4. Two-Sided Randomized Low-Rank GEMM

4.1 Conceptual Overview

Two-sided randomized low-rank GEMM accelerates matrix multiplication when **both** input matrices are approximately low-rank.

Given - $A \in \mathbb{R}^{m \times n}$, - $B \in \mathbb{R}^{n \times p}$,

we want to approximate

$$ C = AB $$

faster than standard GEMM ($O(mnp)$) by exploiting low-rank structure in **both** A and B.

We: 1. Use **Randomized SVD (RSVD)** to obtain low-rank factorizations $$ A \approx U_A \Sigma_A V_A^T, \quad B \approx U_B \Sigma_B V_B^T, $$ where each factor is rank $r \ll \min(m, n, p)$. 2. Multiply using only the small factors, in **factorized form**: $$ C = AB \approx U_A \, \Sigma_A \, (V_A^T U_B) \, \Sigma_B \, V_B^T, $$ replacing one large GEMM by a sequence of much cheaper operations involving $r \times r$, $m \times r$, and $r \times p$ matrices.

This is the algorithmic core of many modern systems: low-rank matrix engines in AI accelerators, neural network compression, and methods like Low-Rank GEMM.

4.2 Low-Rank Factorizations via RSVD

We assume both A and B are **numerically low-rank**, with effective rank r.

Using the RSVD algorithm from Section 3, we compute rank-r approximations:

- For A: $$ A \approx U_A \Sigma_A V_A^T, $$ where $U_A \in \mathbb{R}^{m \times r}$, $\Sigma_A \in \mathbb{R}^{r \times r}$ (diagonal), $V_A^T \in \mathbb{R}^{r \times n}$.

- For B: $$ B \approx U_B \Sigma_B V_B^T, $$ where $U_B \in \mathbb{R}^{n \times r}$, $\Sigma_B \in \mathbb{R}^{r \times r}$, $V_B^T \in \mathbb{R}^{r \times p}$.

These factorizations are obtained **once** per matrix using RSVD with oversampling parameter p (e.g., $p = 5$ or 10).

4.3 Two-Sided Low-Rank GEMM Algorithm

We approximate $C = AB$ using only the low-rank factors.

Starting from $$ C = AB \approx (U_A \Sigma_A V_A^T)(U_B \Sigma_B V_B^T), $$ we group terms as $$ C \approx U_A \, \Sigma_A \, (V_A^T U_B) \, \Sigma_B \, V_B^T. $$

Define: - $M_1 = V_A^T U_B \in \mathbb{R}^{r \times r}$, - $M_2 = \Sigma_A M_1 \Sigma_B \in \mathbb{R}^{r \times r}$.

Then $$ C \approx U_A M_2 V_B^T. $$

Algorithm (Two-Sided Randomized Low-Rank GEMM)

Input: - Matrices $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$ - Target rank r - Oversampling parameter for RSVD (say $p = 10$)

Offline (performed once per A and B):

1. **RSVD of A**
2. Use RSVD to compute a rank-r approximation $$ A \approx U_A \Sigma_A V_A^T. $$
3. **RSVD of B**
4. Use RSVD to compute a rank-r approximation $$ B \approx U_B \Sigma_B V_B^T. $$
5. **Core $r \times r$ coupling matrix**
6. Compute $$ M_1 = V_A^T U_B \in \mathbb{R}^{r \times r}, $$
7. Then $$ M_2 = \Sigma_A M_1 \Sigma_B \in \mathbb{R}^{r \times r}. $$ Because Σ_A and Σ_B are diagonal, this step is mostly element-wise scaling.

Online (for each product $C = AB$):

1. Compute $$ T = U_A M_2 \in \mathbb{R}^{m \times r}. $$
2. Compute the approximate product $$ \tilde{C} = T V_B^T \in \mathbb{R}^{m \times p}, $$ which serves as a low-rank approximation to $C = AB$.

All expensive online work is expressed in terms of matrices with an r dimension, where $r \ll \min(m, n, p)$.

4.4 Approximation Intuition

Let the exact SVDs be $$ A = U_A^ \Sigma_A^ V_A^{T}, \quad B = U_B^ \Sigma_B^ V_B^{T}, $$ with singular values in decreasing order.

The **optimal** rank-r approximations (truncated SVDs) are $$ A_r^ = U_{A,r}^ \Sigma_{A,r}^ V_{A,r}^{T}, \quad B_r^ = U_{B,r}^ \Sigma_{B,r}^ V_{B,r}^{T}, $$ with errors $$ \|A - A_r^\|_F^2 = \sum_{i>r} \sigma_i^A)^2, \quad \|B - B_r^\|_F^2 = \sum_{i>r} \sigma_i^B)^2. $$

RSVD with oversampling approximates these truncated SVDs up to a small multiplicative factor in expectation. Denote the RSVD-based approximations by A_r and B_r.

We then form $$ \tilde{C} = A_r B_r. $$ A rough error decomposition is $$ |C - \tilde{C}|_F = |AB - A_r B_r|_F \lesssim |A - A_r|_F |B|_2 + |A_r|_2 |B - B_r|_F. $$

So the approximation error is small when:

- The discarded singular values of A and B (beyond rank r) are small.
- RSVD accurately captures the top-r subspaces (which holds when A and B are numerically low-rank).

In many real-world matrices (neural network weights, recommender matrices, kernel matrices), singular values decay rapidly, so modest ranks r already yield very accurate approximations.

4.5 Time and Space Complexity

Assume for simplicity that m, n, p are all on the order of N and that $r \ll N$.

4.5.1 RSVD Factorization Cost (Offline)

Using RSVD (Section 3) with sketch dimension $\ell = r + p$:

- RSVD(A): $O(mnr)$
- RSVD(B): $O(npr)$

Total factorization cost (paid once per matrix): $$ T_{\text{RSVD,total}} = O(m n r + n p r). $$

4.5.2 Two-Sided Low-Rank GEMM Cost (Online)

For each approximate product $\tilde{C} = AB$:

1. $M_1 = V_A^T U_B$:
2. V_A^T is $r \times n$, U_B is $n \times r$.
3. Cost: $O(nr^2)$.
4. $M_2 = \Sigma_A M_1 \Sigma_B$:
5. Diagonal scalings; typically $O(r^2)$.
6. $T = U_A M_2$:
7. U_A is $m \times r$, M_2 is $r \times r$.
8. Cost: $O(mr^2)$.
9. $\tilde{C} = T V_B^T$:
10. T is $m \times r$, V_B^T is $r \times p$.
11. Cost: $O(mpr)$.

Total per-product cost: $$ T_{\text{2-sided}} = O(n r^2 + m r^2 + m p r). $$

Compare with full GEMM: $$ T_{\text{GEMM}} = O(m n p). $$

For $m, n, p \sim N$: - Full GEMM: $O(N^3)$ - Two-sided low-rank GEMM: $O(N^2 r + N r^2)$

Asymptotic speedup is roughly $$ \text{speedup} \sim \frac{N^3}{N^2 r} = \frac{N}{r}, $$ ignoring lower-order terms. When $r \ll N$, this is a substantial reduction in arithmetic.

Space usage also drops from storing full A and B ($mn + np$ entries) to predominantly storing their factors ($mr + nr + nr + pr$ entries), which is $O((m + n + p)r)$.

4.6 Experimental Workflow

We evaluate two-sided low-rank GEMM on a common suite of matrix families (matching the RMM and RSVD experiments) to understand how structure affects accuracy, runtime, and speedup.

4.6.1 Matrix Families

1. **Dense Gaussian matrices (worst case)**
2. A, B with i.i.d. $\mathcal{N}(0, 1)$ entries.
3. Spectra are relatively flat; no strong low-rank structure.
4. Low-rank GEMM is expected to require larger r for good accuracy.
5. **Synthetic low-rank matrices**
6. Construct $$ A = U_A^{\text{true}} \Sigma_A^{\text{true}} V_A^{\text{true}T}, \quad B = U_B^{\text{true}} \Sigma_B^{\text{true}} V_B^{\text{true}T}, $$ where $U_A^{\text{true}}, V_A^{\text{true}}, U_B^{\text{true}}, V_B^{\text{true}}$ are random orthonormal matrices and the diagonal entries of $\Sigma_A^{\text{true}}, \Sigma_B^{\text{true}}$ decay rapidly.
7. Optionally add small Gaussian noise to simulate approximate low-rank structure.
8. These represent ideal cases where the true rank is small.
9. **Sparse / structured matrices**
10. Generate sparse matrices at different sparsity levels (e.g., 10%, 5%, 1%, 0.1% nonzeros) while maintaining low effective rank.
11. These model graph-related matrices, high-dimensional text representations, or structured operators.
12. **Neural-network-like matrices**
13. Extract real weight matrices from trained fully connected layers or embeddings (e.g., from a small MLP or transformer block), or generate synthetic matrices with heavy-tailed singular value spectra.
14. These typically exhibit strong low-rank structure and are the most relevant for AI applications.

4.6.2 Experimental Procedure

For each matrix family and a range of sizes (e.g., $N = 512, 1024, 2048$):

1. **Generate A and B**
2. According to the chosen family and size parameters.
3. **Compute ground-truth product**
4. Compute $$ C_{\text{full}} = A B $$ using standard GEMM (NumPy / PyTorch matmul) and record the **baseline runtime**.
5. **Apply RSVD to A and B**
6. For a grid of target ranks $r \in \{16, 32, 64, 128, 256\}$ (adjusting to matrix size), run RSVD on A and B to obtain $$ A_r \approx U_A \Sigma_A V_A^T, \quad B_r \approx U_B \Sigma_B V_B^T. $$
7. Record RSVD runtimes (for completeness), but treat them as offline costs.
8. **Two-sided low-rank GEMM**
9. For each rank r, compute $$ \tilde{C}_r = U_A \Sigma_A (V_A^T U_B) \Sigma_B V_B^T $$ using the factorized algorithm.
10. Measure **online runtime** for this computation (excluding RSVD).
11. **Metrics**
12. **Relative Frobenius error** $$ \text{relErr}*F(r) = *frac{|C}} - \tilde{C*r|_*F}{|C*. $$}}|_F
13. (Optionally) **spectral norm error** $$ \text{relErr}*2(r) = *frac{|C}} - \tilde{C*r|_*2}{|C*. $$}}|_2
14. **Speedup** (focusing on online compute) $$ \text{speedup}(r) = \frac{T_{\text{full GEMM}}}{T_{\text{2-sided LR-GEMM}}(r)}. $$
15. (Optional) **memory footprint**: compare $\text{nnz}(A) + \text{nnz}(B)$ vs $\text{nnz}(U_A, \Sigma_A, V_A^T, U_B, \Sigma_B, V_B^T)$.
16. **Repeats**
17. Repeat each experiment (generation + factorization + multiply) for several random seeds (e.g., 5–10 runs) and average the metrics to reduce variance.

4.6.3 Varying Rank and Structure

To connect to real-world behavior, we systematically vary:

- **Target rank r**:
- Observe how error decays and speedup decreases as r grows.
- Identify the smallest r giving, say, $< 1\%, 5\%, 10\%$ relative error.
- **Intrinsic rank / spectrum shape** (for synthetic low-rank matrices):

- Construct matrices with different singular value decay rates (fast, medium, slow).
- Check how the number of components r needed for a given accuracy changes.
- **Sparsity level** (for sparse/structured matrices):
- Test densities like 10%, 5%, 1%, 0.1%.
- Measure how sparsity interacts with low-rank structure, runtime, and approximation quality.
- **Matrix family (Gaussian vs low-rank vs NN-like)**:
- Compare performance across families to show that real-world matrices (with low-rank structure) benefit much more than adversarial Gaussian matrices.

4.7 Results (Figures to Include)

For each matrix family, we plan to include:

1. **Error vs rank r**
2. Plot $\mathrm{relErr}_F(r)$ against r for several matrix sizes.
3. Separate curves for different matrix families (Gaussian, low-rank, sparse, NN-like).
4. **Speedup vs rank r**
5. Plot $\mathrm{speedup}(r)$ against r.
6. Show that speedup is large for small r, and gradually drops as r approaches $\min(m, n, p)$.
7. **Error vs speedup tradeoff**
8. For each r, plot $(\mathrm{relErr}_F(r), \mathrm{speedup}(r))$, to visualize the tradeoff frontier.
9. **Error vs intrinsic rank / spectrum decay** (for synthetic low-rank matrices)
10. For different decay patterns of singular values, plot the rank r needed to achieve a fixed error threshold (e.g., 5%).
11. **Error vs sparsity level** (for sparse matrices)
12. For a fixed r, plot relErr_F vs sparsity level.
13. Show that as long as effective rank remains low, sparsity does not harm approximation and often helps runtime.
14. **Neural-network-like matrices**
15. Plot error and speedup vs rank for actual NN weight matrices.

16. This directly showcases relevance to AI workloads.

These figures mirror the structure of the RMM results section, but with **rank r** in place of sample count s, and emphasize how low-rank GEMM behaves across different matrix structures.

4.8 Observations and Interpretation

From these experiments, we expect to observe:

- **Gaussian matrices (no structure)**:
- Singular values decay slowly.
- Two-sided low-rank GEMM needs large r to reach low error, so speedups are modest.
- **Synthetic low-rank matrices**:
- Even very small r (close to the true rank) yield near-zero approximation error.
- Speedups are substantial (matching the N/r asymptotic picture).
- **Sparse/structured matrices**:
- When sparsity coexists with low effective rank, factorized GEMM is particularly attractive.
- Computation on sparse low-rank matrices can be dominated by the mr^2, nr^2, mpr terms rather than full mnp, giving large speedups.
- **Neural-network-like matrices**:
- Singular values typically show heavy-tailed decay.
- Moderate ranks (e.g., $r \in [32, 256]$ depending on dimension) can achieve small relative error.
- This aligns with practical low-rank methods in deep learning (e.g., LoRA-style adapters and low-rank compression), which exploit similar structure.

Overall, the experiments should demonstrate that **two-sided low-rank GEMM is most effective when both operands are numerically low-rank**, a condition commonly satisfied in real-world ML, recommendation, and scientific computing workloads.

4.9 Practical Real-World Use Cases

Two-sided low-rank GEMM appears naturally in:

- **Deep learning**:
- Compressing large projection matrices and linear layers on both sides of a pipeline.
- Accelerating blocks in transformers where multiple weight matrices interact and are individually low-rank.
- **Recommender systems**:

- Operating on user and item embedding matrices that are individually low-rank; their interactions can be computed via low-rank GEMM.

- **Kernel methods and Gaussian processes**:

- Approximating kernel matrices and feature maps by low-rank factors and combining them via two-sided GEMM.

- **Scientific computing**:

- Applying two low-rank operators in sequence (e.g., preconditioners followed by system matrices) using factorized multiplies.

These settings match the experimental matrix families and motivate the choice of benchmarks in our evaluation.

4.10 Conclusion

Two-sided randomized low-rank GEMM uses RSVD factorizations of both input matrices to replace a large $m \times n$ by $n \times p$ product with a sequence of cheaper operations involving rank-r factors. Its runtime scales like $O(N^2 r)$ rather than $O(N^3)$, leading to potential speedups of order N/r when $r \ll N$.

By evaluating this method on dense Gaussian, synthetic low-rank, sparse, and neural-network-like matrices, we can clearly see when two-sided low-rank GEMM is effective, how error decays with rank, and how it connects to real-world applications in machine learning, recommendation, and scientific computing. This completes the trio of randomized linear algebra techniques in our project: RMM (sampling-based GEMM), RSVD (low-rank approximation), and two-sided low-rank GEMM (structure-exploiting fast multiplication).