Homework 1: Numerical Stability and Linear Systems

Due September 21, 2023 (100 points + 20 points extra credit)

Problem 1 (20 points)**.** Suppose $\mathbf{x} \in \mathbb{R}^n$ has elements $x_1, \ldots, x_n \in \mathbb{R}$. The *log-sum-exp* function, common in machine learning code, can be written:

$$f(\mathbf{x}) = \log \sum_{i=1}^{n} e^{x_i}.$$

(a) Suppose we implement the formula above numerically. Give examples of cases where the computation of $f(\mathbf{x})$ can led to underflow and overflow.

(b) For $a \in \mathbb{R}$, show $f(\mathbf{x}) = a + \log \sum_{i=1}^{n} e^{x_i - a}$. Explain how taking $a = \max_i x_i$ avoids the issues you identified in the previous part.

(c) For $\mathbf{g} = \nabla f(\mathbf{x})$, show $g_j = \exp(x_j - \log \sum_{i=1}^{n} e^{x_i})$.

(d) Show that $\max_i x_i \leq \frac{1}{t} f(t\mathbf{x}) \leq \max_i x_i + \frac{1}{t} \log n$ for any $t > 0$. How would you use $f(\cdot)$ as a differentiable approximation of the map $\mathbf{x} \mapsto \max_i x_i$?

Problem 2 (20 points)**.** We can use a *condition number* to estimate how difficult it will be to approximate $f(\mathbf{x})$ for some $\mathbf{x}$. Define the relative condition number of $f(\cdot)$ as

$$\operatorname{cond}(f, \mathbf{x}) = \lim_{\varepsilon \to 0} \sup_{\|\Delta\mathbf{x}\| \leq \varepsilon \|\mathbf{x}\|} \frac{|f(\mathbf{x} + \Delta\mathbf{x}) - f(\mathbf{x})|}{\varepsilon |f(\mathbf{x})|}.$$

(a) Suppose we use the ∞-norm: $\|\mathbf{x}\|_\infty = \max_k |x_k|$. For log-sum-exp function f from Problem 1, show

$$\operatorname{cond}_\infty(f, x) = \frac{\max_k |x_k|}{|\log \sum_k e^{x_k}|}.$$

Hint: You may wish to use the Taylor expansion of f about $\mathbf{x}$.

(b) Argue that log-sum-exp is ill-conditioned when $x_i \approx -\log n$ for all i.

(c) Argue that log-sum-exp is well-conditioned when $\max_i x_i = \max_i |x_i|$.

Problem 3 (25 points)**.** Suppose we have a multi-class classification problem, wherein we wish to train a classifier to map from points $\mathbf{x} \in \mathbb{R}^d$ to one of k classes. A simple model for this task is to learn a *softmax classifier*, parameterized by an unknown matrix $A \in \mathbb{R}^{k \times d}$, which maps points $\mathbf{x} \in \mathbb{R}^d$ to probability vectors via:

$$\mathbf{p}(\mathbf{x}; A) = \frac{e^{A\mathbf{x}}}{\mathbf{1}^\top e^{A\mathbf{x}}}.$$

Here, we take the convention that $e^{\mathbf{v}}$ is the vector whose elements are e^{v_i}. We can think of $\mathbf{p}(\mathbf{x}; A)$ as a vector of *probabilities*, where $p_i(\mathbf{x}; A)$ is the likelihood that $\mathbf{x}$ is in class i.

Suppose we are given a dataset $\{(\mathbf{x}_i, c_i)\}_{i=1}^N$, where $c_k \in \{1, \ldots, k\}$. The *negative log likelihood* loss associated to A is:

$$\ell(A) = -\frac{1}{N}\sum_{i=1}^N \log p_{c_i}(\mathbf{x}_i; A).$$

Notice ℓ is *small* when our classifier correctly classifies all the points in the dataset.

(a) Take $X \in \mathbb{R}^{d\times N}$ to be the matrix whose columns are the $\mathbf{x}_i$'s, and suppose $C \in \{0,1\}^{k\times N}$ satisfies

$$C_{ij} = \begin{cases} 1 & \text{if } i = c_j \\ 0 & \text{otherwise.} \end{cases}$$

Show $\ell(A) = \frac{1}{N}(\log \mathbf{1}^\top e^{AX})\mathbf{1} - \frac{1}{N}\text{tr}(C^\top AX)$, where $\mathbf{1}$ denotes the vector of all ones and matrix exponentiation is *elementwise*. What is the gradient of ℓ with respect to A?

(b) Implement $\ell(A)$ and $\nabla_A \ell(A)$ in `nllSoftmax` in `hw1.py`. Make sure your implementation is stable to large values in $A\mathbf{x}$. The assignment includes a dataset and gradient descent code to test the resulting model.

Problem 4 (10 points)**.** Provide an algorithm for computing the determinant of a matrix $A \in \mathbb{R}^{n\times n}$ in $O(n^3)$ time. Justify its correctness and that it works for *all possible* square matrix inputs A, up to numerical precision.

Problem 5 (25 points)**.** In the *regression* problem, we are given a number of pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^N \subset \mathbb{R}^d \times \mathbb{R}$ and wish to find a function $f(\cdot)$ so that $f(\mathbf{x}_i) \approx y_i$.

(a) Suppose we seek to fit a function of the form $f(\mathbf{x}) = \mathbf{a}^\top \mathbf{x}$. Provide a $d \times d$ linear system of equations for recovering $\mathbf{a} \in \mathbb{R}^d$ via least-squares with Tikhonov regularization. Your system should be in terms of $X \in \mathbb{R}^{d\times N}$, the matrix whose columns are the $\mathbf{x}_i$'s, as well as $\mathbf{y} \in \mathbb{R}^N$, the vector whose elements are the y_i's.

(b) Show that $\mathbf{a} \in \mathbb{R}^d$ from the previous part can be written $X\mathbf{c}$ for some $\mathbf{c} \in \mathbb{R}^N$.

(c) Give a $N \times N$ linear system of equations to recover $\mathbf{c}$. You can assume $X^\top X$ is invertible.

(d) Now, suppose we instead use the form $f_\phi(\mathbf{x}) = \mathbf{a} \cdot \phi(\mathbf{x})$, for some function $\phi : \mathbb{R}^d \to \mathbb{R}^k$ and an unknown $\mathbf{a} \in \mathbb{R}^k$. Define $K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y})$. Give an algorithm for evaluating the least-squares fit $f_\phi(\mathbf{z})$ at an arbitrary $\mathbf{z} \in \mathbb{R}^d$ using only evaluations of K—*not* evaluations of ϕ—as well as linear algebra operations.

(e) Which matrix factorization would you use for the linear system of equations in part (d)?

Problem 6 (20 points, extra credit)**.** A typical model of floating-point arithmetic assumes *multiplicative error*. Suppose we attempt to compute $c = a\, op\, b$ where $op \in \{+, -, \times, \div\}$. We will assume after rounding that the computer provides us with a value $\hat{c}$ satisfying $\hat{c} = c \cdot (1 + \delta)$, where $|\delta| < u$ for some fixed constant u. Applying multiple arithmetic operations incurs *different* δ's:

$$\widehat{(a + b) + c} = ((a + b)(1 + \delta_1) + c)(1 + \delta_2).$$

(a) Suppose we have a list of nonnegative values $w_1, \ldots, w_n \geq 0$ and wish to compute $s_k = \sum_{i=1}^{k} w_i$; our code uses a `for` loop, using the relationship $s_i = s_{i-1} + w_i$ with $s_1 = w_1$. Prove the bound $|\hat{s}_{n-1} - s_{n-1}| \leq u \sum_{i=1}^{n-1} (n - i) w_i + O(u^2)$.
Hint: First show $|\hat{s}_i - s_i| \leq u \sum_{j=1}^{i} s_j + O(u^2)$ by induction.

(b) For some $a \in \mathbb{R}$, define $w_i = e^{x_i - a}$. Assuming exponentiation also generates multiplicative error, justify the bound $|\hat{w}_i - w_i| \leq ((1 + |a - x_i|)u + O(u^2))w_i$.
Note: For all parts of this problem, you can assum $\hat{w}_i \geq 0$ for all i.

(c) Suppose we compute $p_k = \sum_{i=1}^{k} e^{x_i - a}$ via the recurrence $p_i = p_{i-1} + w_i$ and $w_i = e^{x_i - a}$ (with $p_0 = 0$). Accounting for rounding error, using the previous two parts, justify the inequality

$$|\hat{p}_{n-1} - p_{n-1}| \leq u \sum_{i=1}^{n-1} (2 + |a - x_i| + n - i) w_i + O(u^2).$$

(d) For simplicity, suppose we reshuffle so that $x_n = \max_i x_i$. Then, Problem 1 computes

$$f(\mathbf{x}) = x_n + \log\left(1 + \sum_{i=1}^{n-1} e^{x_i - x_n}\right).$$

Suppose we code up this forumla to approximate $z = x_n + \log(1 + p_{n-1})$. If we use Python's `log1p` function, we incur error once: $\widehat{\log(1 + x)} = (\log(1 + x))(1 + \delta)$. Derive the bound

$$|\hat{z} - z| \leq u \left[2|z| + n - x_{\min}\right] + O(u^2).$$

(e) If we repeat the analysis above without the log-sum-exp trick from Problem 1, we will find a different inequality $|\hat{z} - z| \leq u \left[|z| + n + 1\right] + O(u^2)$. You've done enough annoying computations: We will not ask you to derive this bound! Based on this formula and the solution to the previous part, explain how shifting from Problem 1 does *not* yield an obvious improvement to accuracy.