Don't Always Do Your Best

Dianne P. O'Leary

"Always do your best" sounds like a good motto, but it is actually quite impossible. If I take time to make the perfect breakfast, I might not make it to class on time. If I focus on keeping my car perfectly centered in my lane, I might not notice the cyclist starting to cross in front of me. If I am the perfect student, I might not be able to be the perfect parent / spouse / daughter / son / employee. "The perfect is the enemy of the good," and in life we need to make reasonable compromises to determine what is *good enough* for a given task.

Similarly, in science and engineering, we put a lot of effort in getting as close as possible to truth – computing solutions as accurately as possible. But we also put a lot of effort into computing solutions that we know are *wrong* but close enough to the truth to be useful. We do this because it is impossible to obtain perfect measurements, and too expensive to compute perfectly accurate results. In fact, approximations can yield very useful results, and we focus in this case study on the approximation of matrices.

We learn in Pointer 5.8 (SCCS textbook) that among all matrices of a given rank, the singular value decomposition (SVD) of a matrix gives the best approximation to a matrix. In this case study, we consider alternatives when the SVD is too expensive.

Notation: We'll try to approximate a matrix $\boldsymbol{A}$ of dimension $m \times n$ where $m \geq n$. The jth column of $\boldsymbol{A}$ will be denoted by $\boldsymbol{a}_j$.

Algorithm 0.1 CU Decomposition

```
Choose a set of $k$ representative columns from $\boldsymbol{A}$, and call these vectors $\boldsymbol{c}_1, \dots, \boldsymbol{c}_k$.
Let $\boldsymbol{C} = [\boldsymbol{c}_1, \dots, \boldsymbol{c}_k]$.
for $j = 1 : n$
    Express $\boldsymbol{a}_j$ as a combination of the columns in $\boldsymbol{C}$: $\boldsymbol{a}_j \approx \boldsymbol{C}\boldsymbol{u}_j$, where the
    coefficients in $\boldsymbol{u}_j$ are chosen to make $\|\boldsymbol{a}_j - \boldsymbol{C}\boldsymbol{u}_j\|$ small.
end
Let $\boldsymbol{U} = [\boldsymbol{u}_1, \dots, \boldsymbol{u}_n]$.
Then $\boldsymbol{A} \approx \boldsymbol{C}\boldsymbol{U}$.
```

The CU Decomposition

If there were a small number of columns of $\boldsymbol{A}$ that were representative of all of the columns, then we could approximate the matrix as in Algorithm 0.1. There are two main tasks in the algorithm:

- Choose the columns to include in $\boldsymbol{C}$.
- Determine the columns of $\boldsymbol{U}$ by solving n least squares problems.

Our goal is to find a good approximation, and we'll measure goodness by taking the Frobenius norm of the error: $E = \|\boldsymbol{A} - \boldsymbol{C}\boldsymbol{U}\|_F$.

We'll consider the second task first.

Computing U using least squares

We can find $\boldsymbol{u}_j$ $(j = 1 : n)$ as the solution of the least squares problem

$$\min_{\boldsymbol{z}} \|\boldsymbol{a}_j - \boldsymbol{C}\boldsymbol{z}\|_2^2.$$

Let's see how good our approximation will be, and how expensive it will be to compute.

CHALLENGE 0.1.

(a) Explain why the E obtained from least squares is the smallest possible E over all choices of $\boldsymbol{U}$.

(b) Solving n least squares problems could be expensive, but each of them involves the same matrix $\boldsymbol{C}$. Suppose we have a rank-revealing QR factorization of $\boldsymbol{C}$. How many multiplications will it take to solve our problems? Compare this with the cost of solving n least squares problems of the same size with different matrices for each problem.

Choosing columns from A

The only remaining question is how to find a good choice of columns to include in the matrix $\boldsymbol{C}$. Here are three reasonable choices:

- Choose a set of columns that are "most important" in the matrix $\boldsymbol{A}$. For example, our first choice might be the column that has maximum norm. Our second might be the column whose component orthogonal to the first choice has maximal norm, since we already have accounted for the component in the direction of the first choice. Our third might be the column whose component orthogonal to the first two is maximal. We continue this way. Our matrix tool for computing this is the pivoted QR decomposition, stopped after k steps. We call this algorithm `CUQR`.

POINTER 0.1. Alternatives not discussed here.

- We could just as well have chosen rows from $\boldsymbol{A}$ instead of columns, and then determined a $\boldsymbol{UR}$ decomposition by choosing $\boldsymbol{U}$ appropriately, where $\boldsymbol{R}$ is a matrix containing the chosen rows.
- We could also have chosen a set of rows and a set of columns, obtaining a $\boldsymbol{CUR}$ decomposition.

- We might use our method of last resort: a random choice of columns. We could do this with or without replacement. "With replacement" means that if we think about choosing column indices out of a hat, we put the one we chose back into the hat before choosing the next one. This can be implemented using k samples from `rand`, renormalized to the interval 0 to n and then rounded up to the next integer.

  "Without replacement" means that we don't allow any duplicates. If we think about choosing column indices out of a hat, we don't put the chosen ones back into the hat. So if `rand` gives a duplicate, we ignore it and choose again.

  We call these algorithms `CUrandWith` and `CUrandWithout`.
- It would be good to include some sort of importance sampling in our randomized algorithms, but we won't do that in this case study.

CHALLENGE 0.2. (a) Implement `CUQR`, `CUrandWith`, and `CUrandWithout` in MATLAB. The three MATLAB functions should be well documented. They should take $\boldsymbol{A}$ and k as input and produce $\boldsymbol{C}$ and $\boldsymbol{U}$ as output.

(b) How many multiplications are needed for each algorithm assuming that that `nz` entries in $\boldsymbol{A}$ are nonzero? Include your answer in the documentation for your functions.

Nonnegative matrix factorization

In many applications, the matrix $\boldsymbol{A}$ has entries that are all nonnegative. When we solve our least squares problems for $\boldsymbol{U}$, we may compute some negative entries in $\boldsymbol{U}$, and this may cause some entries of $\boldsymbol{CU}$ to be negative. This is not a desirable feature.

An alternative to the CU decomposition is a **nonnegative matrix decomposition**. In this case, we approximate $\boldsymbol{A}$ by the product $\boldsymbol{WH}$, where all entries in the $m \times k$ matrix $\boldsymbol{W}$ and the $k \times n$ matrix $\boldsymbol{H}$ are nonnegative. Given initial choices for $\boldsymbol{W}$ and $\boldsymbol{H}$, the algorithm uses an **alternating iteration** (a very slowly converging iteration), updating the choices by the MATLAB formulas

```
H = H .* (W' * A) ./ (W' * W * H + 1.e-9);
W = W .* (A * H') ./ (W * H * H' + 1.e-9);
```

Convergence is not guaranteed, but the iteration often converges to a local minimizer of the function $f(\boldsymbol{W}, \boldsymbol{H}) = \|\boldsymbol{A} - \boldsymbol{WH}\|_F$.

CHALLENGE 0.3.

(a) Write a MATLAB function `NonNegApprox` that takes $\boldsymbol{A}$ and k as input, along with an initial guess $\boldsymbol{H}$ and $\boldsymbol{W}$, and uses the alternating iteration to compute a nonnegative approximation to $\boldsymbol{A}$. Stop the iteration when $\boldsymbol{H}$ and $\boldsymbol{W}$ stop changing too much; this is vague, so document exactly how you implemented the stopping condition.

(b) How many multiplications are needed to perform one iteration, assuming that every entry in $\boldsymbol{H}$ and $\boldsymbol{W}$ is nonzero (i.e., these matrices are **dense**) and that `nz` entries in $\boldsymbol{A}$ are nonzero? Include your answer in the documentation for your function.

Testing our ideas

Let's see how these methods work on some different matrices $\boldsymbol{A}$. For each test, we will measure the relative error in the approximation, $\|\boldsymbol{A} - approximation\|_F / \|\boldsymbol{A}\|_F$, as a function of k.

CHALLENGE 0.4.

(a) Test your matrix approximation methods `CUQR`, `CUrandWith`, and `CUrandWithout`, `NonNegApprox`, and the SVD on the term-document matrix `cisimatrix.mat`, a matrix `bwidata.mat` of hourly temperature records at BWI airport from September 2009 through August 2010, and the matrix `gene.mat` of gene expression data.

Choose values of k between 1 and $n/4$. Use the first k columns and rows of $\boldsymbol{A}$ to initialize the nonnegative matrix decomposition algorithm. For the randomized algorithms, run the algorithms 50 times for each matrix and each value of k and average your results.

Create one figure for each testmatrix. The figure should plot relative error vs. k, one curve for each algorithm, with the curves labeled by `legend`.

Then create another figure for each testmatrix, plotting the number of columns with less than 10% relative error, as a function of k.

(b) Discuss your results, comparing the accuracy, the time, and the storage expense of the algorithms.

POINTER 0.2. Less is more: from Google to Crime to Netflix

The problem that we consider in this case study has many applications. Here are a few examples.

- Search engines like Google have billions of documents that must be indexed by topic for automatic retrieval. We could make a list of all the terms in the documents and number them 1 through m. Then we can represent a document by a column vector whose ith entry measures the importance of that term in that document. (We expect the vector to have mostly zeros, since most terms are not present in a single document.) If we stack all of these columns into a matrix $\boldsymbol{A}$, we have a **term-document** matrix, and it would be good to approximate it so that we do not need to store such a large matrix.

- To compress video files, it would be useful to have a set of important frames in the movie that we could use to compute a representation of every other frame.

- We might collect data on interactions among a group of people, and represent it as a graph. There is one node in the graph for every person, and an edge between each pair of people who interact. The weight on the edge is a measure of the importance of the interaction; for example, it might be the number of email messages between them. Now we can represent this graph as a square matrix with one row and column for each person. The weights (0 if an edge is missing) are the entries in the matrix. If the group is small enough, we have no trouble storing such a matrix, but if the group is large, we need to approximate it.

  Analysis of such graphs is important in sociology (with a famous study of the members of a karate club), in documenting power and information exchange in corporations (e.g., the Enron study), in detecting anomolies in computer network connections that might indicate intrusions, and in many other criminal investigations.

- The famous Netflix data is also a sparse matrix, one row per customer and one column per movie. Each entry in the matrix is a rating of a movie by a customer. The matrix is incomplete, and it is useful to use a matrix approximation to compress the data and fill in guesses for missing entries.