CS3220 Lecture Notes: Singular Value decomposition and applications

Steve Marschner
Cornell University

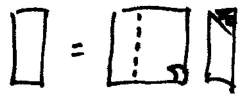
23 March–1 April 2009

This document is a transcription of the notes I have used to give CS322/3220 lectures on the SVD over the years, so it includes a bit more material than I've covered this year. It covers the SVD and what it is, and then applications to finding a matrix's fundamental subspaces, solving rank-deficient least squares problems, deciding matrix rank in the presence of noise, and in principal component analysis.

1 From QR to SVD

We've just been looking at an orthogonal matrix factorization that gives us an orthogonal factor on the left:

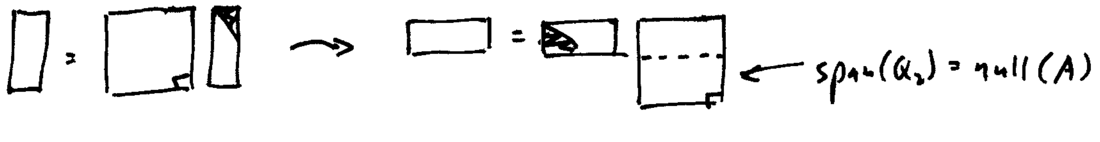
$$A = QR$$



By drawing in that dotted line we separate Q in to a basis for the range of A (the left part) and a basis for the rest of $\mathbb{R}^m$. This Q factor tells us about the *column* space of A.

When we wanted to know about the row space of A instead we factored A^T, resulting in something that might be called the "LQ factorization" (except that it isn't):

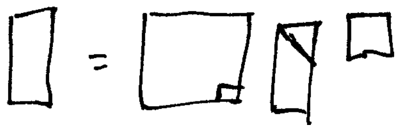
$$A^T = QR \quad \text{or} \quad A = R^T Q^T$$



In the full-rank case the row space of a tall matrix or the column space of a wide matrix are uninteresting, so we can pick one of these two factorizations.

But if the matrix is not full rank (it is rank deficient) we will be stuck choosing one or the other.

The next factorization we'll look at, the SVD, has orthogonal factors on both sides, and it works fine in the presence of rank deficiency:



$$A = U\Sigma V^T \quad (\text{or } AV = U\Sigma)$$

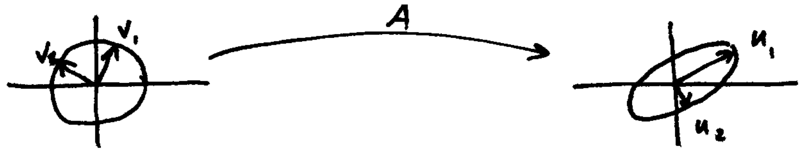
The SVD writes A as a product of two orthogonal transformations with a diagonal matrix (a scaling operation) in between. It says that we can replace any transformtion by a rotation from "input" coordinates into convenient coordinates, followed by a simple scaling operation, followed by a rotation into "output" coordinates. Furthermore, the diagonal scaling Σ comes out with its elements sorted in decreasing order.

2 SVD definitions and interpretation

The pieces of the SVD have names following the "singular" theme. The columns of U are the *left singular vectors* u_i; the entries on the diagonal of Σ are the *singular values*; and the columns of V (which are the rows of V^T are (you guessed it) the *right singular vectors* v_i.

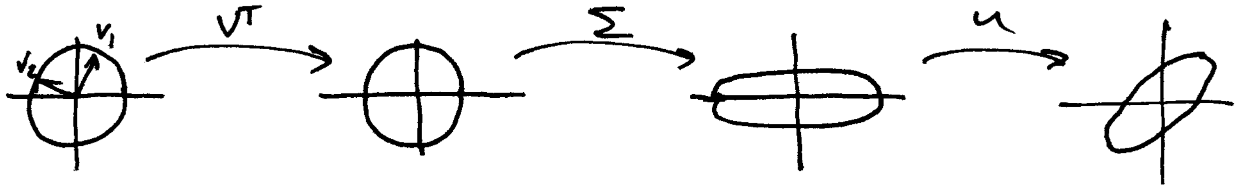
The SVD has a nice, simple geometric interpretation (see also Todd Will's SVD tutorial linked from the Readings page, which has a similar take). It's easiest to draw in 2D.

Let $U = \begin{bmatrix} u_1 & u_2 \end{bmatrix}$ and $V^T = \begin{bmatrix} v_1^T \\ v_2^T \end{bmatrix}$.



If we take the unit circle and transform it by A, we get an ellipse (because A is a linear transformation). The left singular vectors u_1, u_2 are the major and minor axes of that ellipse (being on the left they live in the "output" space). The right singular vectors v_1, v_2 are the vectors that get mapped to the major and minor axes (being on the right they live in the "input" space).

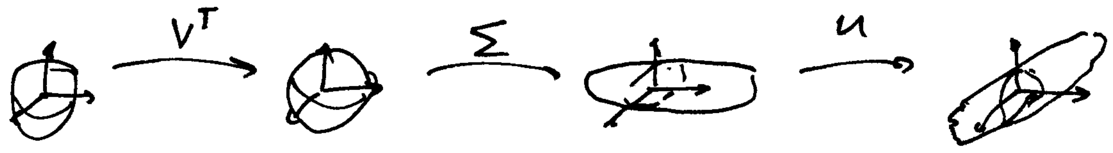
If we break the transformation down into these three stages we see a circle being rotated to align the vs with the coordinate axes, then scaled along those axes, then rotated to align the ellipse with the us:



Another way to say some of this is that $Av_i = \sigma_i u_i$, which you can see by this 2D example:

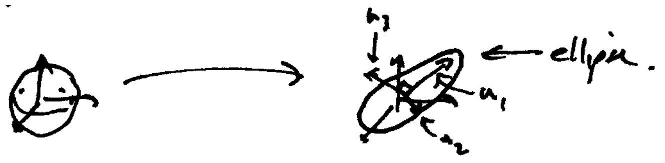
$$\begin{bmatrix} v_1^T \\ v_2^T \end{bmatrix} v_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad ; \quad U\Sigma \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \sigma_1 u_1$$

We can easily promote this idea to 3D, though it becomes harder to draw:



Let's look at what happens when we have a singular (aka. rank-deficient) matrix, in this 3×3 setting.

A rank-deficient matrix is one whose range is a subspace of $\mathbb{R}^3$, not all of $\mathbb{R}^3$, so it maps the sphere to a flat ellipse rather than an ellipsoid:



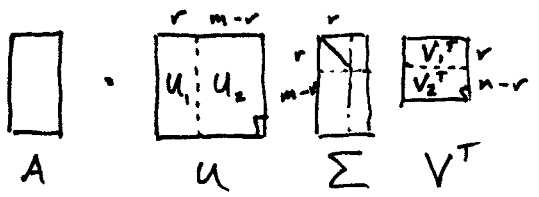
$$A = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix} \begin{bmatrix} a & & \\ & b & \\ & & 0 \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \end{bmatrix}$$

This means one of the singular values (the last one, since we sort them in decreasing order) is zero. The last left singular vector is the normal to that ellipse.

A rank-deficient matrix is also one that has a nontrivial null space: some direction that gets mapped to zero. In this case, that vector is v_3, since

$$V^T v_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \text{and} \quad \Sigma \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

More generally, the SVD of a rank-r matrix looks like this:



In this picture, $r < n < m$. So the matrix is rank-deficient ($r < n$) with an r-dimensional range and an $(n - r)$-dimensional null space.

3 Applications of the SVD

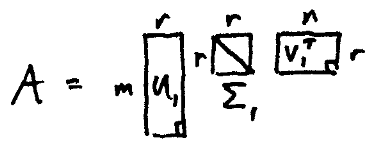
3.1 Fundamental subspaces of a matrix

From our rectangles-and-squares picture of the SVD, we can read off the four fundamental spaces:

- U_1 is a basis for $\text{ran}(A)$ (just like in QR) ($\dim = r$). After multiplication by Σ only the first k entries can be nonzero, so only vectors in $\text{span}(U_1)$ can be produced by multiplication with A.
- U_2 is a basis for $\text{ran}(A)^\perp$ ($\dim = m - r$). Since all the entries of $\Sigma V^T x$ corresponding to those columns of U are zero, no vectors in $\text{span}(U_2)$ can be generated by multiplication with A.
- V_2 is a basis for $\text{null}(A)$ ($\dim = n - r$). Since any vector in $\text{span}(V_2)$ will end up multiplying the zero singular values, it will get mapped to zero.
- V_1 is a basis for the row space of A ($\dim = r$).

Again, when the matrix is rank-deficient, all four spaces are nontrivial.

If we are just interested in a factorization of A from which we can reconstruct A we need only the first r singular vectors and singular values:



3.2 Rank-deficient least squares

As with most of our other linear algebra tools, SVD provides yet another way to solve linear systems.

$$Ax = b \quad x = V\Sigma^{-1}U^T b$$

No problem! It's just three matrix multiplications. The inverse of Σ is easy: it is just $\text{diag}(1/\sigma_1, 1/\sigma_2, \dots, 1/\sigma_n)$. This may be the simplest yet—but not the fastest. Note that back substitution takes less arithmetic than full matrix multiplication.

This process is nice and stable, and it's very clear exactly where the numbers can get bigger: only when they get divided by small singular values. It's also clear up front if the process will fail: it's when we divide by a zero singular value.

One of the strengths of the SVD is that it works when the matrix is singular. How would we go about solving a singular system? More specifically, how can we solve $Ax \approx b$, A square $n \times n$, rank $r < n$?

This is no longer a system of equations we can expect to have an exact solution, since $\text{ran}(A) \subset \mathbb{R}^n$ and b might not be in there. It *also* will not have a unique solution. If $Ax^* - b$ is minimum, so is $A(x^* + y) - b$ for any vector $y \in \text{null}(A)$. Thus a system like this is *both* overdetermined *and* underdetermined.

SVD gives us easy access to the solution space, though. The procedure for doing this is a combination of the procedures we used for over- and underdetermined systems using QR. First we expand the residual we're trying to minimize using the SVD of A and then transform it by U^T (which does not change the norm of the residual):

$$\begin{aligned} \|Ax - b\| &= \|U\Sigma V^T x - b\|^2 = \|\Sigma V^T x - U^T b\|^2 \\ &= \left\| \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix} V^T x - \begin{bmatrix} U_1^T \\ U_2^T \end{bmatrix} b \right\|^2 \end{aligned}$$

Breaking apart the top and bottom rows:

$$= \| \begin{bmatrix} \Sigma_1 & 0 \end{bmatrix} V^T x - U_1^T b\|^2 + \|U_2^T b\|^2$$

The first term can be made zero; the second is the residual. But because of rank deficiency, minimizing the first term is an overdetermined problem—the matrix $\begin{bmatrix} \Sigma_1 & 0 \end{bmatrix} V^T$ is wider than tall. If we let $y = V^T x$ and $c = U_1^T b$, then split y into $\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ the system to be solved is

$$\begin{aligned} \begin{bmatrix} \Sigma_1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} &= c \\ \Sigma_1 y_1 &= c \end{aligned}$$

Since y_2 does not change the answer we'll go for the minimum-norm solution and set $y_2 = 0$. Then $x = V_1 y_1$.

So the solution process boils down to just:

1. Compute SVD.
2. $c = U_1^T b$.
3. $y_1 = \Sigma_1^{-1} c$.

4. $x = V_1 y_1$.

Or, more briefly,

$$x = V_1 \Sigma_1^{-1} U_1^T b. \tag{1}$$

This is just like the full-rank square solution, but with U_2 and V_2 left out.

I can also explain this process in words:

1. Transform b into the U basis.
2. Throw out components we can't produce anyway, since there's no point trying to match them.
3. Undo the scaling effect of A.
4. Fill in zeros for the components that A doesn't care about, since it doesn't matter what they are.
5. Transform out of the V basis to obtain x.

3.3 Pseudoinverse

If A is square and full-rank it has an inverse. Its SVD has a square Σ that has nonzero entries all the way down the diagonal. We can invert Σ easily by taking the reciprocals of these diagonal entries:

$$\Sigma = \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$$
$$\Sigma^{-1} = \operatorname{diag}(1/\sigma_1, 1/\sigma_2, \ldots, 1/\sigma_n)$$

and we can write down the inverse of A directly:

$$A^{-1} = (U\Sigma V^T)^{-1} = (V^T)^{-1}\Sigma^{-1}U^{-1} = V\Sigma^{-1}U^T \tag{2}$$

So the SVD of A^{-1} is closely related to the SVD of A. We simply have swapped the roles of U and V and inverted the scale factors in Σ. Earlier we interpreted U and V as bases for the "output" and "input" spaces of A, so it makes sense that they are exchanged in the inverse. A nice thing about this way of getting the inverse is that it's easy to see before we start whether and where things will go badly: the only thing that can go wrong is for the singular values to be zero or nearly zero. In our other procedures for the inverse we just plowed ahead and waited to see if things would blow up. (Of course, don't forget that computing the SVD is many times more expensive than a procedure like LU or QR.)

If A is non-square and/or rank deficient, it no longer has an inverse, but we can go ahead and generalize the SVD-based inverse, modeling it on the rank-deficient least squares process we just saw. Looking back at (1), we can see that we are treating the matrix

$$A^+ = V_1 \Sigma_1^{-1} U_1^T \tag{3}$$

like an inverse for A, in the sense that we solve the problem $Ax \approx b$ by writing $x = A^+ b$. It is also just like the formula in (2) for A^{-1} except that we have kept only the first r rows and/or columns of each matrix.

The matrix A^+ is known as the *pseudoinverse* of A. Multiplying a vector by the pseudoinverse solves a least-squares problem and gives you the minimum-norm solution. Note that writing down a pseudoinverse for a matrix always involves deciding on a rank for the matrix first.

There's another way in which the pseudoinverse is like an inverse: it is the matrix that comes closest to acting like an inverse, in the sense that multiplying A by A^+ almost produces the identity:

$$\begin{aligned} A^+ &= \min_{X \in \mathbb{R}^{n\times m}} \|AX - I_m\|_F \\ &= \min_{X \in \mathbb{R}^{n\times m}} \|XA - I_n\|_F \end{aligned}$$

We can prove this by looking at X in A's singular vector space:

$$X' = UXV^T$$

so that

$$\begin{aligned} AX - I &= (U\Sigma V^T)(VX'U^T) - I \\ U^T(AX - I)U &= \Sigma X' - I \end{aligned}$$

Since U is orthogonal, multiplying by it didn't change the F-norm of $AX - I$. This means we've reduced the problem of minimizing $\|AX - I\|_F$ to the easier problem of minimizing $\|\Sigma X' - I\|_F$. Since the last $m - r$ rows of Σ are zero, the closest we can hope to match I is to get the first r rows to match. We can do that easily by setting

$$X' = \Sigma^+ = \begin{bmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{bmatrix}$$

so that

$$\Sigma\Sigma^+ = \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix}.$$

Now that we know the optimal X' we can compute the corresponding X, which is A^+, and you can easily verify that A^+ comes out to the expression in (3).

Here are some facts about A^+ for special cases:

- A full rank, square: $A^+ = A^{-1}$ (as it has to be if it's going to be the closes thing to an inverse, when the inverse in fact exists!)
- A full rank, tall: $A^+A = I_n$ but $AA^+ \neq I_m$. By the normal equations we learned earlier, $A^+ = (A^TA)^{-1}A^T$ since both matrices compute the (unique) solution to a full-rank least squares problem.
- A full rank, wide: $AA^+ = I_m$ but $A^TA \neq I_n$. Again by normal equations, $A^+ = A^T(AA^T)^{-1}$.

Said informally, in both cases you can get the small identity but not the big identity, because the big identity has a larger rank than A and A^+.

3.4 Numerical rank

Sometimes we have to deal with rank-deficient matrices that are corrupted by noise. In fact we always have to do this, because when you take some rank-deficient matrix and represent it in floating point numbers, you'll usually end up with a full-rank matrix! (This is just like representing a point on a plane in 3D – it's only if we are lucky that a floating point number happens to lie exactly on the plane; usually it has to be approximated with one near the plane instead.)

When our matrix comes from some kind of measurement that has uncertainty associated with it, it will be quite far from rank-deficient even if the underlying "true" matrix is rank-deficient.

SVD is a good tool for dealing with numerical rank issues because it answers the question:

> Given a rank-k matrix, how far is it from the nearest rank-$(k-1)$ matrix?

That is, How rank-k is it?, or How precisely to we have to know the matrix in order to be sure that it's not actually rank-$(k-1)$?

SVD lets us get at these questions because, since rank and distances are unaffected by the orthogonal transformations U and V, numerical rank questions about A are reduced to questions about the diagonal matrix Σ.

If Σ has k entries on its diagonal, we can make it rank-$(k-1)$ by zeroing the k^{th} diagonal entry. The distance between the two matrices is equal to the entry we set to zero, namely σ_k, in both the 2-norm and the $F-norm$. It's not surprising (though I won't prove it) that this is the closest rank-$(k-1)$ matrix.

If we have a rank-k matrix and perturb it by adding a noise matrix that has norm less then ϵ, it can't change the zero singular values by more than ϵ. This means noise has a nicely definable effect on our ability to detect rank: if the singular values are more than ϵ we know they are for real (they did not just come from the noise) and if they are less then ϵ we don't know whether they came from the "real" matrix or from the noise.

We can roll these ideas up in a generalization of rank known as *numerical rank*: whereas $\text{rank}(A)$ is the (exact) rank of the matrix A, we can write $\text{rank}(A, \epsilon)$ for the highest rank we're sure A has if it could be corrupted by noise of norm ϵ. That is, $\text{rank}(A, \epsilon)$ is the highest r for which there are no rank-r matrices within a distance ϵ of A.

3.5 Principal component analysis

Another way of viewing the SVD is that it gives you a sequence of low-rank approximations to a data matrix A. These approximations become accurate as the rank of the approximation approaches the "true" dimension of the data. To do this, think of the product $U\Sigma V^T$ as a sum of outer products.

[Aside: recall the outer product version of matrix multiply, from way back at the beginning:

$$C = AB \quad ; \quad c_{ij} = \sum_k a_{ik} b_{kj} \quad ; \quad C = \sum_k a_{:k} b_{k:}$$

We can use the same idea even with Σ in the middle.]

We can express the SVD as a sum of outer products of corresponding left and right singular vectors:

$$\begin{aligned} A &= \begin{bmatrix} | & & | \\ u_1 & \cdots & u_m \\ | & & | \end{bmatrix} \Sigma \begin{bmatrix} — & v_1^T & — \\ & \vdots & \\ — & v_n^T & — \end{bmatrix} \\ &= \sum_i \sigma_i u_i v_i^T \end{aligned}$$

This is a sum of a bunch of rank-1 matrices (outer product matrices are always rank 1), and the norms of these matrices are steadily decreasing.

By truncating this sum and including only the first r terms, we wind up with a matrix of rank r that approximates A. In fact, it is the best rank-r approximation, in both the 2-norm and the Frobenius norm.

Interpreting this low-rank approximation in terms of column vectors leads to a powerful and widely used statistical tool known as *principal component analysis.* If we have a lot of vectors that we think are somehow related—they came from some process that only generates certain kinds of vectors—then one way to try to identify the structure in the data is to try and find a subspace that all the vectors are close to. If the subspace fits the data well, then that means the data are low rank: they don't have as many degrees of freedom as there are measurements for each data point. Then, with a basis for that subspace in hand, we can talk about our data in terms of combinations of those basis vectors, which is more efficient, especially if we can make do with a fairly small number of vectors. It can make the data easier to understand by reducing the dimension to where we can manage to think about it, or it can make computations more efficient by allowing us to work with the (short) vectors of coefficients in place of the (long) original vectors.

To be a little more specifc, let's say we have a bunch of m-vectors called $a_1, \ldots, a_n$. Stack them, columnwise, into a matrix A, and then compute the SVD of A.

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

The first n columns of U are the principal components of the data vectors $a_1, \ldots, a_n$. The entries of Σ tell us the relative importance of the principal components. The rows of V give the coefficients of the data vectors in the principal components basis. Once we have this representation of the data, we can look at the singular values to see how many of the principal components seem to be important. If we use only the first k principal components, leaving out the remaining $n - k$ components, we've chosen a k-dimensional subspace, and the error of approximating our data vectors by their projections into that subspace is limited by the magnitude of the singular values belonging to the components we left off. This subspace is optimal in the 2-norm sense, and in the F-norm sense.

One adjustment to this process, which sacrifices a bit of optimality in exchange for more meaningful principal components, is to first subtract the mean of the a_is from each a_i. Of course, you need to keep the mean vector around and add it back in whenever you are approximating vectors from the components.

Sources

- Our textbook: Cheney & Kincaid, *Numerical Mathematics and Computing*, 6e. Section 8.3.
- Golub and Van Loan, *Matrix Computations*, Third edition. Johns Hopkins Univ. Press, 1996. Chapter 5.