资源有着巨大的需求，导致边缘赋能的受阻。[1]

深度学习算法通常计算成本高、耗电大，并且需要大量内存来处理数百万个参数的复杂迭代操作。因此，深度学习模型的训练和推理通常在云中的高性能计算 (HPC) 集群上执行。数据传输到云端会导致高延迟、往返延迟、安全和隐私问题以及无法实时决策。因此，在边缘设备上进行处理可以显着降低云端传输成本。边缘设备是最接近用户的终端设备，例如移动电话、网络物理系统 (CPS)、可穿戴设备、物联网 (IoT)、嵌入式和自治系统以及智能传感器。这些设备的内存、计算资源和功率处理能力有限。因此，硬件和软件层面的优化技术都被开发出来，以有效地处理边缘的深度学习部署。了解现有的研究、挑战和机遇对于利用具有人工智能 (AI) 功能的下一代边缘设备至关重要。

基于边缘计算的深度学习优化技术中一个重要的部分就是训练一个极其高效的模型，该模型可在资源受限的边缘节点设备（从Arduino 到 RaspberryPi）上运行。为了解决深度学习模型网络复杂度高、计算、存储资源消耗大，无法高速的分发部署，并在网络、计算、存储等受限的场景下高效准确推理的问题，研究者们已提出很多深度学习模型优化方案。[2]

为了在边缘设备上进行高效的深度学习推理，主要有四个研究方向：新颖的深度学习架构和算法设计；现有DL方法的优化；算法开发——硬件协同设计；用于深度学习部署的高效加速器设计。

在边缘计算领域中，为了适应不同边缘平台的性能，衍生出了不同的模型优化技术，这其中包含机器学习算法优化和深度学习模型优化技术，矢量量化也可以进行深度学习模型压缩。[3]

**模型轻量化在边缘设备深度学习加速中的作用**

随着数据量的增长，神经网络作为一种有效的数据处理手段已得到广泛关注。然而，高性能的神经网络往往伴随着较高的模型复杂度，难以在各种资源受限的移动设备、嵌入式设备上高效部署。因此，在保障模型精度可接受的情况下，深入研究模型压缩技术以降低模型复杂度，具有重要的理论及应用研究价值。[3]

在边缘计算领域中，为了适应不同边缘平台的性能，衍生出了不同的模型优化技术，这其中包含机器学习算法优化和深度学习模型优化技术，矢量量化也可以进行深度学习模型压缩。

通过压缩的方法把模型复杂度和模型大小尽可能的降低，在此基础上，通过统一模型转换框架把模型移植到边缘节点上，达到在内存极其有限的开发板上运行深度学习模型推理的目的，从而达到将机器学习或深 度学习的人工智能方法向网络边缘延展，在数据产生源头实现复杂的分类或者回归，减少数据上传和数据延迟。[3]

模型轻量化在边缘设备深度学习加速中的作用主要体现在以下几个方面：

1. 降低计算成本, 缘设备可以在本地执行模型推理，减少对云端计算资源的依赖，从而降低计算成本。
2. 提高响应速度：在实时环境中，边缘设备可以在本地执行模型推理，提高响应速度。
3. 保护数据隐私：通过在本地执行模型推理，边缘设备可以避免数据泄露，保护数据隐私。
4. 减小模型大小：模型量化技术通过减少模型中参数的位数来降低模型的存储需求和计算复杂性。
5. 减少冗余权重或神经元：通过模型剪枝技术，去除模型中不重要的连接或神经元，减少计算和存储复杂度，同时尽量保持模型性能。
6. 提升运行效率：模型轻量化技术显著减少AI模型在边缘设备上的计算需求，提升运行效率。

2.模型剪枝

2.1模型剪枝的定义

模型剪枝作为一种重要的模型压缩技术，在深度学习领域中扮演着关键角色。在深度学习的神经网络模型里，包含着海量的参数，如权重和偏置等。这些参数共同构成了模型复杂的结构和强大的学习能力，但同时也带来了一些问题，例如计算资源的高需求和模型存储的大空间占用等。

模型剪枝的核心定义在于，它依据一定的规则和方法，精准地去除神经网络中那些对模型性能贡献微小甚至几乎没有贡献的组成部分。这些部分涵盖了神经元、连接（权重）以及整个网络层等。形象地说，这就如同对一棵枝繁叶茂的大树进行精心修剪，将那些多余的、对树木整体生长和美观没有实质价值的树枝果断剪掉，从而使大树（模型）的结构更加精简，进而提升其运行效率和性能表现，使其能够以更高效的状态应对各种数据处理和分析任务。

2.2模型剪枝的作用

1. 减少计算资源需求降低

计算量：神经网络模型在执行推理任务，比如对输入数据进行分类或预测时，通常需要进行大规模的计算操作。而通过模型剪枝，去除部分连接或神经元后，能够显著减少计算量。以卷积神经网络（CNN）为例，其卷积层的乘法和加法运算数量与连接数量呈直接正相关关系。当进行剪枝操作后，卷积核中的连接数量得以减少，相应地，每层的计算复杂度大幅降低，进而有效地加快了模型的推理速度。这对于那些对实时性要求较高的应用场景，如自动驾驶中的目标识别、工业生产中的实时故障检测等，具有至关重要的意义，能够使模型在更短的时间内给出准确的结果，满足实际应用的实时性需求。

节省存储资源：大型神经网络模型由于其庞大的参数数量，存储时需要占用大量的内存空间。然而，对于众多边缘设备，如移动设备、物联网设备等，它们的存储资源往往十分有限。模型剪枝技术通过减少模型参数的数量，能够有效地降低模型存储所需的空间。例如，一个原本需要占用 100MB 存储空间的深度神经网络模型，经过合理的剪枝处理后，可能仅需 20MB 的存储空间，这使得模型能够更加轻松地部署在这些存储资源受限的设备上，为人工智能在边缘计算领域的广泛应用提供了有力支持，促进了智能设备的小型化和便携化发展。

1. 防止过拟合

过拟合是深度学习中常见的问题之一，表现为模型在训练数据上展现出极高的准确性，但在面对测试数据或新的未知数据时，性能却大打折扣。造成过拟合的一个重要原因是模型中存在大量冗余的参数，这些参数可能会导致模型过度拟合训练数据中的噪声和无关特征，从而失去对新数据的泛化能力。

而模型剪枝则可以有效地解决这一问题，通过去除这些冗余的部分，使模型的结构变得更加简单，复杂度得到显著降低。这样一来，模型能够更好地聚焦于数据中的关键特征和共性规律，从而增强其泛化能力，使其在新的数据上也能够有较为出色的表现。例如，在图像分类任务中，一个过拟合的模型可能会将训练集中每个图像的细节特征，包括噪声和一些与分类无关的特征都一一记住，导致在面对新的图像时，无法准确地进行分类。而经过剪枝后的模型，能够摒弃那些无关紧要的特征，学习到更具代表性和通用性的特征，从而对新的图像实现更准确、更可靠的分类。

1. 提高模型可解释性

复杂的神经网络模型通常被视为难以理解的 “黑盒”，其内部的决策过程和逻辑关系难以直观地被人们所洞察和解释。这在一定程度上限制了模型在一些对可解释性要求较高的领域中的应用，如医疗、金融等。

模型剪枝后的模型结构更加简洁明了，一些不重要的神经元和连接被去除后，研究人员能够更加清晰地观察到模型中关键部分的作用和相互关系。例如，在一个用于医疗影像诊断的深度学习模型中，通过剪枝操作，可以明确地发现哪些神经元或网络层对于识别特定疾病的关键特征起到了主要作用。这不仅有助于医学专家更好地理解模型的诊断依据，提高模型在实际医疗场景中的可信度和可接受度，还能够为医学研究提供有价值的参考，促进医学领域与人工智能技术的深度融合。

1. 提升模型鲁棒性

适当的模型剪枝可以去除一些对输入数据微小变化较为敏感的连接或神经元，使模型在面对数据的噪声、扰动或对抗攻击时更加稳定，从而提高模型的鲁棒性。例如，在图像识别任务中，经过剪枝后的模型对于图像的部分遮挡、光照变化等情况能够有更稳定的表现，减少因这些因素导致的错误识别，更好地适应实际应用中复杂多变的环境。

2.3模型剪枝的意义

1. 推动人工智能在边缘计算的应用

随着物联网技术的飞速发展，越来越多的设备需要在本地具备数据处理和智能决策的能力，如智能家居设备、智能传感器等。这些边缘设备通常具有计算能力和存储资源有限的特点，难以直接部署复杂的深度学习模型。

模型剪枝技术的出现，为解决这一难题提供了有效的途径。它能够使复杂的深度学习模型经过压缩和优化后，适应边缘设备的资源限制，从而将人工智能的强大功能拓展到边缘计算领域。这不仅实现了设备的智能化和自主化，还能够在本地对数据进行快速处理和分析，减少数据传输延迟和对云端服务器的依赖，提高数据的安全性和隐私性，为物联网的发展注入了新的活力，推动了智能生活和智能工业等领域的快速发展。

1. 优化模型训练和部署成本

训练大型神经网络模型往往需要耗费大量的计算资源和时间成本。从计算资源角度来看，需要高性能的硬件设备，如 GPU 集群等，来支持大规模的矩阵运算和参数更新，这无疑增加了硬件投资成本。从时间成本方面，复杂模型的训练可能需要数天甚至数周的时间，严重影响了模型的开发和迭代效率。

而模型剪枝可以显著减少模型的规模，从而有效地缩短模型的训练时间，降低对计算资源的需求。在模型部署阶段，经过剪枝后的模型对存储和计算资源的要求大幅降低，减少了硬件设备的采购成本和运营成本。例如，在大规模数据中心中，如果能够对模型进行有效的剪枝，就可以在相同的硬件资源条件下部署更多的模型，提高数据中心的资源利用率，实现更高效的任务并行处理和服务响应，为企业和科研机构节省了大量的资金和时间成本。

1. 促进深度学习理论的发展

模型剪枝过程涉及到对神经网络结构和参数重要性的深入研究和探索。通过不断地尝试和分析哪些部分的模型可以被剪枝而不影响性能，研究人员能够更加深入地了解神经网络的工作原理和内在机制。

这种探索有助于发现神经网络中不同神经元、连接和层之间的相互关系和协同工作模式，为神经网络架构的设计提供新的思路和方法。例如，通过剪枝研究发现了某些神经元组的协同工作模式，为设计新型神经网络架构提供了重要依据，推动深度学习理论向更加高效、简洁的方向发展。同时，模型剪枝也为模型压缩、优化和泛化等方面的理论研究提供了丰富的实验数据和实践经验，促进了深度学习领域相关理论的不断完善和创新。

（四）助力人工智能伦理与可持续发展

随着人工智能在各个领域的广泛应用，其伦理和可持续发展问题日益受到关注。模型剪枝通过减少计算资源的消耗和能源的使用，降低了人工智能对环境的影响，符合可持续发展的理念。同时，较小的模型更容易被理解和解释，有助于解决人工智能决策过程中的透明度和可解释性问题，从而更好地遵循伦理原则，促进人工智能在社会中的健康、可持续发展，增强公众对人工智能技术的信任和接受度。

**非结构化剪枝**

非结构化剪枝是一种深度学习模型压缩技术，它通过移除模型中不重要的权重来减少模型的参数量和计算量。

非结构化剪枝是指修剪参数的单个元素，比如全连接层中的单个权重、卷积层中的单个卷积核参数元素或者自定义层中的浮点数（scaling floats）。其重点在于，剪枝权重对象是随机的，没有特定结构，因此被称为非结构化剪枝。

非结构化剪枝的工作原理基于评估权重的重要性，然后根据这个重要性评分来确定哪些权重可以被剪枝。重要性评分可以基于梯度、绝对值、L2范数等标准进行计算。剪枝后，通常会对模型进行微调，以恢复其性能。

非结构化剪枝的优势

1. 灵活性和可定制性：非结构化剪枝允许用户根据具体需求实现特定的剪枝逻辑，提供了高度的灵活性。
2. 更高的剪枝比例：理论上，非结构化剪枝可以获得比结构化剪枝更高的剪枝比例，因为它可以在更细粒度上保留重要信息。
3. 压缩存储优势：非结构化剪枝适合不考虑硬件加速的场景，例如压缩存储大小，这对于需要减少模型存储需求的应用场景非常有用。

非结构化剪枝在以下场景中尤为重要：

移动设备：在移动设备上运行的深度学习模型需要较低的计算需求和存储空间，非结构化剪枝可以显著减少模型的体积和计算量，使其更适合在移动设备上运行。

边缘计算：边缘设备通常资源有限，非结构化剪枝可以帮助优化模型性能，使其在边缘设备上实现更好的实时性和能效比。

非结构化剪枝的挑战：

精度损失：剪枝过程中可能会剪掉部分重要信息，导致模型精度下降。

硬件依赖性：非结构化剪枝后的模型通常需要专用硬件才能实现压缩和加速效果。

剪枝策略的选择：如何选择合适的剪枝策略（如剪枝比例、剪枝粒度等）是一个复杂的问题。

非结构化剪枝通过随机移除部分权重，不改变模型结构，提供了一种灵活且有效的模型压缩方法，尤其适用于对模型存储和计算需求有严格要求的场景。然而，它也面临着精度损失和硬件依赖性的挑战。

结构化剪枝是一种模型压缩技术，它通过移除模型中的整个结构单元（如通道、神经元或层）来减少模型的参数量和计算量。

**结构化剪枝**

结构化剪枝的基本概念：

结构化剪枝与非结构化剪枝不同，它不是删除单个权重，而是剪除整个结构单元。这样做的好处是剪枝后的模型在硬件加速和并行计算方面更具优势，因为剪除的是连续的结构单元，便于硬件进行优化。

主要类型：

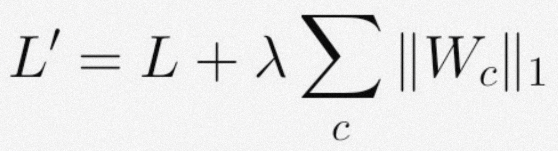
1.通道剪枝：删除卷积层中的某些通道。这种方法通过减少每个卷积层的输出通道数来降低模型复杂度。

2.神经元剪枝：删除全连接层中的某些神经元。这种方法可以减少模型的参数数量和计算量。

3.层剪枝：删除整个层或模块。这种方法可以显著降低模型的深度和复杂度。

结构化剪枝的数学基础：

结构化剪枝可以通过优化问题来实现，例如，通过添加正则项来鼓励权重矩阵的稀疏性。一个常见的方法是在损失函数中加入L1范数，即：



其中，L是原始损失函数，Wc是第c个通道的权重矩阵，λ是正则化系数，||Wc||1是权重矩阵的L1范数。

结构化剪枝的步骤

1.正常训练模型：首先对模型进行正常的训练。

2.模型剪枝：根据设定的剪枝比例和策略，剪除不重要的通道、神经元或层。

3.重新训练模型：剪枝后，通常需要对模型进行微调以恢复性能。

结构化剪枝的方法

1.向量级剪枝：将卷积核中的向量作为修剪的结构单位进行剪枝。

2.内核级剪枝：对滤波器中的二维卷积核进行剪枝。

3.组级剪枝：根据滤波器上的相同稀疏模式进行剪枝，当多个滤波器拥有相同的稀疏模式时，可以表示为一个细化的稠密矩阵，利用组级剪枝算法，卷积可以通过稠密矩阵乘法实现。

4.滤波器剪枝：对卷积滤波器或信道进行剪枝，也称为通道剪枝。

结构化剪枝的优缺点

优点：

硬件友好：剪除的是整个结构单元，便于硬件进行加速和优化。

显著减少计算量：通过删除通道、神经元或层，显著降低模型的计算复杂度和存储需求。

保持模型结构一致性：剪枝后的模型仍然保持良好的结构一致性，便于部署。

缺点：

可能导致性能下降：过度剪枝可能导致模型性能显著下降，需要谨慎选择剪枝比例。

剪枝策略复杂：需要设计有效的评估指标和剪枝策略，确保剪枝效果。

可能需要多次微调：剪枝后通常需要多次微调以恢复模型性能，增加了训练时间。

结构化剪枝通过剪除整个结构单元来实现模型的压缩和加速，它在硬件上实现更高效的加速，尽管可能带来一定的性能损失，但通过合理的剪枝策略和微调过程，可以在保持模型性能的同时，实现模型的高效部署。

三、非结构化剪枝相对于结构化剪枝的优势[4]

1.更高的压缩率：非结构化剪枝可以提供更高的压缩率，因为它可以在更细粒度上移除权重，而结构化剪枝受限于结构单元的数量。

2.更小的精度损失：非结构化剪枝在相同的剪枝比例下，精度损失相对较小，这对于对精度要求较高的应用场景非常有用。

3.灵活性和可定制性：非结构化剪枝允许用户根据具体需求实现特定的剪枝逻辑，提供最大的灵活性。

4.适用于复杂模型：对于模型结构复杂且需要精细调整的场景，非结构化剪枝可以更好地保留关键连接，减少精度下降。

非结构化剪枝在压缩率、精度保持以及灵活性方面具有明显优势，尤其适用于对精度要求较高的复杂模型。然而，这也意味着它可能不如结构化剪枝那样容易在硬件上实现加速，因为非结构化剪枝产生的稀疏矩阵可能需要特殊的处理方式来处理这些稀疏数据。

深度学习模型在各种应用领域中表现出色，但在资源受限的环境

（如移动设备和嵌入式系统）中运行时会遇到许多问题。在这些环境中，内存、电力和计算能力等资源都是有限的，因此高参数量的深度学习模型在这些设备张高效运行是一个巨大的挑战。为了解决这些 问题，模型量化技术应运而生。

模型量化（Quantization）是将深度学习模型的权重和激活值从

32位浮点数转换位8位整数等较低的比特精度，从而减少模型的大小并提高计算速度的一种方法。

32位浮点数用32比特的数据表现实数值。这个方式虽然提供搞

得精密度，但内存用量大，计算成本高。它的优点是精密度高，在学习过程中可以最大限度地提高模型的准确性，但缺点是内存和计算资源消耗大，在移动设备或嵌入式系统中效率低下。

8位整数使用8位数据表示值。主要用于量化（quantuzation） 过程，有利于减少模型尺寸，提高计算速度。它的优点式内存用量

少，计算速度快，可降低功耗。但由于精密度降低，模型的精度可能会损失一部分。然而，使用适当的量化技术可以最大限度地减少精度损失。

模型量化种类将学习模型量子化的技术（PTQ,Post- trainingQuantizaiton）和边学习边量子化的技术（QAT,Quantiziation-Aware Training）。

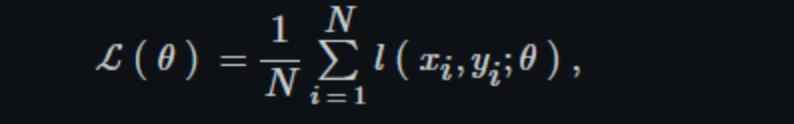
PTQ意味着对已经学习的LLM进行量化的技术,虽然比GAT更容易

实现,但由于权重值的精密度损失,模型准确度可能会降低。 对完成学习的模型进行统计分析,得到阿尔法。

QAT最初是以事先学习过的模型或PTQ模型开始的。 使用PTQ模型时,将发生的损失进行修复。

QAT通过学习时进行量子化的技术,对数据进行微调,以保存性能

。 在学习阶段进行校正、范围估计、剪裁、四舍五入等权重转换过程。 因此,与PTQ相比,虽然可以缓解模型性能的减少,但还需要更 多的运算。 QAT通过学习获得Alpha,在学习完成的模型中增加量子化的层后,重新进行微调,获得适当的Alpha。

 公式 .模型量化公式

量化技术可以通过减少内存使用、加快计算速度和降低功耗等优势，是的深度学习模型在移动设备的嵌入式系统中能够高效运行。

模型量化在各种方面利用。例如，节约存储器、提高运算速度、减少电力消耗。

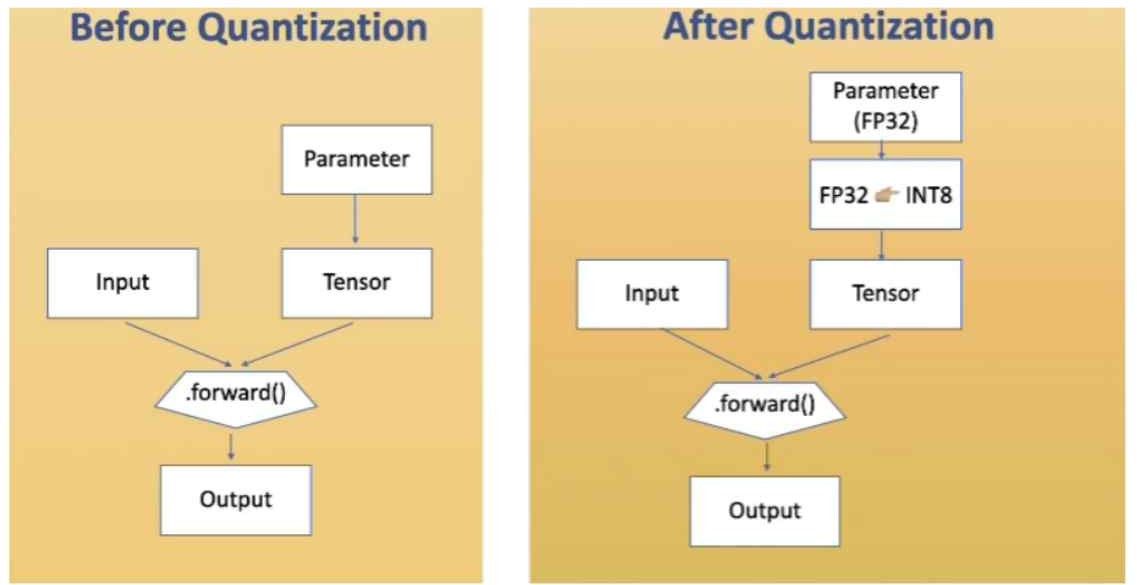
1. 节约存储器：量化是通过将深度学习模型的权重和激活值转换为低位精度来减小模型尺寸的技术。 例如，用8位整数代替32位浮 点，可将模型尺寸减小约四分之一。 这样减少的型号内存使用量少,在存储器资源有限的移动设备或嵌入式系统中也能顺利运行。
2. 提高运算速度：量子化模型采用整数运算代替浮点运算，大大提高了运算速度。 由于整数运算在硬件上比浮点运算更有效，因此具有优势，尤其是在计算资源有限的环境中，例如移动设备。 这导致在实时应用中响应速度加快，整个系统的性能得到改善。
3. 减少电力消耗：量子化模型可以降低功耗。 这是因为净水运算比浮点运算耗电少。 使用量子化的型号可以提高电力效率，延长电池寿命。

Figure 1.模型量化列子

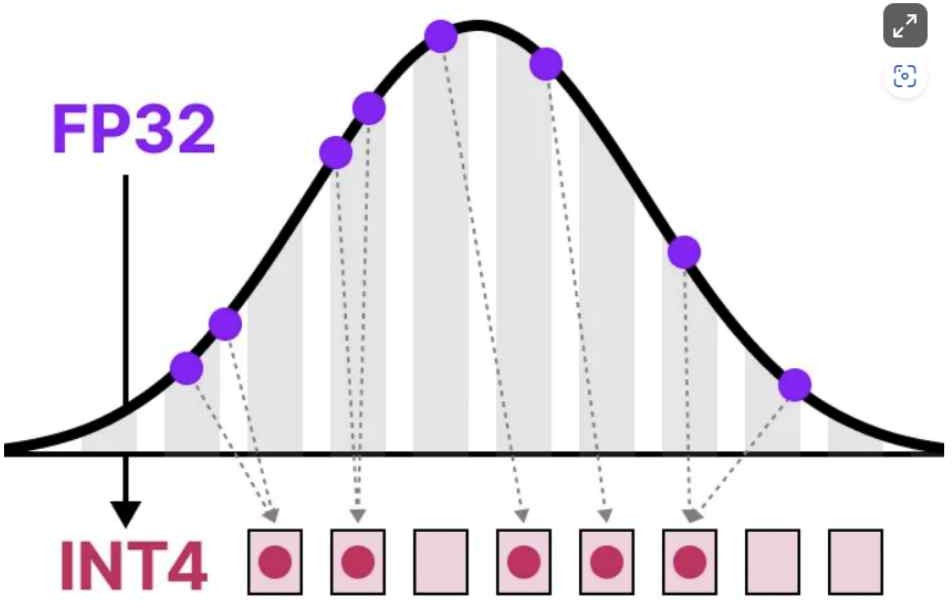
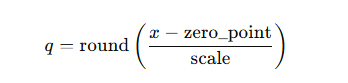


Figure 2。模型量化列子2

随着深度学习模型在各领域的广泛应用，其规模和复杂度也不断增长。大型模型虽然拥有强大的表达能力，但其计算开销和存储需求对硬件设备提出了巨大挑战，特别是在资源受限的嵌入式设备、移动设备和物联网设备中。模型量化（Quantization）作为一种模型压缩技术，通过将浮点数表示的权重和激活值转换为低比特整数（如 INT8），显著降低计算和存储开销，成为模型轻量化的重要手段。根据量化方法和应用时机的不同，量化技术主要分为静态量化（Static Quantization）和动态量化（Dynamic Quantization）。静态量化通过离线完成量化范围的确定，推理时直接使用定点数表示；动态量化则在推理过程中动态完成激活值的量化。这两种方法各有适用场景，但在性能、精度和资源效率上，静态量化通常表现更优。

静态量化是一种在模型部署之前，通过校准数据或量化感知训练（QAT）离线完成量化范围确定的技术。其主要目标是将权重和激活值转换为低比特整数表示，同时最大程度地保留模型的表达能力和预测性能。具体来说，静态量化需要通过校准数据采样，计算每一层激活值的动态范围（min-max），并通过缩放因子（scale）和零点（zero-point）将浮点数映射到整数域。例如，激活值 xxx 的量化公式为：



其中 q 是量化后的整数值。对于静态量化，推理过程中直接使用预先量化的权重和激活值，与硬件加速（如 INT8 算子）结合，显著提升推理效率和能效。其实现通常需要结合专用工具，如 PyTorch 的量化模块或 TensorFlow 的 TensorRT。

动态量化的核心思想是离线量化权重，但激活值的量化范围在推理过程中根据实际输入动态计算。相比静态量化，动态量化无需校准数据，且实现难度更低，适合快速原型开发。动态量化的实现流程通常包括以下步骤：首先对模型的权重进行量化，将浮点数参数转换为低比特整数；其次，在推理时根据每批输入动态确定激活值的量化范围，计算 scale 和 zero-point，并将浮点激活值实时转换为整数表示。这种方法虽然避免了对校准数据的依赖，但由于激活值量化范围是基于运行时数据动态确定的，可能导致范围不稳定，从而引发精度下降。此外，动态量化由于需要实时计算量化范围，其推理效率较静态量化略逊一筹。

静态量化在性能上的优势主要体现在两方面：计算效率和硬件适配性。首先，静态量化在离线阶段完成了所有量化相关的计算（如范围校准、scale 和 zero-point 的确定），推理阶段仅需直接使用量化后的低比特整数进行计算，无需额外的动态开销。这种特性使得静态量化与硬件优化更紧密结合，尤其是在支持 INT8 运算的 CPU、GPU 或专用 AI 芯片上，能够显著提高推理速度。其次，由于静态量化的激活值范围是离线确定的，其量化范围通常更加精准，减少了动态量化中可能因范围不匹配导致的额外运算，从而进一步优化了硬件计算效率。

静态量化的精度优势得益于其对量化过程的精细控制。通过量化感知训练（QAT），模型能够在训练过程中模拟量化对权重和激活值的影响，从而学习到对量化更鲁棒的参数分布。即使不进行 QAT，仅使用校准数据也能通过采样大量输入样本，准确捕获激活值的动态范围，避免量化时信息的丢失。相较之下，动态量化的激活值范围是基于运行时输入动态确定的，可能因输入数据的变化导致范围不准确，从而引发量化误差。此外，静态量化能够支持对非线性操作（如 ReLU、Softmax）的定点实现，而动态量化在这方面的支持较为有限，进一步拉大了两者在精度上的差距。

在内存占用和能耗方面，静态量化也具备显著优势。由于权重和激活值均被量化为低比特整数，静态量化的模型存储需求大幅减少。例如，一个原本以 32 位浮点数存储的模型经过静态量化后可以减少至原大小的四分之一甚至更低。这种压缩效应在大规模模型中尤为明显。此外，整数运算的能耗远低于浮点运算，因此静态量化在推理时能够显著降低能耗，延长嵌入式设备的电池寿命。而动态量化虽然在权重方面同样具备存储优势，但由于激活值仍需以浮点数表示并动态量化，其整体内存节省和能耗降低的效果不如静态量化显著。

静态量化的另一大优势在于其高稳定性和可控性。在离线阶段完成量化范围校准后，模型的推理行为在各种输入条件下都较为可预测，适合对可靠性要求较高的生产环境。而动态量化的推理性能和精度受输入数据分布的影响较大，如果输入数据分布偏离训练时的假设，可能导致激活值量化范围失配，从而影响推理结果。此外，静态量化支持更细粒度的优化，例如对不同层采用不同的量化范围或策略，以适应不同的权重分布特性。这种灵活性使得静态量化能够针对不同的硬件平台和应用需求进行优化，从而进一步提升其实用性。

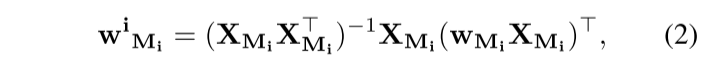
虽然静态量化在性能、精度和资源效率上具有明显优势，但其实现难度较高。量化感知训练需要对训练流程进行改造，增加了开发和训练的复杂性；而校准数据的采样质量也直接影响量化效果。此外，对于一些应用场景（如开发周期较短或对精度要求不高的场合），动态量化以其实现简单、无需校准数据的优势更具吸引力。例如，在快速开发阶段，动态量化能够帮助开发者快速验证模型在低比特表示下的性能，为后续静态量化的实施提供参考。

静态量化和动态量化适用的场景有所不同。静态量化适用于对性能和精度要求较高的生产环境，尤其是在推理效率和能耗至关重要的嵌入式设备和边缘设备中。而动态量化则更适合需要快速部署的场景，如快速开发的原型验证或对模型压缩要求不高的轻量化应用。此外，对于一些无法获取大量校准数据的场景，动态量化提供了一种无需额外数据的量化途径。然而，从长期优化角度看，静态量化的综合性能和资源利用率通常更能满足工业级部署需求。

综上所述，静态量化和动态量化各有其特点和优势。静态量化通过离线校准或量化感知训练，实现了对模型推理性能、精度和资源利用率的全面优化，适合高要求的生产环境；动态量化以其简单快捷的实现方式，在快速开发和测试阶段表现突出。随着硬件技术的进步，量化方法的潜力将进一步释放。例如，量化精度从 INT8 向更低比特（如 INT4、二值化）发展，以及结合剪枝、蒸馏等技术的多模态优化。未来，静态量化与动态量化可能会融合发展，为深度学习模型的高效部署提供更灵活多样的解决方案。

SparseGPT算法

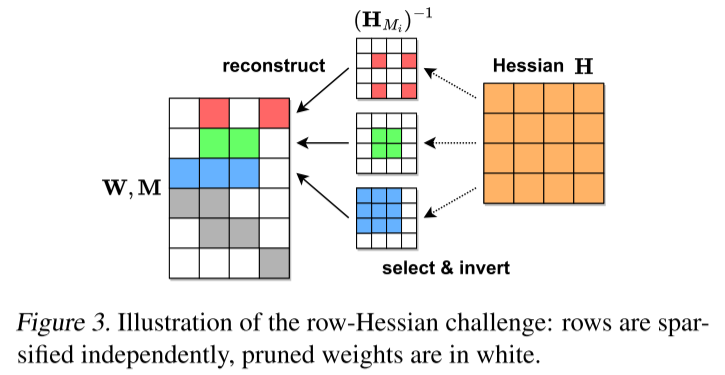
对于固定的[剪枝掩码](https://zhida.zhihu.com/search?content_id=242552777&content_type=Article&match_order=1&q=%E5%89%AA%E6%9E%9D%E6%8E%A9%E7%A0%81&zhida_source=entity)M，权重的最优值可以通过解决稀疏重建矩阵中的每行wi的问题计算出来：



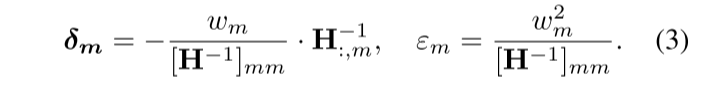
其中，XMi 表示输入特征的子集，其对应的权重在第 i 行未被修剪，wMi 表示它们各自对应的权重。然而这需要我们求逆 Hessian 矩阵HMi=XMiXMi⊤，对应于剪枝掩码 Mi 对第 i 行的保留值，即对 1≤i≤drow 的所有行分别计算 (HMi)−1。这样的一次求逆需要O(dcol3)的时间，所以总复杂度为O(drow·dcol3)。对于Transformer模型，这将正相关于[隐藏维度](https://zhida.zhihu.com/search?content_id=242552777&content_type=Article&match_order=1&q=%E9%9A%90%E8%97%8F%E7%BB%B4%E5%BA%A6&zhida_source=entity" \t "_blank)dhidden的四次方，我们至少需要加速掉一个隐藏维度因子才能实现一个实际有用的算法。

方程2的高计算复杂度主要来自在求解每一行的时候，需要单独对 O(dcol×dcol) 的Hessian矩阵求逆。这是因为行掩码通常是不同的，并且 (HMi)−1≠(H−1)Mi，即掩码Hessian的逆不等于掩码逆Hessian，因此我们每次都要进行求逆操作。如果行掩码都相同，那么只计算一个单独的共享逆即可，因为 H=XX⊤ 只依赖于当前层的输入，这对所有行来说是相同的。

所以，设计一种既准确又高效的[近似算法](https://zhida.zhihu.com/search?content_id=242552777&content_type=Article&match_order=1&q=%E8%BF%91%E4%BC%BC%E7%AE%97%E6%B3%95&zhida_source=entity)的关键在于能够在具有不同修剪掩码的行之间重用Hessians。



假设wm对应掩码确定后的权重，那么可以得到权重补偿和误差：



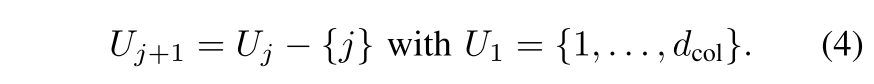
w+δm 就是最优权重重建。进一步，给定掩码M对应的最优稀疏重构w(M)，可以再次应用OBS来寻找掩码 M′=M−{m}的最优重构。因此，这意味着不用求解全掩码 M={m1,...,mp}C，而是迭代地应用OBS逐个将权重 m1 按顺序修剪到 mp，一次一个，将初始完整掩码减少到 M，并最终得到与直接对完整 M 应用封闭形式回归重建相同的最优解，这样就可以避免使用式（2）导致的多次求逆问题。

最优局部更新（OPU）

应用OBS会调整所有可用参数（在当前掩码 M 下）的值来补偿wm的移除。考虑只更新可用参数的一个子集 U∈M，我们仍然可以从[误差补偿](https://zhida.zhihu.com/search?content_id=242552777&content_type=Article&match_order=1&q=%E8%AF%AF%E5%B7%AE%E8%A1%A5%E5%81%BF&zhida_source=entity" \t "_blank)中获益的同时降低应用OBS的成本。这样可以通过简单地使用 HU（对应于 U 的Hessian，而不是HM）计算OBS更新来完成，并且只更新wU。

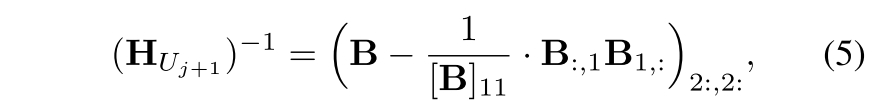
在这种情况下，分层问题的损失对U来说仍然是二次的，因此OBS更新仍然是最优的，只是误差补偿可能没有那么有效，因为可以调整的权重更少。同时，如果 |U|<|M|，那么求逆HU会比 HM快很多。

假设输入特征的固定顺序为 j=1,…,dcol，然后递归地定义一个子集序列：



即从 U1 所有索引的集合开始，通过从前一个子集Uj中删除最小的索引来创建每个子集 Uj+1。

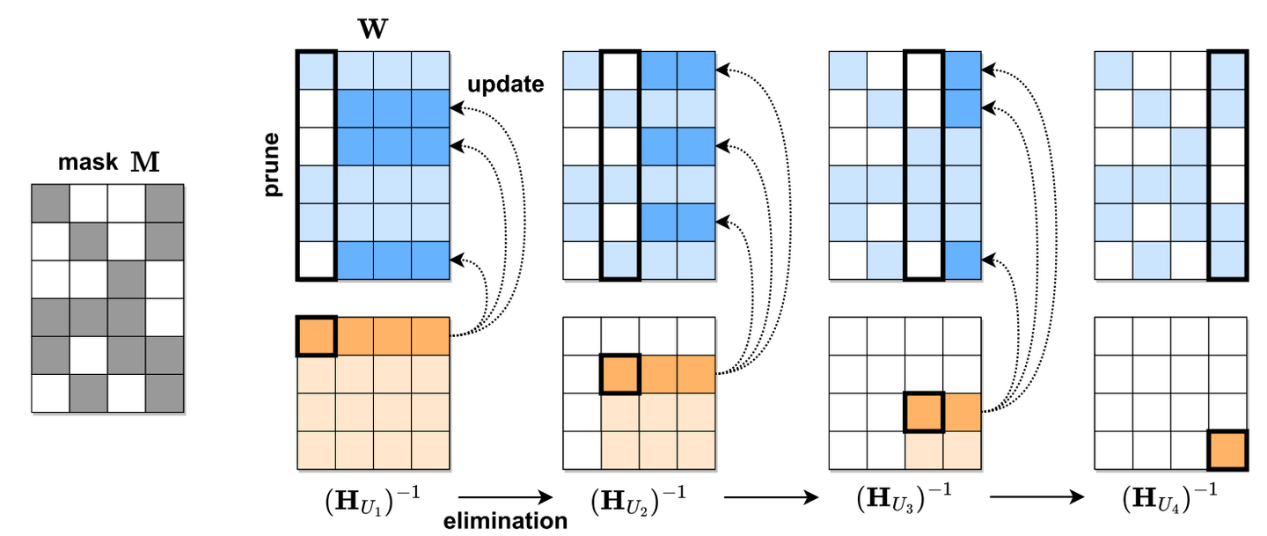
这些子集所使用的海森矩阵的逆 (HUj)−1=((XX⊤)Uj)−1，可以在 W 的各行共享。最重要的是，如OBC一样，可以通过高斯消去在O(dcol2)的时间里，从B=(HUj)−1中移除原始H中对应j的第一行和第一列，从而有效计算更新后的逆Hessian (HUj+1)−1：



而(HU1)−1=H−1，因此，可以在O(dcol3)的时间里递归计算出整个 dcol 序列的逆Hessians，这和在一开始对求逆矩阵H−1的代价相同。

一旦权重wk被修剪，就不应该再更新它了。另外，当修剪wk的时候，希望更新尽可能多的未修剪权值来获得更好的补偿，于是产生以下策略：按顺序迭代Uj和对应的逆Hessian (HUj)−1，并对所有的行i修剪满足的j∉Mi的wj。

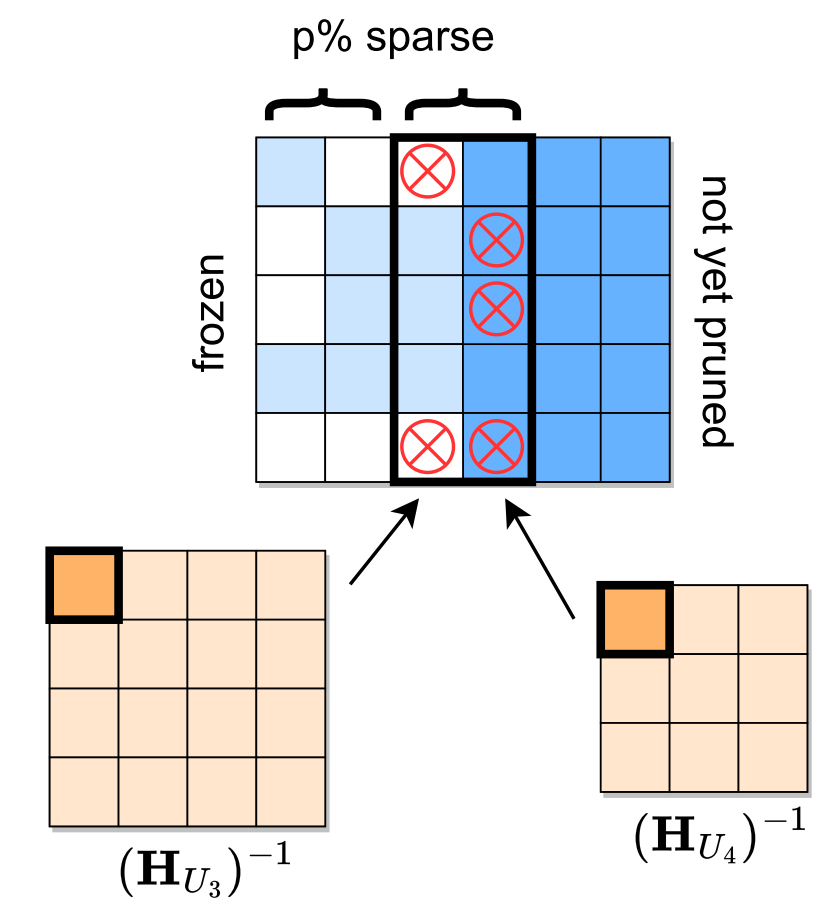
重要的是，每个逆Hessian (HUj)−1 只计算一次，并重复使用以删除所有行中作为剪接掩码一部分的权重j，如图：



上图展示了具体的权重重建的流程：对 Uj 及其对应的海森矩阵的逆 (HUj)−1 按照掩码的列进行有序迭代，如果 j∈Mi，则对该行的 wj 进行修剪（图像的白色），并更新这些行中位于被处理列的“右侧”的其余权重（图中的深蓝色）。由于相关性，修剪期间的更新会显著改变权重，并且在掩码选择中考虑到这一点会产生更好的结果。通过在运行重构时自适应地选择掩码，可以将这种洞察力集成到SparseGPT中。

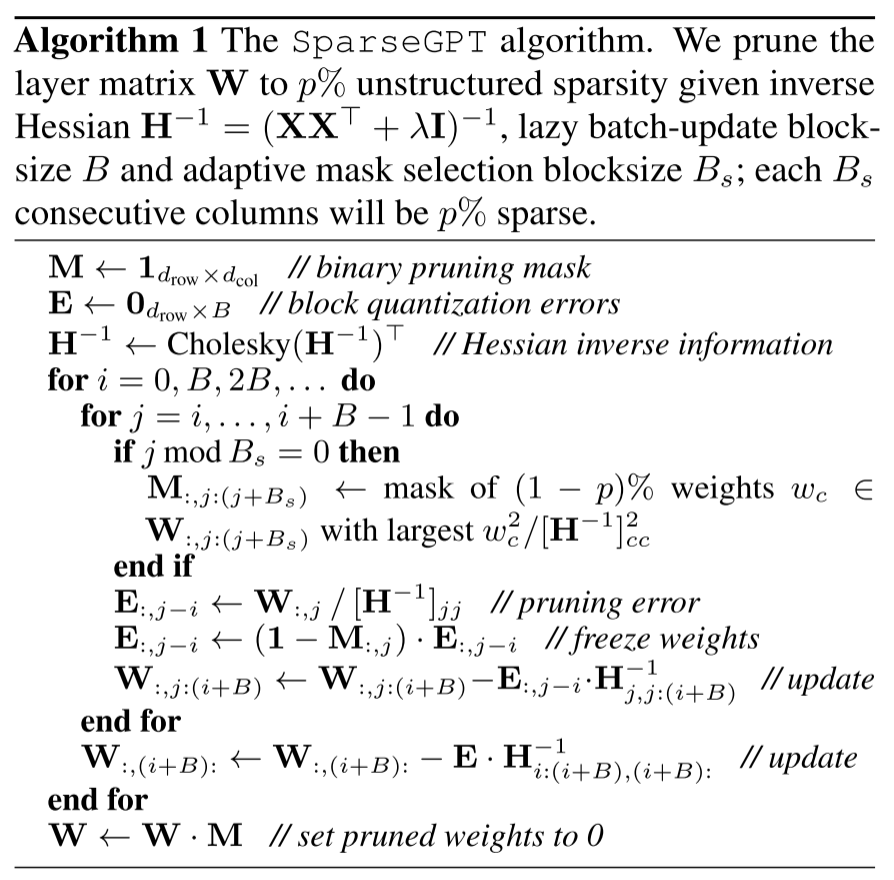
一种简单的方法是在压缩列i时选择p%最无关的权重进行修剪，从而获得 p% 的总体稀疏度。但这种方法的稀疏性不能在列之间非均匀分布，增加了额外的约束，会导致非结构化剪枝的性能下降。对于 GPT 大模型来说，这一问题更为明显，有研究表明 GPT 大模型在早期层中存在大量死亡的ReLU，这导致很多列可以全部被去除，列之间的稀疏性存在极大差异。

为了平衡精度和计算成本，SparseGPT 通过分块迭代求解来实现自适应地掩码选择，下图很好地展示了这一平衡策略——以Bs列为一组进行分块的掩码选择，每块内保持 p% 的稀疏率，确定好掩码后，按照下图的重建算法流程对该块及右侧的列进行权重补偿。



半结构化稀疏的扩展SparseGPT也很容易适应[半结构化](https://zhida.zhihu.com/search?content_id=242552777&content_type=Article&match_order=2&q=%E5%8D%8A%E7%BB%93%E6%9E%84%E5%8C%96&zhida_source=entity" \t "_blank)模式，如流行的 n:m 稀疏性格式。可以简单地选择块大小Bs=m，然后通过公式(3)选择导致最低误差的n个权重，在每行的掩码选择中强制执行零约束。类似的策略也可以应用于其他半结构化修剪模式。

需要注意的是，在这种半结构化场景中，较大的 Bs 是没有用的，0 不能在大小为m的不同[列集](https://zhida.zhihu.com/search?content_id=242552777&content_type=Article&match_order=1&q=%E5%88%97%E9%9B%86&zhida_source=entity" \t "_blank)之间非均匀分布。



项目实现了SparseGPT稀疏化算法搭配GPTQ量化算法对BERTBASE模型进行压缩，并在IMDB数据集上进行测试，得到以下结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 压缩方法 | 稀疏率 | 精度 | 模型大小 | 准确率 |
| 无 | 0% | fp32 | 417.67MB | 91.6% |
| 普通动态量化 | 0% | int8 | 173.1MB | 90.6% |
| GPTQ | 0% | int4 | 89.27MB | 91.4% |
| SparseGPT+GPTQ | 50% | int4 | 89.27MB | 91.1% |
| SparseGPT+GPTQ | 60% | int4 | 89.27MB | 90.9% |
| SparseGPT+GPTQ | 70% | int4 | 89.27MB | 90.8% |

小组分工

陈思达 剪枝量化算法的设计

刘浩洋 剪枝算法的实现

刘同宇 量化算法的实现

王奕然 剪枝量化算法的综合

李琏峻 对模型应用剪枝量化算法

赵宇川 对剪枝量化后的模型进行测试

**参考文献**

[1] 余昉. 深度学习边缘使能及边缘加速关键技术研究[D]. 中国科学院大学,2022.

[2]刘盼雨.面向边缘计算的深度学习模型优化技术研究[D].国防科技大学,2020.DOI:10.27052/d.cnki.gzjgu.2020.000113.

[3]王桥.卷积神经网络的模型轻量化研究与应用[D].西南科技大学,2024.DOI:10.27415/d.cnki.gxngc.2024.001348.

[4] 王宪保, 刘鹏飞, 项圣, 王辛刚. 基于神经架构搜索的非结构化剪枝方法[J]. 模式识别与人工智能, 2023, 36(5): 448-458. WANG Xianbao, LIU Pengfei, XIANG Sheng, WANG Xingang. Unstructured Pruning Method Based on Neural Architecture Search. Pattern Recognition and Artificial Intelligence, 2023, 36(5): 448-458.

**参考文献**

[5] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., & Kalenichenko, D. (2017). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. arXiv preprint arXiv:1712.05877.

[6] Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., & Keutzer, K. (2021). A Survey of Quantization Methods for Efficient Neural Network Inference. arXiv preprint arXiv:2103.13630.

[7] Chen, X., & Zhu, R. (2018). Quantum pumping with adiabatically modulated barriers in three-band pseudospin-1 Dirac-Weyl systems. arXiv preprint arXiv:1812.00344.