Large Scale Matrix Computation, Optimization and Learning (EE270) HW #4

Winter 2021 Due: **March 15 (Monday), 11:59 PM**

Instructions:

- After a genuine attempt to solve the homework problems by yourself, you are free to collaborate with your fellow students to find solutions to the homework problems. Regardless of whether you collaborate with other students, you are required to type up or write your own solutions. Copying homework solutions from another student or from existing solutions is a serious violation of the honor code. Please take advantage of the professor's and TA's office hours. We are here to help you learn, and it never hurts to ask!
- **The assignments should be submitted via Gradescope including your code attached as pdf.**

1. **Randomized Gauss-Newton Algorithm for Training Neural Networks (25 pts)**
In this problem, you will implement randomized Gauss-Newton (GN) algorithm to train a neural network. We will consider a single layer architecture and a squared loss training objective. This special case is also referred to as a nonlinear least squares problem. Consider a set of training data $\{x_i, y_i\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ is the i^{th} data sample and $y_i \in \{0, 1\}$ is the corresponding label. Then, use the following nonlinear function

$$\sigma(w, x) = \frac{1}{1 + e^{-w^T x}}$$

to fit the given data labels. In order to estimate w, we can minimize the sum of squares as follows

$$\begin{aligned} w^* &= \arg\min_w \sum_{i=1}^n (\sigma(w^T x_i) - y_i)^2 \\ &= \arg\min_w \|f(w) - y\|_2^2 \end{aligned}$$

where w^* denotes the optimal parameter vector, y is the vector containing labels $y_1, ..., y_n$, and the vector output function $f(w)$ is defined as

$$f(w) := \sigma(Xw) = \begin{bmatrix} \sigma(x_1^T w) \\ \vdots \\ \sigma(x_n^T w) \end{bmatrix}.$$

The above problem is non-convex unlike the ordinary least squares and logistic regression problems.

Download the provided MNIST dataset (`mnist_all.mat`), where you will use only the samples belonging to digit 0 (`train0`) and digit 1(`train1`). Next, we will apply the Gauss-Newton (GN) heuristic to approximately solve the non-convex optimization roblem. In the GN algorithm, you first need to linearize the nonlinear vector output function $f(w)$. The first order expansion of $f(w)$ around the current estimate, w_k is given by

$$f(w) \approx f(w_k) + J^k(w - w_k) \tag{1}$$

where $J^k \in \mathbb{R}^{n \times d}$ is the Jacobian matrix at the iteration k whose ij^{th} entry is defined as

$$J_{ij}^k = \left.\frac{\partial f_i(w)}{\partial w_j}\right|_{w=w_k} = \left.\frac{\partial \sigma(x_i^T w)}{\partial w_j}\right|_{w=w_k}.$$

Then, Gauss-Newton algorithm performs the update

$$w^{k+1} = \arg\min_w \|f(w_k) + J^k(w - w_k) - y\|_2^2 . \tag{2}$$

(a) Find the Gauss-Newton update (2) explicitly for this problem by deriving the Jacobian. Describe how the sub-problems can be solved. What is the computational complexity per iteration for $n \times d$ data matrices?

One can incorporate a step-size $\mu \in (0, 1]$ in the above update as follows.

$$\begin{aligned} &w^{k+1} = (1 - \mu)w^k + \mu d^k \\ &\text{where } d^k := \arg\min_w \|f(w_k) + J^k(w - w_k) - y\|_2^2 . \end{aligned} \tag{3}$$

This is referred to as damped Gauss-Newton algorithm.

(b) Implement the damped Gauss Newton algorithm on the MNIST dataset to classify digits 0 and 1. Find a reasonable step-size for fast convergence by trial and error. Plot the training error, i.e., $\|\sigma(Xw) - y\|_2^2$, as a function of the iteration index.

(c) You will now apply sampling to reduce computational complexity of the GN algorithm. Apply uniform sampling to the rows of the Least-Squares sub-problem (3) at each iteration. Plot the training error, i.e., $\|\sigma(Xw) - y\|_2^2$, as a function of the iteration index. Repeat this procedure for row norm scores sampling.

2. **Randomized Kaczmarz Algorithm (25 pts)**
In this question, you will implement the Randomized Kaczmarz Algorithm (see Lecture 16 slides) on the YearPredictionMSD dataset, which can be downloaded from https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD. In this file, there are 463,715 training and 51,630 test samples (songs) with 90 audio features and your goal is to predict the release year of each song using least squares regression. After downloading the file, you can import and partition the data using the following command in Python

```
import numpy as np
file_name='YearPredictionMSD.txt'
file = open(file_name, 'r')

data=[]
for line in file.readlines():
    fname = line.rstrip().split(',')
    data.append(fname)

data_arr=np.asarray(data).astype('float')
A=data_arr
b=(data_arr[:,0]-np.min(data_arr[:,0]))/(np.max(data_arr[:,0])-np.min(data_arr[:,0]))
```

You may also normalize the data matrix as follows

```
meanA=np.mean(A,axis=0)
stdA=np.std(A,axis=0)
A=(A-meanA)/stdA
```

(a) Plot the training and test cost vs iteration curves for Randomized Kaczmarz Algorithm (i.e., the one with optimal sampling distribution) and the randomized algorithm with uniform sampling distribution on the same figure. For the uniformly sampled version, you should use an explicit learning rate and tune this parameter by trial and error.

(b) Note that the analysis of the Randomized Kaczmarz Algorithm assumes that the linear system $Ax = b$ is consistent, which is may not be valid in our case. Show that an optimal solution of the least squares problem $\min_x \|Ax - b\|_2^2$ can be found via solving the augmented linear system $\begin{bmatrix} A & -I \\ 0 & A^T \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$.

(c) Since the augmented linear system in part (b) is consistent, we may apply Randomized Kaczmarz Algorithm. Repeat part (a) using Randomized Kaczmarz algorithm on the augmented linear system.

(d) Repeat part (a) using SGD with diminishing step-sizes (see page 11 of Lecture 16 slides) by tuning the initial step size.

3. **Randomized Low-Rank Approximation and Randomized SVD (25 pts)**
In this problem, you will implement a randomized approach in order to obtain a low-rank approximation for a given data matrix. Assume that you are given a data matrix $A \in \mathbb{R}^{n\times d}$ with $U\Sigma V^T$ as its SVD. The best rank-k approximation for the data matrix is $A_k = U_k\Sigma_k V_k^T = \sum_{i=1}^{k} \sigma_i u_i v_i^T$. However, since this is computationally expensive, you need to use another approach to reduce the complexity.

First, you will generate a 5000×1000 data matrix with decaying singular values. To construct such a matrix, you can first generate a random Gaussian matrix with zero mean and identity covariance. Then, compute the SVD of this matrix as $U\Sigma V^T$. Now, replace Σ with a diagonal matrix $\hat{\Sigma}$ with decaying entries, $\hat{\Sigma}_{ii} = i^{-2}$. Then, you can construct the data matrix with decaying singular values as $A = U\hat{\Sigma}V^T$.

(a) One approach to obtain a rank-k approximation is as follows.

- Obtain a random approximation for the data matrix A as C by uniformly sampling k columns and scaling appropriately.
- Verify the approximation by computing the relative error $\|AA^T - CC^T\|_F / \|AA^T\|_F$.
- Compute the randomized rank-k approximation defined as $\tilde{A}_k = CC^\dagger A$.

Plot the approximation error, i.e., $\|A - \tilde{A}_k\|_2^2$, as a function of the rank k. Verify the error bound we have in Lecture 19 slides, i.e., $\|A - \tilde{A}_k\|_2^2 \leq \sigma_{k+1}^2 + \epsilon\|A\|_2^2$, and find the numerical value of ϵ. Repeat this procedure for uniform sampling, column norm score sampling, Gaussian sketch, $+1/-1$ i.i.d. sketch, and randomized Hadamard (FJLT) sketch.

(b) Now we use the low rank approximation in part (a) to produce an approximate Singular Value Decomposition of A. The Randomized SVD algorithm is as follows.

- Generate a sketching matrix S.
- Compute $C = AS$
- Calculate QR decomposition: $C = AS = QR$
- Calculate the SVD of $Q^T A$, i.e., $Q^T A = U\Sigma V^T$
- Approximate SVD of A is given by $A \approx (QU)\Sigma V^T$.
- Note that the randomized rank-k approximation $\tilde{A}_k := (QU)\Sigma V^T = QQ^T A = CC^\dagger A$ as in part (a)

Plot the approximation error, i.e., $\|A - (QU)\Sigma V^T\|_2^2$, as a function of the rank k. To compare with the exact SVD, plot $\|A - A_k\|_2^2$, where A_k is the best rank k approximation of A using the SVD of A. Repeat the whole procedure for uniform sampling, column norm score sampling, Gaussian sketch, $+1/-1$ i.i.d. sketch, and randomized Hadamard (FJLT) sketch.

4. **CUR decomposition (25 pts)**
CUR decomposition is a dimensionality reduction and low-rank approximation method in a similar spirit to Singular Value Decomposition (SVD). One particular difficulty of SVD is that the left and right singular vectors are difficult to interpret since they lack any direct meaning in terms of the original data. On the other hand, CUR decomposition is interpretable since it involves small subsets of the rows/columns of the original data.

Suppose that A is an m $\times$ n matrix of approximate rank k, and that we have two column/row subsampled approximations

$$C := A(:, J_S) \tag{4}$$
$$R := A(I_S, :) \tag{5}$$

where I_S, J_S are appropriate subsets. Then we can approximate the matrix A via

$$A \approx CC^{\dagger}AR^{\dagger}R = CUR\,, \tag{6}$$

where $U := C^{\dagger}AR^{\dagger}$ and the superscript $\dagger$ denotes the pseudoinverse operation. We choose set of columns C and a set of rows R, which play the role of U and V in SVD. We may pick any number of rows and columns. Therefore, this factorization provides an interpretable alternative to the Singular Value Decomposition. Furthermore, CUR has computational advantages especially for sparse matrices. Particularly, since CUR directly select random rows and columns C and R are sparse for a sparse matrix A, whereas U and V in SVD can still be dense matrices.

You will be using the movie ratings dataset from the https://movielens.org/ website provided at https://grouplens.org/datasets/movielens/100k/. The dataset contains 100,000 ratings from 943 users on 1682 movies. The 'usermovies' matrix of dimension 943 x 1682 is very sparse.

(a) For $m = 1, ..., 10$, apply uniform row/column subsampling with sample size m to obtain C and R matrices of rank m and solve U to obtain the CUR decomposition. Plot the approximation error in Frobenius norm and spectral norm as a function of m.

(b) Repeat part (a) using ℓ_2 row/column norm scores for row/column subsampling respectively.

(c) Repeat part (a) using leverage scores of A and A^T for row/column subsampling respectively.