Image Compression Using the SVD

Paul Hewitt

April 9, 2002

The singular value decomposition can be used to compress information, for more efficient transfer or manipulation. A simple example is image compression. Suppose that an image is represented as an $m \times n$ array A of tiny *pixels* (picture elements), with the color in pixel ij stored in A_{ij}. One way to compress the image A is to approximate A by a matrix of smaller rank. If the singular value decomposition of A is

$$\sigma_1 u_1 v_1^* + \cdots + \sigma_n u_n v_n^*$$

then the closest approximation to A — in either the 2-norm or the Frobenius norm — by a matrix of rank k is the truncation of the above sum to the first k terms:

$$A \approx A_k = \sigma_1 u_1 v_1^* + \cdots + \sigma_k u_k v_k^*$$
$$\|A - A_k\|_F = \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_n^2} \leq (n-k)\sigma_{k+1}$$

Sending A in its entirety requires sending mn numbers, whereas sending the rank-k approximation A_k requires sending $(2n+1)k$ numbers, namely the data $\sigma_1, u_1, v_1, \ldots, \sigma_k, u_k, v_k$. Of course, the receiver has to reconstruct A_k from this data but today computation is cheap and bandwidth is expensive.

For example, if we have a 200×200 image, and $\sigma_{25} \approx 10^{-3}$, then A contains 40000 numbers, whereas the rank-24 approximation A_{24} can be reconstructed from fewer than 10000 numbers — a 75% savings! Moreover the entries of A_{24} differ from those of A by about 10^{-3}, on average.

Problems

If there is sufficient interest in this application, then I will post more problems.

1. The text describes a 2-step procedure for computing the singular value decomposition of a matrix A, which is similar to the procedure to compute eigenvalues. The first step is to *bigiagonalize* the matrix, and the second step is an infinite process which converges to the SVD. There are several ways to do step 1. One of these, the Goub-Kahan algorithm multiplies A alternately on the left and right by Householder reflections:

$$\cdots U_3 U_2 U_1 A V_1 V_2 \cdots = \begin{bmatrix} \times & \times & 0 & 0 & \cdots \\ 0 & \times & \times & 0 & \cdots \\ 0 & 0 & \times & \times & \cdots \\ \vdots & \vdots & & \ddots & \ddots \end{bmatrix}$$

It turns out that if m is much bigger than n then it is cheaper first to find the QR factorization of A and then to apply the Golub-Kahan algorithm to R. This is called the Lawson-Hanson-Chan algorithm. There is also a hybrid method, which starts computing the Golub-Kahan algorithm then switches to the L-H-C algorithm when the submatrix remaining to be bidiagonalized is sufficiently tall and skinny. The text points out, however, that all of its analysis is theoretical. On a real machine mileage may vary. For this problem you should implement and test the G-K and L-H-C algorithms. Aim for a picture like that on page ???.

2. Use a linear algebra package (for example MatLab, or its free clone Octave) to generate compressed images using the SVD. Present your results to the class.

3. In a similar way, some people are trying to search and manipulate large text documents using the SVD. To do this each document is first represented by a vector, where each coordinate is the frequency of a particular word, and then the frequency vector is normalized to have length 1. A matrix then represents a whole collection of documents to be searched. To search for documents on a particular topic you form a *query vector* q whose coordinates are 1 or 0, depending on whether the document you are seeking should or should not contain the corresponding word. A document vector which points in nearly the same direction as q is likely to be concerned with the topics indicated by q. Thus, one measure of how close document a is to matching your query is

$$\cos \angle q, a = q^* a.$$

Call this number the *score* of the document a for the query q. If $A \approx A_k = U_k \Sigma_k V_k^*$ then $V_k \Sigma_k$ turns out to be a reasonably good low-dimensional proxy for the collection of documents represented by A. So, if $q^* A_k$ has a high score in column j then we might expect that document j is a good match for the query. In this problem you should grab several documents from the web, form the matrix A and a query q, then use the SVD to find the document which best matches your query. Is this a promising way to search for documents on a particular topic?