深度学习中的高效计算方法-lab02

韩金成 2400012825 信息科学技术学院

2025 年 7 月 11 日

Q1. Implement the matrix multiplication using Strassen Algorithm and compare the speed with original matmul()in lab 01. The shape of matrix A is I ×K and the shape of matrix B is K ×J. The matrix size setting remains the same as lab 01, the value of I, K, J will be fixed at 256, 512 or 1024

解答. 经过实验，我们得到的结果如下：

表 1: 不同矩阵形状下不同矩阵乘法实现的平均运行时间对比

N	`matmul()`	`matmul_ikj()`	`matmul_AT()`	`matmul_BT()`	`matmul_strassen`
			平均运行时间 (s)		
256	0.026	0.006	0.029	0.012	0.014
512	0.336	0.048	0.414	0.083	0.098
1024	4.050	0.371	18.637	0.720	0.675

备注：实验平台:MacBook air、处理器型号:Apple M4、测试方法：运行 10 次取平均。

实验代码见附件中的 strassen.cpp

编译命令为：

```
g++ strassen.cpp -o strassen_mul -O3 && ./strassen_mul
```

Q2. Implement a C++ verison from scratch based on Winograd algorithm and compare the speed with your original im2col implement in lab 01. Please provide analysis on whether or not is your implementation improve the speed performance and why. The Convolution kernel and input size remain the same as lab 01:

- batch: 1
- height feature: 56
- width feature: 56
- in channels: 3
- out channels: 64
- kernel size: 3
- stride: 1
- padding: 0

解答. 经过实验，我们得到的结果如下：

表 2: Im2col 和 Winograd 实现的平均运行时间对比

优化选项	**平均运行时间 (s)**	
	im2col()	**Winograd**
`-O1`	6.87	15.57
`-O2`	6.34	15.33
`-O3`	3.72	5.81

备注：实验平台:MacBook air、处理器型号:Apple M4.

我们发现在这个参数规模下测试，`Winograd()` 方法与 `Im2col` 方法相比并没有显著优势，反而实际结果与理论分析相反。通过分析，我认为问题出在硬件对特定计算过程的优化上。

对于 Im2col 来说，其虽然表面上来看其对缓存的局部性利用率没有 Winograd 那么高，但是由于其涉及到大量矩阵乘法，在现代处理器上，编译器会自动识别这是一个 GEMM 计算，从而自动应用诸如 SIMD、Cache Blocking 等强力优化，从而减小了性能开销。

然而对于 Winograd 来说，虽然其理论性能应当更高，但是我们前期进行了大量的矩阵操作，这些矩阵操作在处理器上并没有能自动优化的手段，因此其在常数级别上慢于 Im2col.

但理论上来说，Winograd 的速度应高于 Im2col，但真的要其达到相同的性能水平，仍需编译器和处理器级别的高效优化。

实验代码见附件中的 winograd.cpp

编译命令为:

```
g++ winograd.cpp -o winograd -O1 && ./winograd

g++ winograd.cpp -o winograd -O2 && ./winograd

g++ winograd.cpp -o winograd -O3 && ./winograd
```