ChatGPT

Singular Value Decomposition (SVD): Mathematical Foundations and Applications

Singular Value Decomposition (SVD) is a fundamental matrix factorization that appears throughout machine learning. For any real $m \times n$ matrix A , the SVD writes

$$A \; = \; U \, \Sigma \, V^T,$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{m \times n}$ is diagonal with nonnegative entries [1] . The diagonal entries $\sigma_1, \sigma_2, \ldots$ of Σ are the *singular values* of A (sorted $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$), and the columns of U and V are the *left* and *right singular vectors*, respectively [2] [3] . Importantly, A always has an SVD, and the number of nonzero singular values equals the rank of A [2] . In practice, one often computes a *reduced* SVD by truncating to the first r nonzero singular values (with $r = \mathrm{rank}(A)$), yielding $A \approx U_r \Sigma_r V_r^T$ of much smaller size.

Mathematical Derivation of SVD

The SVD can be understood via the spectral decomposition of $A^T A$. Let A be real $m \times n$ and consider the symmetric matrix $A^T A$ (size $n \times n$). Because $A^T A$ is symmetric, it has an eigen-decomposition $A^T A = V \Lambda V^T$. One shows that the eigenvalues λ_i of $A^T A$ are all nonnegative, and their square roots are the singular values of A [4] . Concretely, let

$$A^T A \, v_i \; = \; \lambda_i \, v_i, \qquad \text{with unit-norm eigenvectors } v_i.$$

Then $\sigma_i = \sqrt{\lambda_i}$ and $A v_i = \sigma_i u_i$ for some unit vector u_i (if $\sigma_i > 0$) [3] . In fact, u_i can be taken as

$$u_i = \frac{1}{\sigma_i} A \, v_i, \qquad \sigma_i > 0.$$

Thus one column at a time:

$$A v_i = \sigma_i u_i, \quad i = 1, \ldots, r,$$

where r is the number of nonzero singular values. Stacking these as matrices gives

$$A \, V_r \; = \; U_r \, \Sigma_r,$$

where $V_r = [v_1, \ldots, v_r]$, $U_r = [u_1, \ldots, u_r]$, and $\Sigma_r = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$ [5] . Extending U_r, V_r with additional orthonormal columns to full square orthonormal U, V , and padding Σ_r with zeros, yields the full SVD $A = U \Sigma V^T$ [6] . Equivalently, the SVD provides a rank-one decomposition

$$A \; = \; \sum_{i=1}^{r} \sigma_i \, u_i v_i^T,$$

with each term $u_i v_i^T$ an outer product (rank-1 matrix) [6].

In summary, the **computational procedure** is: - Compute $A^T A$ (or AA^T) and find its eigenpairs (λ_i, v_i) .
- Set $\sigma_i = \sqrt{\lambda_i}$. These are the singular values.
- Form $V = [v_1, \ldots, v_n]$ from orthonormal eigenvectors of $A^T A$.
- Compute $u_i = Av_i/\sigma_i$ for $\sigma_i > 0$ to get the left singular vectors, forming $U = [u_1, \ldots, u_m]$.
- The diagonal matrix Σ has $\sigma_1, \ldots, \sigma_r$ on its leading diagonal (rest zeros).

This process yields $A = U\Sigma V^T$. (Any zero singular values correspond to zero columns in Σ and arbitrary orthonormal completions of U, V .) These constructions guarantee $U^T U = I$, $V^T V = I$ and $Av_i = \sigma_i u_i$, $A^T u_i = \sigma_i v_i$ for all i [5] [4]. Thus, each left singular vector u_i is an eigenvector of AA^T and each right singular vector v_i is an eigenvector of $A^T A$. In fact, one can show $A^T A = V\, \Sigma^T \Sigma\, V^T$, so $\Sigma^T \Sigma$ contains the eigenvalues of $A^T A$ [7].

Geometric Interpretation

Geometrically, SVD decomposes any linear transformation $A : \mathbb{R}^n \to \mathbb{R}^m$ into a sequence of a rotation, a stretching, and another rotation. Concretely, V^T rotates or reflects the input space, Σ scales each coordinate axis by σ_i , and U rotates/reflects to the output space. Equivalently, A maps the unit sphere in $\mathbb{R}^n$ to an ellipsoid in $\mathbb{R}^m$ whose principal axes lengths are the singular values. Intuitively, each singular value σ_i is the factor by which A stretches the corresponding direction v_i into direction u_i [8] [9].

More precisely, because U and V are orthonormal, their columns form orthonormal bases of $\mathbb{R}^m$ and $\mathbb{R}^n$ respectively [10]. In this SVD-basis, A acts as

$$A(v_i) \;=\; A(Ve_i) \;=\; U\Sigma V^T(Ve_i) \;=\; \sigma_i u_i,$$

for $i = 1, \ldots, \min(m, n)$. In words, A sends the i -th basis vector of the input basis (given by v_i) to the i -th basis vector of the output basis (given by u_i), scaled by σ_i [9]. All other vectors orthogonal to these bases are sent to 0 if beyond rank. Thus U and V align the coordinate systems so that A simply *stretches* along each coordinate by the singular values [8] [9].

As a consequence, the first r columns of U form an orthonormal basis for the column space of A (the span of its outputs), and the first r columns of V form an orthonormal basis for the row space of A (the span of its inputs) [11]. (The remaining columns of U and V span the left and right nullspaces, respectively.) In summary: A can be viewed as rotating into the V -basis, scaling by Σ , then rotating into the U -basis [8] [9].

Algorithms for Computing SVD

Computing the full SVD of a dense $m \times n$ matrix (e.g. via LAPACK's bidiagonalization) costs $O(\min(mn^2, m^2 n))$ in general and yields all singular values and vectors. However, for large data applications one often needs only a few top singular values/vectors (a *truncated SVD*). Several approaches exist:

- **Power iteration (for top singular vector)**: One can compute the largest singular value σ_1 and corresponding singular vectors u_1, v_1 by iterating on $A^T A$ (or AA^T). Starting with a random unit vector $x_0 \in \mathbb{R}^n$, repeatedly set

  $$x_{k+1} \ \propto \ A^T A \, x_k.$$

  In the limit $x_k \to v_1$ (the principal right singular vector) and $\|Ax_k\| \to \sigma_1$ [12] [13] . In fact the algorithm is:
- Initialize x_0 randomly.
- Iterate $x_k \leftarrow A^T A \, x_{k-1}$, then normalize $v = x_k / \|x_k\|$.
- Set $\sigma_1 = \|Av\|$, $u_1 = (Av)/\sigma_1$.

- Return σ_1, u_1, v .
  This yields the dominant singular triplet (σ_1, u_1, v_1) [12] [13] . To find more vectors one can deflate or perform block methods. Note that this converges slowly if singular values are close.

- **Lanczos/ARPACK (Truncated SVD)**: Efficient libraries (e.g. ARPACK, PROPACK) compute a few largest singular values by working implicitly with $A^T A$ or via a Lanczos bidiagonalization of A . In Python, for example, one can call `scipy.sparse.linalg.svds(A, k)` which uses ARPACK to compute the top k singular values and vectors. The scikit-learn `TruncatedSVD` transformer likewise uses ARPACK or a randomized solver under the hood [14] . These methods are much faster than full SVD when $k \ll \min(m, n)$, especially for sparse or large matrices.

- **Randomized SVD**: Modern algorithms project A onto a low-dimensional random subspace to capture its range, then compute SVD in that subspace. The randomized SVD (e.g. Halko–Martinsson–Tropp method) is highly efficient for very large matrices. It involves generating a random Gaussian matrix $\Omega \in \mathbb{R}^{n \times (k+p)}$, computing $Y = A\Omega$, orthonormalizing $Y \to Q$, and then computing the smaller SVD of $Q^T A$ [14] . Power iterations can be added for accuracy. This yields an approximate rank-k SVD faster than deterministic methods. Libraries like scikit-learn and `scipy.sparse.linalg.randomized_svd` implement this approach with options for oversampling and iterations [14] .

- **Direct Dense Methods**: For smaller matrices, standard LAPACK routines (using bidiagonal reduction) compute the full SVD. These are deterministic and highly accurate, but scale cubically in dimension. Typically one uses such methods (via `numpy.linalg.svd` or similar) when the matrix fits in memory.

In summary, one chooses an SVD algorithm based on matrix size and sparsity. Power iteration is simple for one or a few singular vectors. Lanczos/ARPACK is effective for up to hundreds of vectors. Randomized SVD excels on huge, noisy data when only approximate low-rank structure is needed [14] .

Python Code Examples: NumPy and SciPy

Below are illustrative code snippets demonstrating full and truncated SVD in Python. Each code block is annotated to explain its steps and output.

```python
import numpy as np
from scipy.sparse.linalg import svds

# Example matrix
A = np.array([[2.0, 0.0, 0.0],
              [2.0, 1.0, 0.0],
              [0.0,-2.0, 0.0]])
# Compute full SVD using NumPy (dense)
U, s, Vt = np.linalg.svd(A, full_matrices=True)
Sigma = np.zeros_like(A, dtype=float)       # form Sigma matrix
Sigma[:len(s), :len(s)] = np.diag(s)       # place singular values on diag

print("Singular values (full):", s)
print("U matrix:\n", U)
print("Sigma matrix:\n", Sigma)
print("V^T matrix:\n", Vt)
# Verify reconstruction: U @ Sigma @ V^T should equal A
A_recon = U.dot(Sigma).dot(Vt)
print("Reconstruction error:", np.max(np.abs(A - A_recon)))
```

- We use `np.linalg.svd` to compute the full SVD of matrix `A`. It returns `U`, the singular values `s`, and `Vt = V^T`.
- We construct the diagonal matrix `Sigma` from `s` (zero elsewhere).
- The printout shows that `U` and `V^T` are orthonormal and `s` are nonnegative.
- Finally, we reconstruct `A` as `U @ Sigma @ V^T` and check the maximum absolute error is (near) zero, verifying $A = U\Sigma V^T$.

```python
# Truncated SVD: compute top-2 singular values/vectors using ARPACK (scipy)
k = 2
U2, s2, Vt2 = svds(A, k=k)  # svds returns singular values in ascending order
# Sort in descending order for consistency
idx = np.argsort(s2)[::-1]
s2 = s2[idx]
U2 = U2[:, idx]
Vt2 = Vt2[idx, :]

print("Top-2 singular values (truncated):", s2)
print("U[:, :2] (approx):\n", U2)
print("V^T[:2, :] (approx):\n", Vt2)
# Reconstruct rank-2 approximation (here full rank anyway)
```

```
A2 = U2.dot(np.diag(s2)).dot(Vt2)
print("Error of rank-2 approximation:", np.max(np.abs(A - A2)))
```

- Here we use `scipy.sparse.linalg.svds` to get the top `k=2` singular triplets of `A`. Note that `svds` returns the `k` largest singular values but in ascending order, so we sort them descending.
- The printed `s2` shows the two largest singular values (which match the first two of the full SVD).
- `U2` contains the corresponding left singular vectors (columns), and `Vt2` the right singular vectors (rows).
- We then reconstruct the rank-2 approximation `A2 = U2 diag(s2) Vt2`. The error printed is zero here because `A` has rank 2. In general, truncating to `k<rank(A)` would yield an approximation.

These examples illustrate how to call and interpret SVD routines. One sees that the singular values from `svds` match those from the full SVD, and that the reconstruction `U Σ V^T` reproduces `A`. In practice, one can choose `k` to retain only significant singular values for dimensionality reduction.

Worked Example: SVD of a 3×3 Matrix

Let's compute the SVD of the example matrix

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & -2 & 0 \end{bmatrix}.$$

Step 1: Compute $A^T A$.

$$A^T A = \begin{bmatrix} 2 & 2 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 0 \end{bmatrix}^T \begin{bmatrix} 2 & 2 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 8 & 2 & 0 \\ 2 & 5 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Step 2: Find eigenvalues of $A^T A$. Solve $\det(A^T A - \lambda I) = 0$:

$$\det \begin{pmatrix} 8-\lambda & 2 & 0 \\ 2 & 5-\lambda & 0 \\ 0 & 0 & -\lambda \end{pmatrix} = -\lambda(\lambda-4)(\lambda-9) = 0.$$

Thus the eigenvalues are $\lambda_1 = 9,\ \lambda_2 = 4,\ \lambda_3 = 0$. The singular values are their square roots: $\sigma_1 = 3,\ \sigma_2 = 2,\ \sigma_3 = 0$. We order them descending in Σ.

Step 3: Compute right singular vectors v_i. Find eigenvectors of $A^T A$ for $\lambda_1 = 9$ and $\lambda_2 = 4$. Solving $(A^T A)v = 9v$ and normalizing yields

$$v_1 = \frac{1}{\sqrt{5}} \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}, \quad v_2 = \frac{1}{\sqrt{5}} \begin{pmatrix} 1 \\ -2 \\ 0 \end{pmatrix}.$$

For $\lambda_3 = 0$, one finds $v_3 = (0,0,1)^T$. Stack into $V = [v_1, v_2, v_3]$.

Step 4: Compute left singular vectors u_i . For each nonzero σ_i , compute $u_i = Av_i/\sigma_i$. For $\sigma_1 = 3$:

$$Av_1 = \begin{pmatrix} 2 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & -2 & 0 \end{pmatrix} \frac{1}{\sqrt{5}} \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{5}} \begin{pmatrix} 4 \\ \sqrt{5} \\ -2 \end{pmatrix}.$$

Then $u_1 = (Av_1)/3 = \frac{1}{3\sqrt{5}}(4,\ \sqrt{5},\ -2)^T$. Similarly, for $\sigma_2 = 2$:

$$Av_2 = \frac{1}{\sqrt{5}} \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{5}} \begin{pmatrix} 2 \\ -0 \\ 4 \end{pmatrix} = \frac{1}{\sqrt{5}}(2,\ 0,\ -4)^T,$$

so $u_2 = (Av_2)/2 = \frac{1}{2\sqrt{5}}(2,\ 0,\ -4)^T = \frac{1}{\sqrt{5}}(1,\ 0,\ -2)^T$. (One can check these are unit length.) For $\sigma_3 = 0$, we choose u_3 orthogonal to u_1, u_2 , e.g. $u_3 = (0, 0, 1)^T$. Thus $U = [u_1, u_2, u_3]$.

Step 5: Assemble U, Σ, V^T . We now have

$$U = \begin{pmatrix} \frac{4}{3\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ \frac{1}{3} & 0 & 1 \\ -\frac{2}{3\sqrt{5}} & -\frac{2}{\sqrt{5}} & 0 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad V^T = \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Finally, $A = U\,\Sigma\,V^T$. One can verify by multiplication that this identity holds (any unit-vector column of V is sent by A to the corresponding scaled U -column). The decomposition yields all details: A has rank 2 with singular values 3 and 2, and the above U, V, Σ satisfy $Av_i = \sigma_i u_i$ as expected [5] [6] .

Applications of SVD in Machine Learning

SVD underpins many techniques in data science and ML. Key applications include:

- **Dimensionality Reduction (PCA).** Principal Component Analysis is essentially SVD on centered data. The top singular values/vectors identify directions of greatest variance. By projecting data onto the first k singular vectors (those with largest σ_i), one obtains a best k -dimensional approximation (in least-squares sense) [15] . This reduces noise and compresses data while preserving most important features. In practice, one computes $X = U\Sigma V^T$ for data matrix X and retains only large σ_i (or uses sklearn's `PCA` which uses SVD internally) [15] .

- **Latent Semantic Analysis (LSA).** In NLP, one forms a term–document matrix X and computes its truncated SVD. The factorization $X \approx U_k \Sigma_k V_k^T$ uncovers latent topics: columns of U_k relate to "term features" and V_k to "document features" [16] . Keeping only the largest singular values captures major co-occurrence patterns among words and documents. This helps in identifying synonyms and improving search by representing text in a reduced semantic space [16] .

- **Image Compression.** Any image can be treated as a matrix of pixel intensities. SVD can approximate an image by keeping only the top singular values/vectors. For example, setting smaller σ_i to zero yields a low-rank approximation that preserves the main structure of the image [17] . In practice, storing U_k, Σ_k, V_k with $k \ll \min(m,n)$ greatly reduces storage. Many textbooks demonstrate that even a few tens of singular values can yield visually good approximations. This is essentially how JPEG and other techniques exploit low-rank structure [17].

- **Recommender Systems (Collaborative Filtering).** User–item rating matrices are typically large, sparse, and noisy. SVD (or matrix factorization) reduces this matrix to a low-rank approximation that captures latent preferences. Concretely, if $R \approx U_k \Sigma_k V_k^T$, then $U_k \Sigma_k$ and $\Sigma_k V_k^T$ encode latent user and item features. The low-rank approximation can predict missing ratings by projecting users and items into the shared feature space [18]. This approach is famously used in systems like Netflix and Amazon to infer preferences and make recommendations [18].

- **Other Areas:** SVD is also used for noise reduction (signal denoising), solving ill-conditioned linear systems via pseudoinverse (the Moore–Penrose inverse is computed via SVD), and spectral clustering, among others. In all these cases, SVD provides a principled way to extract the dominant patterns (via large singular values) and discard weaker components (noise or redundancy) [15] [17] [18] [16].

Each of these applications leverages the fact that truncating the SVD to the largest k components yields the best rank-k approximation to the data. In machine learning, one typically chooses k by explained variance (fraction of $\sum \sigma_i^2$), cross-validation, or domain needs. The references above provide more context on these applications.

Sources: The mathematical and algorithmic details above are standard in linear algebra texts and machine learning references [3] [12] [13]. The geometry of SVD is summarized in many sources [8] [9]. Practical guidelines on truncated and randomized SVD appear in modern libraries and articles [14] [15]. The code examples use NumPy/SciPy documentation and typical usage patterns for illustrative purposes.

---

[1] [2] [8] [9] [10] [11] Singular value decomposition - Wikipedia
https://en.wikipedia.org/wiki/Singular_value_decomposition

[3] [4] [5] [6] [13] math.mit.edu
https://math.mit.edu/classes/18.095/2016IAP/lec2/SVD_Notes.pdf

[7] 4.4. Power iteration — MMiDS Textbook
https://mmids-textbook.github.io/chap04_svd/04_power/roch-mmids-svd-power.html

[12] cs.yale.edu
https://www.cs.yale.edu/homes/el327/datamining2013aFiles/07_singular_value_decomposition.pdf

[14] TruncatedSVD — scikit-learn 1.7.2 documentation
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

[15] [16] [17] [18] Applications of Singular Value Decomposition (SVD) — Understand The Math
https://www.understandthemath.com/blog/singular-value-decomposition