Homework 1 [**70 pts**]

Due Wednesday, September 18 at 5pm

If you have not already done so, please read the syllabus, which is available on Blackboard. A few comments:

- Please don't cheat! Don't use LLMs to try to do your homework!
- You may discuss the problems at a general level with your classmates, but your solutions must be your own. If you discuss the assignment with anybody, please mention their name in your submission.
- Your solution (written + Python code) must be submitted electronically through Blackboard. I highly recommend using Latex for the written portion.
- Your submission should include: (1) the written solution to Problem 1, (2) your code for problems 2A and 2B, and (3) a pdf with the plots and explanations asked for in problems 2A and 2B.

Background

In class, we talked about least squares regression where our data consists of pairs (x, y) with $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$; our model has the form

$$h_{w,w_0}(x) = x^\top w + w_0$$

where $w \in \mathbb{R}^d$ and $w_0 \in \mathbb{R}$ are our parameters; and our goal is to minimize the expected squared loss of our model's predictions on a new sample from the data distribution $\mathcal{D}$ (aka the test loss):

$$L(w, w_0) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} (x^\top w + w_0 - y)^2$$

Recall that by adding one additional feature to x that is always equal to one and stacking w and w_0 into one vector:

$$\tilde{x} = \begin{bmatrix} x \\ 1 \end{bmatrix} \in \mathbb{R}^{d+1} \qquad \tilde{w} = \begin{bmatrix} w \\ w_0 \end{bmatrix} \in \mathbb{R}^{d+1}$$

we can write our model as $\tilde{x}^\top \tilde{w}$ (i.e. without the pesky w_0 term).

Given training data

$$S = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

in class, we wrote the training loss in two equivalent ways:

$$L_S(\tilde{w}) = \frac{1}{n}\sum_{i=1}^{n}(\tilde{x}_i^\top \tilde{w} - y_i)^2 = \frac{1}{n}(\tilde{X}\tilde{w} - Y)^\top(\tilde{X}\tilde{w} - Y)$$

where $\tilde{X} \in \mathbb{R}^{n\times d+1}$ is the "design matrix" whose ith row is equal to $\tilde{x}_i$ and $Y \in \mathbb{R}^n$ is the vector whose ith entry is y_i.

1 The gradient of the training loss [10 pts]

In class, I claimed that the gradient of the training loss is

$$\nabla L_S(\tilde{w}) = \frac{2}{n}(\tilde{X}^\top\tilde{X}\tilde{w} - \tilde{X}^\top Y)$$

[**10 pts**] Show that this is the case by calculating the partial derivative

$$\frac{\partial}{\partial \tilde{w}[j]}\left(\frac{1}{n}\sum_{i=1}^{n}(\tilde{x}_i^\top \tilde{w} - y_i)^2\right)$$

(where $\tilde{w}[j]$ is the jth entry of $\tilde{w}$) and showing that it is equal to the jth entry of the vector

$$\frac{2}{n}(\tilde{X}^\top\tilde{X}\tilde{w} - \tilde{X}^\top Y).$$

2 Implementing least squares regression [60 pts]

I have provided you with $N = 50$ samples

$$(x_1, y_1), \ldots, (x_N, y_N) \overset{iid}{\sim} \mathcal{D}$$

from an unknown data distribution $\mathcal{D}$ (well, it's unknown to you ;p) in the files `X.npy` and `Y.npy`. There is only one feature for each example, so $x_i \in \mathbb{R}$ and $y_i \in \mathbb{R}$. Starter code is also provided in `hw1.py`.

2.1 Part A [20 pts]

Do the following:

1. [**2 pts**] Split the provided data into training (80% of the samples) and testing data (the remaining 20% of the samples).
2. [**3 pts**] Create the design matrix (don't forget the constant 1 feature!), this matrix should have dimensions 40×2.

3. [**5 pts**] Compute the parameters $\tilde{w} \in \mathbb{R}^2$ that minimizes the training loss (recall from class that the minimizer of the training loss solves $\nabla L_S(\tilde{w}^\star) = 0$ i.e. $\tilde{X}^\top \tilde{X} \tilde{w}^\star = \tilde{X}^\top Y$). *Hint:* You might be interested in `np.linalg.solve`.

4. [**5 pts**] Report the training and test loss for the parameters you learned.

5. [**5 pts**] Use the provided `plot_data_and_model` function to plot (a) the training data with your model's predictions overlaid (b) the test data with your model's predictions overlaid. Describe what you see and explain why this happened.

2.2 Part B [40 pts]

The linear model from part A performed poorly on the test data, so we are going to go back and try again. To hopefully improve performance, we will add new polynomial features so that our model can represent more a more complex relationship between x and y. Specifically, for a specified value of k we will use new features

$$x \mapsto \tilde{x} = \begin{bmatrix} x^k \\ x^{k-1} \\ \vdots \\ x \\ 1 \end{bmatrix} \in \mathbb{R}^{k+1}$$

Do the following:

1. [**5 pts**] If we learn a linear model based on the polynomial features with $k = 3$, describe what functions the linear model is capable of representing.

2. [**5 pts**] Implement the function `polynomial_features` in `hw1.py`.

3. [**10 pts**] Create the design matrix for degree-20 polynomial features; compute the parameters that minimize the training loss with these features; report the training and test loss; and use `plot_data_and_model` to visualize the model's predictions on the train and test data.

4. [**5 pts**] Describe what you see and explain why this happened.

5. [**5 pts**] Repeat part 3 for each $k \in \{1, 2, \ldots, 15\}$ in order to create two length-15 lists `train_losses` and `train_losses` with `train_losses[k-1]` equal to the train loss for the model that used degree-k polynomial features (and similarly for `test_losses`). Plot the train and test losses against k using the code provided at the bottom of `hw1.py` (currently commented out).

6. [**5 pts**] Describe what you see. Explain why increasing k has the effect that it does.

7. [**5 pts**] Having seen what you've seen, if you wanted one final linear model based on polynomial features for this regression problem, which k would you use? Explain.