From Diagonalisation to SVD

We know about eigenvalues and diagonalisation. For a square, $n \times n$ matrix A, we can find a decomposition in the form

$$A = PDP^{-1},$$

where D is a diagonal matrix with eigenvalues along the diagonal, and P is a matrix with columns that are the corresponding eigenvectors, but you can only do this for a square matrix with n linearly independent eigenvectors.

Can we do something in other cases?

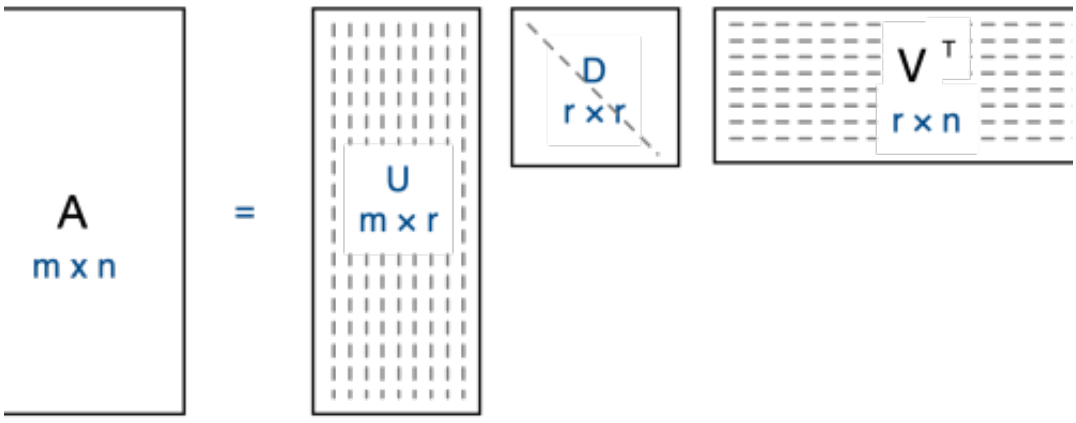
Singular Value Decomposition

The Singular Value Decomposition (SVD) does the same thing, but for any matrix, *i.e.,* for any matrix A we can write it as

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

When A is an $m \times n$ matrix, then

- Σ is a $r \times r$ matrix with the singular values σ_i along its diagonals. By convention we order singular values so that $\sigma_i \geq \sigma_{i+1}$, from which you can also infer that they are real, unlike eigenvalues which may be complex. Here r is the rank of the matrix, which is also the number of non-zero singular values.
- U is a $m \times r$ matrix, whose r columns are the *left singular vectors* of A.
- V is a $n \times r$ matrix, whose r columns are the *right singular vectors* of A.



The singular values are in fact the square root of the eigenvalues of AA^T, and the columns of U are the eigenvectors of AA^T and the columns of V are the eigenvectors of A^TA.

We can always do this because AA^T is a *normal* matrix and hence is always diagonalisable (because ...).

N.B. it is not uncommon to write the SVD slightly differently by taking U to be $m \times m$, Σ to be $m \times n$ and V to be $n \times n$, but then a part of Σ is all zeros and it seems a bit wasteful to me.

Why SVD?

The SVD has many interpretations and uses. I will focus on only one here, but see the associated reading for more.

The use I want to push comes from an alternative expression for the decomposition:

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T,$$

where $\mathbf{u}_i$ and $\mathbf{v}_i$ are the columns of U and V, respectively. We can rewrite this as

$$A = \sum_{i=1}^{r} \sigma_i A_i,$$

where here each A_i is a rank-1 matrix.

Now **here** is the big trick. If we put the singular values in a vector σ (in order) then the vector is **compressible** if there are only a few large values, and the rest are trivial. This is a type of sparsity. In that case we can drop the small values (by hard thresholding) and we are now left with s singular values, and we can come up with a *low-rank approximation* $A^{(s)}$ for A in the form

$$A^{(s)} = \sum_{i=1}^{s} \sigma_i A_i.$$

That is, we use the largest s singular values and their associate rank-1 matrices to form our approximation. It turns out this is the **best** approximation with rank no more than s, where best is in the sense that it minimises the Frobenius norm of the difference, *i.e.,* this approximation solves the problem

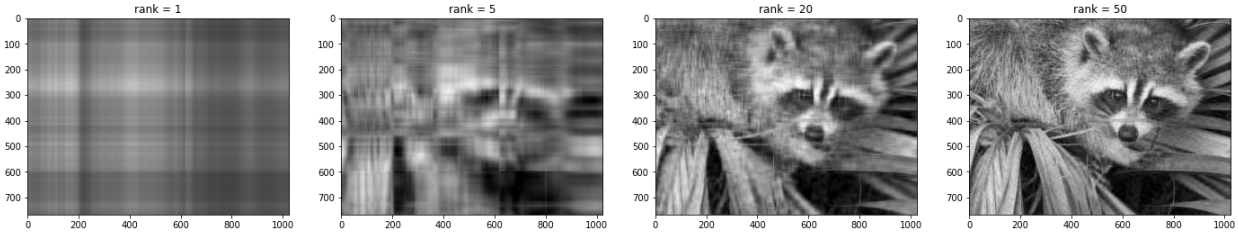
$$A^{(s)} = \text{argmin}_X ||A - X||_F \quad \text{s.t. rank}(X) \leq s.$$

You (may) later also see that this is a good approximation to finding approximations for other problems, in particular in the space of *matrix completion* problems.

Soft thresholding (moving all values towards zero by a fixed amount and dropping terms that hit zero) results in minimising the Frobenius distance plus a *regularizing* term corresponding to the nuclear norm (more on regularisation next).

In the mean time here is a short example. The following image is a standard one from `scipy`.

The following are 4 low-rank approximations:



You can see that if you try to make the rank too small, the result is hopeless, but even by rank 50 the image is almost as good as the original (for context, the rank of the original is more than 700.

How SVD

SVD is a goal, not a technique. There are algorithms out there to achieve it, but this is not a

course about algorithms (at least not these algorithms). So we will just use standard code to perform SVD.

It's ridiculously easy to do you do SVD in `Python` and `PyTorch` if you use the canned routines.

```
>>> import torch
>>> A = torch.randn(5, 3)
>>> U, S, Vh = torch.linalg.svd(A, full_matrices=False)
>>> U.shape, S.shape, Vh.shape
(torch.Size([5, 3]), torch.Size([3]), torch.Size([3, 3]))
>>> torch.dist(A, U @ torch.diag(S) @ Vh)
tensor(1.0486e-06)
```

Here the outputs are

- `U` : the matrix descxribed above;
- `S` : a vector containing the diagonal of Σ, *i.e.,* the singular values sorted in descending order; and
- `Vh` : the Hermitian transpose of V.

The operation `U @ torch.diag(S) @ Vh` multiplies them back together, and we can see the result is very close to the original matrix A.

Now look for `svd_racoon_image.ipynb` to reproduce the pictures I used here.

Limits of SVD

SVD can do a lot, but it isn't enough by itself. In particular, it says nothing about what to do with missing data, but the ideas can lead in that direction (and hence help solve matrix completion problems).

SVD is also based in linear algebra. Real data may lie in some sparse subset of a space, but may not be linear. Some of the techniques we will eventually get to in this course are non-linear.

Finally, although we are solving a problem of finding a good representation with bounded rank, that is not the same as solving a problem to minimise the rank (given some other criteria).

Links

Python and SVD

- https://pytorch.org/docs/stable/generated/torch.linalg.svd.html
- https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html

Matrix completion

- https://hastie.su.domains/TALKS/SVD_hastie.pdf
- https://arxiv.org/pdf/1410.2596.pdf