Nicholas Carolan
email@princeton.edu
COS302-F23

Assignment #5

Due: 6:00pm Wednesday 25 October 2023

Discussants: N/A

Upload at: https://www.gradescope.com/courses/606160/assignments/3456405

Problem 1 (20pts)

We sometimes need to introduce norms for matrices. This comes up when we want to talk about the magnitude of a matrix, or when we need a notion of distance between matrices. One important matrix norm is the Frobenius norm, which can be written in several ways for a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$:

$$||\boldsymbol{A}||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} A_{i,j}^2} = \sqrt{\text{trace}(\boldsymbol{A}^T\boldsymbol{A})}$$

(A) Show that the Frobenius norm is also the square root of the sum of the squared singular values.

(B) Assume that $\boldsymbol{A}$ is square and invertible, with a very small Frobenius norm. What kind of value would you expect to get for the Frobenius norm of $\boldsymbol{A}^{-1}$?

(A) Start by taking the SVD of A: $A = U\Sigma V^T$. The matrix Σ contains the singular values of A on the diagonal.

We can then compute A^TA with substitution:

$$A^TA = (U\Sigma V^T)^T U\Sigma V^T$$
$$A^TA = V\Sigma^T U^T U\Sigma V^T$$
$$A^TA = V\Sigma^T\Sigma V^T$$

Recalling that Σ contains only the singular values $\sigma_0 ... \sigma_m$ of A along its diagonal, we know that $\Sigma^T\Sigma$ contains those singular values squared along its diagonal ($\sigma_0^2 ... \sigma_m^2$).

The Frobenius norm is defined as $\sqrt{\text{trace}(A^TA)}$. Using the cyclic property of trace:

$$\text{trace}(A^TA) = \text{trace}(V\Sigma^T\Sigma V^T) = \text{trace}(\Sigma^T\Sigma V^TV) = \text{trace}(\Sigma^T\Sigma).$$

Because Σ is diagonal with entries $\sigma_0 ... \sigma_m$ (and zeros otherwise), $\text{trace}(\Sigma^T\Sigma) = \sum_{i=0}^{m} \sigma_i^2$. With substitution:

$$||\boldsymbol{A}||_F = \sqrt{\text{trace}(A^TA)} = \sqrt{\sum_{i=0}^{m} \sigma_i^2}$$

i.e. the Frobenius norm is the square root of the sum of the squared singular values.

(B) We illustrated in **(A)** that the Frobenius norm of a matrix A is proportional to the sum of its squared singular values. We know that an inverted matrix A^{-1} has eigenvalues that are the inverse of the eigenvalues of A, i.e. $\frac{1}{\sigma_0} ... \frac{1}{\sigma_m}$. Therefore, we expect:

$$||\boldsymbol{A}^{-1}||_F = \sqrt{\sum_{i=0}^{m} (\frac{1}{\sigma_i})^2}$$

If the eigenvalues of A are sufficiently small, we expect the Frobenius norm of its inverse to be very large.

Problem 2 (25pts)

In general, computing the determinant of an $n \times n$ matrix scales as n^3 in computational cost. When the matrix is highly structured, however, it can sometimes be possible to take advantage of that structure to save computation for quantities such as the determinant. One example of such structure is in *tridiagonal* matrices, which look like this:

$$\boldsymbol{T} = \begin{bmatrix} a_1 & b_1 & 0 & 0 & \cdots & 0 \\ c_1 & a_2 & b_2 & 0 & & 0 \\ 0 & c_2 & a_3 & \ddots & & \vdots \\ 0 & 0 & \ddots & \ddots & b_{n-2} & 0 \\ \vdots & & & c_{n-2} & a_{n-1} & b_{n-1} \\ 0 & 0 & \cdots & 0 & c_{n-1} & a_n \end{bmatrix}$$

Such matrices can come up when simulating a physical system with local structure, e.g., a spring-mass system.

(A) We would like to compute the determinant of the matrix $\boldsymbol{T}$ above, which we are assuming is invertible. Let d_m denote the determinant of the upper left $m \times m$ submatrix. So, $d_1 = a_1$ and $d_n = |\boldsymbol{T}|$ (computing the whole determinant). Use expansion by minors to compute d_n in terms of d_{n-1}, d_{n-2}, and any of the a_i, b_i, or c_i. If necessary you can take $d_0 = 1$ and $d_i = 0$ for $i < 0$. Carefully show how you arrived at your answer.

(B) Based on this recurrence relation, how would you expect the computational cost to scale for computing the determinant of a tridiagonal matrix?

(A) Let d_m be the determinant of the upper-left $m \times m$ block of T. We know that $d_0 = 1$ and $d_1 = a_1$.

We can use expansion by minors to iteratively find the determinant d_{m+1} given the value of d_m, setting $i = m + 1$.

$$d_{m+1} = \sum_{j=1}^{m+1} (-1)^{i+j} a_{ij} M_{ij}$$

Per (source), "$M_{(ij)}$ is a so-called minor of A, obtained by taking the determinant of A with row i and column j crossed out." Because the matrix T is tridiagonal, there are only two non-zero elements a_{ij}: c_m in column m and a_{m+1} in column $m + 1$. As a result:

$$d_{m+1} = \sum_{j=1}^{m+1} (-1)^{i+j} a_{ij} M_{ij} = (-1)^{(m+1)+m} c_m M_{m+1,m} + (-1)^{(m+1)+(m+1)} a_{m+1} M_{m+1,m+1}$$

$$(-1)^{2m+1} = -1 \text{ and } (-1)^{2m+2} = 1 \;\rightarrow\; d_{m+1} = -c_m M_{m+1,m} + a_{m+1} M_{m+1,m+1}$$

This neatly expresses the determinant as a function of two minors. $M_{m+1,m+1}$ is the determinant of the upper-left $m \times m$ block, i.e. $M_{m+1,m+1} = d_m$.

$M_{m+1,m}$ involves the sub-matrix with row $m + 1$ and column m removed. In the resulting matrix, the last column (original column $m + 1$) has a single nonzero entry b_m in row m. Expanding that minor along its last column gives $M_{m+1,m} = (-1)^{2m}\, b_m\, d_{m-1} = b_m\, d_{m-1}$.

We can complete our relationship with substitution: $d_{m+1} = a_{m+1}d_m - b_m c_m d_{m-1}$ with $d_0 = 1,\ d_1 = a_1$. This establishes a recursive algorithm that expands to find $d_n = |\boldsymbol{T}|$.

(B) Using the recurrence relation established in (A), we can compute the determinant d_n with n steps, each with constant $O(1)$ work. As a result, the total computational cost for computing the tridiagonal determinant is linear, $O(n)$.

Problem 3 (23pts)

In this problem, you'll look at singular value decomposition as a way to find a low-rank approximation to a matrix.

(A) Upload any (tasteful) image you want to Colab. Note that life is a little easier if you put it in your Google drive and access it from there so you don't have to constantly re-upload it every time you come back after a break. Load the image as a NumPy array. There are various ways to do this, but the easiest may be to use `imread` from matplotlib. There is a useful tutorial here also. Once you have a NumPy array, look at its shape: it is probably $width \times height \times 3$ or something to that effect, with the third dimension usually reflecting red, green, and blue channels. Make the image grayscale by taking the mean across this third dimension. This should make it into a $width \times height$ array. Use `imshow` as in previous homeworks to render the grayscale image.

(B) Import `numpy.linalg` and use the `svd` function to compute the singular value decomposition of the image matrix. This will return three things: a $\boldsymbol{U}$ matrix, a vector containing the singular values, and a $\boldsymbol{V}^T$ matrix. Print the shapes of these arrays, and then figure out how to "reassemble" these three arrays with multiplication to reconstruct the image. Render the reconstruction and verify that it looks like the original image.

(C) Plot the cumulative sum of the singular values. Sum them from largest to smallest.

(D) Create three different low-rank approximations to your image. Create one of rank 5, one of rank 10, and one of rank 25. (Hint: you can do this by reconstructing the image matrix as above, but with zeros for all the singular values with index larger than the rank of your approximation. Render each of the three images.

(E) Create three new matrices, each with the pixel-wise squared difference between the original image and the low-rank approximation. Render these as images and qualitatively describe what kind of visual structure seems to be lost in the low-rank approximations.

Problem 4 (30pts)

In this problem you will use SVD to model a text corpus, a small subset of New York Times articles. You will use a "bag of words" representation in which documents are represented by the counts of words, usually excluding very common "stop words" like *the* and *and*. Download `nyt.pkl.gz` and upload it to your Google drive so you don't have to upload it every time you open the Colab; it's a fairly big file. Here's some code to get you going:

```
import pickle as pkl
import numpy as np
import gzip

filename = 'drive/My Drive/COS 302/nyt.pkl.gz'
with gzip.open(filename, 'rb') as fh:
  nyt = pkl.load(fh)
documents = nyt['docs']
vocab = nyt['vocab']

# Create reverse lookup table.
vocab_indices = dict([(w, i) for (i, w) in enumerate(vocab)])

M = len(documents)
N = len(vocab)
print('%d documents, %d words' % (M,N))

count_mat = np.zeros((M,N))
for mm, doc in enumerate(documents):
  for word, count in doc.items():
    count_mat[mm, vocab_indices[word]] = count
```

(A) Typically, raw counts don't lead to discovery of interesting structure. Instead, it is common to use something like TF-IDF, which stands for *term frequency-inverse document frequency*. Term frequency is the number of times word n appeared in document m, divided by the total number of words in document m.

$$\text{tf}_{m,n} = \frac{\text{\# times word } n \text{ appears in doc } m}{\text{total \# of words in doc } m}$$

Transform `count_mat` from the code above into a term frequency matrix.

(B) Inverse document frequency is typically the natural log of the number of documents, divided by the number of documents in which word m appears. (The plus-one ensures that you're not dividing by zero.)

$$\text{idf}_n = \log \frac{\text{total \# of documents}}{1 + \text{\# of documents with word } n}$$

Compute the idf vector from `count_mat`.

(C) Now compute the TF-IDF matrix by multiplying (broadcasting) $\text{tf}_{m,n}$ and idf_n. Use `numpy.linalg.svd` to take the SVD of this TF-IDF matrix; it may take a minute since the matrix is relatively large. Plot the singular values in decreasing order.

(D) The right singular vectors (columns of the V matrix) will ideally represent interesting topical dimensions. For each of the top 20 right singular vectors: identify the words in the vocabulary that have the largest entries in the vector and print them. That will probably mean looping over the first 20 right singular vectors, doing an appropriate `argsort` and then finding that entry in the `vocab` variable. Do you see any interesting qualitative structure in these groups of words?

Problem 5 (2pts)
Approximately how many hours did this assignment take you to complete?

My notebook URL: https://colab.research.google.com/drive/1LadbLhqzlv9UqPgYgXvubTH0no6Xe6QE?usp=sharing

Changelog

- 2 October 2023 – Initial F23 version.