

2025.12.04

内容由AI生成·小艺深度研究报告

矩阵乘法：神经网络的计算基石与性能引擎

2025.08.05

内容由AI生成·小艺深度研究报告

美联储降息25基点：全球市场震荡与中国双刃剑效应

## 摘要

矩阵乘法是神经网络计算的基石，它通过将神经元的加权求和抽象为向量点积，并将整个网络层的计算统一为矩阵运算，构成了连接算法设计与硬件执行的关键纽带。这一统一范式不仅驱动了前向传播的端到端计算流水线，还通过在反向传播中高效计算误差梯度，成为模型训练的核心引擎。其应用贯穿于现代深度学习的复杂架构，例如在卷积神经网络（CNN）中，通过im2col算法将卷积操作高效转换为高度优化的矩阵乘法（GEMM）；在Transformer中，则直接构成了自注意力机制的核心计算基础。

矩阵乘法的卓越地位不仅体现在理论层面，更转化为强大的工程优势。它极大地释放了现代GPU等硬件的并行计算潜能，通过高度优化的线性代数库（如cuBLAS）和专用加速器（如NVIDIA Hopper GPU的张量核心），将计算性能推向极致，例如在特定实现中可实现高达139倍的性能加速。同时，向量化的编程范式显著提升了代码的简洁性与健壮性。因此，矩阵乘法既是统一神经网络计算的理论语言，也是驱动其高效运行的工程引擎，其持续优化是推动人工智能发展的核心动力。

目录

[1. 矩阵乘法：神经网络计算的基石 4](#__RefHeading___Toc14313_2302805567)

[1.1 从单个神经元到整个网络层的抽象 4](#__RefHeading___Toc14315_2302805567)

[1.2 前向传播：矩阵乘法驱动的端到端计算流水线 4](#__RefHeading___Toc14317_2302805567)

[1.3 统一计算范式：连接算法与硬件的关键纽带 5](#__RefHeading___Toc14319_2302805567)

[2. 复杂网络架构中的矩阵乘法：从卷积到注意力 6](#__RefHeading___Toc14321_2302805567)

[2.1 卷积神经网络（CNN）：通过im2col实现的GEMM转换 6](#__RefHeading___Toc14323_2302805567)

[2.2 Transformer架构：自注意力机制的核心计算 12](#__RefHeading___Toc14325_2302805567)

[3. 训练之魂：矩阵乘法在反向传播中的核心地位 14](#__RefHeading___Toc14327_2302805567)

[3.1 理论基石：链式法则与梯度流的矩阵表示 16](#__RefHeading___Toc14329_2302805567)

[3.2 核心机制：误差梯度的高效回传与矩阵运算 21](#__RefHeading___Toc14331_2302805567)

[3.3 参数更新：权重梯度计算与外积运算的本质 24](#__RefHeading___Toc14333_2302805567)

[4. 性能引擎：矩阵乘法带来的工程与计算优势 26](#__RefHeading___Toc14335_2302805567)

[4.1 释放并行潜能：批处理与数据并行计算 26](#__RefHeading___Toc14337_2302805567)

[4.2 榨干硬件性能：高度优化的矩阵库与专用加速器 28](#__RefHeading___Toc14339_2302805567)

[4.3 提升开发效能：向量化编程的简洁与健壮 34](#__RefHeading___Toc14341_2302805567)

[5. 优化与前沿：矩阵乘法的加速策略与未来趋势 35](#__RefHeading___Toc14343_2302805567)

[5.1 算法层面的计算优化：从理论到实践的权衡 35](#__RefHeading___Toc14345_2302805567)

[5.2 硬件层面的架构革新：从通用到专用的演进 35](#__RefHeading___Toc14347_2302805567)

[5.3 软件与系统层面的智能调度：从手动调优到自动化 36](#__RefHeading___Toc14349_2302805567)

[5.4 未来展望：超越传统范式的新挑战与新机遇 38](#__RefHeading___Toc14351_2302805567)

[6. 结论：矩阵乘法——神经网络的计算基石与未来引擎 39](#__RefHeading___Toc14353_2302805567)

[6.1 理论基石：统一计算范式的核心地位 39](#__RefHeading___Toc14355_2302805567)

[6.2 工程引擎：性能与开发效率的双重驱动 39](#__RefHeading___Toc14357_2302805567)

[6.3 未来展望：持续演进中的核心引擎与优化路径 39](#__RefHeading___Toc14359_2302805567)

## 1. 矩阵乘法：神经网络计算的基石

在人工智能的宏伟蓝图中，矩阵乘法并非仅仅是一个基础的线性代数运算，而是驱动现代深度学习的核心引擎与理论基石。它如同神经网络的“计算DNA”，将复杂的信息处理过程抽象为一种简洁、高效且可并行的数学语言。从单个神经元的微小决策，到整个模型的宏观推理，矩阵乘法无处不在，构成了连接算法设计与硬件执行的关键纽带。本章旨在深入剖析这一核心计算范式，阐明其如何从微观层面统一神经网络的计算，并最终成为支撑现代AI发展的工程与理论支柱。

### 1.1 从单个神经元到整个网络层的抽象

神经网络的计算始于神经元。一个神经元的核心任务是接收一组输入信号，通过加权求和进行整合，并最终输出一个结果。这个看似简单的过程，在数学上可以被精确地描述为一个向量点积。例如，对于一个神经元，其加权求和的输出 z 可以表示为：

z = w₁x₁ + w₂x₂ + ... + wₙxₙ + b

其中，xᵢ 是输入信号，wᵢ 是对应的权重，而 b 是偏置项。这个公式在数学上等价于输入向量 x 与权重向量 w 的点积，再加上偏置 b。当我们将输入和权重都视为列向量时，这个计算可以被优雅地表示为矩阵乘法的形式：z = wᵀx + b。

这一从标量运算到向量运算的转变，其意义远超数学上的简洁性。它揭示了神经网络计算的本质—— **并行性** 。当多个神经元被组织成一层时，每一个神经元都在执行独立的加权求和操作。如果我们将所有输入信号按样本组织成一个矩阵 X，并将所有神经元的权重按行排列成一个权重矩阵 W，那么整个网络层的计算就可以被高效地归约为一次单一的矩阵乘法操作：Z = WX + B，其中 Z 是输出矩阵，B 是偏置向量。

这种抽象带来了革命性的效率提升。它将数百甚至数千个独立的、串行的计算任务，打包成一个可以被现代计算机硬件大规模并行执行的宏任务。这不仅极大地简化了模型的设计与实现，更重要的是，它为神经网络的计算从理论走向工程实践铺平了道路。

### 1.2 前向传播：矩阵乘法驱动的端到端计算流水线

如果说单个神经元的计算是神经网络的“细胞”，那么前向传播（Forward Propagation）就是由这些“细胞”组成的完整“有机体”的生命流程。在前向传播过程中，输入数据从网络的第一层（输入层）开始，经过层层变换，最终在输出层产生预测结果。这一过程可以被精确地描述为一个由矩阵乘法和非线性激活函数交替构成的计算流水线。

以经典的多层感知机（MLP）为例，其前向传播过程可以被清晰地拆解为以下步骤：

1. **输入处理** ：输入数据 X（一个矩阵，每行代表一个样本）进入第一层。
2. **线性变换** ：第一层的权重矩阵 W₁ 与输入矩阵相乘，得到线性组合结果 Z₁ = W₁X + B₁。
3. **非线性激活** ：为了让网络能够学习到复杂的非线性关系，线性结果 Z₁ 会通过一个非线性激活函数（如ReLU），产生激活值 A₁ = σ(Z₁)。
4. **信息传递** ：激活值 A₁ 作为第二层的输入，重复上述线性变换和激活过程，即 Z₂ = W₂A₁ + B₂ 和 A₂ = σ(Z₂)。
5. **输出生成** ：这一过程持续到最后一层，最终的激活值 Aₗ 即为模型的输出。

在这个流水线中，矩阵乘法扮演着核心的“信息路由器”角色。每一层的输出（激活值矩阵）自动成为下一层的输入，使得整个网络的计算可以被系统性、自动化地描述为一系列矩阵运算。这种端到端的计算模式，不仅保证了计算的高效性，也为后续的反向传播算法提供了清晰、可追溯的计算路径。

### 1.3 统一计算范式：连接算法与硬件的关键纽带

矩阵乘法的真正威力，体现在其作为一种统一计算范式的能力上。它并非全连接层的专属，而是一种能够兼容并高效执行现代深度学习中几乎所有核心计算的通用语言。无论是处理图像的卷积神经网络（CNN），还是处理序列数据的Transformer，其底层的核心运算最终都可以被“翻译”成标准的矩阵乘法。

这种统一性的实现，关键在于算法层面的巧妙转换。以CNN为例，其核心的卷积操作——卷积核对输入特征图进行滑动窗口计算——本质上是一种局部的、不规则的计算模式。然而，通过im2col（Image to Column）等算法，可以将输入特征图中每个滑动窗口的像素值重排成列向量，同时将卷积核参数重排成行向量。这样一来，原本复杂的卷积计算就被转化为一个标准的通用矩阵乘法（GEMM）问题。这一转换使得CNN的计算能够无缝接入为矩阵乘法高度优化的软件库和硬件单元。

由此可见，矩阵乘法扮演了连接高层算法设计与底层硬件执行的“关键纽带”角色。它为不同架构的神经网络提供了一个共同的计算接口，极大地简化了深度学习框架的设计与实现。更重要的是，这种统一的范式使得开发者能够充分利用现代硬件（如GPU和专用AI芯片）的并行计算能力，将复杂模型的训练时间从数月缩短至数小时甚至更短。可以说，没有矩阵乘法这一强大的抽象，现代深度学习的繁荣与发展将无从谈起。

## 2. 复杂网络架构中的矩阵乘法：从卷积到注意力

在第一章确立了矩阵乘法作为神经网络计算基石的宏观地位后，本章将深入具体的复杂网络架构，剖析矩阵乘法如何在不同的计算范式中被特殊化和高效应用[11]。我们将聚焦于两种代表性的现代深度学习模型：卷积神经网络（CNN）和Transformer。前者通过巧妙的数据重排，将其核心的局部特征提取操作映射到标准的矩阵乘法上；后者则在其自注意力机制的设计中，天然地依赖大规模的矩阵运算来实现全局信息整合。通过对这两种架构的深入分析，本章旨在揭示矩阵乘法作为一种统一计算范式的强大生命力，它不仅是连接算法与硬件的关键纽带，更是驱动现代AI模型创新与性能飞跃的核心引擎。

### 2.1 卷积神经网络（CNN）：通过im2col实现的GEMM转换

卷积神经网络的成功，根植于其通过卷积层高效提取空间局部特征的能力。然而，传统的卷积计算——卷积核对输入特征图进行滑动窗口的元素级乘加（MAC）——本质上是一种不规则、局部化的计算模式，这在现代高度并行的硬件上效率低下。为了克服这一挑战，业界发展出了一种核心优化策略：通过im2col（Image-to-Column）算法，将卷积运算统一为标准的通用矩阵乘法（GEMM）问题[4][5][6]。

im2col算法的核心思想是通过数据重排，将空间上的局部计算转化为内存中连续的线性代数运算[6][7][8]。其实现过程可分解为三个关键步骤：

1. **输入特征图变换 (Input Transformation):** 算法首先将输入特征图（通常为三维张量：通道数 × 高度 × 宽度）进行“展开”。对于每个输出特征图的位置，算法提取其对应的输入局部窗口（由卷积核大小决定），并将这些窗口中的所有元素按特定顺序排列成一个列向量。所有这些列向量按输出特征图的空间顺序（从左到右，从上到下）拼接在一起，最终形成一个巨大的二维矩阵[7]。这个矩阵的行数等于单个卷积核的参数数量（输入通道数 × 卷积核高度 × 卷积核宽度），而列数则等于输出特征图的总像素数（输出高度 × 输出宽度）。
2. **卷积核变换 (Kernel Transformation):** 与此同时，算法将三维的卷积核张量（输出通道数 × 输入通道数 × 高度 × 宽度）也进行“展开”[7]。对于每个输出通道，其对应的三维卷积核被展平成一个行向量[4][5]。所有这些行向量按输出通道的顺序拼接，形成另一个二维矩阵。这个矩阵的行数等于输出通道数，列数则与输入特征图变换后矩阵的行数完全相同[5]。
3. **矩阵相乘 (Matrix Multiplication):** 经过上述变换，原本复杂的卷积计算被简化为一次标准的矩阵乘法。将变换后的卷积核矩阵与变换后的输入特征图矩阵相乘，得到的结果矩阵的每个元素，恰好对应输出特征图中对应位置的像素值[4]。最后，只需将结果矩阵重塑为三维的输出特征图张量，即可完成整个卷积操作[7]。

这种转换带来的性能优势是系统性的，源于硬件、软件和数据特性的多重协同作用[5]。首先，它将不规则的内存访问模式转化为规则的、连续的内存块访问，极大地提升了缓存命中率，减少了内存延迟[6]。其次，它使得卷积计算可以无缝接入数十年积累的、针对CPU、GPU乃至专用AI加速器（如TPU）高度优化的线性代数库（如BLAS, cuBLAS）[4]。这些库通过分块（tiling）、低精度量化等技术，将矩阵乘法的性能推向极致。例如，在NVIDIA Hopper GPU上，通过im2col转换的卷积实现，其性能相较于直接卷积可获得高达139倍的加速[1-1-0]。

然而，im2col并非没有代价[7]。数据重排本身会引入额外的计算和内存开销，在某些实现中，这一步骤甚至可能占据整个推理时间的相当一部分（例如，高达28.8%）[1-0-0]。为了缓解这一问题，业界持续对im2col算法进行优化，例如通过消除内层循环中的条件判断、优化带填充（padding）情况的处理等方式，显著提升转换效率[1-0-0]。这表明，im2col是一个持续演进的环节，其目标是在转换开销与GEMM计算效益之间找到最佳平衡点。

{"data": [{"content": "【GiantPandaCV导语】大家好，国庆闲在家里偶然看到了这篇对Im2Col+GEMM实现卷积进行改进的文章，然后去读了读顺便实现了一下，并在本文给出了一个测试结果。从结果来看，这个算法在速度和内存消耗都优于Im2Col+GEMM方式的卷积，想更多的知识和BenchMark可以阅读本文和原始论文。原论文地址：https://arxiv.org/abs/1706.06873v1 。本文复现代码开源在：https://github.com/BBuf/Memory-efficient-Convolution-for-Deep-Neural-Network\n1. 前言\n前面介绍了Im2Col+GEMM来实现卷积以在某些条件下获得更好的访存和计算效率，详见：详解Im2Col+Pack+Sgemm策略更好的优化卷积运算 。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fblog.csdn.net%2Fjust\_sort%2Farticle%2Fdetails%2F108964587", "duration": 0, "header\_picture\_sim": "3", "height": 305, "image\_aesthetics": "47.8", "image\_category": ["图表公式"], "image\_entropy": 2.300942195396486, "image\_file\_size": "63.20", "image\_ocr\_text": "Solution A1005 gemmh-n-w-cn-h-w-c1Solution B15 gemm10-1n-h-w-cn-h-w-cFigure 3. MEC with mini-batch examplehttps://blog.csdn.net/just\_sort", "image\_phash": "ae4ed3a5a593997c", "image\_prefix\_context": "然后下面的Figure3是它的示例图：", "image\_quality": "75.3", "image\_source": "refer\_false", "image\_suffix\_context": "从伪代码里可以看到这里有2种计算方法：", "image\_url": "https://i-blog.csdnimg.cn/blog\_migrate/dfd030bff961d73d0b6e07a818362523.png#pic\_center", "image\_view\_height": 0, "image\_view\_ratio": "", "image\_view\_width": 0, "img-fit": "none", "max\_context\_sim": 0.41663259332298697, "max\_header\_text\_sim": 0.7929960164385025, "name": "", "packageName": "", "prefix\_answer": "`im2col`算法的核心思想是通过数据重排，将空间上的局部计算转化为内存中连续的线性代数运算<rsup>6</rsup><rsup>7</rsup><rsup>8</rsup>。其实现过程可分解为三个关键步骤：", "prefix\_answer\_picture\_sim": "2", "publish\_time": "", "rank\_text\_sim": 0.7929960164385025, "second\_header\_image\_prefix\_context\_sim": 0.1576688258137711, "second\_header\_image\_suffix\_context\_sim": 0.41663259332298697, "second\_header\_text\_sim": 0.4186733977704295, "second\_header\_title": "## 2. 复杂网络架构中的矩阵乘法：从卷积到注意力", "siteLogo": "", "siteName": "CSDN博客", "third\_header\_image\_prefix\_context\_sim": 0.23746766211240525, "third\_header\_image\_suffix\_context\_sim": 0.1410613708921857, "third\_header\_text\_sim": 0.7929960164385025, "third\_header\_title": "### 2.1 卷积神经网络（CNN）：通过im2col实现的GEMM转换", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/7aa77842961b5b08cf7343cdcda9a694.jpg", "thumb\_view": "", "title": "Im2Col+GEMM的改进方法MEC，一种更加高效的卷积计算策略", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/7aa77842961b5b08cf7343cdcda9a694.jpg", "weblink": "https://blog.csdn.net/just\_sort/article/details/108964587", "width": 789}, {"content": "使用 Im2Col 将输入矩阵展开一个大矩阵，矩阵每一列表示卷积核需要的一个输入数据，按行向量方式存储。使用上面转换的矩阵进行 Matmul 运算，得到的数据就是最终卷积计算的结果。一般图像的三通道卷积，其输入为 3 维张量(H, W, 3) ，其中H ,W 为输入图像的高和宽，3 为图像的通道数；卷积核为 4 维张量(N, C, KH, KW) ，其中N 为卷积核的个数,KH ,KW 为卷积核的高和宽，C 为卷积核的通道数，卷积核的通道数应与输入图像的通道数一致；输出为 3 维张量(N, H, W) ,其中H ,W 为输入图像的高和宽，N 为输出图像的通道数，输出图的通道数应与卷积核个数一致。此段中H,W 只是代指，并不表示数值通用，输出图像的宽高具体数值需要按照公式另行计算。在神经网络中，卷积默认采用数据排布方式为NHWC ，意为（样本数，高，宽，通道数）。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fcloud.tencent.com%2Fdeveloper%2Farticle%2F2474315", "duration": 0, "header\_picture\_sim": "3", "height": 1021, "image\_aesthetics": "45.6", "image\_category": ["图表公式"], "image\_entropy": 1.2254417849216819, "image\_file\_size": "158.91", "image\_ocr\_text": "-----2233531123211+0000+1100|0|1000001ZOMI酱0000", "image\_phash": "d8ececc82d032333", "image\_prefix\_context": "根据上述式子可知，每个窗口下对应的卷积计算和两个一维向量的点乘计算是等价的。由此可以推知，在未改变直接卷积计算的参数量和连接数的情况下，这种将每个窗口中的特征子矩阵展开成一维的行向量后再进行矩阵乘计算的思路，能够通过减少计算过程中的访存需求，优化整体的计算时间。", "image\_quality": "78.1", "image\_source": "refer\_false", "image\_suffix\_context": "image", "image\_url": "https://developer.qcloudimg.com/http-save/11307734/51fe848698f3f0d9dab06fc480456a1b.png", "image\_view\_height": 1021, "image\_view\_ratio": "1:1", "image\_view\_width": 1021, "img-fit": "none", "max\_context\_sim": 0.7687717762864382, "max\_header\_text\_sim": 0.6521562849099982, "name": "", "packageName": "", "prefix\_answer": "卷积神经网络的成功，根植于其通过卷积层高效提取空间局部特征的能力。然而，传统的卷积计算——卷积核对输入特征图进行滑动窗口的元素级乘加（MAC）——本质上是一种不规则、局部化的计算模式，这在现代高度并行的硬件上效率低下。为了克服这一挑战，业界发展出了一种核心优化策略：通过`im2col`（Image-to-Column）算法，将卷积运算统一为标准的通用矩阵乘法（GEMM）问题<rsup>4</rsup><rsup>5</rsup><rsup>6</rsup>。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.7687717762864382, "second\_header\_image\_prefix\_context\_sim": 0.7687717762864382, "second\_header\_image\_suffix\_context\_sim": 0.2545167363815923, "second\_header\_text\_sim": 0.4139155340158033, "second\_header\_title": "## 2. 复杂网络架构中的矩阵乘法：从卷积到注意力", "siteLogo": "", "siteName": "腾讯云", "third\_header\_image\_prefix\_context\_sim": 0.3953692845183468, "third\_header\_image\_suffix\_context\_sim": 0.31596097040081056, "third\_header\_text\_sim": 0.6521562849099982, "third\_header\_title": "### 2.1 卷积神经网络（CNN）：通过im2col实现的GEMM转换", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/ff87cee7db69e1b4cd071d89a31f0f85.jpg", "thumb\_view": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/ff87cee7db69e1b4cd071d89a31f0f85\_view.jpg", "title": "【AI系统】Im2Col 算法", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/ff87cee7db69e1b4cd071d89a31f0f85.jpg", "weblink": "https://cloud.tencent.com/developer/article/2474315", "width": 1080}, {"content": "使用 Im2Col 将输入矩阵展开一个大矩阵，矩阵每一列表示卷积核需要的一个输入数据，按行向量方式存储。使用上面转换的矩阵进行 Matmul 运算，得到的数据就是最终卷积计算的结果。一般图像的三通道卷积，其输入为 3 维张量(H, W, 3) ，其中H ,W 为输入图像的高和宽，3 为图像的通道数；卷积核为 4 维张量(N, C, KH, KW) ，其中N 为卷积核的个数,KH ,KW 为卷积核的高和宽，C 为卷积核的通道数，卷积核的通道数应与输入图像的通道数一致；输出为 3 维张量(N, H, W) ,其中H ,W 为输入图像的高和宽，N 为输出图像的通道数，输出图的通道数应与卷积核个数一致。此段中H,W 只是代指，并不表示数值通用，输出图像的宽高具体数值需要按照公式另行计算。在神经网络中，卷积默认采用数据排布方式为NHWC ，意为（样本数，高，宽，通道数）。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fcloud.tencent.com%2Fdeveloper%2Farticle%2F2474315", "duration": 0, "header\_picture\_sim": "3", "height": 774, "image\_aesthetics": "37.8", "image\_category": ["图表公式"], "image\_entropy": 1.7333413458740918, "image\_file\_size": "151.42", "image\_ocr\_text": "003332001000100010010ZOMI酱2213133112121", "image\_phash": "f8fbfc5407910791", "image\_prefix\_context": "2. 使用上面转换的矩阵进行 Matmul 运算，得到的数据就是最终卷积计算的结果。", "image\_quality": "83.7", "image\_source": "refer\_false", "image\_suffix\_context": "卷积过程", "image\_url": "https://developer.qcloudimg.com/http-save/yehe-11307734/8eea8548951f36268219f452e0b21bf4.png", "image\_view\_height": 0, "image\_view\_ratio": "", "image\_view\_width": 0, "img-fit": "none", "max\_context\_sim": 0.677701178127692, "max\_header\_text\_sim": 0.6521562849099982, "name": "", "packageName": "", "prefix\_answer": "卷积神经网络的成功，根植于其通过卷积层高效提取空间局部特征的能力。然而，传统的卷积计算——卷积核对输入特征图进行滑动窗口的元素级乘加（MAC）——本质上是一种不规则、局部化的计算模式，这在现代高度并行的硬件上效率低下。为了克服这一挑战，业界发展出了一种核心优化策略：通过`im2col`（Image-to-Column）算法，将卷积运算统一为标准的通用矩阵乘法（GEMM）问题<rsup>4</rsup><rsup>5</rsup><rsup>6</rsup>。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.677701178127692, "second\_header\_image\_prefix\_context\_sim": 0.677701178127692, "second\_header\_image\_suffix\_context\_sim": 0.5632471548062855, "second\_header\_text\_sim": 0.4139155340158033, "second\_header\_title": "## 2. 复杂网络架构中的矩阵乘法：从卷积到注意力", "siteLogo": "", "siteName": "腾讯云", "third\_header\_image\_prefix\_context\_sim": 0.5476258046565602, "third\_header\_image\_suffix\_context\_sim": 0.4564880956544651, "third\_header\_text\_sim": 0.6521562849099982, "third\_header\_title": "### 2.1 卷积神经网络（CNN）：通过im2col实现的GEMM转换", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/f4aa8b3a98c320726e3048fe70b8a12e.jpg", "thumb\_view": "", "title": "【AI系统】Im2Col 算法", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/f4aa8b3a98c320726e3048fe70b8a12e.jpg", "weblink": "https://cloud.tencent.com/developer/article/2474315", "width": 1080}, {"content": "使用 Im2Col 将输入矩阵展开一个大矩阵，矩阵每一列表示卷积核需要的一个输入数据，按行向量方式存储。使用上面转换的矩阵进行 Matmul 运算，得到的数据就是最终卷积计算的结果。一般图像的三通道卷积，其输入为 3 维张量(H, W, 3) ，其中H ,W 为输入图像的高和宽，3 为图像的通道数；卷积核为 4 维张量(N, C, KH, KW) ，其中N 为卷积核的个数,KH ,KW 为卷积核的高和宽，C 为卷积核的通道数，卷积核的通道数应与输入图像的通道数一致；输出为 3 维张量(N, H, W) ,其中H ,W 为输入图像的高和宽，N 为输出图像的通道数，输出图的通道数应与卷积核个数一致。此段中H,W 只是代指，并不表示数值通用，输出图像的宽高具体数值需要按照公式另行计算。在神经网络中，卷积默认采用数据排布方式为NHWC ，意为（样本数，高，宽，通道数）。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fcloud.tencent.com%2Fdeveloper%2Farticle%2F2474315", "duration": 0, "header\_picture\_sim": "3", "height": 346, "image\_aesthetics": "41.8", "image\_category": ["图表公式"], "image\_entropy": 2.586872269602102, "image\_file\_size": "53.60", "image\_ocr\_text": "ImageKW\*KH\*IC?39ト846ω8672IHIWOH\*O次KW\*KKW\*KH52", "image\_phash": "cfbfb1d8f161ec44", "image\_prefix\_context": "Input 重排 对 Input 进行重排，得到的矩阵见下图右侧，矩阵的行数对应输出 OH\*OW 个数，也就是卷积核在 Input 上的滑动次数；每个行向量里，先排列计算一个输出点所需要输入上第一个通道的 KH\*KW 个数据，根据卷积窗的大小逐行拼接成一段行向量，排完当前通道的数据后，以同样模式再按次序排列之后的通道的数据，直到第 IC 个通道，最终构成前述完整的一个行向量。", "image\_quality": "79.2", "image\_source": "refer\_false", "image\_suffix\_context": "2. 权重数据重排 对权重数据进行重排，将 N 个卷积核展开为权重矩阵的一行，因此共有 N 行，每个行向量上先排列第一个输入通道上 KH∗KW 数据，根据卷积窗的大小逐行拼接成一段行向量，排完当前通道的数据后，以同样模式再依次排列后面的通道数据直到IC。", "image\_url": "https://developer.qcloudimg.com/http-save/yehe-11307734/acbff29969ce52fa3346a9b1987b1953.png", "image\_view\_height": 0, "image\_view\_ratio": "", "image\_view\_width": 0, "img-fit": "none", "max\_context\_sim": 0.6735255539261807, "max\_header\_text\_sim": 0.6521562849099982, "name": "", "packageName": "", "prefix\_answer": "卷积神经网络的成功，根植于其通过卷积层高效提取空间局部特征的能力。然而，传统的卷积计算——卷积核对输入特征图进行滑动窗口的元素级乘加（MAC）——本质上是一种不规则、局部化的计算模式，这在现代高度并行的硬件上效率低下。为了克服这一挑战，业界发展出了一种核心优化策略：通过`im2col`（Image-to-Column）算法，将卷积运算统一为标准的通用矩阵乘法（GEMM）问题<rsup>4</rsup><rsup>5</rsup><rsup>6</rsup>。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.6735255539261807, "second\_header\_image\_prefix\_context\_sim": 0.6351465210637269, "second\_header\_image\_suffix\_context\_sim": 0.6735255539261807, "second\_header\_text\_sim": 0.4139155340158033, "second\_header\_title": "## 2. 复杂网络架构中的矩阵乘法：从卷积到注意力", "siteLogo": "", "siteName": "腾讯云", "third\_header\_image\_prefix\_context\_sim": 0.48891532301830615, "third\_header\_image\_suffix\_context\_sim": 0.44609318079650245, "third\_header\_text\_sim": 0.6521562849099982, "third\_header\_title": "### 2.1 卷积神经网络（CNN）：通过im2col实现的GEMM转换", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/4925aa5966411c71b913163eefaa4482.jpg", "thumb\_view": "", "title": "【AI系统】Im2Col 算法", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/4925aa5966411c71b913163eefaa4482.jpg", "weblink": "https://cloud.tencent.com/developer/article/2474315", "width": 1080}, {"content": "使用 Im2Col 将输入矩阵展开一个大矩阵，矩阵每一列表示卷积核需要的一个输入数据，按行向量方式存储。使用上面转换的矩阵进行 Matmul 运算，得到的数据就是最终卷积计算的结果。一般图像的三通道卷积，其输入为 3 维张量(H, W, 3) ，其中H ,W 为输入图像的高和宽，3 为图像的通道数；卷积核为 4 维张量(N, C, KH, KW) ，其中N 为卷积核的个数,KH ,KW 为卷积核的高和宽，C 为卷积核的通道数，卷积核的通道数应与输入图像的通道数一致；输出为 3 维张量(N, H, W) ,其中H ,W 为输入图像的高和宽，N 为输出图像的通道数，输出图的通道数应与卷积核个数一致。此段中H,W 只是代指，并不表示数值通用，输出图像的宽高具体数值需要按照公式另行计算。在神经网络中，卷积默认采用数据排布方式为NHWC ，意为（样本数，高，宽，通道数）。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fcloud.tencent.com%2Fdeveloper%2Farticle%2F2474315", "duration": 0, "header\_picture\_sim": "3", "height": 516, "image\_aesthetics": "50.7", "image\_category": ["图表公式"], "image\_entropy": 1.7569657300228774, "image\_file\_size": "51.22", "image\_ocr\_text": "Filter1...Filter 2123123456KH789KW\*KH145678963NKW123456789KW\*KHMICFilter NFilter…ZOMI酱2", "image\_phash": "df5f83afacb1e160", "image\_prefix\_context": "2. 权重数据重排 对权重数据进行重排，将 N 个卷积核展开为权重矩阵的一行，因此共有 N 行，每个行向量上先排列第一个输入通道上 KH∗KW 数据，根据卷积窗的大小逐行拼接成一段行向量，排完当前通道的数据后，以同样模式再依次排列后面的通道数据直到IC。", "image\_quality": "77.3", "image\_source": "refer\_false", "image\_suffix\_context": "通过数据重排，完成 Im2Col 的操作之后会得到一个输入矩阵，卷积的 Weights 也可以转换为一个矩阵，卷积的计算就可以转换为两个矩阵相乘的求解，得到最终的卷积计算结果。", "image\_url": "https://developer.qcloudimg.com/http-save/yehe-11307734/28525ceb165d8e4009afc796f2ba1da0.png", "image\_view\_height": 0, "image\_view\_ratio": "", "image\_view\_width": 0, "img-fit": "none", "max\_context\_sim": 0.6735255539261807, "max\_header\_text\_sim": 0.6521562849099982, "name": "", "packageName": "", "prefix\_answer": "卷积神经网络的成功，根植于其通过卷积层高效提取空间局部特征的能力。然而，传统的卷积计算——卷积核对输入特征图进行滑动窗口的元素级乘加（MAC）——本质上是一种不规则、局部化的计算模式，这在现代高度并行的硬件上效率低下。为了克服这一挑战，业界发展出了一种核心优化策略：通过`im2col`（Image-to-Column）算法，将卷积运算统一为标准的通用矩阵乘法（GEMM）问题<rsup>4</rsup><rsup>5</rsup><rsup>6</rsup>。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.6735255539261807, "second\_header\_image\_prefix\_context\_sim": 0.6735255539261807, "second\_header\_image\_suffix\_context\_sim": 0.6225107346601957, "second\_header\_text\_sim": 0.4139155340158033, "second\_header\_title": "## 2. 复杂网络架构中的矩阵乘法：从卷积到注意力", "siteLogo": "", "siteName": "腾讯云", "third\_header\_image\_prefix\_context\_sim": 0.44609318079650245, "third\_header\_image\_suffix\_context\_sim": 0.6248755927476252, "third\_header\_text\_sim": 0.6521562849099982, "third\_header\_title": "### 2.1 卷积神经网络（CNN）：通过im2col实现的GEMM转换", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/c6b29c8baf556117ab413198ba3e7ade.jpg", "thumb\_view": "", "title": "【AI系统】Im2Col 算法", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/c6b29c8baf556117ab413198ba3e7ade.jpg", "weblink": "https://cloud.tencent.com/developer/article/2474315", "width": 1080}], "uiTemplate": "grid"}

### 2.2 Transformer架构：自注意力机制的核心计算

与CNN专注于局部空间特征不同，Transformer架构通过自注意力机制（Self-Attention）实现了对序列中所有位置的全局信息整合，这一特性使其在自然语言处理等领域取得了革命性的成功。自注意力机制的计算核心，正是大规模的矩阵乘法[9][10][12]。

自注意力机制的工作流程可以被清晰地分解为以下几个步骤，其中矩阵乘法扮演着不可或缺的角色：

1. **线性变换生成Q、K、V矩阵：** 对于输入序列中的每个元素（例如一个词向量），模型首先通过三个独立的线性变换（由权重矩阵和偏置项构成），生成对应的查询（Query, Q）、键（Key, K）和值（Value, V）向量[9]。从数学上看，这相当于将输入序列矩阵 X 分别与三个权重矩阵 W\_Q, W\_K, W\_V 相乘，得到 Q = XW\_Q, K = XW\_K, V = XV\_V。这一步骤本质上是矩阵乘法的直接应用。
2. **计算注意力得分：** 注意力得分衡量了序列中不同位置之间的相关性。它通过查询向量与所有键向量的点积来计算。在矩阵形式下，这相当于 Q 矩阵与 K 矩阵的转置相乘，即 Scores = QK^T。这个矩阵乘法的结果是一个得分矩阵，其中每个元素 Scores 代表第i个位置对第j个位置的关注程度。
3. **归一化与加权求和：** 为了确保关注的“焦点”集中，得分矩阵会经过Softmax函数进行归一化，得到注意力权重矩阵 Attention = Softmax(Scores / sqrt(d\_k))，其中 d\_k 是键向量的维度，用于防止点积值过大导致梯度消失。最后，将注意力权重矩阵与值矩阵 V 相乘，得到最终的上下文向量矩阵 Output = Attention \* V。

由此可见，自注意力机制的每一个关键环节都深度依赖于矩阵乘法。然而，这种强大的全局信息整合能力也带来了严峻的性能挑战。自注意力机制的计算复杂度为 O(n²)，其中 n 是序列的长度。这意味着，随着序列长度的增加（例如在处理长文本时），计算量会呈二次方增长，迅速成为模型训练和推理的性能瓶颈。

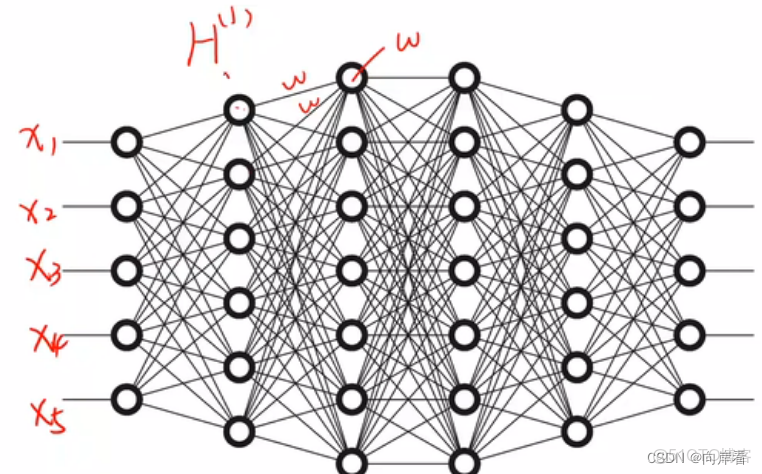
为了应对这一挑战，业界发展出了一系列优化技术。例如，FlashAttention系列算法通过引入分块（tiling）和异步内存访问等策略，将原本内存受限的注意力计算转化为计算受限的问题，从而显著提升了计算效率[10]。这些优化的本质，是在不改变矩阵乘法这一核心计算范式的前提下，通过更智能的调度和内存管理，最大限度地榨取硬件的计算潜能。

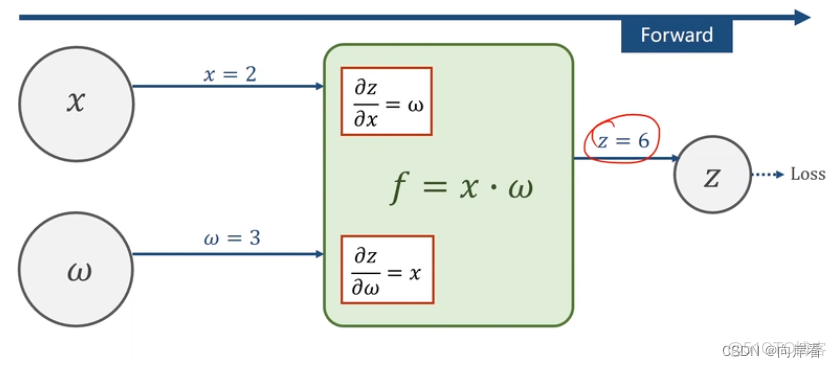
下表总结了矩阵乘法在卷积神经网络（CNN）和Transformer这两种复杂架构中的应用差异：

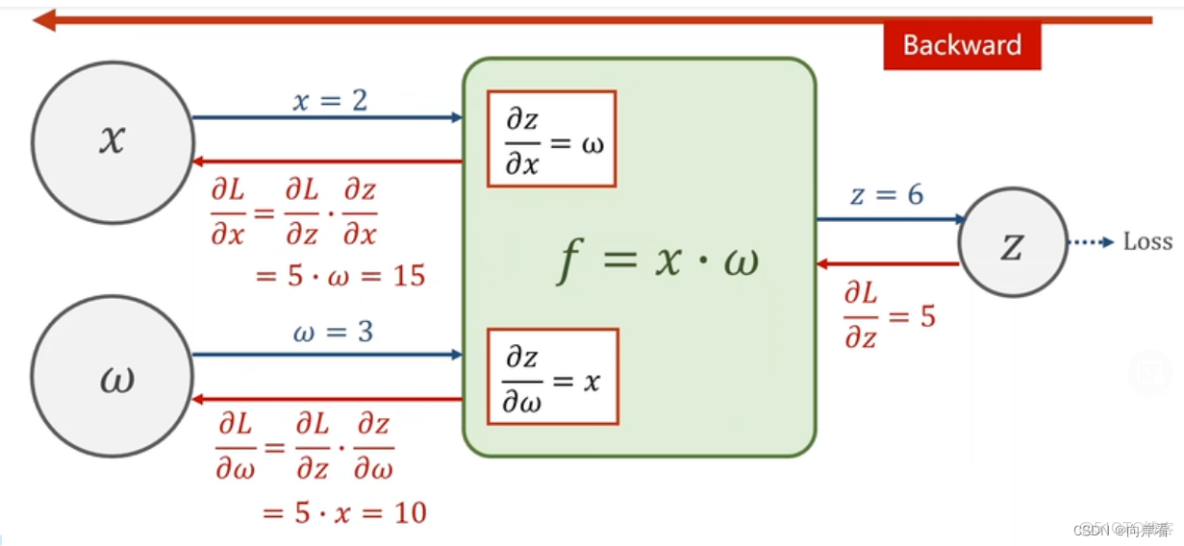
| 对比维度 | 卷积神经网络 (CNN) | Transformer |
| --- | --- | --- |
| **核心计算单元** | 卷积层 (Convolutional Layer) | 自注意力机制 (Self-Attention Mechanism) |
| **矩阵乘法的作用** | 将卷积运算转换为标准的GEMM问题，以利用高度优化的线性代数库[4][5]。 | 计算查询(Query)、键(Key)和值(Value)矩阵之间的乘积，以生成注意力得分和上下文向量。 |
| **数据重排策略** | 使用im2col算法将输入特征图的局部窗口重排成矩阵[4][5]。 | 无数据重排，直接对输入序列进行线性变换生成Q、K、V矩阵。 |
| **计算复杂度** | 通常为O(N)或O(N log N)，取决于具体实现（如GEMM或Winograd）。 | 自注意力机制的计算复杂度为O(n²)，其中n是序列长度。 |
| **性能瓶颈** | 内存访问延迟和计算量，尤其是在高分辨率图像处理中。 | 计算复杂度（O(n²)）随序列长度呈二次方增长，成为性能瓶颈。 |
| **硬件优化方向** | 利用GPU张量核心（Tensor Core）进行高度并行的矩阵乘法。 | 针对注意力机制的特定优化（如FlashAttention），以解决计算和内存瓶颈[10]。 |

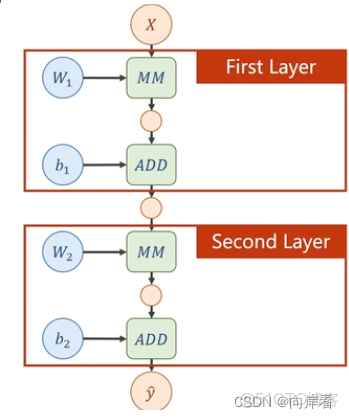
## 3. 训练之魂：矩阵乘法在反向传播中的核心地位

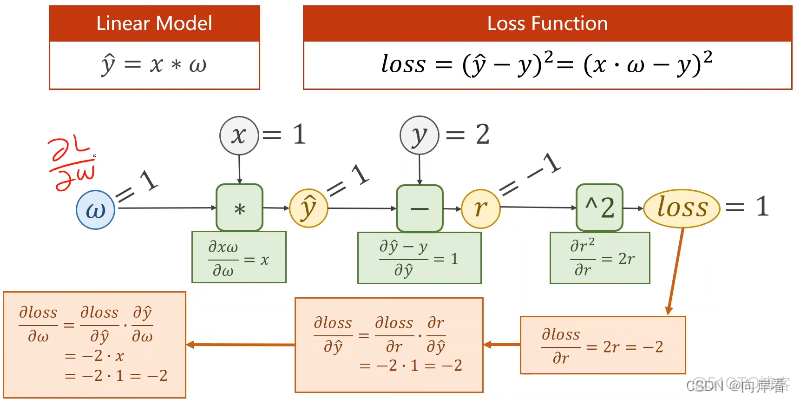
如果说前向传播是神经网络展示其预测能力的“表演”，那么反向传播（Backpropagation, BP）就是其自我反思、不断精进的“灵魂”[15]。在这一过程中，矩阵乘法不再仅仅是一个计算工具，而是成为了驱动整个训练流程的数学引擎。它将复杂的梯度计算转化为一系列标准的线性代数运算，使得神经网络能够高效、准确地更新其内部参数，从而从海量数据中学习到深刻的模式。本章将深入剖析矩阵乘法在反向传播算法中的基础性作用，揭示其如何通过链式法则的矩阵化表达，实现误差梯度的高效回传与参数的精准更新[16][17][24]。











### 3.1 理论基石：链式法则与梯度流的矩阵表示

反向传播算法的本质，是微积分中链式法则（Chain Rule）在由数百万个神经元构成的计算图上的高效应用[18]。其核心目标是计算损失函数（Loss Function）相对于网络中每一个权重和偏置的梯度，即 ∂J/∂W 和 ∂J/∂b，其中 J 代表损失函数，W 和 b 分别代表权重矩阵和偏置向量[13][19][20]。这些梯度是梯度下降算法调整模型参数的“导航图”，指明了参数应该向哪个方向、以多大的步长进行更新，以最小化预测误差[14]。

然而，直接对一个由数十甚至上百层构成的深度网络进行求导，其计算复杂度是灾难性的。反向传播算法通过一种巧妙的“反向”计算策略，将这个复杂问题分解为一系列简单的局部计算[14][15][18]。它从输出层开始，计算损失函数对输出的梯度，然后逐层向后（朝向输入层）传播这些梯度，最终计算出所有参数的梯度[13][18][20]。

在这一过程中，矩阵乘法扮演了至关重要的角色，它将梯度的传播从标量运算提升到了向量和矩阵运算的层面[12]。通过将一个批次（mini-batch）的输入数据组织成一个矩阵 X，并将每一层的激活值也组织成矩阵 A，反向传播的计算就可以被系统性地描述为矩阵之间的运算[24]。这种向量化的表示不仅极大地简化了算法的数学描述，更重要的是，它为利用现代硬件的并行计算能力铺平了道路。可以说，没有矩阵乘法这一统一的数学语言，反向传播算法就无法从一个理论概念演变为驱动现代深度学习的工程实践。

{"data": [{"content": "摘要\n本文是CUDA矩阵乘法系列文章的上篇。\n这个系列会从一个最简单的实现出发，逐步优化到cuBLAS标准库86%的性能，并详细介绍其中涉及到的CUDA性能优化技巧。\n本文首先给出了一个开箱即用的实验源代码，然后介绍了GPU硬件知识以及3种简单实现，逐步展示了把性能从cuBLAS的0.39%优化到16%，即性能提升40倍的“魔法”。<span style=\"color:var(--color-public-accent, #0A59F7)\">矩阵乘法在当今的AI世界扮演着至关重要的角色，神经网络的前向传播，注意力机制的计算等最终都可以使用矩阵乘法来实现，一次大模型的推理背后是数以亿计的矩阵乘法操作。</span>因此，矩阵乘法的执行性能是一个需要重点关注的优化目标。\n目前CUDA平台上已经有很多高效的矩阵乘法的实现，例如cuBLAS，CUTLASS。\n为了探究这些高效实现背后的原理，本文会从一个最简单的矩阵乘法内核出发，通过逐步优化的方式来逐渐逼近cuBLAS的表现。\n本系列文章会分为上下两篇，上篇会介绍一下实验环境，一些本系列会用到的GPU硬件知识，以及3种较为简单的实现；下篇会继续介绍剩下的4种更为复杂的实现。本文实验的源代码已开源到了GitHub，链接：", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fwww.cnblogs.com%2Fqzero233%2Fp%2F19183768", "duration": 0, "header\_picture\_sim": "2", "height": 1354, "image\_aesthetics": "55.7", "image\_category": ["图表公式"], "image\_entropy": 0.33423841262960663, "image\_file\_size": "95.38", "image\_ocr\_text": "WoWI", "image\_phash": "edf4f22b96cb9792", "image\_prefix\_context": "对于矩阵乘法这个操作而言，确实是可行的。我们以C中的某一个元素为例，如下图所示：可以把A的对应行和B的对应列拆分为多个小块W0，W1......，只需把对应的块加载然后相乘之后累加，就能得到正确结果。", "image\_quality": "78.0", "image\_source": "refer\_false", "image\_suffix\_context": "推广到整个Block，我们就可以只加载所需行和列的部分数据到SMEM里，然后在SMEM里完成计算后再继续加载，参考资料作者的图可以很清晰地说明这一点：", "image\_url": "https://img2024.cnblogs.com/blog/3477704/202511/3477704-20251101194759804-70243160.png", "image\_view\_height": 1330, "image\_view\_ratio": "1:1", "image\_view\_width": 1330, "img-fit": "none", "max\_context\_sim": 0.6495645397573081, "max\_header\_text\_sim": 0.6482541832258925, "name": "", "packageName": "", "prefix\_answer": "在这一过程中，矩阵乘法扮演了至关重要的角色，它将梯度的传播从标量运算提升到了向量和矩阵运算的层面[1]。通过将一个批次（mini-batch）的输入数据组织成一个矩阵 `X`，并将每一层的激活值也组织成矩阵 `A`，反向传播的计算就可以被系统性地描述为矩阵之间的运算[2]。这种向量化的表示不仅极大地简化了算法的数学描述，更重要的是，它为利用现代硬件的并行计算能力铺平了道路。可以说，没有矩阵乘法这一统一的数学语言，反向传播算法就无法从一个理论概念演变为驱动现代深度学习的工程实践。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.6495645397573081, "second\_header\_image\_prefix\_context\_sim": 0.6495645397573081, "second\_header\_image\_suffix\_context\_sim": 0.370289748368138, "second\_header\_text\_sim": 0.6482541832258925, "second\_header\_title": "## 3. 训练之魂：矩阵乘法在反向传播中的核心地位", "siteLogo": "", "siteName": "博客园", "third\_header\_image\_prefix\_context\_sim": 0.5280988246746859, "third\_header\_image\_suffix\_context\_sim": 0.3847421260835086, "third\_header\_text\_sim": 0.593186827429537, "third\_header\_title": "### 3.1 理论基石：链式法则与梯度流的矩阵表示", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/022241c769987692415adbc00f3c1dba.jpg", "thumb\_view": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/022241c769987692415adbc00f3c1dba\_view.jpg", "title": "手撕深度学习之CUDA矩阵乘法（上篇）：从朴素实现到40倍性能提升的优化之旅", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/022241c769987692415adbc00f3c1dba.jpg", "weblink": "https://www.cnblogs.com/qzero233/p/19183768", "width": 1330}, {"content": "在前馈神经网络中，各神经元分别属于不同的层。每一层的神经元可以接收前一层神经元的信号，并产生信号输出到下一层。第0层称为输入层，最后一层称为输出层，其他中间层称为隐藏层。整个网络中无反馈，信号从输入层向输出层，可用一个单向传播表示。在FNN中处理二维矩阵输入时，激活函数通常作用于每个元素上。也就是说，对于二维矩阵中的每个元素，都会独立地应用激活函数f。这个过程通常发生在网络的隐藏层，其中每个神经元接收来自前一层的加权输入，然后通过激活函数转换成一个新的矩阵，用作下一层的输入。\n在实际的神经网络中，隐藏层的每个神经元通常会接收来自前一层所有神经元的加权和，然后通过激活函数。这意味着，虽然激活函数是逐元素应用的，但在计算每个隐藏层神经元的输出时，会先进行一个线性变换（即 𝑤⋅𝑥+𝑏，其中 𝑤是权重，𝑏是偏置）。\n二、多层感知机MLP\n多层感知机（Multilayer Perceptron，MLP）是一种特殊的前馈人工神经网络，是前馈神经网络的一个子集。\n所有的多层感知机都是前馈神经网络，但并非所有的前馈神经网络都是多层感知机。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fblog.csdn.net%2Fzhenz0729%2Farticle%2Fdetails%2F138873711", "duration": 0, "header\_picture\_sim": "2", "height": 286, "image\_aesthetics": "46.1", "image\_category": ["图表公式"], "image\_entropy": 2.9079102648708064, "image\_file\_size": "139.29", "image\_ocr\_text": "輸入層隱藏層輸出層VikWkjx1h1Z1WkjZ1살子xiSku.....ZmZm六↑", "image\_phash": "beb191a46699d96e", "image\_prefix\_context": "输出层到隐藏层（权重为wkj）：", "image\_quality": "77.1", "image\_source": "refer\_false", "image\_suffix\_context": "链式求导（chain rule）:", "image\_url": "https://i-blog.csdnimg.cn/blog\_migrate/8561f8bf213a8a9402232e6411d7cd9e.png", "image\_view\_height": 0, "image\_view\_ratio": "", "image\_view\_width": 0, "img-fit": "none", "max\_context\_sim": 0.5969253494479725, "max\_header\_text\_sim": 0.4045665666585273, "name": "", "packageName": "", "prefix\_answer": "反向传播算法的本质，是微积分中链式法则（Chain Rule）在由数百万个神经元构成的计算图上的高效应用[1]。其核心目标是计算损失函数（Loss Function）相对于网络中每一个权重和偏置的梯度，即 `∂J/∂W` 和 `∂J/∂b`，其中 `J` 代表损失函数，`W` 和 `b` 分别代表权重矩阵和偏置向量[1][1][2]。这些梯度是梯度下降算法调整模型参数的“导航图”，指明了参数应该向哪个方向、以多大的步长进行更新，以最小化预测误差[1]。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.5969253494479725, "second\_header\_image\_prefix\_context\_sim": 0.3133227048859126, "second\_header\_image\_suffix\_context\_sim": 0.3937575213929684, "second\_header\_text\_sim": 0.4045665666585273, "second\_header\_title": "## 3. 训练之魂：矩阵乘法在反向传播中的核心地位", "siteLogo": "", "siteName": "CSDN博客", "third\_header\_image\_prefix\_context\_sim": 0.35241359748279694, "third\_header\_image\_suffix\_context\_sim": 0.5969253494479725, "third\_header\_text\_sim": 0.3399672077301023, "third\_header\_title": "### 3.1 理论基石：链式法则与梯度流的矩阵表示", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/6dfb18b5abd4af56a7b0df30877b74fc.jpg", "thumb\_view": "", "title": "CNN学习（4）：前馈神经网络FNN、多层感知机MLP和反向传播推导", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/6dfb18b5abd4af56a7b0df30877b74fc.jpg", "weblink": "https://blog.csdn.net/zhenz0729/article/details/138873711", "width": 759}, {"content": "在前馈神经网络中，各神经元分别属于不同的层。每一层的神经元可以接收前一层神经元的信号，并产生信号输出到下一层。第0层称为输入层，最后一层称为输出层，其他中间层称为隐藏层。整个网络中无反馈，信号从输入层向输出层，可用一个单向传播表示。在FNN中处理二维矩阵输入时，激活函数通常作用于每个元素上。也就是说，对于二维矩阵中的每个元素，都会独立地应用激活函数f。这个过程通常发生在网络的隐藏层，其中每个神经元接收来自前一层的加权输入，然后通过激活函数转换成一个新的矩阵，用作下一层的输入。\n在实际的神经网络中，隐藏层的每个神经元通常会接收来自前一层所有神经元的加权和，然后通过激活函数。这意味着，虽然激活函数是逐元素应用的，但在计算每个隐藏层神经元的输出时，会先进行一个线性变换（即 𝑤⋅𝑥+𝑏，其中 𝑤是权重，𝑏是偏置）。\n二、多层感知机MLP\n多层感知机（Multilayer Perceptron，MLP）是一种特殊的前馈人工神经网络，是前馈神经网络的一个子集。\n所有的多层感知机都是前馈神经网络，但并非所有的前馈神经网络都是多层感知机。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fblog.csdn.net%2Fzhenz0729%2Farticle%2Fdetails%2F138873711", "duration": 0, "header\_picture\_sim": "3", "height": 328, "image\_aesthetics": "47.2", "image\_category": ["图表公式"], "image\_entropy": 1.3218971967339477, "image\_file\_size": "82.25", "image\_ocr\_text": "aE(1)AE(1)可以拆成兩項一和azi)所以awkjazjawkj第一項（aE(1)azja()(-0)2\_/2(2()-)2azjazjzm=∑(°-y/)f段（2/\")j=0", "image\_phash": "bb1fb73a79a59e2f", "image\_prefix\_context": "链式求导（chain rule）:", "image\_quality": "73.5", "image\_source": "refer\_false", "image\_suffix\_context": "隐藏层到输入层", "image\_url": "https://i-blog.csdnimg.cn/blog\_migrate/b9644d422ec85487dbd657bc63ba9b7c.png", "image\_view\_height": 0, "image\_view\_ratio": "", "image\_view\_width": 0, "img-fit": "none", "max\_context\_sim": 0.5969253494479725, "max\_header\_text\_sim": 0.4045665666585273, "name": "", "packageName": "", "prefix\_answer": "反向传播算法的本质，是微积分中链式法则（Chain Rule）在由数百万个神经元构成的计算图上的高效应用[1]。其核心目标是计算损失函数（Loss Function）相对于网络中每一个权重和偏置的梯度，即 `∂J/∂W` 和 `∂J/∂b`，其中 `J` 代表损失函数，`W` 和 `b` 分别代表权重矩阵和偏置向量[1][1][2]。这些梯度是梯度下降算法调整模型参数的“导航图”，指明了参数应该向哪个方向、以多大的步长进行更新，以最小化预测误差[1]。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.5969253494479725, "second\_header\_image\_prefix\_context\_sim": 0.3937575213929684, "second\_header\_image\_suffix\_context\_sim": 0.18534888061672303, "second\_header\_text\_sim": 0.4045665666585273, "second\_header\_title": "## 3. 训练之魂：矩阵乘法在反向传播中的核心地位", "siteLogo": "", "siteName": "CSDN博客", "third\_header\_image\_prefix\_context\_sim": 0.5969253494479725, "third\_header\_image\_suffix\_context\_sim": 0.26296856333318464, "third\_header\_text\_sim": 0.3399672077301023, "third\_header\_title": "### 3.1 理论基石：链式法则与梯度流的矩阵表示", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/8fd2845080c50367e7cb25be2738c4bb.jpg", "thumb\_view": "", "title": "CNN学习（4）：前馈神经网络FNN、多层感知机MLP和反向传播推导", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/8fd2845080c50367e7cb25be2738c4bb.jpg", "weblink": "https://blog.csdn.net/zhenz0729/article/details/138873711", "width": 697}, {"content": "在前馈神经网络中，各神经元分别属于不同的层。每一层的神经元可以接收前一层神经元的信号，并产生信号输出到下一层。第0层称为输入层，最后一层称为输出层，其他中间层称为隐藏层。整个网络中无反馈，信号从输入层向输出层，可用一个单向传播表示。在FNN中处理二维矩阵输入时，激活函数通常作用于每个元素上。也就是说，对于二维矩阵中的每个元素，都会独立地应用激活函数f。这个过程通常发生在网络的隐藏层，其中每个神经元接收来自前一层的加权输入，然后通过激活函数转换成一个新的矩阵，用作下一层的输入。\n在实际的神经网络中，隐藏层的每个神经元通常会接收来自前一层所有神经元的加权和，然后通过激活函数。这意味着，虽然激活函数是逐元素应用的，但在计算每个隐藏层神经元的输出时，会先进行一个线性变换（即 𝑤⋅𝑥+𝑏，其中 𝑤是权重，𝑏是偏置）。\n二、多层感知机MLP\n多层感知机（Multilayer Perceptron，MLP）是一种特殊的前馈人工神经网络，是前馈神经网络的一个子集。\n所有的多层感知机都是前馈神经网络，但并非所有的前馈神经网络都是多层感知机。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fblog.csdn.net%2Fzhenz0729%2Farticle%2Fdetails%2F138873711", "duration": 0, "header\_picture\_sim": "3", "height": 264, "image\_aesthetics": "39.3", "image\_category": ["图表公式"], "image\_entropy": 1.1677323007295286, "image\_file\_size": "49.09", "image\_ocr\_text": "第二項（-):azja∑i=owkihk=hk³awkjawkj所以MaE(4∑の1°-y/)fだ(29)·hxawkjj=0", "image\_phash": "e38d1792956bec68", "image\_prefix\_context": "链式求导（chain rule）:", "image\_quality": "76.4", "image\_source": "refer\_false", "image\_suffix\_context": "隐藏层到输入层", "image\_url": "https://i-blog.csdnimg.cn/blog\_migrate/a031377996c9dbb9da2ddb2df7953c4d.png", "image\_view\_height": 0, "image\_view\_ratio": "", "image\_view\_width": 0, "img-fit": "none", "max\_context\_sim": 0.5969253494479725, "max\_header\_text\_sim": 0.4045665666585273, "name": "", "packageName": "", "prefix\_answer": "反向传播算法的本质，是微积分中链式法则（Chain Rule）在由数百万个神经元构成的计算图上的高效应用[1]。其核心目标是计算损失函数（Loss Function）相对于网络中每一个权重和偏置的梯度，即 `∂J/∂W` 和 `∂J/∂b`，其中 `J` 代表损失函数，`W` 和 `b` 分别代表权重矩阵和偏置向量[1][1][2]。这些梯度是梯度下降算法调整模型参数的“导航图”，指明了参数应该向哪个方向、以多大的步长进行更新，以最小化预测误差[1]。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.5969253494479725, "second\_header\_image\_prefix\_context\_sim": 0.3937575213929684, "second\_header\_image\_suffix\_context\_sim": 0.18534888061672303, "second\_header\_text\_sim": 0.4045665666585273, "second\_header\_title": "## 3. 训练之魂：矩阵乘法在反向传播中的核心地位", "siteLogo": "", "siteName": "CSDN博客", "third\_header\_image\_prefix\_context\_sim": 0.5969253494479725, "third\_header\_image\_suffix\_context\_sim": 0.26296856333318464, "third\_header\_text\_sim": 0.3399672077301023, "third\_header\_title": "### 3.1 理论基石：链式法则与梯度流的矩阵表示", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/47eeebec171c62b378603bda79dac414.jpg", "thumb\_view": "", "title": "CNN学习（4）：前馈神经网络FNN、多层感知机MLP和反向传播推导", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/47eeebec171c62b378603bda79dac414.jpg", "weblink": "https://blog.csdn.net/zhenz0729/article/details/138873711", "width": 582}, {"content": "在前馈神经网络中，各神经元分别属于不同的层。每一层的神经元可以接收前一层神经元的信号，并产生信号输出到下一层。第0层称为输入层，最后一层称为输出层，其他中间层称为隐藏层。整个网络中无反馈，信号从输入层向输出层，可用一个单向传播表示。在FNN中处理二维矩阵输入时，激活函数通常作用于每个元素上。也就是说，对于二维矩阵中的每个元素，都会独立地应用激活函数f。这个过程通常发生在网络的隐藏层，其中每个神经元接收来自前一层的加权输入，然后通过激活函数转换成一个新的矩阵，用作下一层的输入。\n在实际的神经网络中，隐藏层的每个神经元通常会接收来自前一层所有神经元的加权和，然后通过激活函数。这意味着，虽然激活函数是逐元素应用的，但在计算每个隐藏层神经元的输出时，会先进行一个线性变换（即 𝑤⋅𝑥+𝑏，其中 𝑤是权重，𝑏是偏置）。\n二、多层感知机MLP\n多层感知机（Multilayer Perceptron，MLP）是一种特殊的前馈人工神经网络，是前馈神经网络的一个子集。\n所有的多层感知机都是前馈神经网络，但并非所有的前馈神经网络都是多层感知机。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fblog.csdn.net%2Fzhenz0729%2Farticle%2Fdetails%2F138873711", "duration": 0, "header\_picture\_sim": "2", "height": 432, "image\_aesthetics": "37.3", "image\_category": ["图表公式"], "image\_entropy": 1.6286578012369293, "image\_file\_size": "193.85", "image\_ocr\_text": "六↑S1...SkhkSp", "image\_phash": "e11e1ffc7c0007c3", "image\_prefix\_context": "隐藏层到输入层（权重为vik）：", "image\_quality": "76.3", "image\_source": "refer\_false", "image\_suffix\_context": "链式求导（chain rule）:", "image\_url": "https://i-blog.csdnimg.cn/blog\_migrate/1e07a6bd531d21ba83e759e726fc72bf.png", "image\_view\_height": 0, "image\_view\_ratio": "", "image\_view\_width": 0, "img-fit": "none", "max\_context\_sim": 0.5969253494479725, "max\_header\_text\_sim": 0.4045665666585273, "name": "", "packageName": "", "prefix\_answer": "反向传播算法的本质，是微积分中链式法则（Chain Rule）在由数百万个神经元构成的计算图上的高效应用[1]。其核心目标是计算损失函数（Loss Function）相对于网络中每一个权重和偏置的梯度，即 `∂J/∂W` 和 `∂J/∂b`，其中 `J` 代表损失函数，`W` 和 `b` 分别代表权重矩阵和偏置向量[1][1][2]。这些梯度是梯度下降算法调整模型参数的“导航图”，指明了参数应该向哪个方向、以多大的步长进行更新，以最小化预测误差[1]。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.5969253494479725, "second\_header\_image\_prefix\_context\_sim": 0.23199007145554618, "second\_header\_image\_suffix\_context\_sim": 0.3937575213929684, "second\_header\_text\_sim": 0.4045665666585273, "second\_header\_title": "## 3. 训练之魂：矩阵乘法在反向传播中的核心地位", "siteLogo": "", "siteName": "CSDN博客", "third\_header\_image\_prefix\_context\_sim": 0.338309771756714, "third\_header\_image\_suffix\_context\_sim": 0.5969253494479725, "third\_header\_text\_sim": 0.3399672077301023, "third\_header\_title": "### 3.1 理论基石：链式法则与梯度流的矩阵表示", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/3d388428bfb9b43d393ba7ba3183bd83.jpg", "thumb\_view": "", "title": "CNN学习（4）：前馈神经网络FNN、多层感知机MLP和反向传播推导", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/3d388428bfb9b43d393ba7ba3183bd83.jpg", "weblink": "https://blog.csdn.net/zhenz0729/article/details/138873711", "width": 1124}], "uiTemplate": "grid"}

### 3.2 核心机制：误差梯度的高效回传与矩阵运算

反向传播的核心在于“误差梯度”（Error Gradient）的计算与传递[14][16][21]。误差梯度，通常用符号 δ 表示，定义为损失函数对某一层神经元加权输入 z 的偏导数，即 δ^(l) = ∂J/∂z^(l)。这个梯度向量承载了从输出层传递回来的“错误信号”，指示当前层的神经元输出需要如何调整。

误差梯度的计算遵循一个清晰的递推关系，而矩阵乘法及其转置操作是实现这一递推的关键。计算过程分为两步：

1. **输出层误差计算** ：对于输出层，误差梯度 δ^(L) 直接由损失函数的梯度和该层激活函数的导数共同决定[13][15][16]。例如，在均方误差损失下，δ^(L) = (y\_pred - y\_true) ⊙ f'(z^(L)，其中 ⊙ 表示逐元素相乘（Hadamard product），f' 是激活函数的导数。
2. **隐藏层误差回传** ：对于隐藏层，误差梯度的计算则依赖于下一层（更靠近输出层）的误差梯度[15][16][22]。其计算公式为：

* δ^(l) = (W^(l+1))^T \* δ^(l+1) ⊙ f'(z^(l))。

这个公式的物理意义极为深刻。(W^(l+1))^T 是下一层权重矩阵的转置。它的作用是将上层的误差信号 δ^(l+1) 根据连接权重 W^(l+1) 的强度，“反向分配”给当前层的每一个神经元[23]。这正是链式法则的直观体现：当前层的误差不仅取决于自身的激活，更取决于它对下一层误差的贡献。权重越大的连接，其反向传播的误差贡献也越大。通过矩阵乘法，这一复杂的依赖关系被高效地计算出来，使得误差能够以极高的效率从输出层“流回”输入层。

{"data": [{"content": "文章标签：\n#神经网络#深度学习#机器学习\n在计算图中，基于链式法则的计算图反向求导是模型训练的关键算法，下面用图例说明为什么反向传播的时候，权重矩阵需要转置\n按着图中箭头方向“从左向右进行计算”称为正向传播,即从计算图的出发点到结束点的传播, “从右往左计算”称为反向传播.\n另一个例子:\n当反向传播进来的是误差对上一层的导数时，反向传播矩阵使用正向传播矩阵的转置.\n结束！", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fblog.csdn.net%2Ftugouxp%2Farticle%2Fdetails%2F120171884", "duration": 0, "header\_picture\_sim": "2", "height": 442, "image\_aesthetics": "41.3", "image\_category": ["图表公式"], "image\_entropy": 1.096684841413743, "image\_file\_size": "39.61", "image\_ocr\_text": "x1X2y1Y3CSDN @tugouxpbSรS……", "image\_phash": "a455a4aa4b2a4840", "image\_prefix\_context": "另一个例子:", "image\_quality": "69.5", "image\_source": "refer\_false", "image\_suffix\_context": "当反向传播进来的是误差对上一层的导数时，反向传播矩阵使用正向传播矩阵的转置.", "image\_url": "https://i-blog.csdnimg.cn/blog\_migrate/d40a4ef52671dbdac5d8b3fd86d6c035.png", "image\_view\_height": 439, "image\_view\_ratio": "16:9", "image\_view\_width": 781, "img-fit": "none", "max\_context\_sim": 0.7144744988823875, "max\_header\_text\_sim": 0.670024957429752, "name": "", "packageName": "", "prefix\_answer": "这个公式的物理意义极为深刻。`(W^(l+1))^T` 是下一层权重矩阵的转置。它的作用是将上层的误差信号 `δ^(l+1)` 根据连接权重 `W^(l+1)` 的强度，“反向分配”给当前层的每一个神经元[2]。这正是链式法则的直观体现：当前层的误差不仅取决于自身的激活，更取决于它对下一层误差的贡献。权重越大的连接，其反向传播的误差贡献也越大。通过矩阵乘法，这一复杂的依赖关系被高效地计算出来，使得误差能够以极高的效率从输出层“流回”输入层。", "prefix\_answer\_picture\_sim": "2", "publish\_time": "", "rank\_text\_sim": 0.7144744988823875, "second\_header\_image\_prefix\_context\_sim": 0.23626315154000052, "second\_header\_image\_suffix\_context\_sim": 0.7144744988823875, "second\_header\_text\_sim": 0.670024957429752, "second\_header\_title": "## 3. 训练之魂：矩阵乘法在反向传播中的核心地位", "siteLogo": "", "siteName": "CSDN博客", "third\_header\_image\_prefix\_context\_sim": 0.182717056903077, "third\_header\_image\_suffix\_context\_sim": 0.7015495983671342, "third\_header\_text\_sim": 0.654932186474359, "third\_header\_title": "### 3.2 核心机制：误差梯度的高效回传与矩阵运算", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/db386fad6a7125accb6d06121fce6103.jpg", "thumb\_view": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/db386fad6a7125accb6d06121fce6103\_view.jpg", "title": "计算图中反向传播求导权重矩阵需要转置的说明", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/db386fad6a7125accb6d06121fce6103.jpg", "weblink": "https://blog.csdn.net/tugouxp/article/details/120171884", "width": 781}, {"content": "文章标签：\n#神经网络#深度学习#机器学习\n在计算图中，基于链式法则的计算图反向求导是模型训练的关键算法，下面用图例说明为什么反向传播的时候，权重矩阵需要转置\n按着图中箭头方向“从左向右进行计算”称为正向传播,即从计算图的出发点到结束点的传播, “从右往左计算”称为反向传播.\n另一个例子:\n当反向传播进来的是误差对上一层的导数时，反向传播矩阵使用正向传播矩阵的转置.\n结束！", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fblog.csdn.net%2Ftugouxp%2Farticle%2Fdetails%2F120171884", "duration": 0, "header\_picture\_sim": "2", "height": 404, "image\_aesthetics": "42.4", "image\_category": ["图表公式"], "image\_entropy": 1.0891261388602496, "image\_file\_size": "29.29", "image\_ocr\_text": "Y1Y1y2x2y2y2CSDN @tugouxp", "image\_phash": "a05ba4ba495b485b", "image\_prefix\_context": "按着图中箭头方向“从左向右进行计算”称为正向传播,即从计算图的出发点到结束点的传播, “从右往左计算”称为反向传播.", "image\_quality": "71.7", "image\_source": "refer\_false", "image\_suffix\_context": "另一个例子:", "image\_url": "https://i-blog.csdnimg.cn/blog\_migrate/d689d7de685557564685c278420a927b.png", "image\_view\_height": 0, "image\_view\_ratio": "", "image\_view\_width": 0, "img-fit": "none", "max\_context\_sim": 0.51637299679492, "max\_header\_text\_sim": 0.670024957429752, "name": "", "packageName": "", "prefix\_answer": "这个公式的物理意义极为深刻。`(W^(l+1))^T` 是下一层权重矩阵的转置。它的作用是将上层的误差信号 `δ^(l+1)` 根据连接权重 `W^(l+1)` 的强度，“反向分配”给当前层的每一个神经元[2]。这正是链式法则的直观体现：当前层的误差不仅取决于自身的激活，更取决于它对下一层误差的贡献。权重越大的连接，其反向传播的误差贡献也越大。通过矩阵乘法，这一复杂的依赖关系被高效地计算出来，使得误差能够以极高的效率从输出层“流回”输入层。", "prefix\_answer\_picture\_sim": "2", "publish\_time": "", "rank\_text\_sim": 0.670024957429752, "second\_header\_image\_prefix\_context\_sim": 0.51637299679492, "second\_header\_image\_suffix\_context\_sim": 0.23626315154000052, "second\_header\_text\_sim": 0.670024957429752, "second\_header\_title": "## 3. 训练之魂：矩阵乘法在反向传播中的核心地位", "siteLogo": "", "siteName": "CSDN博客", "third\_header\_image\_prefix\_context\_sim": 0.4315678210477977, "third\_header\_image\_suffix\_context\_sim": 0.182717056903077, "third\_header\_text\_sim": 0.654932186474359, "third\_header\_title": "### 3.2 核心机制：误差梯度的高效回传与矩阵运算", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/8417e000fd47a4a46ce7cfc1eb5f82ea.jpg", "thumb\_view": "", "title": "计算图中反向传播求导权重矩阵需要转置的说明", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/8417e000fd47a4a46ce7cfc1eb5f82ea.jpg", "weblink": "https://blog.csdn.net/tugouxp/article/details/120171884", "width": 587}, {"content": "文章标签：\n#机器学习#人工智能\n雅可比矩阵（Jacobian Matrix）详解 | 多变量函数微积分的基石\n在深度学习、计算图、优化算法、机器人控制、流形学习等众多领域中，“雅可比矩阵（Jacobian Matrix）”是一个非常核心的数学工具。\n这篇文章将结合一张视觉化图示，帮助大家从零理解什么是雅可比矩阵、它的数学形式、几何含义以及实际应用场景。\n一、什么是雅可比矩阵？\n雅可比矩阵（Jacobian Matrix）是描述多变量向量函数的一阶偏导数矩阵。\n图示定义（来自 Chris Albon）：\n“当一个函数的输入和输出都是向量的时候，\n包括了所有一阶偏导的矩阵叫做雅可比矩阵。”\n数学定义：\n假设一个向量函数：\n那么它的雅可比矩阵 为：\n二、雅可比矩阵的几何含义\n雅可比矩阵可以被理解为多变量函数在某一点附近的线性近似。\n在一元函数中，我们用导数表示变化率；在多元函数中，雅可比矩阵就像是“变化率”的推广：它描述了输入变量微小变化如何影响输出变量的每一维度。\n举例：\n在二维到二维的函数中，Jacobian 矩阵是一个 2×2 矩阵。这个矩阵可以看作是一个局部线性变换：比如一个向量场的“旋转 + 缩放”。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fblog.csdn.net%2FIT\_ORACLE%2Farticle%2Fdetails%2F148039738", "duration": 0, "header\_picture\_sim": "2", "height": 735, "image\_aesthetics": "35.1", "image\_category": ["图表公式"], "image\_entropy": 3.246915537959675, "image\_file\_size": "60.05", "image\_ocr\_text": "雅可比矩阵Jacobian Matrix当一个函数的输入和输出都是向量的时候，包括了所有一阶偏导的矩阵叫做雅可比矩阵。ChrisAlbon为客…写客答…看哥", "image\_phash": "ac3868391bc09193", "image\_prefix\_context": "雅可比矩阵（Jacobian Matrix）详解 | 多变量函数微积分的基石", "image\_quality": "68.1", "image\_source": "refer\_false", "image\_suffix\_context": "在深度学习、计算图、优化算法、机器人控制、流形学习等众多领域中，“雅可比矩阵（Jacobian Matrix）”是一个非常核心的数学工具。", "image\_url": "https://i-blog.csdnimg.cn/direct/efea89f212654787a58ef673aa3ebbbc.jpeg", "image\_view\_height": 0, "image\_view\_ratio": "", "image\_view\_width": 0, "img-fit": "none", "max\_context\_sim": 0.4884606105945505, "max\_header\_text\_sim": 0.4920243940412712, "name": "", "packageName": "", "prefix\_answer": "反向传播的核心在于“误差梯度”（Error Gradient）的计算与传递[1][1][2]。误差梯度，通常用符号 `δ` 表示，定义为损失函数对某一层神经元加权输入 `z` 的偏导数，即 `δ^(l) = ∂J/∂z^(l)`。这个梯度向量承载了从输出层传递回来的“错误信号”，指示当前层的神经元输出需要如何调整。", "prefix\_answer\_picture\_sim": "2", "publish\_time": "", "rank\_text\_sim": 0.4920243940412712, "second\_header\_image\_prefix\_context\_sim": 0.4141635199774573, "second\_header\_image\_suffix\_context\_sim": 0.4884606105945505, "second\_header\_text\_sim": 0.4920243940412712, "second\_header\_title": "## 3. 训练之魂：矩阵乘法在反向传播中的核心地位", "siteLogo": "", "siteName": "CSDN博客", "third\_header\_image\_prefix\_context\_sim": 0.3956460493231985, "third\_header\_image\_suffix\_context\_sim": 0.433391066737022, "third\_header\_text\_sim": 0.4602476388604866, "third\_header\_title": "### 3.2 核心机制：误差梯度的高效回传与矩阵运算", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/9cf6310e698a74c70536d220ad0a704f.jpg", "thumb\_view": "", "title": "【漫话机器学习系列】266.雅可比矩阵（Jacobian Matrix）", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/9cf6310e698a74c70536d220ad0a704f.jpg", "weblink": "https://blog.csdn.net/IT\_ORACLE/article/details/148039738", "width": 1101}], "uiTemplate": "grid"}

### 3.3 参数更新：权重梯度计算与外积运算的本质

一旦所有层的误差梯度 δ^(l) 都被计算出来，下一步便是更新对应的权重矩阵 W^(l)。权重梯度的计算公式揭示了矩阵乘法的另一项核心作用：

∂J/∂W^(l) = δ^(l) \* (a^(l-1))^T。

这个公式的本质是一个 **外积（Outer Product）** 运算。它将当前层的误差向量 δ^(l) 与上一层的激活值矩阵 a^(l-1) 的转置相乘，得到一个与权重矩阵 W^(l) 形状完全相同的梯度矩阵。

外积运算的意义在于，它精确地量化了每一个权重对最终损失的贡献。梯度矩阵中的每一个元素 ∂J/∂W\_ij^(l)，都代表了第 i 个输入特征与第 j 个神经元之间的权重 W\_ij^(l) 应该如何调整。这个值的大小由两方面共同决定：上一层第 i 个神经元的激活强度 a\_i^(l-1)，以及当前层第 j 个神经元的误差 δ\_j^(l)。只有当输入特征被强烈激活（a\_i^(l-1) 较大）且当前神经元存在显著误差（δ\_j^(l) 较大）时，对应的权重才需要进行大幅度调整。

通过外积运算，反向传播算法将所有神经元的局部误差信息，高效地聚合为一个全局的、可用于更新所有连接权重的矩阵[22]。这不仅保证了参数更新的数学正确性，更使得整个训练过程可以被简洁地描述为一系列矩阵运算，从而能够被高度优化的线性代数库（如cuBLAS）所执行，极大地加速了深度学习模型的训练。

## 4. 性能引擎：矩阵乘法带来的工程与计算优势

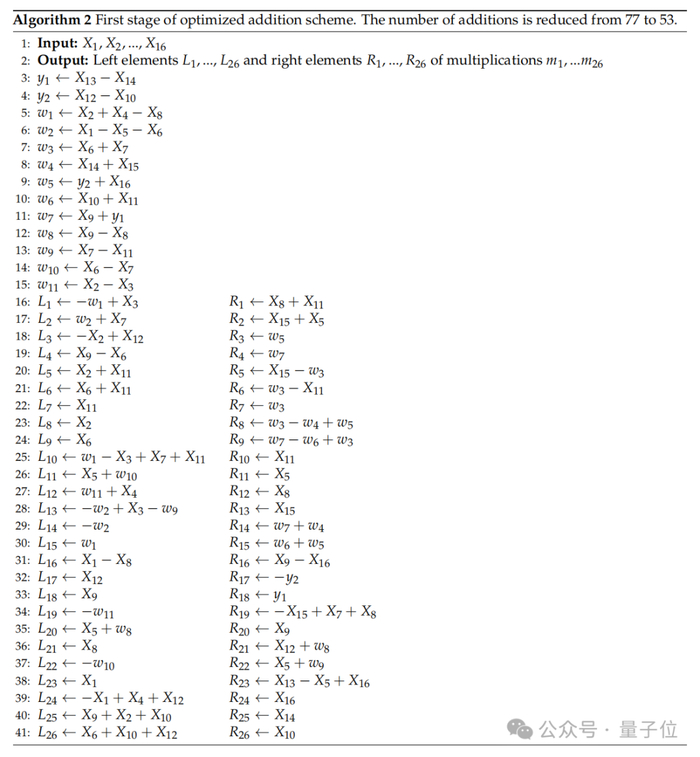
在前序章节中，我们已经深入剖析了矩阵乘法作为神经网络计算基石的理论基础与算法核心。然而，理论的先进性若不能转化为工程实践的效率，其价值将大打折扣。本章将聚焦于矩阵乘法所带来的巨大性能与工程优势，从计算、硬件和软件工程三个维度，系统阐述其为何不仅是理论上的“计算DNA”，更是驱动现代深度学习高效运行的“性能引擎”。我们将论证，矩阵乘法通过释放并行潜能、极致利用现代硬件以及提升开发效能，共同构成了深度学习从理论走向繁荣的关键支柱。

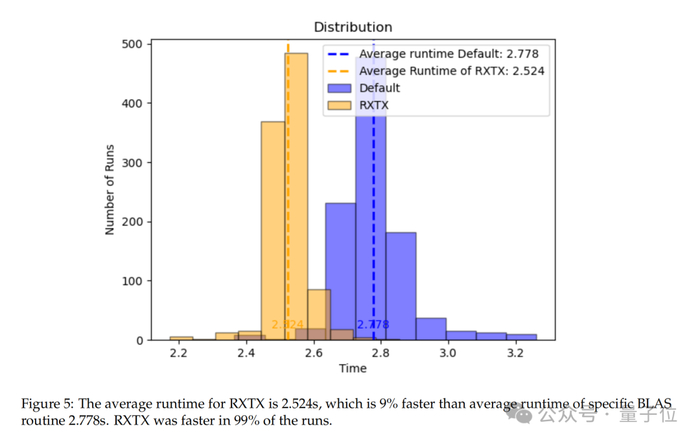
### 4.1 释放并行潜能：批处理与数据并行计算

矩阵乘法最核心的优势之一，在于其天然支持大规模数据并行计算的能力，而这正是批处理（Batch Processing）技术得以高效实现的基石。批处理是现代神经网络训练的标准范式，它将多个样本的计算合并在一起，一次性完成，从而将计算吞吐量推向极致。

这种并行性根植于矩阵乘法的数学本质。当我们将batch\_size个样本的输入数据组织成一个矩阵X ∈ ℝ^(batch\_size × d)时，一次矩阵乘法操作Y = W \* X即可同时计算出所有样本的输出Y。这意味着，计算图中的每一个节点都可以被并行执行，极大地提升了计算效率[30]。这种从算法层面就定义好的并行性，使得矩阵乘法能够完美契合GPU等并行处理器的架构，将其海量的计算核心（如NVIDIA的CUDA Cores）充分利用起来。研究表明，GPU的并行计算能力对矩阵乘法的加速效果极为显著，其性能提升可随矩阵阶数的增加而呈指数级增长，最高可达数百倍以上[36][1-0-0][31][32][37]。

批处理带来的收益远不止于计算速度的提升。在训练阶段，对一个批次的样本计算出的梯度平均值，比单个样本的梯度更能稳定地反映数据的真实分布，这有助于模型更快地收敛到更优的解[14]。因此，矩阵乘法不仅是加速计算的工具，更是稳定训练过程、提升模型性能的关键使能技术[38][39]。





### 4.2 榨干硬件性能：高度优化的矩阵库与专用加速器

矩阵乘法的卓越性能，离不开底层高度优化的软件库与专用硬件的协同工作[37][40][41]。它们共同构成了连接高层算法与底层物理计算的桥梁，将矩阵运算的性能推向极限。

首先，以BLAS（Basic Linear Algebra Subprograms）及其GPU实现cuBLAS为代表的高度优化线性代数库，是性能提升的关键[33][40]。这些库由顶尖专家团队编写，深度利用了处理器的底层特性，如缓存（Cache）层级、内存预取（Prefetching）和单指令多数据流（SIMD）指令集。一项在MATLAB环境下的对比实验清晰地显示，使用标准矩阵乘法（调用BLAS库）的速度比使用手写循环结构快5到8倍[1-0-1]。其根本原因在于，这些库通过向量化编程和精细的内存管理，最大限度地减少了计算瓶颈，确保了每一个CPU或GPU核心都在满负荷运行。

其次，为了进一步突破性能瓶颈，业界推出了一系列专用硬件加速器。以Google的TPU和NVIDIA的Tensor Cores为代表，这些专用单元专门为矩阵乘法进行了优化[34][35][42]。例如，NVIDIA的Tensor Cores通过硬件电路直接加速矩阵乘加运算，在支持混合精度计算（如FP16）的同时，极大地提升了计算密度和能效[34]。在NVIDIA Hopper GPU上，通过im2col算法将卷积转换为矩阵乘法后，其性能相较于直接卷积实现了高达139倍的飞跃[1-1-1]。这充分证明，矩阵乘法已成为驱动现代计算硬件架构演进的核心引擎[39]。

{"data": [{"content": "此外，先前的研究表明，LLM模型采用S8格式的权重和F16格式的输入进行矩阵乘法运算，能够在保持可接受的准确性的同时提高效率。这一技术被称为仅权重量化（weight-only quantization），需要对带有混合输入的矩阵乘法进行高效实现，例如半精度输入与8位整数相乘。因为硬件加速器（包括GPU）支持一组固定的数据类型，因此，混合输入矩阵乘法需要通过软件转换来映射到硬件操作。\n为此，本文重点关注将混合输入的矩阵乘法映射到NVIDIA Ampere架构上。我们提出了解决数据类型转换和布局一致性的软件技术，以有效地将混合输入矩阵乘法映射到硬件支持的数据类型和布局上。结果显示，在软件中进行额外工作的计算开销很小，并且可以实现接近硬件峰值的性能。本文所介绍的软件技术已在开源的NVIDIA/CUTLASS库（github.com/NVIDIA/cutlass/pull/1084）中发布。\n175亿参数的LLM模型在不同数据类型格式下的内存占用。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fgitcode.csdn.net%2F662760ac9ab37021bfb04633.html", "duration": 0, "header\_picture\_sim": "2", "height": 668, "image\_aesthetics": "56.1", "image\_category": ["图表公式"], "image\_entropy": 2.4573672668789754, "image\_file\_size": "43.88", "image\_ocr\_text": "=A\*BKKCMA", "image\_phash": "eec73d387d87f3d3", "image\_prefix\_context": "张量核心的mma运算通过指定两个输入矩阵（如下图所示的A和B）来计算生成结果矩阵C。mma运算本身支持混合精度。混合精度张量核心允许混合输入（A和B）数据类型与结果（C）数据类型。相比之下，混合输入矩阵乘法涉及混合输入数据类型，这在硬件上不受支持，因此需要通过软件实现。", "image\_quality": "76.2", "image\_source": "refer\_false", "image\_suffix\_context": "对M乘K的输入矩阵A和K乘N的输入矩阵B进行的M乘N乘K的张量核心操作，", "image\_url": "https://i-blog.csdnimg.cn/blog\_migrate/afab695e019ab7ea499005a44929f36a.png", "image\_view\_height": 0, "image\_view\_ratio": "", "image\_view\_width": 0, "img-fit": "none", "max\_context\_sim": 0.6030316943733957, "max\_header\_text\_sim": 0.7819536299953126, "name": "", "packageName": "", "prefix\_answer": "其次，为了进一步突破性能瓶颈，业界推出了一系列专用硬件加速器。以Google的TPU和NVIDIA的Tensor Cores为代表，这些专用单元专门为矩阵乘法进行了优化[3][3][4]。例如，NVIDIA的Tensor Cores通过硬件电路直接加速矩阵乘加运算，在支持混合精度计算（如FP16）的同时，极大地提升了计算密度和能效[3]。在NVIDIA Hopper GPU上，通过`im2col`算法将卷积转换为矩阵乘法后，其性能相较于直接卷积实现了高达139倍的飞跃[1-1-1]。这充分证明，矩阵乘法已成为驱动现代计算硬件架构演进的核心引擎[3]。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.7819536299953126, "second\_header\_image\_prefix\_context\_sim": 0.5511195088941505, "second\_header\_image\_suffix\_context\_sim": 0.6030316943733957, "second\_header\_text\_sim": 0.7819536299953126, "second\_header\_title": "## 4. 性能引擎：矩阵乘法带来的工程与计算优势", "siteLogo": "", "siteName": "GitCode 开源社区", "third\_header\_image\_prefix\_context\_sim": 0.4221884419866255, "third\_header\_image\_suffix\_context\_sim": 0.40879689634200955, "third\_header\_text\_sim": 0.6261115951318701, "third\_header\_title": "### 4.2 榨干硬件性能：高度优化的矩阵库与专用加速器", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/05c03a04463da5ac30b43166da5aa3c9.jpg", "thumb\_view": "", "title": "混合输入矩阵乘法的性能优化", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/05c03a04463da5ac30b43166da5aa3c9.jpg", "weblink": "https://gitcode.csdn.net/662760ac9ab37021bfb04633.html", "width": 1039}, {"content": "此外，先前的研究表明，LLM模型采用S8格式的权重和F16格式的输入进行矩阵乘法运算，能够在保持可接受的准确性的同时提高效率。这一技术被称为仅权重量化（weight-only quantization），需要对带有混合输入的矩阵乘法进行高效实现，例如半精度输入与8位整数相乘。因为硬件加速器（包括GPU）支持一组固定的数据类型，因此，混合输入矩阵乘法需要通过软件转换来映射到硬件操作。\n为此，本文重点关注将混合输入的矩阵乘法映射到NVIDIA Ampere架构上。我们提出了解决数据类型转换和布局一致性的软件技术，以有效地将混合输入矩阵乘法映射到硬件支持的数据类型和布局上。结果显示，在软件中进行额外工作的计算开销很小，并且可以实现接近硬件峰值的性能。本文所介绍的软件技术已在开源的NVIDIA/CUTLASS库（github.com/NVIDIA/cutlass/pull/1084）中发布。\n175亿参数的LLM模型在不同数据类型格式下的内存占用。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fgitcode.csdn.net%2F662760ac9ab37021bfb04633.html", "duration": 0, "header\_picture\_sim": "2", "height": 669, "image\_aesthetics": "38.0", "image\_category": ["图表公式"], "image\_entropy": 3.42389393841589, "image\_file\_size": "215.84", "image\_ocr\_text": "(a)Mixed-inputSoftware solutionMixed-precision(LLM application)(Tensor Cores in Hardware)Datatype conversionLayout conformanceMatrix BMatrix BF32=F16\*U8(Weights)(Weights)F32=F16\*F1632 bits32 bitsWarpWarp(32 threads)(32 threads)Warp32 bits32 bits(32 threads)Matrix AMatrix A(User input)(User Input)TOTOT0T0WarpWarpWarpWarp(32 threads)WarpWarp(32 threads)(32 threads)(32 threads)(32 threads)(32 threads)TOTOTTOTOWarpWarpWarpWarp(32 threads)WarpWarp(32 threads)(32 threads)(32 threads)(32 threads)(32 threads)", "image\_phash": "d9f22b2be7e33322", "image\_prefix\_context": "下图展示了一个mma操作，它从寄存器中提取矩阵A和矩阵B，然后在寄存器中生成矩阵C，这个过程分布在一个warp中。其中，线程T0被突出显示，并对其进行了放大，以展示权重矩阵B经过数据类型转换，需要符合布局一致性才能映射到硬件支持的张量核心操作。", "image\_quality": "74.0", "image\_source": "refer\_false", "image\_suffix\_context": "将软件中的混合输入（F32=F16U8）操作映射到硬件中原生支持的warp级张量核心（F32=F16F16）。原图来源：《在NVIDIA A100上开发CUDA核心以充分发挥张量核心的性能极限》。", "image\_url": "https://i-blog.csdnimg.cn/blog\_migrate/7354ecf97e1c6663617edfa8af65a370.png", "image\_view\_height": 607, "image\_view\_ratio": "16:9", "image\_view\_width": 1080, "img-fit": "none", "max\_context\_sim": 0.6114012976480379, "max\_header\_text\_sim": 0.7819536299953126, "name": "", "packageName": "", "prefix\_answer": "其次，为了进一步突破性能瓶颈，业界推出了一系列专用硬件加速器。以Google的TPU和NVIDIA的Tensor Cores为代表，这些专用单元专门为矩阵乘法进行了优化[3][3][4]。例如，NVIDIA的Tensor Cores通过硬件电路直接加速矩阵乘加运算，在支持混合精度计算（如FP16）的同时，极大地提升了计算密度和能效[3]。在NVIDIA Hopper GPU上，通过`im2col`算法将卷积转换为矩阵乘法后，其性能相较于直接卷积实现了高达139倍的飞跃[1-1-1]。这充分证明，矩阵乘法已成为驱动现代计算硬件架构演进的核心引擎[3]。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.7819536299953126, "second\_header\_image\_prefix\_context\_sim": 0.6114012976480379, "second\_header\_image\_suffix\_context\_sim": 0.3646957991650695, "second\_header\_text\_sim": 0.7819536299953126, "second\_header\_title": "## 4. 性能引擎：矩阵乘法带来的工程与计算优势", "siteLogo": "", "siteName": "GitCode 开源社区", "third\_header\_image\_prefix\_context\_sim": 0.5734903660468577, "third\_header\_image\_suffix\_context\_sim": 0.5082709554953593, "third\_header\_text\_sim": 0.6261115951318701, "third\_header\_title": "### 4.2 榨干硬件性能：高度优化的矩阵库与专用加速器", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/be9103eb9d5c00a09a793fadc0218168.jpg", "thumb\_view": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/be9103eb9d5c00a09a793fadc0218168\_view.jpg", "title": "混合输入矩阵乘法的性能优化", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/be9103eb9d5c00a09a793fadc0218168.jpg", "weblink": "https://gitcode.csdn.net/662760ac9ab37021bfb04633.html", "width": 1080}, {"content": "此外，先前的研究表明，LLM模型采用S8格式的权重和F16格式的输入进行矩阵乘法运算，能够在保持可接受的准确性的同时提高效率。这一技术被称为仅权重量化（weight-only quantization），需要对带有混合输入的矩阵乘法进行高效实现，例如半精度输入与8位整数相乘。因为硬件加速器（包括GPU）支持一组固定的数据类型，因此，混合输入矩阵乘法需要通过软件转换来映射到硬件操作。\n为此，本文重点关注将混合输入的矩阵乘法映射到NVIDIA Ampere架构上。我们提出了解决数据类型转换和布局一致性的软件技术，以有效地将混合输入矩阵乘法映射到硬件支持的数据类型和布局上。结果显示，在软件中进行额外工作的计算开销很小，并且可以实现接近硬件峰值的性能。本文所介绍的软件技术已在开源的NVIDIA/CUTLASS库（github.com/NVIDIA/cutlass/pull/1084）中发布。\n175亿参数的LLM模型在不同数据类型格式下的内存占用。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fgitcode.csdn.net%2F662760ac9ab37021bfb04633.html", "duration": 0, "header\_picture\_sim": "3", "height": 637, "image\_aesthetics": "65.7", "image\_category": ["图表公式"], "image\_entropy": 2.7259752390958996, "image\_file\_size": "117.13", "image\_ocr\_text": "Mixed-input Matrix Multiplication Performance on NVIDIA A100 40GB SXM, Clock 1410MHz, CUDA toolkit 12.3Problem Shape (m=3456,n=4096, k=2048)Mixed-input (Upcast A) = Mixed-input (Upcast B) =Mixed-precision250000235907238219238439226187224724210238208974200000200003188121176175150000500000S8\*BF16U8\*BF16S8\*F16U8\*F16BF16\*S8BF16\*U8 F16\*S8F16\*U8BF16\*BF16 F16\*F16ed-inut(UpcastAMixed-precisionMixed-input (Upcast B)", "image\_phash": "fec18137a63fac7c", "image\_prefix\_context": "值得注意的是，相对于最后两个矩阵乘法，前八个需要额外的操作，因为混合精度变体直接针对硬件加速的张量核心操作，无需数据类型转换和布局一致性。即便如此，在混合输入矩阵乘法性能上，我们的方法仅略低于或与混合精度相当。", "image\_quality": "77.2", "image\_source": "refer\_false", "image\_suffix\_context": "在NVIDIA A100 40GB SMX4芯片上，针对一个计算受限的矩阵问题，测试混合输入矩阵乘法的性能，其矩阵大小为m=3456，n=4096，k=2048。", "image\_url": "https://i-blog.csdnimg.cn/blog\_migrate/fad5380f72b8eecabc74ad27b8e460e6.png", "image\_view\_height": 607, "image\_view\_ratio": "16:9", "image\_view\_width": 1080, "img-fit": "none", "max\_context\_sim": 0.6517180722421891, "max\_header\_text\_sim": 0.7819536299953126, "name": "", "packageName": "", "prefix\_answer": "其次，为了进一步突破性能瓶颈，业界推出了一系列专用硬件加速器。以Google的TPU和NVIDIA的Tensor Cores为代表，这些专用单元专门为矩阵乘法进行了优化[3][3][4]。例如，NVIDIA的Tensor Cores通过硬件电路直接加速矩阵乘加运算，在支持混合精度计算（如FP16）的同时，极大地提升了计算密度和能效[3]。在NVIDIA Hopper GPU上，通过`im2col`算法将卷积转换为矩阵乘法后，其性能相较于直接卷积实现了高达139倍的飞跃[1-1-1]。这充分证明，矩阵乘法已成为驱动现代计算硬件架构演进的核心引擎[3]。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.7819536299953126, "second\_header\_image\_prefix\_context\_sim": 0.6428941048538364, "second\_header\_image\_suffix\_context\_sim": 0.6517180722421891, "second\_header\_text\_sim": 0.7819536299953126, "second\_header\_title": "## 4. 性能引擎：矩阵乘法带来的工程与计算优势", "siteLogo": "", "siteName": "GitCode 开源社区", "third\_header\_image\_prefix\_context\_sim": 0.5563379680564244, "third\_header\_image\_suffix\_context\_sim": 0.6358515702935696, "third\_header\_text\_sim": 0.6261115951318701, "third\_header\_title": "### 4.2 榨干硬件性能：高度优化的矩阵库与专用加速器", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/68667158c7af81cdd0987c1688830676.jpg", "thumb\_view": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/68667158c7af81cdd0987c1688830676\_view.jpg", "title": "混合输入矩阵乘法的性能优化", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/68667158c7af81cdd0987c1688830676.jpg", "weblink": "https://gitcode.csdn.net/662760ac9ab37021bfb04633.html", "width": 1080}, {"content": "知识酷Pro 👆\n学显示行业知识\n找小酷！来源：内容编译自谷歌，谢谢。\n关于 TPU 的工作原理、它们如何联网以实现多芯片训练和推理，以及它们如何限制我们最喜欢的算法的性能。虽然这看起来有点枯燥，但对于真正提高模型效率来说，它非常重要。\n<span style=\"color:var(--color-public-accent, #0A59F7)\">TPU 基本上是一个专门用于矩阵乘法的计算核心（称为 TensorCore），连接到一堆快速内存（称为高带宽内存或 HBM）</span>，以下是图表：\n图： TPU 芯片的基本组件。TensorCore 是左侧灰色框，包含矩阵乘法单元 (MXU)、矢量单元 (VPU) 和矢量内存 (VMEM)。\n您可以将 TensorCore 视为一款出色的矩阵乘法机，但它还有一些其他值得注意的功能。TensorCore 有三个关键单元：\nMXU （矩阵乘法单元）是 TensorCore 的核心。对于大多数 TPU 代，它执行一次矩阵bfloat16[8,128] @ bf16[128,128] -> f32[8,128]乘法1每 8 个周期使用一个脉动阵列。在 TPU v5e 上，1.5GHz 时每个 MXU大约为5e13bf16 FLOPs/s。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fwww.eet-china.com%2Fmp%2Fa380283.html", "duration": 0, "header\_picture\_sim": "3", "height": 360, "image\_aesthetics": "53.9", "image\_category": ["图表公式"], "image\_entropy": 3.3990086408969784, "image\_file\_size": "22.58", "image\_ocr\_text": "The Scalar Unit sort of actslike a CPU 'dispatching'instructions to the VPU andMXUHBM stores theSmemweights, activations,optimiser states, newbatch data etcScalar UnitVectorThe VPU performsVector UnitMemoryHigh(VPU)Bandwidthelementwise operations(e.g. activations), loadsdata into the MXU(Vmem)Memory(HBM)Matrix Multiply Unit (MXU)The MXU performsmatrix multiplications-and is therefore ourdriver of chip FLOP/s.Abstract layout of a TPU TensorCore.HBM bandwidth: determineshow fast data goes to andfrom the computational体行业观察elements", "image\_phash": "a131cc87738e8ff2", "image\_prefix\_context": "TPU 基本上是一个专门用于矩阵乘法的计算核心（称为 TensorCore），连接到一堆快速内存（称为高带宽内存或 HBM），以下是图表：", "image\_quality": "72.0", "image\_source": "refer\_false", "image\_suffix\_context": "图： TPU 芯片的基本组件。TensorCore 是左侧灰色框，包含矩阵乘法单元 (MXU)、矢量单元 (VPU) 和矢量内存 (VMEM)。", "image\_url": "https://static.mianbaoban-assets.eet-china.com/xinyu-images/MBXY-CR-7dbe74656562612d8a4c250e14f004d6.png", "image\_view\_height": 0, "image\_view\_ratio": "", "image\_view\_width": 0, "img-fit": "none", "max\_context\_sim": 0.5850338226975953, "max\_header\_text\_sim": 0.478829354764782, "name": "", "packageName": "", "prefix\_answer": "其次，为了进一步突破性能瓶颈，业界推出了一系列专用硬件加速器。以Google的TPU和NVIDIA的Tensor Cores为代表，这些专用单元专门为矩阵乘法进行了优化[3][3][4]。例如，NVIDIA的Tensor Cores通过硬件电路直接加速矩阵乘加运算，在支持混合精度计算（如FP16）的同时，极大地提升了计算密度和能效[3]。在NVIDIA Hopper GPU上，通过`im2col`算法将卷积转换为矩阵乘法后，其性能相较于直接卷积实现了高达139倍的飞跃[1-1-1]。这充分证明，矩阵乘法已成为驱动现代计算硬件架构演进的核心引擎[3]。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.5850338226975953, "second\_header\_image\_prefix\_context\_sim": 0.5192657273935078, "second\_header\_image\_suffix\_context\_sim": 0.370505427242225, "second\_header\_text\_sim": 0.3120180056409598, "second\_header\_title": "## 4. 性能引擎：矩阵乘法带来的工程与计算优势", "siteLogo": "", "siteName": "Eet-china", "third\_header\_image\_prefix\_context\_sim": 0.5850338226975953, "third\_header\_image\_suffix\_context\_sim": 0.4780880216932713, "third\_header\_text\_sim": 0.478829354764782, "third\_header\_title": "### 4.2 榨干硬件性能：高度优化的矩阵库与专用加速器", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/2ac0feafe21b782ad6b6342714035059.jpg", "thumb\_view": "", "title": "谷歌撰文，深度剖析TPU", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/2ac0feafe21b782ad6b6342714035059.jpg", "weblink": "https://www.eet-china.com/mp/a380283.html", "width": 718}, {"content": "知识酷Pro 👆\n学显示行业知识\n找小酷！来源：内容编译自谷歌，谢谢。\n关于 TPU 的工作原理、它们如何联网以实现多芯片训练和推理，以及它们如何限制我们最喜欢的算法的性能。虽然这看起来有点枯燥，但对于真正提高模型效率来说，它非常重要。\n<span style=\"color:var(--color-public-accent, #0A59F7)\">TPU 基本上是一个专门用于矩阵乘法的计算核心（称为 TensorCore），连接到一堆快速内存（称为高带宽内存或 HBM）</span>，以下是图表：\n图： TPU 芯片的基本组件。TensorCore 是左侧灰色框，包含矩阵乘法单元 (MXU)、矢量单元 (VPU) 和矢量内存 (VMEM)。\n您可以将 TensorCore 视为一款出色的矩阵乘法机，但它还有一些其他值得注意的功能。TensorCore 有三个关键单元：\nMXU （矩阵乘法单元）是 TensorCore 的核心。对于大多数 TPU 代，它执行一次矩阵bfloat16[8,128] @ bf16[128,128] -> f32[8,128]乘法1每 8 个周期使用一个脉动阵列。在 TPU v5e 上，1.5GHz 时每个 MXU大约为5e13bf16 FLOPs/s。", "deeplink": "superlink://vassistant?pkg=com.huawei.browser&pop=false&startmode=cct&h5=https%3A%2F%2Fwww.eet-china.com%2Fmp%2Fa380283.html", "duration": 0, "header\_picture\_sim": "3", "height": 718, "image\_aesthetics": "47.9", "image\_category": ["图表公式"], "image\_entropy": 3.903011359997598, "image\_file\_size": "23.71", "image\_ocr\_text": "SmemSmemScalar UnitScalar UnitVectorVectorVectorVectorUnit (VPU)MemoryUnit (VPU)Memory(Vmem)(Vmem)Matrix Multiply Unit (MXU)Matrix Multiply Unit (MXU)High Bandwidth Memory (HBM)公众号．半导体行业观察", "image\_phash": "af0dc7d07a7ad07a", "image\_prefix\_context": "TPU 芯片通常（但并非总是）由两个共享内存的 TPU 核心组成，可以视为一个具有两倍 FLOP 的大型加速器。自 TPU v4（称为“兆核”）以来一直如此。在较旧的 TPU 芯片上，它们具有单独的内存，被视为两个单独的加速器（TPU v3 及更早版本）。像 TPU v5e 这样的推理优化芯片每个芯片只有一个 TPU 核心。", "image\_quality": "83.4", "image\_source": "refer\_false", "image\_suffix\_context": "芯片以 4 个为一组排列在通过 PCIe 网络连接到CPU 主机的“托盘”（tray）上。这是大多数读者熟悉的格式，即通过 Colab 或单个 TPU-VM 公开的 4 个芯片（8 个核心，但通常被视为 4 个逻辑兆核）。对于像 TPU v5e 这样的推理芯片，我们每个主机有 2 个托盘，而不是 1 个，但每个芯片只有 1 个核心，这样我们就有 8 个芯片 = 8 个核心。", "image\_url": "https://static.mianbaoban-assets.eet-china.com/xinyu-images/MBXY-CR-3bc55f30bbc65ad8c9b6985026308c11.png", "image\_view\_height": 0, "image\_view\_ratio": "", "image\_view\_width": 0, "img-fit": "none", "max\_context\_sim": 0.5425850027273904, "max\_header\_text\_sim": 0.478829354764782, "name": "", "packageName": "", "prefix\_answer": "其次，为了进一步突破性能瓶颈，业界推出了一系列专用硬件加速器。以Google的TPU和NVIDIA的Tensor Cores为代表，这些专用单元专门为矩阵乘法进行了优化[3][3][4]。例如，NVIDIA的Tensor Cores通过硬件电路直接加速矩阵乘加运算，在支持混合精度计算（如FP16）的同时，极大地提升了计算密度和能效[3]。在NVIDIA Hopper GPU上，通过`im2col`算法将卷积转换为矩阵乘法后，其性能相较于直接卷积实现了高达139倍的飞跃[1-1-1]。这充分证明，矩阵乘法已成为驱动现代计算硬件架构演进的核心引擎[3]。", "prefix\_answer\_picture\_sim": "3", "publish\_time": "", "rank\_text\_sim": 0.5425850027273904, "second\_header\_image\_prefix\_context\_sim": 0.28149361526696376, "second\_header\_image\_suffix\_context\_sim": 0.3563870697836585, "second\_header\_text\_sim": 0.3120180056409598, "second\_header\_title": "## 4. 性能引擎：矩阵乘法带来的工程与计算优势", "siteLogo": "", "siteName": "Eet-china", "third\_header\_image\_prefix\_context\_sim": 0.48934209079640584, "third\_header\_image\_suffix\_context\_sim": 0.5425850027273904, "third\_header\_text\_sim": 0.478829354764782, "third\_header\_title": "### 4.2 榨干硬件性能：高度优化的矩阵库与专用加速器", "thumb": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/53fb59508cf63af0f00f9f25943d8952.jpg", "thumb\_view": "", "title": "谷歌撰文，深度剖析TPU", "type": "image", "url": "https://search-img-drcn.dbankcdn.com/tmp/upload/image\_content\_ec/53fb59508cf63af0f00f9f25943d8952.jpg", "weblink": "https://www.eet-china.com/mp/a380283.html", "width": 928}], "uiTemplate": "grid"}

### 4.3 提升开发效能：向量化编程的简洁与健壮

除了在计算和硬件层面带来的巨大优势，矩阵乘法在软件工程领域同样扮演着不可或缺的角色。它极大地提升了代码的简洁性、可读性和可维护性，降低了开发复杂模型的门槛。

与标量、循环式的代码相比，向量化的矩阵表达式是一种更高级的“声明式”编程范式。开发者不再需要关心每一个元素的计算细节和循环索引，而是直接描述计算的数学意图。例如，在PyTorch中，实现矩阵转置只需一行代码transposed\_matrix = matrix.transpose(0, 1)，而用纯循环实现则需要数十行代码。这种简洁性不仅使代码更短，更重要的是，它使得算法的逻辑一目了然，极大地降低了出错的概率和后续的维护成本。

向量化代码的健壮性也得到了保障。高度优化的矩阵库经过了长期的测试和验证，其计算结果的正确性和数值稳定性有充分的保证。相比之下，手写的循环结构在处理边界条件、并行同步或特殊数据类型（如稀疏矩阵）时，极易引入细微的错误。因此，采用矩阵运算不仅是技术上的最优选择，更是工程实践中保障代码质量和开发效率的必然路径。

## 5. 优化与前沿：矩阵乘法的加速策略与未来趋势

随着人工智能模型规模的指数级增长，矩阵乘法（GEMM）作为神经网络计算的核心，正面临着前所未有的性能与能效挑战[37]。从Transformer架构中占据70%以上计算量的自注意力机制，到反向传播过程中产生的海量梯度计算，每一次模型迭代都对底层线性代数运算的速度、功耗和内存带宽提出了更为严苛的要求。因此，对矩阵乘法的优化已不再是单纯的技术改进，而是决定下一代AI模型能否落地的关键技术瓶颈[37]。本章将系统性地梳理当前在算法、硬件、软件及特定场景下的优化策略，并探讨其前沿发展趋势，旨在揭示矩阵乘法作为AI核心引擎的持续演进路径。

### 5.1 算法层面的计算优化：从理论到实践的权衡

算法层面的优化旨在通过数学变换，从根本上减少计算量或访存次数。然而，这些优化策略往往伴随着理论与实践的权衡，其应用场景也受到严格限制。

**经典算法的理论突破与现实局限**

Strassen算法是矩阵乘法算法优化的里程碑[45][51][52]。该算法通过分治策略，将一个大矩阵的乘法分解为多个小矩阵的运算，并利用7次乘法和18次加减法的组合来替代传统方法所需的8次乘法，从而将时间复杂度从经典的O(n³)降低至O(n^log₂7) ≈ O(n².807)。这一理论上的巨大飞跃，使其在处理超大规模矩阵时展现出巨大潜力。然而，在深度学习的实践中，Strassen算法的应用却受到诸多限制[52]。首先，其数值稳定性较差，在低精度计算环境下容易导致误差累积，影响模型精度。其次，算法的常数因子较大，且需要额外的内存来存储中间结果，这在处理深度学习中常见的中等规模矩阵时，往往无法抵消其带来的开销。因此，尽管Strassen算法为理论研究提供了重要方向，但其在主流深度学习框架中的应用仍十分有限[52]。

相比之下，Winograd算法在特定场景下的实用性更强。它通过预处理和后处理步骤，将矩阵乘法转化为一系列点积运算，从而显著减少了乘法的次数。这一特性使其在处理小尺寸矩阵（如卷积神经网络中的卷积核）时优势明显。通过将卷积操作转换为GEMM，Winograd算法能够利用高度优化的矩阵乘法库，实现远超直接卷积的性能[6][43][46]。然而，该算法也存在缺点，其复杂的变换过程会引入额外的计算开销，且在某些矩阵尺寸下可能并不适用。

**数据稀疏性驱动的计算范式转变**

随着模型剪枝等技术的普及，稀疏矩阵乘法（SpGEMM）正成为一个日益重要的优化方向。在经过剪枝的模型中，权重矩阵中存在大量的零元素。SpGEMM算法通过仅计算非零元素的乘积和累加，巧妙地跳过了所有与零相关的运算，从而在计算量和内存访问上都实现了巨大的节省。这不仅显著加速了剪枝模型的推理过程，更重要的是，它预示着计算范式的转变——从追求极致的稠密计算，转向利用数据内在稀疏性的智能计算。随着模型规模的不断扩大，如何高效利用稀疏性将成为决定计算效率的关键[37]。

### 5.2 硬件层面的架构革新：从通用到专用的演进

为了应对矩阵乘法日益增长的计算需求，硬件架构正经历一场从通用计算到专用加速的深刻变革[37][53]。

**脉动阵列：为矩阵乘法而生的专用引擎**

谷歌的TPU等专用AI加速器，其核心便是脉动阵列（Systolic Array）架构[35][47][54]。这是一种专为矩阵乘法设计的二维硬件阵列，它通过高效的数据流组织和数据复用，最大化了计算吞吐量。与通用GPU相比，脉动阵列能够以极高的效率执行GEMM操作，其性能优势在处理大规模矩阵时尤为突出[35][43][48]。例如，在VGG16网络的前向计算中，TPU v3的性能比GPU提升了6.79倍[1-3-1]。这种架构上的深度定制，使得脉动阵列成为当前AI计算领域的性能标杆，也推动了硬件与算法的深度融合。

**存内计算：颠覆冯·诺依曼架构的未来挑战**

存内计算（Computing-in-Memory, CIM）则代表了一种更为激进的变革方向。它旨在从根本上解决困扰计算机科学界数十年的“内存墙”问题——即数据在计算单元和存储单元之间搬运的速度远慢于计算速度。CIM架构将计算单元与存储单元集成在一起，使得矩阵乘法等运算可以直接在内存内部完成。这意味着数据无需长途跋涉，从而在物理层面消除了访存瓶颈，有望带来数量级的能效提升。尽管目前CIM技术在精度、可编程性和制造工艺上仍面临严峻挑战，但它为突破传统计算范式的极限提供了可能，是未来AI硬件发展的重要探索方向。

### 5.3 软件与系统层面的智能调度：从手动调优到自动化

随着硬件的日益复杂和算法的多样化，软件层面的优化已从简单的库函数调用，演变为智能化的调度与编译。

**深度学习编译器：软硬件协同的桥梁**

以Apache TVM为代表的深度学习编译器，扮演着连接高层算法与底层硬件的关键角色。它们通过在编译阶段进行算子融合、循环优化和自动调优，为特定的硬件和矩阵形状生成高度优化的执行代码。这种“编译时优化”的模式，使得开发者能够摆脱对特定硬件的依赖，实现“一次编写，处处高效”的愿景。TVM等工具能够处理变维批处理等复杂场景，其生成的代码在CPU和GPU上均能取得数倍于通用库（如MKL, cuBLAS）的性能提升。

**自动算法选择器：AI驱动的性能调优**

面对海量的矩阵尺寸和硬件平台，手动选择最优算法已变得不切实际。因此，机器学习驱动的自动算法选择器应运而生。例如，AlphaTensor项目利用深度强化学习，将矩阵乘法算法的发现过程建模为一个优化问题，成功找到了超越人类专家设计的新算法[44][51][55]。在4×4矩阵乘法中，AlphaTensor发现的算法比传统的Strassen算法更快[1-3-0][31][44][45]。这种“AI驱动AI”的模式，标志着性能调优正从一门艺术转变为一门科学，为应对未来更复杂的计算场景提供了强大的自动化工具。

下表系统性地总结了本章所讨论的核心优化技术：

| 优化维度 | 技术/算法名称 | 核心工作原理 | 主要优势/特点 | 相关性能/数据 |
| --- | --- | --- | --- | --- |
| **算法层面** | Strassen算法 | 基于分治策略，通过将矩阵分解为子矩阵，并利用7次乘法和18次加减法组合来计算结果，而非传统的8次乘法。 | 理论上乘法次数少于传统算法，尤其适用于大规模矩阵。 | 时间复杂度为  。 |
| **算法层面** | Winograd算法 | 一种快速矩阵乘法算法，通过预处理和后处理步骤，将矩阵乘法转化为点积运算，减少了计算量。 | 在特定场景下（如小尺寸矩阵）性能优势显著。 | - |
| **算法层面** | 稀疏矩阵乘法 (SpGEMM) | 针对稀疏矩阵（如模型剪枝后的权重矩阵），仅计算非零元素的乘积和累加，跳过大量零元素的运算。 | 极大减少计算量和内存访问，显著加速剪枝模型的推理。 | - |
| **硬件层面** | 脉动阵列 (Systolic Array) | 一种专用的二维阵列硬件结构，通过数据流的高效组织和复用，最大化计算吞吐量。 | 专为矩阵乘法设计，能高效执行GEMM操作，是现代AI加速器的核心。 | 谷歌TPU等专用芯片广泛采用，性能远超通用GPU。 |
| **硬件层面** | 存内计算 (CIM) | 将计算单元与存储单元集成在一起，在内存内部执行矩阵乘法，从根本上解决“内存墙”问题。 | 消除了数据在计算单元和存储单元之间的长距离搬运，大幅降低能耗和延迟。 | - |
| **软件层面** | 深度学习编译器 (如TVM) | 在编译时为特定硬件和矩阵形状生成高度优化的代码，进行算子融合、循环优化等。 | 实现软硬件协同优化，自动生成高效的执行计划，提升跨平台性能。 | - |
| **软件层面** | 自动算法选择器 (如AlphaTensor) | 利用强化学习等机器学习技术，根据输入矩阵的具体特征（如形状、稀疏性）自动搜索最优算法。 | 动态适应不同场景，发现超越人类专家设计的新算法，实现性能调优自动化。 | - |
| **前沿方向** | 低精度计算 (如FP8) | 利用硬件对低精度浮点数（如8位）的原生支持，加速矩阵乘法并降低内存带宽需求。 | 在保证模型精度的前提下，显著提升计算吞吐量和能效比。 | NVIDIA H100 GPU上，cuBLAS库实现了2.7倍于A100的FP8性能加速[1-2-0]。 |
| **前沿方向** | 混合专家模型 (MoE) 布局 | 针对MoE模型中稀疏激活的特点，设计专用的矩阵乘法布局，避免计算未被激活的专家参数。 | 优化MoE模型的计算效率，降低推理成本。 | DeepGEMM库支持MoE布局，在特定场景下性能超越专家优化内核。 |

### 5.4 未来展望：超越传统范式的新挑战与新机遇

展望未来，矩阵乘法的优化将继续在挑战中寻求突破，驱动整个AI领域向更高层次演进[55]。

**低精度计算的极限与机遇**

Transformer架构的普及正将矩阵乘法推向低精度计算的时代。以NVIDIA H100 GPU为代表的新一代硬件，通过对FP8等更低精度格式的原生支持，在保证模型精度的同时，实现了计算吞吐量和能效比的显著提升[1-2-0][49][50]。然而，低精度计算也带来了新的挑战，如数值溢出、动态范围限制以及对量化误差的敏感性。未来的优化将更加注重混合精度策略和误差补偿技术，以在精度与性能之间找到最佳平衡点。

**模型稀疏性带来的范式转变**

随着模型规模的不断扩大，稀疏性将不再是一种“优化技巧”，而成为一种必然的“计算范式”。无论是通过剪枝还是自然稀疏，模型权重矩阵中的零元素将越来越多。因此，针对各类稀疏模式（如动态稀疏注意力）的SpGEMM算法将成为标准配置。同时，硬件架构也需要向支持稀疏计算的方向演进，例如在脉动阵列中集成稀疏感知的数据流，从而在根本上提升稀疏模型的计算效率。

**新型计算范式的颠覆性变革**

最后，我们必须认识到，矩阵乘法的未来可能并不局限于传统的硅基芯片。光子计算、量子计算等新兴范式正展现出巨大的潜力。例如，基于马赫-曾德尔干涉仪（MZI）阵列的光子矩阵乘法器，理论上可以实现比数字计算低数个数量级的能耗[1-5-0]。尽管这些技术目前仍处于早期探索阶段，面临着精度、稳定性和可编程性等诸多挑战，但它们为最终突破冯·诺依曼架构的物理极限，开启全新的AI计算时代，提供了令人激动的可能性。由此可见，矩阵乘法作为AI的核心引擎，其优化之路将永无止境[38][39]。

## 6. 结论：矩阵乘法——神经网络的计算基石与未来引擎

本报告系统性地剖析了矩阵乘法在神经网络中的核心地位与广泛应用。通过从理论基础到工程实践的多维度分析，我们得出结论：矩阵乘法不仅是驱动现代深度学习的计算基石，更是连接算法设计与硬件执行的关键纽带，其重要性贯穿于神经网络的整个生命周期。

### 6.1 理论基石：统一计算范式的核心地位

矩阵乘法是神经网络计算的理论基石。它通过将单个神经元的加权求和抽象为向量点积，并进一步将整个网络层的计算统一为矩阵运算，为神经网络的前向传播提供了简洁、高效的数学表达。这种统一的计算范式不仅适用于全连接层，还通过算法转换（如im2col）成功将卷积神经网络（CNN）的局部特征提取，以及Transformer模型的全局自注意力机制，都高效地映射到标准的矩阵乘法框架之下。在训练阶段，矩阵乘法更是反向传播算法的数学引擎，它通过链式法则的矩阵化实现，高效地计算并传递误差梯度，从而精准更新模型参数。因此，矩阵乘法构成了现代深度学习理论体系的统一语言。

### 6.2 工程引擎：性能与开发效率的双重驱动

矩阵乘法的卓越理论地位直接转化为强大的工程实践优势。在计算层面，它是释放现代硬件（尤其是GPU和专用AI芯片）并行计算潜能的核心载体，通过批处理和高度优化的矩阵库，极大地提升了计算效率。在硬件层面，矩阵乘法已成为驱动专用计算架构（如脉动阵列）和未来存内计算等前沿技术发展的关键需求。从软件工程角度看，向量化的矩阵运算极大地简化了代码实现，提升了开发效率与代码健壮性。可以说，矩阵乘法是将复杂的AI算法转化为高效、可扩展的工程实践的核心引擎。

### 6.3 未来展望：持续演进中的核心引擎与优化路径

展望未来，随着神经网络模型规模的持续扩大和应用场景的不断深化，矩阵乘法作为核心计算单元的地位将愈发凸显。其优化路径将呈现出算法、硬件、软件协同演进的趋势。算法层面将更加注重利用数据稀疏性等特性进行智能化优化；硬件层面将向存内计算等颠覆性架构探索，以解决“内存墙”瓶颈；软件层面则通过自动化算法选择等技术，实现对复杂硬件的智能调度。因此，持续对矩阵乘法进行深度优化，是推动人工智能技术迈向更高性能、更低能耗新纪元的必然路径。

## 参考来源

[1] 基于Matrix Core的小尺寸批量矩阵乘法设计与优化-华南理工大学 <https://zrb.bjb.scut.edu.cn/CN/abstract/abstract14459.shtml>

[2] 面向GPU计算平台的神经网络卷积性能优化-计算机研究与发展 <https://crad.ict.ac.cn/cn/article/doi/10.7544/issn1000-1239.20200985>

[3] DeepSeek 开源进度 35：深度学习利器 DeepGEMM-IT之家 <https://www.ithome.com/0/833/664.htm>

[4] 卷积的基本操作学习（二）-CSDN博客 <https://blog.csdn.net/hello_liuniuniu/article/details/144869048>

[5] 【AI系统】Im2Col 算法-腾讯云 <https://cloud.tencent.com/developer/article/2474315>

[6] 大模型-img2col与Linear to Conv的比较区别-CSDN博客 <https://blog.csdn.net/jx232515/article/details/148905513>

[7] 【AI系统】Im2Col 算法-CSDN博客 <https://blog.csdn.net/ling_ling_l/article/details/144432985>

[8] Im2Col+GEMM的改进方法MEC，一种更加高效的卷积计算策略-CSDN博客 <https://blog.csdn.net/just_sort/article/details/108964587>

[9] Transformer中的Q，K，V\_transformer-CSDN博客 <https://blog.csdn.net/vivi_cin/article/details/131371433>

[10] FlashAttention3 全解析：速度、精度、显存的再平衡FlashAttention-3 进一步将优化推向极致，-掘金 <https://juejin.cn/post/7523185496702910490>

[11] DeepSeek开源周 Day03：从DeepGEMM看大模型算力提速的矩阵乘法-开发者社区 <https://developer.volcengine.com/articles/7476296708092559410>

[12] 手撕深度学习之CUDA矩阵乘法（上篇）：从朴素实现到40倍性能提升的优化之旅-博客园 <https://www.cnblogs.com/qzero233/p/19183768>

[13] 【深度学习 | 反向传播】释放反向传播的力量: 让训练神经网络变得简单-腾讯云 <https://cloud.tencent.com/developer/article/2345703>

[14] 深度学习基础之反向传播算法-腾讯云 <https://cloud.tencent.com/developer/article/2458698>

[15] 神经网络之反向传播算法（BP）公式推导-CSDN博客 <https://blog.csdn.net/november_chopin/article/details/103900937>

[16] 51. 9-2.反向传播算法-哔哩哔哩 <https://www.bilibili.com/video/BV1iH4bevEEf/?p=51>

[17] python 反向传播代码 pytorch反向传播原理-51CTO博客 <https://blog.51cto.com/u_16099304/8806095>

[18] 反向传播算法-CSDN博客 <https://blog.csdn.net/qq_39697468/article/details/146420435>

[19] CNN学习（4）：前馈神经网络FNN、多层感知机MLP和反向传播推导-CSDN博客 <https://blog.csdn.net/zhenz0729/article/details/138873711>

[20] 神经网络之前向传播与反向传播矩阵手动推导-哔哩哔哩 <https://www.bilibili.com/video/BV1q7411W7pj/>

[21] 【漫话机器学习系列】266.雅可比矩阵（Jacobian Matrix）-CSDN博客 <https://blog.csdn.net/IT_ORACLE/article/details/148039738>

[22] 深度学习中权重更新与优化策略的深入探讨-CSDN博客 <https://blog.csdn.net/weixin_29903713/article/details/147514649>

[23] 计算图中反向传播求导权重矩阵需要转置的说明-CSDN博客 <https://blog.csdn.net/tugouxp/article/details/120171884>

[24] 深度学习中的反向传播：链式法则的矩阵形式与梯度计算证明-腾讯云 <https://cloud.tencent.com/developer/article/2560446>

[25] Multiplier-free In-Memory Vector-Matrix Multiplication Using Distributed Arithmetic-arXiv <https://arxiv.org/pdf/2510.02099>

[26] StrassenNets: Deep learning with a multiplication budget-arXiv <https://arxiv.org/pdf/1712.03942>

[27] 带有AVX-512的Intel KNL处理器上矩阵矩阵乘法的实现-X-MOL学术平台 <https://www.x-mol.com/paper/1335485563578839040/t?recommendPaper=1219518825157906432>

[28] 【MatLab】矩阵for循环与矩阵乘法运算速度对比-CSDN博客 <https://blog.csdn.net/bigFatCat_Tom/article/details/91188717>

[29] 矩阵乘法速度对比：循环VS向量化-CSDN博客 <https://blog.csdn.net/qq_27590277/article/details/88228334>

[30] 《脉动阵列：AI硬件加速的“秘密武器”》-阿里云开发者社区 <https://developer.aliyun.com/article/1650661>

[31] GPU是如何加速矩阵乘法的？-CSDN博客 <https://blog.csdn.net/weixin_49393016/article/details/138362792>

[32] CPU与GPU在大型矩阵计算时的性能对比-CSDN博客 <https://blog.csdn.net/weixin_42470012/article/details/103360458>

[33] cuBLAS-NVIDIA 开发者 <https://developer.nvidia.cn/cublas>

[34] 混合输入矩阵乘法的性能优化-GitCode 开源社区 <https://gitcode.csdn.net/662760ac9ab37021bfb04633.html>

[35] 谷歌云代理商：谷歌云TPUGPU如何加速您的AI模型训练和推理-CSDN博客 <https://blog.csdn.net/CLOUDACE/article/details/149276080>

[36] Convolutional Layers User’s Guide - NVIDIA Docs-NVIDIA Docs <https://docs.nvidia.com/deeplearning/performance/dl-performance-convolutional/index.html>

[37] DeepGEMM：Hopper架构下的矩阵乘法革新-易源AI资讯 | 万维易源-万维易源 <https://www.showapi.com/news/article/67bfd24e4ddd79f11a00c94e>

[38] 矩阵乘法可以算得更快了！港中文10页论文证明：能源、时间均可节省-新浪 <https://news.sina.cn/ai/2025-05-18/detail-inewyfte4063120.d.html>

[39] 《解锁 C++矩阵运算优化秘籍，助力人工智能算法“光速”飞驰》-腾讯云开发者社区-腾讯云-腾讯云 <https://cloud.tencent.com/developer/article/2475576>

[40] 通过 cuBLAS 中的浮点仿真释放 Tensor Core 性能-NVIDIA 开发者 <https://developer.nvidia.cn/blog/unlocking-tensor-core-performance-with-floating-point-emulation-in-cublas/>

[41] 算子优化 | 将GEMM的性能提升200倍AutoKernel算子优化工具正式开源(附源码链接） - 文章 - 开发者 …-火山引擎开发者社区 <https://developer.volcengine.com/articles/7382360837308022793>

[42] 谷歌撰文，深度剖析TPU-Eet-china <https://www.eet-china.com/mp/a380283.html>

[43] 面向GPU计算平台的神经网络卷积性能优化-计算机研究与发展 <https://crad.ict.ac.cn/fileJSJYJYFZ/journal/article/jsjyjyfz/HTML/2022-06-1181.shtml>

[44] DeepMind攻克50年数学难题！AlphaZero史上最快矩阵乘法算法登Nature封面-腾讯云 <https://cloud.tencent.com/developer/article/2135487>

[45] 分治算法之Strassen矩阵乘法详细解读（附带Java代码解读）-CSDN博客 <https://blog.csdn.net/m0_61840987/article/details/142248685>

[46] cuDNN 的 IMPLICIT\_GEMM 算法-CSDN博客 <https://blog.csdn.net/eloudy/article/details/149300311>

[47] 神经网络加速器技术详解-CSDN博客 <https://blog.csdn.net/qq_44648285/article/details/148936707>

[48] 利用GEMM算法优化深度学习模型的高性能计算方法深度学习模型是一种强大的机器学习方法，可以用于解决图像识别、语音识别、自-掘金 <https://juejin.cn/post/7257044686815903801>

[49] 类GPT模型训练提速26.5%，清华朱军等人用INT4算法加速神经网络训练-51CTO <https://www.51cto.com/article/759192.html>

[50] 清华朱军团队新作：使用4位整数训练Transformer，比FP16快2.2倍，提速35.1%，加速AGI到来！-腾讯云 <https://cloud.tencent.com/developer/article/2308682>

[51] 矩阵乘法新突破！XX^T原来可以更快！RL助力搜索，世界纪录又被提升了5%-腾讯网 <https://news.qq.com/rain/a/20250524A03GKM>

[52] 矩阵乘法计算速度再次突破极限，我炼丹能更快了吗？| 哈佛、MIT -量子位 <https://www.qbitai.com/2021/03/22553.html>

[53] 深入探索DeepGEMM：提升FP8矩阵乘法效率的利器-万维易源 <https://www.showapi.com/news/article/67bec70f4ddd79f11a002dca>

[54] TPU 内部架构解析：从芯片设计到超大规模集群-博客园 <https://www.cnblogs.com/wujianming-110117/p/19101965>

[55] 矩阵乘法领域的技术突破：AlphaTensor与AlphaEvolve的革命性进展-易源AI资讯 | 万维易源-万维易源 <https://www.showapi.com/news/article/6833d0fa4ddd79013c03e28b>