

GEMM Convolution

GEMM convolution 的核心说法是对的：

“把卷积中的大量 patch 与 filter 的点积，重新排列成一次矩阵乘法。”

这里需要先说明一个小细节：深度学习框架里常说的 convolution，通常实际计算的是 cross-correlation，也就是不翻转卷积核。下面沿用深度学习里的习惯，仍然叫 convolution。

1. 一个数值例子

这个例子只有一张输入图片、一个输入通道，所以可以先把输入写成一个二维矩阵。对应到完整符号里，就是 $N = 1, H = 4, W = 4, C = 1$ 。

令输入为：

$$I = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \quad (1)$$

先只看一个输出通道，也就是 $K = 1$ 。令 filter 为：

$$F = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \quad (2)$$

左上角输出位置 ($p = 0, q = 0$) 对应的 input patch 是：

$$X = \begin{pmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \end{pmatrix} \quad (3)$$

这个输出点就是 patch 和 filter 的点积：

$$O_{0,0,0,0} = 1 \cdot 1 + 2 \cdot 0 + 3 \cdot (-1) + 5 \cdot 1 + 6 \cdot 0 + 7 \cdot (-1) + 9 \cdot 1 + 10 \cdot 0 + 11 \cdot (-1) = -6 \quad (4)$$

如果用 GEMM 表达，先把这个 patch 展成 A 的第一行：

$$A_{0,:} = [1, 2, 3, 5, 6, 7, 9, 10, 11] \quad (5)$$

把 filter 展成 B 的第一列：

$$B_{:,0} = [1, 0, -1, 1, 0, -1, 1, 0, -1]^T \quad (6)$$

于是：

$$G_{0,0} = A_{0,:} B_{:,0} = -6 \quad (7)$$

第二个输出元素对应 ($p = 0, q = 1$)，也就是窗口向右移动一格：

$$X_1 = \begin{pmatrix} 2 & 3 & 4 \\ 6 & 7 & 8 \\ 10 & 11 & 12 \end{pmatrix} \quad (8)$$

所以：

$$O_{0,0,1,0} = 2 \cdot 1 + 3 \cdot 0 + 4 \cdot (-1) + 6 \cdot 1 + 7 \cdot 0 + 8 \cdot (-1) + 10 \cdot 1 + 11 \cdot 0 + 12 \cdot (-1) = -6 \quad (9)$$

用 GEMM 看，就是 A 的第二行乘以 B ：

$$A_{1,:} = [2, 3, 4, 6, 7, 8, 10, 11, 12] \quad (10)$$

$$G_{1,0} = A_{1,:} B_{:,0} = -6 \quad (11)$$

这个例子中一共有 $2 \times 2 = 4$ 个输出空间位置，所以完整的 A 是：

$$A = \begin{pmatrix} 1 & 2 & 3 & 5 & 6 & 7 & 9 & 10 & 11 \\ 2 & 3 & 4 & 6 & 7 & 8 & 10 & 11 & 12 \\ 5 & 6 & 7 & 9 & 10 & 11 & 13 & 14 & 15 \\ 6 & 7 & 8 & 10 & 11 & 12 & 14 & 15 & 16 \end{pmatrix} \quad (12)$$

完整的 B 是：

$$B = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 1 \\ 0 \\ -1 \\ 1 \\ 0 \\ -1 \end{pmatrix} \quad (13)$$

于是完整的 GEMM 是：

$$G = AB = \begin{pmatrix} 1 & 2 & 3 & 5 & 6 & 7 & 9 & 10 & 11 \\ 2 & 3 & 4 & 6 & 7 & 8 & 10 & 11 & 12 \\ 5 & 6 & 7 & 9 & 10 & 11 & 13 & 14 & 15 \\ 6 & 7 & 8 & 10 & 11 & 12 & 14 & 15 & 16 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ -1 \\ 1 \\ 0 \\ -1 \\ 1 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} -6 \\ -6 \\ -6 \\ -6 \end{pmatrix} \quad (14)$$

最后把 G 的 4 行还原成输出的 (p, q) 空间位置：

$$O_{0,,:,0} = \begin{pmatrix} -6 & -6 \\ -6 & -6 \end{pmatrix} \quad (15)$$

其他输出位置也是同样的点积，只是换成不同的 input patch。如果有多个输出通道 K ，就是让 B 有多列；一次矩阵乘法同时算出多个 filter 的结果。

2. 卷积公式

假设输入采用 NHWC 布局：

$$I \in \mathbb{R}^{N \times H \times W \times C} \quad (16)$$

NHWC 表示输入张量的四个维度顺序是：

- N : batch size, 也就是一次处理多少张图片
- H : input height, 输入特征图高度
- W : input width, 输入特征图宽度
- C : input channels, 输入通道数

所以 $I_{n,h,w,c}$ 表示第 n 个样本、第 h 行、第 w 列、第 c 个输入通道的值。

卷积核采用 RSCK 布局：

$$F \in \mathbb{R}^{R \times S \times C \times K} \quad (17)$$

RSCK 表示 filter 张量的四个维度顺序是：

- R : filter height, 卷积核高度
- S : filter width, 卷积核宽度
- C : input channels, 和输入张量的 C 对齐
- K : output channels, 也就是卷积核个数

所以 $F_{r,s,c,k}$ 表示第 k 个 filter 在位置 (r, s) 、输入通道 c 上的权重。

输出为：

$$O \in \mathbb{R}^{N \times P \times Q \times K} \quad (18)$$

这里可以把输出布局记成 NPQK：

- N : batch size, 和输入的 N 一样
- P : output height, 输出特征图高度
- Q : output width, 输出特征图宽度
- K : output channels, 和 filter 的 K 一样

所以 $O_{n,p,q,k}$ 表示第 n 个样本、第 p 行、第 q 列、第 k 个输出通道的值。

如果 stride 为 u, v , padding 为 a_h, a_w , dilation 为 d_h, d_w , 那么输出空间大小是：

$$P = \left\lfloor \frac{H + 2a_h - d_h(R - 1) - 1}{u} \right\rfloor + 1 \quad (19)$$

$$Q = \left\lfloor \frac{W + 2a_w - d_w(S - 1) - 1}{v} \right\rfloor + 1 \quad (20)$$

每个输出元素是：

$$O_{n,p,q,k} = \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} \sum_{c=0}^{C-1} I_{n,pu+rd_h-a_h,qv+sd_w-a_w,c} F_{r,s,c,k} \quad (21)$$

其中越过输入边界的位置按 padding 规则处理，最常见的是补 0。

这个公式表达的是：固定一个输出位置 (n, p, q, k) 后，把输入中的一个局部 patch 和第 k 个 filter 做点积。

3. 从卷积到矩阵乘法

GEMM 是通用矩阵乘法：

$$C = AB \quad (22)$$

要把卷积改写成 GEMM，可以把输出的前三个维度 (n, p, q) 展平成矩阵行，把卷积核内部的三个维度 (r, s, c) 展平成矩阵的归约维度。

定义：

$$m = (nP + p)Q + q \quad (23)$$

$$\ell = (rS + s)C + c \quad (24)$$

于是逻辑矩阵 A 、 B 、 G 的 shape 是：

$$A \in \mathbb{R}^{(NPQ) \times (RSC)} \quad (25)$$

$$B \in \mathbb{R}^{(RSC) \times K} \quad (26)$$

$$G \in \mathbb{R}^{(NPQ) \times K} \quad (27)$$

它们的元素对应关系是：

$$A_{m,\ell} = I_{n,pu+rd_h-a_h,qv+sd_w-a_w,c} \quad (28)$$

$$B_{\ell,k} = F_{r,s,c,k} \quad (29)$$

$$G_{m,k} = O_{n,p,q,k} \quad (30)$$

把这些代入矩阵乘法：

$$G_{m,k} = \sum_{\ell=0}^{RSC-1} A_{m,\ell} B_{\ell,k} \quad (31)$$

再把 ℓ 还原成 (r, s, c) ，就得到前面的卷积公式：

$$G_{m,k} = \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} \sum_{c=0}^{C-1} I_{n,pu+rd_h-a_h,qv+sd_w-a_w,c} F_{r,s,c,k} = O_{n,p,q,k} \quad (32)$$

所以，GEMM convolution 并不是改变了数学结果，而是改变了计算组织方式。

4. im2col 与 implicit GEMM

如果显式生成 A ，这个过程通常叫 im2col：

input tensor -> im2col matrix A -> GEMM -> output tensor

这样做的好处是实现简单，可以直接调用成熟的矩阵乘法库。问题是 A 可能非常大，因为相邻 patch 会重复包含大量相同输入元素。

implicit GEMM convolution 的做法是：

input tensor -> kernel 内按 $A(m, \ell)$ 的映射取数 -> GEMM 风格计算 -> output tensor

也就是说， A 是一个逻辑矩阵，而不是一个提前写入内存的真实大矩阵。kernel 内部按 GEMM 的 tile、warp、thread 组织方式工作，需要 $A_{m,\ell}$ 时，再根据 m 和 ℓ 算出原始 input tensor 中的坐标去加载。

所以：

- GEMM convolution：把卷积写成 $C = AB$
- explicit GEMM convolution：先真的生成 A ，再做 GEMM

- implicit GEMM convolution: 不生成完整 A , 只在 kernel 内隐式访问 $A_{m,\ell}$

5. 一个形状例子

假设:

$$N = 1, H = 4, W = 4, C = 1 \quad (33)$$

$$R = 3, S = 3, K = 2 \quad (34)$$

stride 为 1, padding 为 0, dilation 为 1。

那么:

$$P = 4 - 3 + 1 = 2 \quad (35)$$

$$Q = 4 - 3 + 1 = 2 \quad (36)$$

输出 shape 是:

$$O \in \mathbb{R}^{1 \times 2 \times 2 \times 2} \quad (37)$$

GEMM 中三个矩阵的 shape 是:

$$A : (NPQ) \times (RSC) = 4 \times 9 \quad (38)$$

$$B : (RSC) \times K = 9 \times 2 \quad (39)$$

$$G : (NPQ) \times K = 4 \times 2 \quad (40)$$

其中 A 的 4 行分别对应 4 个输出空间位置:

(n=0, p=0, q=0)

(n=0, p=0, q=1)

(n=0, p=1, q=0)

(n=0, p=1, q=1)

每一行有 9 个元素, 对应一个 $3 \times 3 \times 1$ 的输入 patch。

6. 一句话总结

卷积可以看成大量 patch-filter 点积。把每个输出位置对应的 patch 放到矩阵 A 的一行, 把每个 filter 放到矩阵 B 的一列, 卷积就变成了 $C = AB$ 。implicit GEMM convolution 保留这个 GEMM 视角, 但不显式生成完整 im2col 矩阵, 而是在 kernel 中按映射关系从原始 tensor 取数。