Matrix multiplication is one of the main operations used in deep learning, most notably in large language models. Matrix is a table of numbers. It has rows and columns. A single row and a single column is called a vector – a sequence of numbers. Matrix multiplication uses two matrices, A and B, as an input and produces matrix C as an output. Matrix A has dimensions (M, K). Matrix B has dimensions (K, N). Both matrix A and B share the same dimension K. In other words, rows of matrix A have the same length as columns of matrix B. When you multiply A by B, you get a new matrix C with dimensions (M, N):

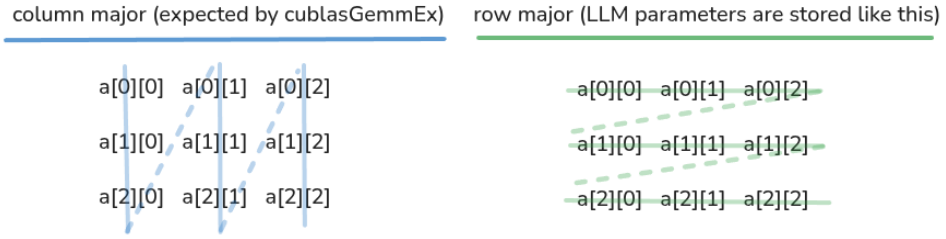
$$A(M, K) \times B(K, N) = C(M, N)$$

Every element of matrix C (c_{ij}) is a dot product of i-th row of A and j-th column of B. Dot product is a sum of all pairs, where each pair is a result of multiplying numbers from i-th row of A with numbers from j-th row of B on the same indices within their vectors (both row and column are vectors):

$$c_{ij} = a_{i0}b_{0j} + a_{i1}b_{1j} + ... + a_{ik}b_{kj} = \sum_{x=0}^{k} a_{ix}b_{xj}$$

Back to large language models. Matrix multiplication happens when computing attention and Q, K and V projections. Most popular hardware to compute it efficiently are NVIDIA GPUs. They provide an important library, cuBLAS, which allows you to run high performance linear algebra computations on their GPUs, including matrix multiplication, using cublasGemmEx function.

The problem with cublasGemmEx is that it expects you to provide the matrices in column-major format. And the LLMs, like Llama 3.2 1B Instruct, are distributed in row-major format.



Now the reason why I write this article – it turns out we don't have to modify the data format to use cuBLAS matrix multiplication functions. All thanks to these properties:

$$[A^T]_{ij} = [A]_{ji}, \qquad C^T = B^T \times A^T, \qquad (A^T)^T = A \tag{1}$$

The "^{T}" means that we transpose the matrix. Transposing a matrix turns columns into rows, and rows into columns. When you store the matrix in row-major format, and cuBLAS reads it in column-major format, it's an equivalent of transposing the matrix.

Let's see an example to understand it better: we want to compute $C = A \times B$, where A has dimensions (5, 2048) and B has dimensions (512, 2048). Our desired dimension of C is (5, 512). Right now, A and B dimensions are incompatible: $A(5, 2048)$ and $B(512, 2048)$. Do you remember that to get $C(M, N)$ we need $A(M, K)$ and $B(K, N)$? In other words, the second dimension of A and first dimension of B need to be equal. To achieve that, we need to transpose B. The formula becomes now: $C = A \times B^T$. The dimensions are ok now: $A(5, 2048) \times B(2048, 512) = C(5, 512)$. Okay, so we would like to use cuBLAS now to compute the C. But cuBLAS expects column-major format of A and B. Row-major transposed will give us column-major. So let's transpose the formula $C = A \times B^T$, using the property $C^T = B^T \times A^T$ and we get $C^T = (B^T)^T \times A^T$. From the third property of the matrix above, we know that a transposition of a transposition is equal to the original matrix, so we simplify $(B^T)^T$ to just B. The final formula is: $C^T = B \times A^T$. Let's check if dimensions are still correct. $B(512, 2048) \times A^T(2048, 5) = C^T(512, 5)$. The result dimensions seem inverse of what we wanted to get – (5, 512) – but notice that we still talk about C^T. The actual C, the output of the cublasGemmEx, is not transposed, so the final dimension is correct (5, 512). Now the code:

```
cublasGemmEx(cublas_handle, CUBLAS_OP_T, CUBLAS_OP_N, KV_DIM, num_active_slots, EMBEDDING_LENGTH,
↪   &k_proj_alpha, weights.w_k[layer], CUDA_R_16BF, EMBEDDING_LENGTH, rms_norms, CUDA_R_16BF,
↪   EMBEDDING_LENGTH, &k_proj_beta, k_proj_batched_buffer, CUDA_R_16BF, KV_DIM, CUBLAS_COMPUTE_32F,
↪   CUBLAS_GEMM_DEFAULT);
```

I want to preempt the last confusion you might have if you actually dig into the code. The flags CUBLAS_OP_T and CUBLAS_OP_N tell the cuBLAS which matrices to transpose. And we just derived the formula $C^T = B \times A^T$, so why do we now tell the cuBLAS to transpose the first matrix B? To understand it, think about column- / row-major again. From cuBLAS perspective, our row-major B is transposed B^T, because cuBLAS reads it as if it were column-major. So we need to tell cuBLAS to transpose it, to get back the B we derived. Similarly, since we derived that the second argument should be A^T, and cuBLAS reads row-major A as a column-major A^T, then don't transpose it again, because it's how we wanted to provide it to the cublasGemmEx. Q.E.D. :D

This article is part of my tiny-vllm course for building a high performance LLM inference engine in C++ and CUDA. Thanks for reading and happy hacking