1 Randomized SVD (RSVD)

1.1 Conceptual Overview

The core challenge in modern data analysis is dealing with matrices $A \in \mathbb{R}^{m \times n}$ that are too large to fit in memory or too computationally expensive to decompose using classical algorithms. The classical Singular Value Decomposition (SVD) requires $O(mn \min(m, n))$ floating-point operations. For massive datasets, this cubic scaling is prohibitive.

Randomized SVD (RSVD) rests on a profound geometric concentration phenomenon: Random vectors in high-dimensional spaces tend to be nearly orthogonal to any fixed low-dimensional subspace, but they capture the energy of the dominant subspace with high probability.

The "Range Finder" Intuition

If a matrix A has exact rank k, its columns span a k-dimensional subspace of $\mathbb{R}^m$. If we multiply A by a set of k random vectors $\omega_1, \ldots, \omega_k$, the resulting vectors $y_i = A\omega_i$ will essentially be random linear combinations of the basis vectors of the column space of A. With probability 1 (for Gaussian vectors), these y_i vectors will be linearly independent and will span the exact same column space as A.

However, real-world matrices are not exactly rank-k, but have a fast spectral decay. This means the singular values $\sigma_1, \sigma_2, \ldots$ drop off quickly. In this scenario, applying A to random vectors acts as a filter:

$$y = A\omega = \sum_{i=1}^{n} \sigma_i u_i (v_i^T \omega)$$

Since ω is random, the scalar projection $v_i^T \omega$ spreads energy across all directions roughly equally. However, the weighting by σ_i amplifies the directions corresponding to the large singular values (the "signal") and suppresses the small singular values (the "noise"). Consequently, the vectors $Y = [y_1, \ldots, y_\ell]$ align themselves with the dominant left singular vectors $u_1, \ldots, u_k$.

1.2 Algorithm Description

The RSVD algorithm is a "matrix decomposition" technique that splits the problem into a probabilistic step (finding the subspace) and a deterministic step (exact SVD on a small matrix).

Inputs

- Matrix $A \in \mathbb{R}^{m \times n}$
- Target rank k
- Oversampling parameter p (usually $5 \leq p \leq 10$).
- Total random vectors $\ell = k + p$.

Step 1: Gaussian Test Matrix Generation

We draw a Gaussian random matrix $\Omega \in \mathbb{R}^{n \times \ell}$ such that every entry $\Omega_{ij} \sim \mathcal{N}(0, 1)$.

Justification: We choose the standard normal distribution because it is rotationally invariant. This means the algorithm's performance does not depend on the specific basis in which the input matrix A is represented. Other distributions (like Rademacher variables ± 1) work but require slightly more complex theoretical analysis.

Step 2: Sampling the Range (The Sketch)

Compute the sketch matrix Y:

$$Y = A\Omega \in \mathbb{R}^{m \times \ell}$$

This step dominates the computational cost if implemented naively. In practice, this is a dense matrix-matrix multiplication (GEMM).

Effect: Y is a "compressed" representation of the column space of A. If we define the SVD of $A = U\Sigma V^T$, then $Y \approx U_\ell \Sigma_\ell (V_\ell^T \Omega)$.

Step 3: Orthonormalization (Basis Generation)

We compute the QR factorization of Y:

$$Y = QR$$

where $Q \in \mathbb{R}^{m \times \ell}$ is a matrix with orthonormal columns, and R is upper triangular.

Crucial Detail: We discard R. We only need Q.

Why is this necessary? Without orthonormalization, the columns of Y would become increasingly collinear (parallel to the principal singular vector u_1) as the singular value gaps increase, leading to severe floating-point errors. Q provides a numerically stable basis for the range of Y.

Step 4: Projection to Low-Dimension

We form the projected matrix B:

$$B = Q^T A \in \mathbb{R}^{\ell \times n}$$

Dimension Reduction: We have reduced the dimension from m (which could be millions) to ℓ (usually hundreds).

Approximation: Since Q captures the range of A, $A \approx QQ^T A = QB$.

Step 5: Deterministic SVD

Compute the SVD of the small matrix B:

$$B = \tilde{U}\Sigma V^T$$

where $\tilde{U} \in \mathbb{R}^{\ell \times \ell}$ and $V \in \mathbb{R}^{n \times n}$.

Efficiency: This step is extremely fast because ℓ is small. $O(\ell^2 n)$.

Step 6: Lifting (Reconstruction)

We construct the approximate left singular vectors of A by transforming $\tilde{U}$ back to the original coordinate system using Q:

$$U = Q\tilde{U} \in \mathbb{R}^{m\times \ell}$$

The approximate SVD is then:

$$A \approx U\Sigma V^T$$

We typically truncate this to the top k components to return the rank-k approximation.

1.3 Theoretical Analysis and Proofs

This section provides the mathematical justification for why the randomized approach yields an accurate approximation. We rely on the seminal analysis by Halko, Martinsson, and Tropp (2011).

1.3.1 The Objective

We want to bound the approximation error $\|A - QQ^T A\|$ (where $\|\cdot\|$ denotes either the spectral norm $\|\cdot\|_2$ or Frobenius norm $\|\cdot\|_F$). Here, $P_Q = QQ^T$ is the orthogonal projector onto the range of $Y = A\Omega$.

1.3.2 Partitioning the Spectrum

Let the exact SVD of A be partitioned into the "target" k components and the "tail":

$$A = U\Sigma V^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix}$$

- U_1 contains the top k left singular vectors.
- Σ_1 contains $\sigma_1, \dots, \sigma_k$.
- Σ_2 contains the tail singular values $\sigma_{k+1}, \dots, \sigma_{\min(m,n)}$.

Ideally, the range of our test matrix Q would align perfectly with U_1.

1.3.3 Projection Analysis

Let Ω be partitioned to match V:

$$\Omega_1 = V_1^T\Omega \quad \text{and} \quad \Omega_2 = V_2^T\Omega$$

The sketch matrix Y can be written in the basis of U:

$$Y = A\Omega = U\Sigma V^T\Omega = U_1\Sigma_1\Omega_1 + U_2\Sigma_2\Omega_2$$

Intuitively, if Σ_2 is small (fast decay) and Ω_1 is well-conditioned (which Gaussian matrices are, with high probability), then the range of Y is dominated by U_1.

1.3.4 The Deterministic Error Bound

If Ω_1 has full row rank (which is true with probability 1 since $\ell \geq k$), the error can be structurally bounded by:

$$\|A - QQ^T A\|^2 \leq \|\Sigma_2\|^2 + \|\Sigma_2 \Omega_2 \Omega_1^\dagger\|^2$$

where $\Omega_1^\dagger$ is the Moore-Penrose pseudoinverse of Ω_1.

The first term $\|\Sigma_2\|$ is the theoretically optimal error (the baseline). The second term represents the "penalty" for using randomization. It depends on the tail energy Σ_2 and the ratio of the norms of the random blocks.

1.3.5 Probabilistic Average-Case Bound

Taking the expectation over the Gaussian distribution of Ω, we obtain specific bounds.

For a target rank k and oversampling parameter $p \geq 2$:

$$\mathbb{E}\|A - A_k^{\text{rsvd}}\|_2 \leq \left(1 + \frac{4\sqrt{k+p}}{p-1}\sqrt{\min(m,n)}\right)\sigma_{k+1}$$

While this looks complex, the intuition is simpler. As we increase the oversampling p, the error factor approaches 1 rapidly.

$$\mathbb{E}\|A - A_k^{\text{rsvd}}\|_F \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}$$

1.3.6 The Role of Power Iterations

In practice, if the singular values do not decay rapidly (the spectrum is flat), the mixing of U_1 and U_2 in the sketch Y is strong. To fix this, we apply power iterations.

We replace $Y = A\Omega$ with:

$$Y^{(q)} = (AA^T)^q A\Omega$$

Analytically, this is equivalent to sketching the matrix:

$$B = A(A^T A)^q$$

The singular values of this matrix are σ_j^{2q+1}.

If the ratio between the signal and noise singular values was $\frac{\sigma_k}{\sigma_{k+1}}$, after q iterations, the ratio becomes $(\frac{\sigma_k}{\sigma_{k+1}})^{2q+1}$. Even a small gap is amplified exponentially, forcing the range of Y to align almost perfectly with U_1.

The error bound with q iterations becomes:

$$\mathbb{E}\|A - A_{k,q}^{\text{rsvd}}\|_2 \leq (1+\epsilon)^{1/(2q+1)}\,\sigma_{k+1}$$

As q increases, the error converges to the theoretically optimal σ_{k+1}.

1.4 Complexity Analysis

A rigorous analysis of computational complexity demonstrates why RSVD is superior to deterministic methods for low-rank approximations. We analyze the Floating Point Operations (FLOPs) required.

Assumptions:

- Matrix A has dimensions $m \times n$.
- We assume $m \geq n$ (if $n > m$, the analysis is symmetric).
- Target rank k and oversampling p define the sketch size $\ell = k + p$.
- We assume $\ell \ll n$, meaning we are looking for a significant dimension reduction.

1.4.1 Step-by-Step Cost Breakdown

- **Generation of Test Matrix (Ω):** Generating an $n \times \ell$ matrix of Gaussian random variables. Cost: $O(n\ell)$. Note: This is computationally negligible compared to matrix multiplication.
- **Range Sampling ($Y = A\Omega$):** This is a dense matrix multiplication of $(m \times n)$ by $(n \times \ell)$. Cost: $C_{\text{mult}} = 2mn\ell$ FLOPs. Significance: This is usually the dominant cost in the "one-pass" algorithm (where $q = 0$).
- **Orthonormalization ($Y = QR$):** QR factorization of an $m \times \ell$ matrix using Householder reflectors or Modified Gram-Schmidt. Cost: $C_{\text{qr}} \approx 2m\ell^2$ FLOPs. Significance: Since $\ell \ll n$, this term $m\ell^2$ is much smaller than $mn\ell$.
- **Projection ($B = Q^T A$):** Matrix multiplication of $(\ell \times m)$ by $(m \times n)$. Cost: $C_{\text{proj}} = 2mn\ell$ FLOPs.
- **SVD of Small Matrix (B):** SVD of an $\ell \times n$ matrix. Cost: $C_{\text{svd}} \approx O(n\ell^2)$.
- **Lifting ($U = Q\tilde{U}$):** Matrix multiplication of $(m \times \ell)$ by $(\ell \times \ell)$. Cost: $O(m\ell^2)$.

1.4.2 Total Complexity Comparison

Summing the leading terms, the total cost for Basic RSVD ($q = 0$) is:

$$T_{\text{RSVD}} \approx 4mn\ell + O(m\ell^2 + n\ell^2)$$

Since $\ell \ll n$, the $O(mn\ell)$ term dominates.

Comparison with Classical Deterministic SVD: Golub-Reinsch SVD: Requires bidiagonalization followed by QR iteration. Cost: $O(mn^2)$.

Speedup Factor:

$$\frac{\text{Classical}}{\text{RSVD}} \approx \frac{mn^2}{mn\ell} = \frac{n}{\ell}$$

If $n = 10,000$ and we want the top $k = 100$ components (with $\ell \approx 110$), RSVD is roughly 100x faster.

1.4.3 Complexity with Power Iterations

If we perform q power iterations to improve accuracy, we perform q additional multiplications by A and A^T. Each iteration adds: $2 \times (mn\ell)$ FLOPs.

Total Cost ($q > 0$):

$$T_{\text{RSVD}}^{(q)} \approx 2(2q+2)mn\ell = O(qmn\ell)$$

Even with $q = 2$ or $q = 3$, the algorithm remains linear in m and n, preserving the massive advantage over the cubic/quadratic scaling of classical SVD.

1.5 Approximation Quality and Error Intuition

While Section 3.3 provided the raw probability bounds, this section focuses on the Eckart-Young-Mirsky theorem and the geometric intuition of why the error behaves as it does.

1.5.1 The Baseline: Eckart-Young-Mirsky Theorem

This theorem establishes the fundamental limit of low-rank matrix approximation. For any matrix A and any matrix A_k of rank at most k:

$$\min_{\text{rank}(X) \le k} \|A - X\|_2 = \sigma_{k+1}$$

No algorithm can beat this error in the spectral norm. The goal of RSVD is to produce an approximation $\hat{A}_k$ such that:

$$\|A - \hat{A}_k\|_2 \approx \sigma_{k+1}$$

1.5.2 Interpretation of Oversampling (p)

The oversampling parameter p acts as a safety buffer.

The Problem: If we pick exactly k random vectors, there is a non-zero probability that one of our random vectors falls orthogonal to one of the top k principal components (singular vectors) of A. If this happens, we "miss" that component entirely.

The Solution: By selecting $\ell = k + p$ vectors, we span a slightly larger subspace. The probability that none of the ℓ random vectors capture the energy of the i-th singular vector decreases exponentially with p.

Result: A small p (e.g., $p = 5$ or $p = 10$) is sufficient to make the failure probability negligible ($< 10^{-7}$).

1.5.3 Intuition for Power Iterations (q)

Power iterations are necessary when the singular value spectrum decays slowly (the "flat spectrum" problem). Consider the ratio of the "signal" (the k-th singular value) to the "noise" (the $(k+1)$-th singular value). Let $\gamma = \frac{\sigma_{k+1}}{\sigma_k}$.

- If $\gamma \approx 0$ (fast decay), the signal is distinct.
- If $\gamma \approx 1$ (slow decay), the signal is hard to distinguish from the tail.

When we compute $Y = (AA^T)^q A\Omega$, we are effectively sampling from a matrix with singular values σ_i^{2q+1}. The new ratio becomes:

$$\gamma_{\text{new}} = \left(\frac{\sigma_{k+1}}{\sigma_k}\right)^{2q+1}$$

Example: If $\sigma_k = 1.0$ and $\sigma_{k+1} = 0.9$ (a difficult case):

- $q = 0$: Ratio is 0.9. Separation is poor.
- $q = 2$: Exponent is 5. Ratio is $(0.9)^5 \approx 0.59$. Separation improves.
- $q = 5$: Exponent is 11. Ratio is $(0.9)^{11} \approx 0.31$. Separation is excellent.

By artificially sharpening the decay of the singular values, we force the random vectors to align with the top k singular vectors much faster.

1.6 Implementation Details (Python)

Implementing RSVD requires attention to numerical stability, specifically regarding the precision of floating-point operations and the order of matrix multiplications.

1.6.1 Key Design Choices

- **BLAS Level 3 Operations:** In Python, using `numpy.dot` or the `@` operator calls underlying BLAS (Basic Linear Algebra Subprograms) routines. These are highly optimized for modern CPU caches.
- **QR Factorization Mode:** We use `mode='reduced'` (or economic). We do not want the full $m \times m$ Q matrix, only the $m \times \ell$ active columns.
- **Numerical Stability:** For the Power Iteration, repeating $Y = A(A^T Y)$ is susceptible to round-off error if q is large. In very high-precision requirements, one might orthonormalize between iterations, but for standard ML applications ($q < 5$), direct multiplication is sufficient and faster.

1.6.2 Python Implementation Code

```
import numpy as np

def rsvd(A, k, p=10, q=2):
    """
    Computes the Randomized SVD of matrix A.

    Args:
        A (np.ndarray): Input matrix of shape (m, n).
        k (int): Target rank.
        p (int): Oversampling parameter.
        q (int): Number of power iterations.
    Returns:
        U_k (np.ndarray): Top k left singular vectors (m, k).
        S_k (np.ndarray): Top k singular values (k,).
        Vt_k (np.ndarray): Top k right singular vectors (k, n).
    """
    m, n = A.shape
    ell = k + p

    # 1. Generate Random Test Matrix
    # We use standard normal distribution.
    Omega = np.random.randn(n, ell)

    # 2. Sample the Range (with Power Iteration)
    # Initial projection
    Y = A @ Omega

    # Power Iterations to separate signal from noise
    for _ in range(q):
        Y = A @ (A.T @ Y)

    # 3. Orthonormalization (QR Decomposition)
    # 'reduced' ensures Q is (m, ell), not (m, m)
    Q, _ = np.linalg.qr(Y, mode='reduced')

    # 4. Project A into low-dimensional subspace
    # B is small: (ell, n)
    B = Q.T @ A

    # 5. Deterministic SVD on small matrix B
    # full_matrices=False ensures we get compact SVD
    U_tilde, S, Vt = np.linalg.svd(B, full_matrices=False)

    # 6. Lift singular vectors back to original space
    U = Q @ U_tilde

    # 7. Truncate to rank k
    U_k = U[:, :k]
    S_k = S[:k]
    Vt_k = Vt[:k, :]

    return U_k, S_k, Vt_k

# Example Usage Verification
if __name__ == "__main__":
    # Create a synthetic low-rank matrix
```

```
m, n = 1000, 500
r = 20 # True rank

# Generate random factors
U_true = np.random.randn(m, r)
V_true = np.random.randn(n, r)
# Force orthogonality for clearer singular values
U_true, _ = np.linalg.qr(U_true)
V_true, _ = np.linalg.qr(V_true)

# Singular values decay
S_true = np.linspace(100, 1, r)

# Construct A
A = (U_true * S_true) @ V_true.T

# Add some noise
A += 0.01 * np.random.randn(m, n)

# Run RSVD
k_target = 20
U_approx, S_approx, Vt_approx = rsvd(A, k=k_target, p=10, q=2)

# Check reconstruction error
A_approx = (U_approx * S_approx) @ Vt_approx
error = np.linalg.norm(A - A_approx) / np.linalg.norm(A)
print(f"Relative Reconstruction Error: {error:.5f}")
```

1.6.3 Memory Management Note

For extremely large matrices that do not fit in RAM, standard NumPy cannot be used. In such cases, this algorithm can be adapted for Out-of-Core processing or distributed systems (e.g., PySpark or Dask). The logic remains identical, but the matrix multiplications $A\Omega$ and $Q^T A$ are performed by streaming blocks of A from disk.

1.7 Experimental Workflow for RSVD

To rigorously validate the theoretical claims of RSVD, we must design an experimental framework that isolates the effects of rank (k), oversampling (p), power iterations (q), and matrix structure (spectral decay).

1.7.1 Dataset Generation (Matrix Families)

The performance of RSVD depends heavily on the singular value spectrum of the input matrix. We categorize test matrices into three distinct families:

Fast Decay (Ideal Case):

- **Description:** Matrices where singular values decay exponentially ($\sigma_j \approx e^{-\alpha j}$).
- **Source:** Real-world datasets like "Eigenfaces" (face recognition), user-item recommendation matrices, and certain physical simulations (e.g., heat diffusion).
- **Expectation:** RSVD should achieve near-optimal error with $q = 0$ or $q = 1$.

Slow Decay (The "Heavy Tail"):

- **Description:** Matrices where singular values decay polynomially ($\sigma_j \approx j^{-1}$).
- **Source:** Large-scale graph Laplacians (social networks), term-document matrices in NLP (Zipf's law).
- **Expectation:** Basic RSVD ($q = 0$) will likely perform poorly compared to deterministic SVD. This is the critical test case for Power Iterations ($q \geq 2$).

Structured Random Matrices (Worst Case):

- **Description:** Dense Gaussian matrices scaled by magnitude.
- **Source:** Synthetic noise benchmarks.
- **Expectation:** These have a "flat" spectrum where $\sigma_k \approx \sigma_{k+1}$. RSVD helps with speed, but accuracy gains are harder to visualize because the "true" low-rank approximation is mathematically ill-defined (no clear separation between signal and noise).

1.7.2 Metrics for Evaluation

We quantify performance using two primary dimensions: Accuracy and Efficiency.

1. Relative Reconstruction Error: We measure how close the RSVD approximation is to the original matrix relative to its energy.

$$E_{\text{rel}} = \frac{\|A - U_k \Sigma_k V_k^T\|_F}{\|A\|_F}$$

Why Frobenius norm? It is computationally cheaper to estimate than the spectral norm for large matrices.

2. Approximation Ratio (Competitive Analysis): We compare RSVD against the "Gold Standard" (deterministic truncated SVD computed via ARPACK/scipy.sparse.linalg.svds).

$$\rho = \frac{\|A - A_k^{\text{rsvd}}\|_F}{\|A - A_k^{\text{optimal}}\|_F}$$

- $\rho = 1.0$: RSVD is perfect.
- $\rho \approx 1.05$: RSVD is within 5% of the optimal error.

3. Speedup Factor:

$$S = \frac{T_{\text{deterministic}}}{T_{\text{rsvd}}}$$

We measure wall-clock time, ensuring we exclude the time taken to generate the synthetic data.

1.7.3 Experimental Procedure

For each matrix family and size $N \in \{1000, 5000, 10000\}$:

- **Baseline:** Compute Exact SVD once to get ground truth singular values and vectors.
- **Grid Search:** Run RSVD varying:
  - Target rank $k \in \{10, 50, 100\}$.
  - Oversampling $p \in \{0, 5, 10, 20\}$.
  - Power Iterations $q \in \{0, 1, 2, 4\}$.
- **Plotting:**
  - **Plot 1 (Convergence):** Error vs. Oversampling (p). Hypothesis: Error drops sharply until $p = 10$, then plateaus.
  - **Plot 2 (Robustness):** Error vs. Power Iterations (q) for "Slow Decay" matrices. Hypothesis: Significant error reduction from $q = 0$ to $q = 2$, diminishing returns afterwards.
  - **Plot 3 (Scalability):** Runtime vs. Matrix Size (n). Hypothesis: RSVD scales linearly; Deterministic scales super-linearly.

1.8 Practical Applications of RSVD

RSVD is not just a theoretical improvement; it is the engine behind many modern large-scale machine learning systems.

1.8.1 Dimensionality Reduction (PCA)

Principal Component Analysis (PCA) is mathematically equivalent to SVD on the centered covariance matrix.

- **Challenge:** In genomics (e.g., single-cell RNA sequencing), we often have matrices with 10^5 genes (features) and 10^5 cells (samples). Computing full covariance is $O(n^2)$, which is impossible.
- **RSVD Solution:** We can apply RSVD directly to the data matrix X to get the top principal components without ever forming the covariance matrix $X^T X$. This allows PCA on massive biological datasets.

1.8.2 Image Compression and Denoising

- **Compression:** An image is a matrix of pixel intensities. By keeping only the top $k = 50$ singular values, we can store a high-resolution image with 10-20% of the original file size.
- **Denoising:** Noise in images typically manifests as high-frequency, low-energy variations. These naturally fall into the "tail" of the singular value spectrum. RSVD acts as a filter: by reconstructing the image using only the top k components, we inherently discard the noise (which lives in the range of $\sigma_{k+1} \dots \sigma_n$).

1.8.3 Latent Semantic Analysis (LSA) in NLP

- **Context:** Document-Term matrices (Bag of Words) are massive and sparse.
- **Application:** RSVD extracts "topics" (singular vectors) representing clusters of related words. Because RSVD only requires matrix-vector products, it can exploit the sparsity of the input matrix ($A\Omega$ is fast if A is sparse), making it far more efficient than dense SVD solvers.

1.8.4 Pre-processing for Matrix Multiplication

This is the specific use case for your broader project.

- **Problem:** We want to multiply two massive matrices $A \times B$.
- **Strategy:**
  1. Approximate $A \approx U_A \Sigma_A V_A^T$ using RSVD (rank k).
  2. Approximate $B \approx U_B \Sigma_B V_B^T$ using RSVD (rank k).
  3. Compute $A \times B \approx (U_A \Sigma_A V_A^T)(U_B \Sigma_B V_B^T)$.
- **Benefit:** Instead of $O(n^3)$ operations, we perform multiplications on the smaller factor matrices ($O(nk^2)$), achieving massive speedups when $k \ll n$.

1.9 Conclusion

Randomized SVD represents a paradigm shift in numerical linear algebra. It moves away from the 20th-century focus on deterministic, worst-case guarantees (like full QR iteration) toward a probabilistic, average-case perspective suitable for the Big Data era.

1.9.1 Summary of Key Findings

- **Efficiency:** RSVD reduces the complexity of low-rank approximation from cubic $O(mn^2)$ to linear $O(mnk)$ (in terms of input size). This makes it feasible to process matrices that were previously intractable.
- **Accuracy:** Theoretical bounds and empirical evidence confirm that with modest oversampling ($p = 10$) and minimal power iterations ($q = 2$), RSVD produces approximations that are indistinguishable from the optimal truncated SVD for all practical purposes.
- **Simplicity:** The algorithm is structurally simple—relying primarily on matrix multiplication and standard QR/SVD on small matrices. This makes it highly parallelizable and suitable for GPU acceleration.