Numerical analysis: Project 1

Instructor: Anil Damle
Due: March 11, 2024

Policies

You are permitted to work in groups of up to three for this project and each group only needs to submit a single report. While it is okay to discuss the project with other groups (similar to the HW collaboration policy), every group must produce their own code and report. The report should address all of the items referred to as **goals** or **questions** in the project description. You may refer to inanimate sources (e.g., textbooks) to help you answer the questions and tackle the goals. However, your sources must be explicitly cited. For example, one of the goals requires writing pseudocode for an algorithm that exists, if you go to an external source to read more about it and help generate the pseudocoade you must cite where it came from.

Part of the purpose of this project is to provide you with an opportunity to practice writing more free form reports and carefully choosing what plots, results, etc. are needed to convince us your solution is correct. Therefore, some of the goals are leading, but do not give a concrete list of exactly what has to be included. The questions tend to be more concrete.

The report, including plots and requested output from your code, should be typeset and submitted via the Gradescope as a pdf file. This file must be self contained for grading. Additionally, please submit any code written for the assignment as zip file to the separate Gradescope assignment for code.

Preamble

The goal of this project is to develop a method to compress deep learning models based on variants of tools we have learned about in class. In particular, in this project you will learn about the column-pivoted QR factorization (a variant on the QR factorization) and implement one, understand why "pivoting" is useful and in some cases necessary for QR factorizations, and then use your implementation to compress a simple machine learning model for classifying images in the FashionMNIST data set. Please note that while we will be working with ML models, nothing we do will require "training" or "fine-tuning" and therefore the entire project can be easily completed in Python (with NumPy), Julia, or Matlab; there is no need to use ML frameworks such as PyTorch or TensorFlow.

Column-pivoted QR factorizations

In class we developed a method based on Householder reflectors to compute the factorization

$$A = QR$$

given $A \in \mathbb{R}^{m \times n}$.[1] In particular, for $m \leq n$ we used Householder reflectors to build a sequence of matrices $Q^{(1)}, \ldots, Q^{(n)}$ such that

$$Q^{(n)} \cdots Q^{(1)} A = R, \tag{1}$$

and then let $Q = Q^{(1)} \cdots Q^{(n)}$. If $m > n$ we "redefined" Q to be only the first n columns of $Q^{(1)} \cdots Q^{(n)}$ and R to be the top $n \times n$ block of R from eq. (1).

While for the uses we discussed in class a standard QR factorization suffices, there are many other settings where it is useful to introduce the concept of pivoting to a QR factorization. One example is the construction of low-rank approximations that can sometimes be interpreted more readily than the optimal one produced by the SVD. Another is the construction of sparse solutions to least squares problems. Let's consider the latter example first, followed by low-rank approximations.

Basic solutions to least-squares problems

Assume $m > n$ and consider the least-squares problem

$$\min_x \|b - Ax\|_2^2. \tag{2}$$

We can interpret this problem as trying to fit the vector b using a linear combination of the columns of A. However, if A is highly ill-conditioned there are many vectors x that yield nearly the same predictions Ax—so we get to choose which one to pick. In class we saw one approach is to try and choose x to have small norm. However, another approach is to try and choose x to be sparse.[2]

Let's pretend for a moment that we would like a factorization of A that can be used to solve LS problems for many b.[3] In that case, we may want to identify a subset of columns of A that for any b can be used to achieve a nearly optimal fit. More specifically we may want to identify some integer k and k columns of A denoted by the index set $\mathcal{C}$ such that for any b the solution to

$$\min_z \|b - A(:\mathcal{C})z\|_2^2 \tag{3}$$

has nearly the same optimal value as eq. (2). Solving eq. (3) then yields a k-sparse approximate solution to eq. (2) by setting $x(\mathcal{C}) = z$ and all other entries of x to zero. Beyond sparsity, if we have interpretable labels for columns of A (e.g., they are features for a regression task), $\mathcal{C}$ tells us which features are "most important" in some sense.

Interpolative decompositions

Given the SVD of a matrix A denoted $A = U\Sigma V^T$, we can recover the optimal rank-k approximation of A as $A_k = U_1 \Sigma_1 V_1^T$ where U_1 is the first k columns of U, V_1 is the first k columns of V, and Σ_1 is the upper left $k \times k$ block of Σ. However, if A is structured (e.g., the columns are sparse or have clear meaning) that structure may be lost in the SVD. For example, even if the columns of A are sparse, the left singular vectors will generally not be. Therefore, it can be desirable to build a low-rank approximation of A that preserves structure and/or interpretation.

Similar to above, this can be accomplished by identifying an integer k and k columns of A denoted by the index set $\mathcal{C}$ such that

$$A \approx A(:, \mathcal{C})B$$

[1] Recall that there are subtle variations we can consider if $m > n$; here we will always be referring to the reduced QR factorization.

[2] Finding sparse solutions is a huge field, we are just considering one simple approach and not discussing others.

[3] For any fixed b there is a slight variant on the idea we will discuss later that is preferable.

for some $k \times n$ matrix B. The interpretation here is that we identify k columns of A that can be used to approximately represent any column of A (to some desired accuracy). As before, these columns can be thought of as the "most important" columns of A and if columns of A are structured than the representation of the range of the low-rank factorization retains this structure—as it is built out of columns of A themselves.

The mathematics of a column-pivoted QR factorizations

We will now how we can use a (pivoted) QR factorization to solve the above problems. However, first let's continue our discussion omitting pivoting; we will then see why it is necessary. To simplify our discussion, we will assume throughout this project that A has full-rank in the sense that $\sigma_{\min(m,n)} > 0$ (i.e., A has linearly independent rows if $m \leq n$ and linearly independent columns if $m \geq n$). For any $k < \min(m,n)$ we can partition a QR factorization as

$$A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \\ & R_{22} \end{bmatrix},$$

where $\ell = \min(m,n)$, $Q_1 \in \mathbb{R}^{m \times k}$, $Q_2 \in \mathbb{R}^{m \times \ell - k}$, $R_{11} \in \mathbb{R}^{k \times k}$, $R_{12} \in \mathbb{R}^{k \times n-k}$, and $R_{22} \in \mathbb{R}^{\ell-k \times n-k}$. Such a partition reveals how a QR factorization can be used for the above problems.

Question 1: Show that if z solves

$$\min_z \|A(:, 1:k)z - b\|_2^2$$

then

$$\left\| A \begin{bmatrix} z \\ 0 \end{bmatrix} - b \right\|_2 \leq \|R_{22}\|_2 \|x_{\text{LS}}\|_2 + \|b - Ax_{\text{LS}}\|_2,$$

where x_{LS} solves

$$\min_x \|Ax - b\|_2^2.$$

In other words, the residual from the sparse solution cannot be much larger than the optimal residual.

Question 2: Show that there exists a matrix B such that

$$\|A - A(:, 1:k)B\|_2 = \|R_{22}\|_2$$

and

$$\|A(:, 1:k) - A(:, 1:k)B(:, 1:k)\|_2 = 0.$$

The second condition ensures that k columns of A are exactly represented by the rank-k factorization $A(:, 1:k)B$.

In both of the above results, we see that the effectiveness of using a QR factorization to solve our problem hinged on $\|R_{22}\|_2$ being small. However, for a QR factorization without any pivoting this is unlikely to be the case. In fact, $\|R_{22}\|_2$ could be quite large. Therefore, we will adapt our QR factorization algorithm to try and ensure $\|R_{22}\|_2$ is small (when possible).

Question 3: Show that

$$\|R_{22}\|_2 \leq \sigma_1(A)$$

and that for any m, n, and k with $m \geq n$ and $k < \min(m, n)$ there exists a matrix A such that

$$\|R_{22}\|_2 = \sigma_1(A).$$

This shows that while R_{22} is bounded in size by the scale of A, there are examples where it is that large.

As stated, eq. (1) does not admit any flexibility to control R_{22}. So, we will instead consider a factorization of the form

$$A\Pi = QR \tag{4}$$

where Π is an $n \times n$ permutation matrix and Q and R are as in eq. (1). The key here is that by allowing for our algorithm to also compute a permutation matrix Π we have some ability to force R_{22} to be small. In the next section we will discuss how this is accomplished, but let's first see how such a factorization can help address the prior two problems. For any $k < \min(m, n)$ let

$$A \begin{bmatrix} \Pi_1 & \Pi_2 \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \\ & R_{22} \end{bmatrix} \tag{5}$$

be a partition of eq. (4) where Π_1 is the first k columns of Π and Π_2 is the remaining $n-k$ columns; the remainder of the factorization is partitioned as before. Moreover, let the set $\mathcal{C}$ denote the set of indices "selected" by Π_1. In other words $\mathcal{C}$ is such that $A(:, \mathcal{C}) = A\Pi_1$.

Question 4: Show that if we have a factorization of the form eq. (5) then there exists a matrix B such that

$$\|A - A(:, \mathcal{C})B\|_2 = \|R_{22}\|_2$$

and

$$\|A(:, \mathcal{C}) - A(:, \mathcal{C})B(:, \mathcal{C})\|_2 = 0.$$

Here we are just rewriting the earlier result using the permutations. A similar thing can be done for the least-squares problem.

An algorithm to compute a column-pivoted QR factorization

We are now finally ready to discuss an algorithm to compute eq. (4). Trying to solve something like $\min_\Pi \|R_{22}\|_2$ is provably hard, so we will resort to a greedy heuristic.[4]

Structurally, we will compute our factorization similar to what we did for pivoted LU factorizations—interlace computing permutations and "reductions." In particular, instead of eq. (1) we will compute a factorization of the form

$$Q^{(n)} \cdots Q^{(1)} A \Pi^{(1)} \cdots \Pi^{(n-1)} = R, \tag{6}$$

by alternating the computation of $\Pi^{(i)}$ and $Q^{(i)}$. I.e., we will compute $\Pi^{(1)}$ to swap the first column of A with some other column and then compute the matrix $Q^{(1)}$ to reduce the first column of $A\Pi^{(1)}$ to upper triangular form, compute $\Pi^{(2)}$ to swap the second column of $Q^{(1)}A\Pi^{(1)}$ with some

[4]The heuristic is based on work by Businger and Golub in 1965 and while there have been further developments in the area, the original algorithm remains the most widely used and is highly effective in practice.

column whose index is greater than or equal to 2 and so on. We then let $Q = Q^{(1)} \cdots Q^{(n)}$ and $\Pi = \Pi^{(1)} \cdots \Pi^{(n)}$.

The greedy heuristic we will follow is to remove the largest column from R_{22} at each step. Specifically, lets say we have completed k steps of our process have have the partial factorization

$$Q^{(k)} \cdots Q^{(1)} A \Pi^{(1)} \cdots \Pi^{(k)} = \begin{bmatrix} R_{11} & R_{12} \\ & M \end{bmatrix}, \tag{7}$$

where $R_{11} \in \mathbb{R}^{k \times k}$ is upper triangular and $M \in \mathbb{R}^{(m-k)\times(n-k)}$ is dense (because we have not yet reduced it to upper triangular form).

Question 5: Show that if we continue the factorization in eq. (7) without any additional permutations then $\|R_{22}\|_2 = \|M\|_2$.

To accomplish our goal, we will pick $\Pi^{(k+1)}$ to "remove" the largest column of M from R_{22} at step $k+1$. Specifically, let

$$j = \underset{i=k+1,\ldots,n}{\arg\max} \|M(:,i)\|_2.$$

We then let $\Pi^{(k+1)}$ be the permutation that swaps columns $k+1$ and j. After that we proceed to compute $Q^{(k+1)}$ to reduce the appropriate part of the column to upper triangular form an continue. Note that we could also try and pick j such that $\|R_{22}\|_2$ at the next step is minimized, but that is substantially more expensive since we have to compute/apply $n-k$ Householder reflectors and then compute an equal number of spectral norms—this actually affects the asymptotic complexity of the method.

Question 6: Show that Π_1 in eq. (5) only depends on $\Pi^{(1)}$ through $\Pi^{(k)}$. In other words, if we just want/need $\mathcal{C}$ we can stop the factorization after k steps. We can also recover the matrix B in our low-rank factorization without proceeding further.

Goal 1: Write pseudocode for the described algorithm, and show that its arithmetic complexity is $\mathcal{O}(mnk)$ if we stop at step k.[a]

[a] In practice there is a scheme to efficiently maintain the norms of the columns of M and update them at each step. Feel free to work out how this works, i.e., how the norms get updated by Householder reflectors. Such a scheme is practically far more efficient (albeit not without a few drawbacks), but does not change the asymptotic complexity.

Goal 2: Implement a column pivoted QR factorization and for $k = 20$ show that your code scales linearly in m and n and quadratically if $m = n$.

Goal 3: Build a matrix $G \in \mathbb{R}^{200\times200}$ where $G_{i,j} = e^{-2\|x_i - x_j\|_2^2}$ and $\{x_i\}_{i=1}^{200}$ are i.i.d. random points from the uniform distribution over the unit cube in 2d. Use your column pivoted QR factoration to compute a low-rank approximation of G using its columns for $k = 1, 2, \ldots, 100$ and show how the error compares with the optimal error for a rank-k approximation. This shows us how much we are "loosing" by constraining our low-rank approximation.

Compressing a simple neural network

We are finally ready to develop a method for compressing a simple neural network using a column-pivoted QR factorization. To illustrate how, we will start with a simple one hidden layer model

that has already been trained to achieve good accuracy. Let $W \in \mathbb{R}^{d\times n}$ and $a \in \mathbb{R}^n$ represent the weights in a model

$$y = \sigma(x^T W)a, \tag{8}$$

where $x \in \mathbb{R}^d$ is a data point and σ is a non-linear function applied entrywise to $x^T W$. The question we want to ask is if all n "neurons" (i.e., columns of W) are necessary. In other words, can we identify some weights $\widehat{W} \in \mathbb{R}^{d\times n'}$ and $a \in \mathbb{R}^{n'}$ with $n' < n$ such that for some target accuracy $\epsilon > 0$

$$|\sigma(x^T W)a - \sigma(x^T \widehat{W})\hat{a}| \leq \epsilon$$

for any reasonable data point x.[5] Note that here we are requesting good representation of the model with out any additional retraining of the weights.

To model "reasonable" x we will simply use i.i.d. samples from the training data. Specifically, let $X_c \in \mathbb{R}^{d\times n_c}$ represent n_c data points that we refer to as a compression set (i.e., each column of X_c is a data point). We can then re-frame the above question as finding $\widehat{W}$ and $\widehat{a}$ such that

$$\|\sigma(X_c^T W)a - \sigma(X_c^T \widehat{W})\hat{a}\|_2 \leq \epsilon. \tag{9}$$

Let $Z = \sigma(X_c^T W)$, and say we have computed a column-pivoted QR factorization of Z. Then, for any k if we have that

$$\|Z(:, \mathcal{C})T - Z\|_2 = \|R_{22}\|_2$$

where $T = \begin{bmatrix} I & R_{11}^{-1}R_{12} \end{bmatrix} \Pi^T$. Moreover, because the non-linearity is applied entrywise $Z(:, \mathcal{C}) = \sigma(X_c^T W(:, \mathcal{C}))$. Combining these facts we have that

$$\|\sigma(X_c^T W)a - \sigma(X_c^T W(:, \mathcal{C}))Ta\|_2 \leq \|R_{22}\|_2 \|a\|_2.$$

Letting $\widehat{W} = W(:, \mathcal{C})$ and $\hat{a} = Ta$ we recover a model that satisfies eq. (9) with $n_c = k$ provided that $\|R_{22}\|_2$ is sufficiently small.[6] This gives us a scheme to compress our model: (1) compute Z using some compression data, (2) use the column-pivoted QR factorization to select k important neurons, and (3) use the "interpolation" matrix T to correct the weights on the output.

A practical simplification

While it would be great to verify that $\|R_{22}\|_2 \|a\|_2 \leq \epsilon$ for our chosen ϵ, doing so becomes prohibitively expensive in practice—we end up computing too many spectral norms. It turns out that a reasonable heuristic to use is to simply choose k such that the diagonal element of R is sufficiently small. Specifically, we choose k to ensure that $|r_{k+1,k+1}|/|r_{1,1}| \leq \epsilon/\|a\|_2$—notably, this can be checked as soon as $\Pi^{(k+1)}$ and $Q^{(k+1)}$ have been computed and applied. This is much faster to compute and works well in practice.

Compressing a neural network

To extend the above idea to a multilayered network (albeit with only fully connected layers) we simply iterate the above idea over layers. Say

$$f_i(x) = \sigma(xW^{(i)})$$

[5] There are several ways we could be more formal about this, e.g., writing an expectation over the (unknown) data distribution, but for our purposes the slightly informal version suffices. The key point is that we only need the approximation to be accurate for reasonable data.

[6] Specifically, $\|R_{22}\|_2 < \epsilon/\|a\|_2$.

and our complete m layer model is

$$M(x) = s_m(f_m(f_{m-1}(\cdots f_1(x^T)))W^{(s)}). \tag{10}$$

In other words, it is a composition of m fully connected layers followed by a softmax layer denoted s_m with weights $W^{(s)}$.[7] We can then do the following: (1) using X_c compress f_1 as

$$\hat{f}_1(x^T) = \sigma(x^T W^{(i)}(:, \mathcal{C}))T,$$

(2) fold T into $W^{(2)}$ as

$$f_2(\hat{f}_1(x^T)) = \sigma(\sigma(x^T W^{(i)}(:, \mathcal{C}))(\widehat{W}^{(2)})),$$

where $\widehat{W}^{(2)} = TW^{(2)}$, and (3) repeat by using $\hat{f}_1(X_c^T)$ to compress $\widehat{W}^{(2)}$ and so on. For simplicity, will not do any compression of $W^{(s)}$ as we need to maintain a fixed output size.

ReLU and softmax

For this project we need to discuss two additional details. First, the nonlinearity σ we will use is known as ReLU and mathematically defined as $\sigma(x) = x$ if $x \geq 0$ and $\sigma(x) = 0$ if $x < 0$. Second. we will use our model to solve a classification task, i.e., given a data point x_i we would like to classify it into one of k categories. To accomplish this we pass the output of the final layer through a softmax layer denoted s_m. Specifically, given a vector $z \in \mathbb{R}^k$

$$(s_m(z))_i = \frac{e^{z_i}}{\sum_i e^{z_i}}.$$

In other words, it converts k numbers into probabilities that we can interpret as the probability that data point x_i has a given label. So, considering eq. (10) we have that the probability that x_i is assigned label j is $(M(x_i))_j$. We will then convert this to a simple prediction by labeling x_i as $\arg\max_j (M(x_i))_j$.

A concrete model

We now consider a specific problem. On the website you will find data that contains all the weights $W^{(1)}$ through $W^{(7)}$ and $W^{(s)}$ (i.e., the model has seven layers plus the softmax) for a model trained on the Fashion-MNIST[8] data set. This data set contains 70,000 28×28 grayscale images of clothing representing 10 types of item. The models task is to predict the class a given image belongs to, e.g., is it a t-shirt or a coat. You will also find two parts of the data set, a compression data set, X_c and a test data set X_{test} along with its labels y_{test} encoded as numbers 1-10. We will measure the accuracy of our model using the test set as

$$\text{acc}(M) = \frac{\text{number of correctly predicted labels for images in } X_{\text{test}}}{\text{total number of images in } X_{\text{test}}}.$$

Goal 4: Implement the model in eq. (10) using the provided weights. Compute and report the test accuracy.

[7] More on this later.

[8] https://github.com/zalandoresearch/fashion-mnist

Goal 5: Form the matrix

$$\sigma(X_c^T W^{(1)})$$

and show how the error of a rank-k approximation using a column-pivoted QR compares with that of an SVD.[a]

[a]Note that while the SVD is optimal, it is not at all clear how we could use an SVD of $\sigma(X_c^T W^{(1)})$ to sub-select columns of $W^{(1)}$—the structure of our low-rank factorization based on a column-pivoted QR factorization is key.

Goal 6: Use the scheme outlined above to compress the first layer of the provided model, i.e., we want to reduce the number of columns in $W^{(1)}$, and show how the accuracy varies as the size of the model is reduced (i.e., start with some small enough ϵ such that no neurons are removed and then gradually increase ϵ to get smaller and smaller models until the accuracy degrades significantly). Your results/discussion should include a plot of the accuracy vs the fraction of columns $W^{(1)}$ that remain. What do you observe?

Goal 7: Compress the whole model for a few ϵ you choose based on your answer to the prior goal (e.g., picking points that seem like a reasonable trade off between model size and accuracy). Again, report the accuracy of your compressed model vs the fraction of total number of columns in $\widehat{W}^{(1)}$ through $\widehat{W}^{(7)}$ that remain. What do you observe? Comment on your results and why (or why not) you feel the scheme implemented is practical/useful.

Goal 8: Finally, we come to the most open ended goal. The scheme outlined here for compressing the complete model works through layers progressively compressing them one at a time. However, compressing the first layer may impact how well later layers can be compressed. So, it is not clear such an approach is anywhere near optimal if we are concerned with the total number of parameters in our final model. Can you think up schemes that produce a better overall trade-off between accuracy and model parameters (i.e., they achieve a better accuracy than the simple approach for a given number of parameters)? Implement your ideas and see how good of a scheme you can get. If you have other ideas to improve the method (that do not involve "fine turning" the weights after compression), feel free to discuss and implement them.

What to do if your column-pivoted QR factorization code does not work

If you are not able to get a reliable implementation of a column-pivoted QR factorization working it is still possible to complete these later tasks. Specifically, Matlab, Julia, and NumPy all contain functions to compute such a factorization (as it is part of LAPACK). The available routines will not stop early and instead compute the full factorization always. However, this can still be used for our task by retrospectively analyzing the diagonal of R to determine k and then splitting the factorization appropriately. If you use these built-in routines, please say so in your report.