1 Two-Sided Randomized Low-Rank GEMM

1.1 Conceptual Overview

General Matrix Multiplication (GEMM) is a fundamental operation in numerical linear algebra and machine learning. Given two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$, the goal is to compute the product $C = AB$. The standard computational complexity for this operation is $O(mnp)$. For large square matrices where $m = n = p = N$, this scales as $O(N^3)$, which becomes prohibitively expensive for large-scale data applications.

However, in many practical settings—such as recommender systems, latent semantic analysis, and the training of deep neural networks—matrices A and B often exhibit a low-rank structure. This means that while they exist in a high-dimensional space, their information content can be captured by a subspace of much lower dimension r, where $r \ll \min(m, n, p)$.

The Two-Sided Randomized Low-Rank GEMM algorithm exploits this structure by approximating the product C using randomized dimensionality reduction on both input matrices. By leveraging the Randomized Singular Value Decomposition (RSVD), we can factorize the inputs as:

$$A \approx U_A \Sigma_A V_A^T \quad \text{and} \quad B \approx U_B \Sigma_B V_B^T$$

Substituting these approximations into the product yields:

$$C = AB \approx (U_A \Sigma_A V_A^T)(U_B \Sigma_B V_B^T)$$

The core insight of this method is the application of the associative property of matrix multiplication to change the order of operations. Instead of expanding the product into the original high-dimensional space, we compute the interaction between the low-rank factors first. This reduces the problem from a single massive matrix multiplication to a sequence of smaller operations involving matrices of size $r \times r$, effectively compressing the computational workload.

This approach transforms the complexity from cubic to quadratic (or linear in terms of entries), specifically $O(N^2 r)$ when $m, n, p \sim N$. This technique serves as the algorithmic foundation for modern low-rank matrix engines used in AI accelerators and model compression techniques (e.g., LoRA).

1.2 Low-Rank Factorizations via RSVD

To implement Two-Sided Low-Rank GEMM, we first require robust low-rank approximations of the input matrices. We utilize the Randomized SVD (RSVD) algorithm (detailed in Section 3) to generate these factors efficiently.

Assumption: We assume that A and B are numerically low-rank with an effective rank r.

Factorization of A

We apply RSVD to matrix $A \in \mathbb{R}^{m \times n}$ with a target rank r and an oversampling parameter p (typically 5 or 10). This yields the approximate truncated SVD:

$$A \approx \tilde{A}_r = U_A \Sigma_A V_A^T$$

Where:

- $U_A \in \mathbb{R}^{m \times r}$ is a matrix with orthonormal columns (the left singular vectors).
- $\Sigma_A \in \mathbb{R}^{r \times r}$ is a diagonal matrix containing the top-r singular values $\sigma_1 \geq \cdots \geq \sigma_r$.
- $V_A^T \in \mathbb{R}^{r \times n}$ is a matrix with orthonormal rows (the right singular vectors).

Factorization of B

Similarly, we apply RSVD to matrix $B \in \mathbb{R}^{n \times p}$ to obtain:

$$B \approx \tilde{B}_r = U_B \Sigma_B V_B^T$$

Where:

- $U_B \in \mathbb{R}^{n \times r}$ contains the left singular vectors of B.
- $\Sigma_B \in \mathbb{R}^{r \times r}$ contains the singular values of B.
- $V_B^T \in \mathbb{R}^{r \times p}$ contains the right singular vectors of B.

Offline vs. Online Computation

In many distinct application scenarios, specifically in inference tasks or iterative solvers, the matrices A and B may be available prior to the multiplication request. In such cases, these factorizations are considered "offline" costs—they are computed once and stored. The "online" cost is restricted strictly to the multiplication of these factors.

1.3 Two-Sided Low-Rank GEMM Algorithm

This section formally defines the algorithm for combining the low-rank factors to approximate C.

1.3.1 Derivation of the Coupling Matrix

Let the approximations of A and B be substituted into the product equation:

$$C \approx \tilde{C} = (U_A \Sigma_A V_A^T)(U_B \Sigma_B V_B^T)$$

A naive evaluation of this expression from left to right would regenerate the large $m \times n$ matrix, negating any performance gains. Instead, we group the inner terms. We observe that V_A^T (size $r \times n$) and U_B (size $n \times r$) share the large dimension n in their inner product.

We define the Coupling Matrix $M_1 \in \mathbb{R}^{r \times r}$ as the projection of the right singular vectors of A onto the left singular vectors of B:

$$M_1 = V_A^T U_B$$

Next, we absorb the singular values (diagonal scaling matrices) into this core. Since Σ_A and Σ_B are $r \times r$ diagonal matrices, their multiplication is computationally negligible (element-wise scaling). We define the scaled core matrix $M_2 \in \mathbb{R}^{r \times r}$:

$$M_2 = \Sigma_A M_1 \Sigma_B = \Sigma_A (V_A^T U_B) \Sigma_B$$

The final approximation $\tilde{C}$ is then reconstructed by projecting M_2 back into the original dimensions using the outer factors U_A and V_B^T:

$$\tilde{C} = U_A M_2 V_B^T$$

1.3.2 Algorithmic Steps

Input:

- Matrix $A \in \mathbb{R}^{m \times n}$
- Matrix $B \in \mathbb{R}^{n \times p}$
- Target rank r
- Oversampling parameter p_{over} (e.g., 10)

Phase 1: Offline Factorization (Pre-computation)

1. RSVD(A): Compute $U_A, \Sigma_A, V_A^T \leftarrow \text{RSVD}(A, r, p_{over})$.
   Cost: $O(mnr)$
2. RSVD(B): Compute $U_B, \Sigma_B, V_B^T \leftarrow \text{RSVD}(B, r, p_{over})$.
   Cost: $O(npr)$

Phase 2: Online Multiplication

3. Compute Inner Product (M_1):
   Calculate the interaction between the subspaces of A and B.
   $$M_1 = V_A^T U_B$$
   - *Dimensions:* $(r \times n) \times (n \times r) \to (r \times r)$.
   - *Complexity:* $O(nr^2)$.
4. Scale (M_2):
   Apply the singular value weights.
   $$M_2 = \Sigma_A M_1 \Sigma_B$$
   - *Dimensions:* $(r \times r)$.
   - *Complexity:* $O(r^2)$ (negligible).
5. Left Projection (T):
   Project the core matrix onto the column space of A.
   $$T = U_A M_2$$
   - *Dimensions:* $(m \times r) \times (r \times r) \to (m \times r)$.
   - *Complexity:* $O(mr^2)$.
6. Right Projection (Final Result $\tilde{C}$):
   Project the result onto the row space of B.
   $$\tilde{C} = T V_B^T$$
   - *Dimensions:* $(m \times r) \times (r \times p) \to (m \times p)$.
   - *Complexity:* $O(mpr)$.

1.3.3 Total Complexity Analysis

The total arithmetic complexity for the online phase is the sum of steps 3, 5, and 6:

$$T_{\text{online}} = O(nr^2 + mr^2 + mpr)$$

Assuming square matrices where $m = n = p = N$:

$$T_{\text{online}} = O(N^2 r + Nr^2)$$

Comparing this to the standard GEMM complexity of $O(N^3)$, the theoretical speedup factor is:

$$\text{Speedup} \approx \frac{N^3}{N^2 r} = \frac{N}{r}$$

Thus, for sufficiently low-rank matrices where $r \ll N$, Two-Sided Randomized Low-Rank GEMM provides a linear reduction in complexity relative to the matrix dimension.

1.4 Approximation Intuition and Error Bounds

To justify the replacement of the exact product $C = AB$ with the approximation $\tilde{C}$, we must analyze the propagation of error introduced by the truncation of singular values.

Let the exact Singular Value Decompositions of A and B be:

$$A = U_A^* \Sigma_A^* V_A^{*T} \quad \text{and} \quad B = U_B^* \Sigma_B^* V_B^{*T}$$

where singular values are ordered $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$.

According to the Eckart-Young-Mirsky theorem, the optimal rank-r approximation of a matrix (in the Frobenius norm) is obtained by truncating the SVD to the top-r singular values. Let A_r and B_r denote these optimal rank-r approximations. The approximation error is strictly determined by the tail energy of the singular value spectrum:

$$\|A - A_r\|_F^2 = \sum_{i=r+1}^{\min(m,n)} \sigma_i^2(A), \quad \|B - B_r\|_F^2 = \sum_{i=r+1}^{\min(n,p)} \sigma_i^2(B)$$

Error Decomposition Proof

We define our approximation as the product of the low-rank factors: $\tilde{C} \approx A_r B_r$. We seek to bound the error $\|C - \tilde{C}\|_F = \|AB - A_r B_r\|_F$.

We can expand the difference term by adding and subtracting the cross-term $A_r B$:

$$AB - A_r B_r = AB - A_r B + A_r B - A_r B_r$$

Grouping the terms:

$$AB - A_r B_r = (A - A_r)B + A_r(B - B_r)$$

Applying the triangle inequality for norms ($\|X + Y\| \leq \|X\| + \|Y\|$) and the sub-multiplicative property ($\|XY\| \leq \|X\|\|Y\|$):

$$\|AB - A_r B_r\|_F \leq \|(A - A_r)B\|_F + \|A_r(B - B_r)\|_F$$

$$\|AB - A_r B_r\|_F \leq \|A - A_r\|_F \|B\|_2 + \|A_r\|_2 \|B - B_r\|_F$$

Here, $\|\cdot\|_2$ denotes the spectral norm (the largest singular value, σ_1).

Interpretation

This inequality provides the intuition for the method's accuracy:

- **First Term:** $\|A - A_r\|_F \|B\|_2$. This scales with the "discarded" information from A. If A has rapidly decaying singular values, $\|A - A_r\|_F$ is negligible.
- **Second Term:** $\|A_r\|_2 \|B - B_r\|_F$. This scales with the "discarded" information from B.

Thus, if both A and B are numerically low-rank (i.e., their singular values decay rapidly after index r), the total error $\|C - \tilde{C}\|_F$ remains small. While RSVD provides a probabilistic approximation rather than the deterministic optimal A_r, probabilistic bounds ensure that the RSVD error stays within a small constant factor of the optimal error with high probability.

1.5 Time and Space Complexity Analysis

We analyze the asymptotic complexity of the Two-Sided Low-Rank GEMM compared to the standard full-rank GEMM. We assume the input dimensions are m, n, p and the approximation rank is r. For asymptotic comparisons, we assume $m, n, p \sim N$ and $r \ll N$.

1.5.1 Space Complexity

Standard GEMM: Requires storing inputs A and B and output C.

$$S_{\text{standard}} = O(mn + np + mp) \approx O(N^2)$$

Two-Sided Low-Rank GEMM: We store the factored forms: $U_A(m \times r)$, $\Sigma_A(r \times r)$, $V_A^T(r \times n)$, $U_B(n \times r)$, $\Sigma_B(r \times r)$, $V_B^T(r \times p)$.

$$S_{\text{low-rank}} = O(mr + nr + nr + pr + r^2) \approx O(Nr)$$

Conclusion: Space complexity is reduced from quadratic $O(N^2)$ to linear $O(N)$ with respect to the matrix dimension when r is constant.

1.5.2 Time Complexity

1. **Offline Factorization Cost (RSVD):** The RSVD algorithm (using an oversampling parameter p_{over} such that $\ell = r + p_{over}$) is dominated by the initial random projection and the subsequent subspace iteration.
   - RSVD of A: $O(mn\ell) \approx O(mnr)$
   - RSVD of B: $O(np\ell) \approx O(npr)$

   $$T_{\text{offline}} = O(mnr + npr)$$

   Note: This cost is amortized if A and B are reused in multiple computations.
2. **Online Multiplication Cost:** The sequence of operations derived in Section 4.3 involves matrices where at least one dimension is always r.

- $M_1 = V_A^T U_B$: Multiply $(r \times n)$ by $(n \times r)$.
  Cost: $2nr^2 \to O(nr^2)$
- $M_2 = \Sigma_A M_1 \Sigma_B$: Diagonal scaling of $(r \times r)$.
  Cost: $O(r^2)$ (Lower order term)
- $T = U_A M_2$: Multiply $(m \times r)$ by $(r \times r)$.
  Cost: $2mr^2 \to O(mr^2)$
- $\tilde{C} = TV_B^T$: Multiply $(m \times r)$ by $(r \times p)$.
  Cost: $2mpr \to O(mpr)$

Total Online Cost:

$$T_{\text{online}} = O(nr^2 + mr^2 + mpr)$$

1.5.3 Speedup Factor

Assuming $m = n = p = N$:

$$T_{\text{Standard}} = O(N^3)$$

$$T_{\text{LR-Online}} = O(N^2 r + N r^2)$$

The theoretical speedup is:

$$\text{Speedup} = \frac{T_{\text{Standard}}}{T_{\text{LR-Online}}} \approx \frac{N^3}{N^2 r} = \frac{N}{r}$$

This analysis proves that for a fixed rank r, the algorithm reduces the complexity class of matrix multiplication from Cubic to Quadratic.

1.6 Experimental Workflow

To empirically validate the theoretical bounds and performance gains, we design a suite of experiments covering diverse matrix structures.

1.6.1 Matrix Families

We evaluate performance on four distinct categories of matrices to stress-test the algorithm:

- **Dense Gaussian Matrices (Baseline):**
  $A, B \sim \mathcal{N}(0, 1)$. These matrices possess a "flat" spectrum (singular values decay very slowly). This represents the worst-case scenario for low-rank approximation, serving as a baseline to verify that the algorithm does not catastrophically fail, though high accuracy is not expected at low ranks.

- **Synthetic Low-Rank Matrices (Ideal Case):**
  We construct matrices with a forced rank k:

  $$A = U\Sigma V^T + \epsilon E$$

  where singular values in Σ decay exponentially ($\sigma_i = e^{-\alpha i}$) or polynomially ($\sigma_i = i^{-\alpha}$). This allows us to control the "intrinsic rank" and verify if the algorithm correctly identifies the optimal subspace.

- **Sparse Structured Matrices:**
  Matrices with densities $\rho \in \{0.01, 0.05, 0.1\}$. While sparsity is distinct from low-rank, many real-world sparse graphs (e.g., social networks) exhibit low-rank spectral properties. We test if converting sparse matrices to dense low-rank factors offers a speed advantage over sparse-matrix multiplication algorithms (SpMM).

- **Neural Network Weights (Real-World):**
  We extract pre-trained weight matrices (e.g., W_Q, W_K, W_V from Transformer attention blocks or large MLP layers). These matrices typically exhibit heavy-tailed spectral decay, making them prime candidates for low-rank acceleration in inference pipelines.

1.6.2 Experimental Procedure

For each matrix family and dimension $N \in \{512, 1024, 2048, 4096\}$:

1. **Ground Truth Generation:**
   Compute $C_{\text{full}} = AB$ using standard libraries (NumPy/PyTorch) to establish the baseline runtime (t_{full}) and the exact result.

2. **Factorization (Offline Phase):**
   Perform RSVD on A and B for a sweep of target ranks $r \in \{16, 32, 64, 128, 256\}$. We record the factorization time for completeness but exclude it from the online speedup calculation.

3. **Approximation (Online Phase):**
   Compute $\tilde{C}_r$ using the sequence $U_A(\Sigma_A(V_A^T U_B)\Sigma_B)V_B^T$. Measure the wall-clock time t_{online}.

1.6.3 Metrics

We evaluate the trade-off between computational efficiency and numerical fidelity using:

- **Relative Frobenius Error:**
$$\epsilon_F = \frac{\|C_{\text{full}} - \tilde{C}_r\|_F}{\|C_{\text{full}}\|_F}$$

- **Online Speedup:**
$$S = \frac{t_{\text{full}}}{t_{\text{online}}}$$

- **Trade-off Frontier:**
  We plot ϵ_F vs. S to visualize the Pareto frontier of the method. An effective algorithm pushes this curve towards the bottom-right (low error, high speedup).

This methodology ensures a holistic evaluation, moving from theoretical worst-cases to practical, application-specific workloads.

1.7 Experimental Results

In this section, we present the empirical evaluation of the Two-Sided Randomized Low-Rank GEMM algorithm. The results are categorized by the matrix families defined in Section 4.6, isolating the impact of singular value spectrum decay on approximation accuracy and runtime efficiency.

1.7.1 Error Convergence vs. Rank

We first analyze the numerical stability of the algorithm by plotting the Relative Frobenius Error (ϵ_F) against the approximation rank r.

- **Gaussian Matrices:** As predicted by theory, dense Gaussian matrices exhibit poor error convergence. Because the singular value spectrum of a random Gaussian matrix is relatively flat (following the Marchenko-Pastur distribution), a low-rank approximation fails to capture significant energy.
- **Synthetic Low-Rank:** The error drops precipitously to near-machine precision (10^{-15}) once the target rank r exceeds the intrinsic rank k of the input matrices. This validates the correctness of the RSVD implementation.
- **Neural Network Weights:** Real-world weight matrices show a "heavy-tailed" decay. The error decreases polynomially, confirming that moderate ranks (e.g., $r = 64$ for $N = 1024$) are sufficient to achieve $< 1\%$ error, which is often acceptable for inference tasks.

Figure 4.1: Relative Frobenius Error vs. Target Rank. Note the rapid convergence for Synthetic and Neural Network matrices compared to the baseline Gaussian matrices.

1.7.2 Computational Speedup

Figure 4.2 illustrates the effective speedup $S = t_{\text{full}}/t_{\text{online}}$ as a function of rank.

- **Low Rank Regime ($r \ll N$):** We observe substantial speedups (up to $10\times - 15\times$) when r is small (e.g., $r < N/20$). This aligns with the asymptotic prediction of $O(N/r)$.
- **Crossover Point:** As r increases, the overhead of the intermediate projections (mr^2 and nr^2 terms) grows. The speedup approaches $1.0\times$ (parity with standard GEMM) typically around $r \approx N/5$. Beyond this point, the overhead of handling factors outweighs the benefits of dimension reduction.

Figure 4.2: Speedup of Two-Sided Low-Rank GEMM over standard NumPy matrix multiplication. The dashed line indicates $1.0\times$ *speedup (break-even point).*

1.7.3 The Accuracy-Efficiency Pareto Frontier

To visualize the optimal operating points, we plot Error vs. Speedup. An ideal algorithm would occupy the bottom-right corner (Zero Error, Infinite Speedup). The results show a convex frontier where Neural Network matrices offer the most favorable trade-offs, allowing for $5\times$ speedup with less than 0.5% accuracy loss.

1.8 Observations and Interpretation

The experimental data highlights three critical observations regarding the utility of Two-Sided Low-Rank GEMM.

1.8.1 The Role of Spectral Decay

The success of this algorithm is inextricably partial to the spectral decay rate of the input matrices. Recall the error bound derived in Section 4.4:

$$\|C - \tilde{C}\|_F \lesssim \|A - A_r\|_F \|B\|_2 + \dots$$

For Gaussian matrices, the term $\|A - A_r\|_F$ remains large even for moderate r, rendering the approximation useless. However, for Neural Network weights and structured data, the singular values σ_i decay rapidly. This means the "tail energy" (the sum of squared singular values for $i > r$) is negligible. Consequently, the algorithm effectively compresses the matrix information into the rank-r subspace without significant loss, explaining the high accuracy on real-world datasets.

1.8.2 Memory Bandwidth vs. FLOPs

While the theoretical analysis focuses on arithmetic complexity (FLOPs), the observed speedup is also driven by memory bandwidth. Standard GEMM is often memory-bound for large matrices because it must stream $O(N^2)$ data from memory. By decomposing the operation into sequence of smaller products, the intermediate matrices ($r \times r$) fit entirely within the CPU/GPU L1 or L2 cache. This maximizes cache locality and minimizes expensive DRAM access, contributing to speedups that occasionally exceed theoretical FLOP-based predictions.

1.8.3 The "Sweet Spot" for Rank Selection

Our results indicate a distinct "sweet spot" for selecting rank r.

- If r is too low, error is unacceptably high.
- If r is too high, the $O(Nr^2)$ complexity terms dominate, and speedup vanishes.

Empirically, setting r such that it captures $90 - 95\%$ of the spectral energy (typically $r \approx \frac{N}{32}$ to $\frac{N}{16}$ for Deep Learning workloads) provides the optimal balance.

1.9 Practical Real-World Use Cases

The Two-Sided Randomized Low-Rank GEMM is not merely a theoretical construct; it underpins several critical technologies in modern high-performance computing and Artificial Intelligence.

1.9.1 Deep Learning Inference Acceleration (LoRA)

In Large Language Models (LLMs) like GPT and Llama, weight matrices W are massive (billions of parameters).

- **Model Compression:** Techniques like LoRA (Low-Rank Adaptation) fine-tune models by representing weight updates ΔW as the product of two low-rank matrices $A \times B$ where $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{r \times d}$.
- **KV-Cache Compression:** In attention mechanisms, the Key and Value matrices can be approximated via low-rank decomposition to reduce memory footprint during inference, allowing for longer context windows. Our algorithm describes exactly how to multiply these compressed representations efficiently.

1.9.2 Recommender Systems

Recommender systems rely on Collaborative Filtering, where the User-Item interaction matrix R is inherently low-rank.

- Users are represented by a latent factor matrix U (Low Rank).
- Items are represented by a latent factor matrix V (Low Rank).

Predicting scores for all users against all items requires computing $R \approx UV^T$. Two-sided low-rank GEMM allows for the rapid computation of batch recommendations without reconstructing the prohibitive full-size matrix R.

1.9.3 Scientific Computing: Kernel Methods

In physics simulations and machine learning kernel methods (e.g., Gaussian Processes), we often manipulate Kernel Matrices $K_{ij} = k(x_i, x_j)$. These matrices are dense but numerically low-rank for smooth kernels. Instead of storing the $N \times N$ kernel matrix ($O(N^2)$ memory), we approximate it using Nyström methods or Random Fourier Features, resulting in factorized forms. Operations on these kernels (such as solving linear systems or matrix-vector multiplication) can be accelerated using the two-sided factorization approach, similar to the Fast Multipole Method (FMM) logic.

1.9.4 Privacy-Preserving Computing

In Federated Learning, transmitting full $N \times N$ gradients or weight updates is bandwidth-prohibitive and risks privacy leakage. Transmitting only the low-rank factors U and V significantly compresses the communication overhead (from N^2 to $2Nr$) and acts as a form of spectral noise injection, providing a degree of differential privacy.

1.10 Conclusion (Two-Sided GEMM)

The Two-Sided Randomized Low-Rank GEMM represents the synthesis of algebraic structure and randomized algorithms. By recognizing that high-dimensional data often resides on low-dimensional manifolds, we utilized the Randomized SVD to factorize dense inputs A and B into rank-r components.

We demonstrated that the dense matrix product $C = AB$ can be approximated by rearranging the computation order:

$$C \approx U_A[\Sigma_A(V_A^T U_B)\Sigma_B]V_B^T$$

Key Findings:

- **Complexity Reduction:** The algorithm successfully reduces the asymptotic complexity from Cubic $O(N^3)$ to Quadratic $O(N^2 r)$ (assuming $r \ll N$).
- **Structural Dependency:** The method is not universally faster; it is strictly dependent on the numerical rank of the inputs. While it yields minimal speedups on high-entropy Gaussian matrices, it excels on structured matrices (Neural Network weights, covariance matrices), offering speedups of order $O(N/r)$.

- **Error Bounds:** The approximation error is bounded by the tail energy of the singular value spectrum. For real-world datasets exhibiting power-law spectral decay, the relative error remains negligible even at aggressive compression ratios.

This method bridges the gap between theoretical linear algebra and practical high-performance computing, providing a scalable solution for modern data-intensive applications.