Homework 0 of CS 291K (Fall 2022)

University of California, Santa Barbara

Due by 09/27

Notes:

- The homework is a self-assessment for your readiness in math and coding to enjoy the course.
- It counts a small 5% towards the final evaluation, and has a short deadline. Please start doing it right away.

---

1 Why should I do this homework?

Machine learning often models and processes data coming from a complex real world problem. The data often involve with many variables and are uncertain and probabilistic. Formally learning ML requires us to use elegant notations and languages from linear algebra, probability theory and statistics so that we can express these ideas concisely. In this homework, we will go over the basic notations to help you remember things you learned from your calculus / linear algebra / probability / statistics courses (**Problem 1**), and for you to learn basic tools for python / Numpy programming (**Problem 2**).

Note that if you have trouble solving a small fraction of the problems, don't worry! That is what this homework is for. Try your best and refer to the materials we provide, and get help from the TA / instructors.

That said, if you are having a lot of trouble with most of the problems, then this course might require more effort than a typical course for you to enjoy the materials. We would suggest you to consider taking CS165B instead for a more gentle introduction to ML and come back to us in the future!

2 Standard notations

Before starting on the questions, you may wish to review the following standard notations. You can also come back to these whenever you find equations / symbols that you don't understand.

Some standard notations in optimization: $\max, \min, \arg\max, \arg\min$ are commonly used when working with optimizations programs. The subscripts of these operators denote the "argument / variable" that you are optimizing over. For example, in $\min_{\theta \in \Theta} f(\theta)$, f is the objective function or

criterion function, θ is the argument or variable that you are optimizing over. Θ is the domain that you can choose θ from (sometimes abbreviated when it is the whole space or clear from context). $\min_{\theta\in\Theta} f(\theta)$ returns the minimum objective function value; $\arg\min_\theta f(\theta)$ returns the argument θ^* that achieves the minimum function value.

For example, let's say you are running a convenience store. θ is how much inventory of each item you keep, $f(\theta)$ is the expected business cost minus revenue each month, and Θ describes the constraints, such as size of your storage space, available fund and so on. You goal is to optimize your inventory so as to maximize your net profit.

Some standard notations in linear algebra: We will denote scalars and vectors by lower case letters, e.g., x, a, v, θ. Whether they are scalars or vectors (a vector is just an array of scalars) are usually clear from context and should be specified by, e.g., $x \in \mathbb{R}^d$ — indicating that $x = [x_1, x_2, ..., x_d]^T$. The $(\cdot)^T$ denotes the transpose, which makes x a column vector and $\mathbb{R}$ is the space of real numbers, indicating that each entry of x is numerical. x_i denotes the ith element of the vector x in the above, but sometimes it could denote the ith data point in a dataset $\{x_1, ..., x_n\}$ where each $x_i \in \mathbb{R}^d$ is a d-dimensional vector. We use capital letters, e.g., X, A, Θ to denote matrices. For example, the above dataset can be represented by a matrix of size $d \times n$, i.e., $X \in \mathbb{R}^{d\times n}$ where the ith column of matrix X is the ith data point x_i. When we write $X\Theta$ it refers to matrix multiplication and it is always a good idea to do a dimension check to see that the matrix multiplication is compatible. For example, if $x, a \in \mathbb{R}^d$ we can write $x^T a$ which will return a scalar, because we are multiplying a matrix of $1 \times d$ to a matrix of size $d \times 1$. In comparison, xa is not a valid notation because the inner dimension does not match between the two matrices.

Python codes for most AI agents are usually direct translation of these matrix and vector manipulations. Being able to write down what you are going to code in linear algebraic notations will help you to write efficient, elegant and bug-free code. It will also allow you to more easily learn how to use popular AI software such as sklearn, tensorflow or pytorch.

3 Homework problems

Problem 1. Refreshers on Optimization and probability fundamentals.

(a) (Continuous optimization) Let $x_1, \ldots, x_n$ be real values. Consider a quadratic function $f(\theta) = \sum_{i=1}^n w_i(x_i - \theta)^2$. Assume $w_i > 0$. Derive the optimal solution θ^* that minimizes $f(\theta)$ - denoted by $\theta^* = \arg\min_\theta f(\theta)$? What happens if some w_i are negative?

(b) (counting and combinatorics) If $2n$ kids are randomly divided into two equal subgroups, find the probability that the two tallest kids will be: (i) in the same subgroup; (ii) in different subgroups.

(Hint: Try putting the tallest and the second tallest into Group 1 and Group 2, then count the total number of different group assignments in each of the four possibilities.

Hint 2: Check if your solution is correct for $n = 2$.)

(c) (Bayes rule) In answering a question on a multiple choice test, a candidate either knows the answer with probability p ($0 \le p < 1$) or does not know the answer with probability $1 - p$.

If she knows the answer, she puts down the correct answer with probability 0.99, whereas if she guesses, the probability of his putting down the correct result is $1/k$ (k choices to the answer). Find the conditional probability that the candidate knew the answer to a question, given that she has made the correct answer.

(d) (Likelihood and maximum likelihood) Consider a biased coin with the probability of turning up head $0 < p < 1$. We flip the coin 10 times and get a sequence of outcomes: $\{T, H, H, T, T, H, H, H, T, H\}$. We know that the probability (likelihood) of observing this sequence

$$L(p) := (1-p) \cdot p \cdot p \cdot (1-p) \cdot (1-p) \cdot p \cdot p \cdot p \cdot (1-p) \cdot p = p^6(1-p)^4$$

Calculate the value of p that maximizes the likelihood $L(p)$.

(Hint: you may wish to consider maximizing $\log L(p)$ instead. Why does it preserve the $\arg\max$ — the p^* that maximizes $L(p)$?)

(e) (Calculus, gradients) Let $w \in \mathbb{R}^d$ be a column vector. Let $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$ be column vectors of the same dimension d and $y_1, \ldots, y_n \in \mathbb{R}$ be scalars. Let $\lambda \in \mathbb{R}$ be a non-negative scalar value. Consider function $F(w) = \sum_{i=1}^n (x_i^T w - y_i)^2 + \lambda \sum_{i=1}^d w_i^2$. Calculate the gradient of F. Recall that the gradient with respect to a vector of variables is the vector of partial derivatives with respect to each variable w_i:

$$\nabla f(w) = \left[\frac{\partial F(w)}{\partial w_1}, \ldots, \frac{\partial F(w)}{\partial w_d} \right]^T \in \mathbb{R}^d.$$

(Hint: It is easier to work out the partial derivatives one at a time, then concatenate the partial derivatives into a gradient by applying the definition above. The same hint also applies to other instances where you need to calculate the gradient.)

(f) (Chain-rule and softmax function) Let $x_1, \ldots, x_n$ be real values. Let

$$f(x_1, \ldots, x_n) = \log \sum_{i=1}^n \exp(x_i).$$

This is called the softmax function or the log-sum-exp function[1] Calculate the gradient of f w.r.t. vector $x = (x_1, \ldots, x_n)^T$.

(g) (Simple mathematical proof) For the soft-max function in part (f). Prove that $\max_i x_i \leq f(x_1, \ldots, x_n) \leq \max_i(x_i) + \log n$.

(Hint: A good practice when writing proof is to write a sequence of inequalities and explain every line in the following form:

"$f(x_1, \ldots, x_n) = \log \sum_{i=1}^n \exp(x_i) \leq \ldots \leq \ldots \leq \max_i(x_i) + \log n$. The first inequality is by definition, the second inequality is because ..., the third inequality is because ... ")

[1]Note that this softmax function is a scalar function $\mathbb{R}^n \to \mathbb{R}$ and it is different from the "softmax" transformation typically used in machine learning to convert any score vector to a probability vector, i.e., a vector valued function from $\mathbb{R}^n \to \mathbb{R}^n$. The latter is a misnomer but has a wide-spread misuse to the point that it becomes a convention. We will refer to the softmax transform, i.e., $F(x_1, \ldots, x_n) = \frac{[e^{x_1}, \ldots, e^{x_n}]}{\sum_{i=1}^n e^{x_i}}$, by soft-argmax, because it returns a probability distribution vector that approximates the one-hot representation of an argmax output.

Problem 2. Refreshers on time complexity, data structure and python / numpy. Please write codes for python3.

For easy prototyping, it is a good idea to install "Jupyter Notebook" so that you can easily debug your code.

(a) In numpy, generate two matrices A and B with size 5 by 4 and size 4 by 3 respectively. In matrix A, make sure that the values are $1, 2, 3, \ldots, 19, 20$, from left-to-right then top to bottom. In matrix B, make sure that the values are $1, 2, 3, \ldots, 11, 12$ from top to bottom, then left to right. Write a function that takes matrix A, matrix B as inputs and return the matrix product of AB.

(Hint: Note that A@B denotes matrix multiplication, while A*B aims at doing pointwise multiplication. To make it more explicitly, you can use use numpy.dot for matrix multiply.)

(b) (Sparse matrix-vector multiplication) What is the worst case time complexity (in Big O notation) of multiplying a matrix A of dimension $\mathbb{R}^{n\times n}$ with a dense vector $v \in \mathbb{R}^n$? What if matrix A is sparse, denote the number of non-zero elements by $\text{nnz}(A)$? Write a python function that takes a nonnegative integer n and outputs a sparse matrix A of size $(n-1)\times n$, such that for any $x \in \mathbb{R}^n$, $Ax = [x_1 - x_2, ..., x_{n-1} - x_n]^T$.

(Hint: you can use numpy.array to represent the vector x, but use the sparse matrix library in python (scipy.sparse) to construct matrix A. In this way, your code will take $O(n)$ rather than $O(n^2)$ time and space.)

(c) (Manipulating text using python, data structures) Write a python function to take a data file (data_example.txt) and return the unique words and the corresponding number of times they each appeared in the file. Each line of the data file (a plain text file) contains a paragraph of text. What are the data-structures you used and what is the time-complexity of your function in Big O notation. What is the space complexity of your code?

(Hint: You first need to read the text file line by line. See `https://stackoverflow.com/questions/3277503/how-to-read-a-file-line-by-line-into-a-list` for an example. The for each line, you should split them into words. Finally, you should try finding the unique words that appeared as well as the number of times they appeared. among the build-in data structures, eg., list, dict, set, etc., which one best suits this problem?)

(d) (Recursion with bookkeeping in python) The Fibonacci numbers are a sequence of the form $[0, 1, 1, 2, 3, 5, 8, ...]$, namely, each number is the sum of two numbers before it. Implement a *recursive* python function with a name "fibonacci" that takes a positive integer n and outputs the nth Fibonacci number. A naive recursion-based implementation will have exponential time, but we can keep a global variable (a dictionary) that keeps the result of all input $m \leq n$ so each one of them is computed at most once. What is the time complexity of this version of recursion with book keeping?