CME 323: Distributed Algorithms and Optimization

Instructor: Reza Zadeh (rezab@stanford.edu)

TA:Alex Yang (yangzj@stanford.edu)

HW#2 – Due Thursday May 9 11:59pm (on Gradescope)

1. **List Prefix Sums** As described in class, List Prefix Sums is the task of determining the sum of all the elements before each element in a list. Let us consider the following simple variation.

   - Select each element from the list randomly and independently with probability $1/\log n$ and add it to a set S. Add the head of the list to this set, and mark all these elements in the list.
   - Start from each element $s \in S$, and in parallel traverse the lists until you find the next element in S (by detecting the mark) or the end of the list. For $s \in S$, call this element found in this way $\texttt{next}(s)$. While traversing, calculate the sum from s to $\texttt{next}(s)$ (inclusive of s but exclusive of $\texttt{next}(s)$), and call this $\texttt{sum}(s)$.
   - Create a list by linking each $s \in S$ to $\texttt{next}(s)$ and with each node having weight $\texttt{sum}(s)$.
   - Compute the List Prefix Sums on this list using pointer jumping. Call the result $\texttt{prefixsum}(s)$.
   - Go back to the original list, and again traverse from each s to $\texttt{next}(s)$ starting with the value $\texttt{prefixsum}(s)$ and adding the value at each node to a running sum and writing this into the node. Now all elements in the list should have the correct prefix sum.

   Analyze the work and depth of this algorithm. These should both be given with high probability bounds.

2. **Shortest Path for Weighted Directed Graph** Consider a directed graph $G = (V, E)$ with non-negative weights $\omega : E \to \mathbb{R}^+$. The task is to develop an algorithm to efficiently find the shortest paths from the source $s \in V$ to any other vertex $v \in V$, i.e.

$$p^*(v) = \text{argmin}_{p\,\text{valid}} \sum_{i=1}^{n_p} \omega_i(u_i, u_{i+1}),$$

   where a valid path from source s to v is a sequence $p = (u_0, u_1, \ldots, u_{n_p})$ satisfying:

   - $u_0 = s$;
   - $u_{n_p} = v$;
   - $(u_i, u_{i+1}) \in E$ for $i = 0, \ldots, n_p$.

   Now assume that the operation of finding neighbors of a vertex can be performed in constant time and constant depth (both $O(1)$). Design an algorithm that achieves $O(nm)$ work and $O(n \log n)$ depth. ($n = |V|$ and $m = |E|$.)

3. **Singular Value Decomposition for SPSD Matrices** Given a symmetric positive definite matrix $A \in \mathbb{R}^{n \times n}$, in this problem we explore how to efficiently compute its singular value decomposition $A = U^T S U$, where U is orthogonal matrix, S is diagonal. Here we assume the singular values satisfy $\lambda_1 > \lambda_2 > \cdots > \lambda_n$.

   (a) Instead of directly performing QR-iteration on A, we want to first convert matrix A to a symmetric tridiagonal matrix using Householder reflection, i.e. constructing an orthogonal matrix Q_0, s.t. $T = Q_0^T A Q_0$ is symmetric tridiagonal. Give a parallel algorithm to solve this problem, and analyze its work and depth. (Hint: Householder reflection $H = I_n - 2\frac{(e_\alpha - e_\beta)(e_\alpha - e_\beta)^T}{(e_\alpha - e_\beta)^T(e_\alpha - e_\beta)}$ projects e_α to e_β where $e_\alpha = \alpha/||\alpha||_2$ and $e_\beta = \beta/||\beta||_2$.)

   (b) Now we perform QR-iteration on symmetric tridiagonal matrix T to achieve SVD for T using Givens rotation. Obviously, if we get the SVD of T, i.e. $S = \tilde{Q}^T T \tilde{Q}$, then we have $S = (Q_0\tilde{Q})^T A Q_0 \tilde{Q}$, which is the SVD of A. Now, let's explore the following algorithm:

```
function QR-ITERATION FOR SYMMETRIC TRIDIAGONAL MATRIX($\boldsymbol{s}[1 \dots n]$)
    Let $Q_0 \leftarrow I_n$, $T_0 \leftarrow T$
    for $t = 1$ to $T$ do
        Let $Q_t \leftarrow Q_{t-1}$, $T_t \leftarrow T_{t-1}$
        for $k = 1$ to $n - 1$ do
            Let $\alpha \leftarrow T_t[k : k+1, k]$, $(c, s)^T \leftarrow \alpha/||\alpha||_2$
            $G_k \leftarrow \text{Givens}(k, c, s)$
            $T_t \leftarrow G_k * T_t * G_k^T$
            $Q_t \leftarrow G_k * Q_t$
        end for
    end for
    return $T_T$, $Q_T$
end function
```

   Here, the matrix representation of Givens Rotation is

$$\text{Givens}(k, c, s) = \begin{bmatrix} I_{k-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \begin{bmatrix} c & s \\ -s & c \end{bmatrix} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I_{n-k-1} \end{bmatrix}.$$

   Analyze the work and depth of this algorithm, then compare it with part (a). Write a few sentences about your findings.

4. **Stochastic Gradient Descent**

   (a) In class we proved that gradient descent on L-smooth functions is guaranteed to decrease the function value at each iteration. Stochastic gradient descent, on the other hand, does not have the same guarantee. Provide an example where stochastic gradient descent does not produce a descent step. Specifically, find a function $f(x) = \sum_{i=1}^{m} f_i(x)$, and an iterate x_0 such that for all step sizes, there exist i such that $f(x_1) > f(x_0)$ (where $x + 1 := x_0 - \alpha \nabla f_i(x)$).

(b) This exercise will guide you through the convergence proof of SGD. As a reminder, we are proving that if there exists a constant G such that $\mathbb{E}[\|\nabla f_i(x)\|^2] \leq G^2$ and $f(x)$ is μ-strongly convex. Then, with step-sizes $\gamma_k = \frac{1}{\mu k}$, we have

$$\mathbb{E}[\|x_k - x_*\|^2] \leq \frac{\max\{\|x_1 - x_*\|^2, \frac{G^2}{\mu^2}\}}{k}.$$

- Using strong convexity, prove that

$$\langle \nabla f(x_k) - \nabla f(x_*), x_k - x_* \rangle = \langle \nabla f(x_k), x_k - x_* \rangle \geq \mu \|x_k - x*\|^2$$

- Apply the previous step, to express $\mathbb{E}[\|x_{k+1} - x_*\|^2]$ in terms of $\mathbb{E}[\|x_k - x_*\|^2]$, γ_k, G, and μ.
- Prove the convergence of SGD using induction.

5. **HOGWILD!** This exercise will provide examples applying the main theorem of HOGWILD!. Recall that in HOGWILD!, the objective function we want to minimize is :

$$f(x) = \sum_{e \in E} f_e(x_e)$$

where we define the hyperedge e to be the subset of variables that f_e depends on. Figure 1 depicts such a graph. Then, if we denote the average degree of the conflict graph as $\overline{\Delta}_C$, convergence is still guaranteed if the core delay is less than $\tau \leq \frac{n}{2\overline{\Delta}_C}$ (i.e., no more than τ samples are being processed while a core is processing one).

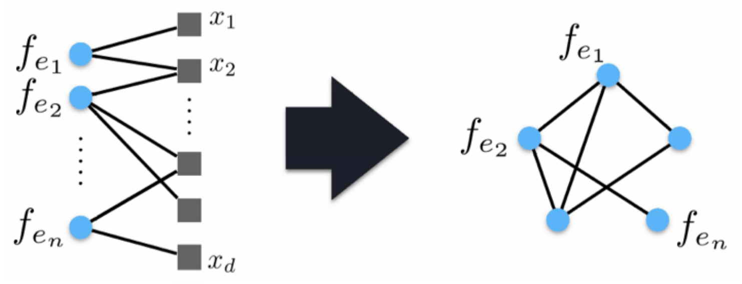

Figure 1: The function-variable and conflict graph for sparse functions.

- **Graph Cuts** In graph cuts problems, we are given a sparse matrix W which indexes similarity between node. We want to match each node to a list of D classes i.e., we want assign a vector $x_i \in \{v \in \mathbb{R}^D | \sum_{j=1}^D v_j = 1, v_j \geq 0\}$ that solve the following optimization problem.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{(u,v)\in E} w_{uv} \|x_u - x_v\|_1 \\ \text{subject to} \quad & x_u \in \{v \in \mathbb{R}^D | \sum_{j=1}^D v_j = 1, v_j \geq 0\}. \end{aligned} \tag{1}$$

Prove that

$$\frac{\overline{\Delta}_C}{n} = \mathcal{O}\left(\text{Avg. deg.}\right)$$

6. **Implement logistic regression using tensorflow.** Use the following code to generate train and test data. Note that we have set seed (using "random_state=42"). Use cross-entropy loss and gradient descent optimizer with a learning rate of 0.01. Use batch_size of 100, and run for 500 steps. Report the accuracy on test set.

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Generate data
X_data, y_data = make_classification(n_samples=200, n_features=2,
n_redundant=0, random_state=42)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,
test_size=0.2, random_state=42)

# Plot training data
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
plt.show()
```