LIGN 167: Problem Set 1

October 3, 2023

Instructions

Collaboration policy: You may collaborate with up to three other students on this problem set (that is, total group size of 4). You should submit **one response** per group, using group submissions on Gradescope. When you submit your work, you must indicate who was in the group, and what each of your individual contributions were.

Starter code: We are providing starter code for the problem set. This starter code contains function signatures for all of the functions that you are defining. You **must** use this starter code when writing answers to the problems. If you do not follow the format of the starter code, **you will not receive credit.**

IMPORTANT: For each problem, you are expected to do 5 things:

1. Provide the correct answer to the problem.
2. Provide the solution that GPT-3.5 gives.
3. Provide the solution that GPT-4 gives.
4. If there are errors in the responses from either GPT-3.5 or GPT-4, explain where they occur.
5. For each problem, include links to your conversations with GPT-3.5 and GPT-4. See here for an example: `https://chat.openai.com/share/90efb80c-705e-4ed1-9e1e-68f13f4306cb`

Please submit this work in two files. One file should be named **ps1.py**. This should contain the correct answer to the problem. The second file should be named **gpt.py**. This should contain the code generated by GPT-3.5 and GPT-4, as well as your explanations of the mistakes that occurred.

In class we talked about *least squares regression.* In least-squares regression, we have a dataset that consists of two variables, $X = (x_1, ..., x_n)$ and $Y = (y_1, ..., y_n)$. We are trying to predict the values in Y from the values in X using the linear equation $y = a \cdot x + b$. More precisely, for each x_i, we use this equation to compute a predicted value:

$$\hat{y_i} = a \cdot x_i + b \tag{1}$$

Our goal is to minimize the total error of our predictions on the dataset. We want to find the values of a and b that minimize the quantity:

$$L = \sum_{i=1}^{n} (y_i - \hat{y_i})^2 \tag{2}$$
$$= \sum_{i=1}^{n} (y_i - a \cdot x_i - b)^2 \tag{3}$$

In class, we found equations for the optimal values of the slope a and intercept b. These equations define *estimators* of the slope and intercept. The equations used the definition of the mean of a variable: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. The equations for the estimators of a and b that we found were:

$$a(x, y) = \frac{\sum_i x_i \cdot y_i - n\bar{x}\bar{y}}{\sum_i x_i^2 - n\bar{x}^2} \tag{4}$$
$$b(x, y) = \bar{y} - a(x, y) \cdot \bar{x} \tag{5}$$

Note that we are treating these estimators as functions: $a(x, y)$ takes in the values of x and y, and returns an estimated value of the slope, and similarly for $b(x, y)$.

Problem 1. Write a Python function *compute_slope_estimator*, which takes in two input variables, x and y. The variables x and y should be 1-dimensional NumPy arrays that have the same length n. The function should return the optimal value of the slope from Equation 4.

(If you are new to Python or Numpy, see this great tutorial: `http://cs231n.github.io/python-numpy-tutorial`. Also, please don't hesitate to come to office hours.)

Problem 2. Write a Python function *compute_intercept_estimator*, which takes in two input variables, x and y. The variables x and y should be 1-dimensional NumPy arrays that have the same length n. The function should return the optimal value of the intercept from Equation 5.

Problem 3. Write a function *train_model*, which takes in two 1-dimensional NumPy arrays of the same length, x and y. It should use *compute_slope_estimator* and *compute_intercept_estimator*, and return a tuple of values: the optimal value of the slope and the optimal value of the intercept.

The elements in the array y can be considered the labels in our training set: we use them to estimate the optimal values of the slope and intercept.

Problem 4. Write a function *dL_da* which takes four arguments: *x_vals*, *y_vals*, a, and b. The variables *x_vals* and *y_vals* are 1-dimensional NumPy arrays of length n. Your function should return the partial derivative $\frac{1}{n} \cdot \frac{\partial L}{\partial a}$:

$$\frac{1}{n} \cdot \frac{\partial L}{\partial a} = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{\partial f(x_i, y_i, a, b)}{\partial a} \tag{6}$$

In this equation, the function $f(x_i, y_i, a, b)$ is defined as:

$$f(x_i, y_i, a, b) = (y_i - a \cdot x_i - b)^2 \tag{7}$$

Problem 5. Write a function dL_db which takes four arguments: x_vals, y_vals, a, and b. The variables x_vals and y_vals are 1-dimensional NumPy arrays of length n. Your function should return the partial derivative $\frac{1}{n} \cdot \frac{\partial L}{\partial b}$:

$$\frac{1}{n} \cdot \frac{\partial L}{\partial b} = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{\partial f(x_i, y_i, a, b)}{\partial b} \tag{8}$$

The function f is defined in the previous problem.

Problem 6. Write a function $gradient\_descent\_step$ which takes five arguments: x_vals, y_vals, a, b, and k. The variable k is a positive real number. This function will perform a single step of *gradient descent*. Given the input values of a and b, the function will compute updated values $a_{updated}$ and $b_{updated}$:

$$a_{updated} = a - \frac{k}{n} \cdot \frac{\partial L}{\partial a} \tag{9}$$

$$b_{updated} = b - \frac{k}{n} \cdot \frac{\partial L}{\partial b} \tag{10}$$

In order to calculate the partial derivatives $\frac{\partial L}{\partial a}$ and $\frac{\partial L}{\partial b}$, you should call the functions that you wrote in Problems 4 and 5.

The function should return a tuple of the values $a_{updated}$ and $b_{updated}$.

(Note that in the function signature provided for you, the variable k is set to a default value. Please leave this value unchanged.)

Problem 7. In the previous problem, you wrote a function which performs a single step of gradient descent. Each time a gradient descent step is performed, the parameter values a and b will improve by a small amount. As we will soon discuss in class, the loss function generally decreases after a gradient descent step, meaning that the model provides a better fit to the data.

The full gradient descent algorithm is simple: iteratively perform many gradient descent steps, getting improved parameter values with each step. More precisely, we start with initial parameter values a_0 and b_0. At step t, we compute the updated parameters a_t and b_t by performing a gradient descent step on a_{t-1} and b_{t-1}, the parameters from step $t-1$.

Write a function $gradient\_descent$, which takes five arguments: x_vals, y_vals, a_0, b_0, and k. a_0 and b_0 are initial parameter values for the algorithm. k is a positive integer, which is the number of gradient descent steps that should be performed. The function should implement the gradient descent algorithm, performing k gradient descent steps. It should return the final parameter values a_k and b_k as a tuple.

(Note that in the function signature provided for you, the variables a_1, b_1, and k are set to default values which you should leave unchanged.)

Problem 8. In the following problems, we will be looking at Einstein Summation (np.einsum). As we will discuss in class, Einstein Summation will allow us to greatly simplify the coding that we do later in the course.

Write a function that takes two arguments, A and B. Both A and B are 2-dimensional NumPy arrays of the same size. (A 2-dimensional array is a matrix. Each array has n rows and m columns.) The function should return a 2-D NumPy array C of the same size, defined as follows:

$$C_{i,j} = A_{i,j} \cdot B_{i,j} \tag{11}$$

$C_{i,j}$ is the element of C in the i'th row and j'th column.

For example, suppose that the two input arrays are:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{12}$$

Then the output array will be:

$$C = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 5 & 0 \end{bmatrix} \tag{13}$$

Your function must use np.einsum to perform this operation. See documentation here: `https://numpy.org/doc/stable/reference/generated/numpy.einsum.html`

Problem 9. Write a function that takes two arguments, A and B. A is a 2-D NumPy array, and B is a 1-D array. The size of A is (n,m), while the size of B is (m). (This means that matrix A has n rows and m columns, and vector B has m rows.) The function should return a 2-D array C of size (n,m), defined as follows:

$$C_{i,j} = A_{i,j} \cdot B_j \tag{14}$$

For example, suppose that the two input arrays are:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \tag{15}$$

Then the output array will be:

$$C = \begin{bmatrix} 0 & 2 & 6 \\ 0 & 5 & 12 \end{bmatrix} \tag{16}$$

Your function must use np.einsum to perform this operation.

Problem 10. This example will involve 3-D NumPy arrays. The 3-D arrays we will be considering have size (b, n, m). You can think of this as a collection of b matrices which each have size (n, m). For example, here is a 3-D array with size (2,3,4):

$$A = \begin{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \\ \\ \begin{bmatrix} 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 \end{bmatrix} \end{bmatrix}$$

As you can see, this 3-D array is made up of 2 matrices, each of size (3,4).

Since there are 3 dimensions in the array, we need 3 indices in order to pick out a specific number. For example, $A_{0,2,3} = 12$, and $A_{1,0,2} = 15$. Make sure that you understand how this indexing works before moving on.

Write a function that takes two arguments, A and B. A is a 3-D NumPy array, and B is a 2-D array. The size of A is (b,n,m), and the size of B is (b,m). The function should return a 2-D array C of size (b,n), defined as follows:

$$C_{i,j} = \sum_k A_{i,j,k} \cdot B_{i,k} \tag{17}$$

This is batch matrix-vector multiplication, something that is used very often in deep learning.

Problem 11. Write a function that takes two arguments, A and B. A and B are both 3-D NumPy arrays. The size of A is (b,n,m), and the size of B is (b,m,p). The function should return a 3-D array C of size (b,n,p), defined as follows:

$$C_{i,j,q} = \sum_k A_{i,j,k} \cdot B_{i,k,q} \tag{18}$$

This is batch matrix-matrix multiplication, another common operation.