Homework 3:
Linear regression, optimization, and regularization

Problem 1: Linear regression three ways [32 points]

Consider the following problem of fitting a linear model. Given n data points $x_1, \ldots, x_n \in \mathbb{R}^d$ with labels $y_1, \ldots, y_n \in \mathbb{R}$, our goal is to find a coefficient vector $\theta \in \mathbb{R}^d$ that minimizes the squared error loss function on the training set:

$$L_{\text{train}}(\theta) = \sum_{i=1}^{n} \left(\langle \theta, x_i \rangle - y_i\right)^2 \ , \tag{1}$$

where $\langle u, v \rangle = \sum_{i=1}^{d} u_i v_i$ is the standard inner product for vectors.

We will draw the data for this problem from the following *synthetic* model. For the training data, we will select $x_1, \ldots, x_n$ drawn independently from a d-dimensional standard Gaussian distribution, **except we divide the first coordinate of each vector x_i by** 20. Note that a standard d-dimensional Gaussian is a random vector whose coordinates are independent $\mathcal{N}(0, 1)$ random variables.

Then, we choose a true coefficient vector $\theta^* \in \mathbb{R}^d$ from a d-dimensional standard Gaussian distribution, and for $i = 1, \ldots, n$, the corresponding label y_i is given by

$$y_i = \langle \theta^*, x_i \rangle + \eta_i \ ,$$

where η_i is random noise drawn from $\mathcal{N}(0, (0.3)^2)$.

We will take $d = 20$, and $n = 2000$.

(a) [3 points] It's a classical fact that least-squares regression (i.e., find θ minimizing (1) on the training set) has a simple closed-form formula. Let $X \in \mathbb{R}^{n \times d}$ be a matrix whose rows are x_i, i.e.

$$X = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix} ,$$

and let $y \in \mathbb{R}^n$ denote the (column) vector of the labels, i.e. $y = (y_1, \ldots, y_n)^T$. Then, the minimizer of (1) is given by $\widehat{\theta} = (X^T X)^{-1} X^T y$. Solve for $\widehat{\theta}$ and report the value of the objective. For comparison, what is the total squared error for θ being the vector of all zeros?

The main downside of this approach is that it's quite slow, since it requires a matrix-matrix multiplication and inversion, so in practice, people often use faster methods that give approximations, which is what we'll explore next. However, since our datasets are pretty small, we can still run this method, and it will serve as a good sanity check for the rest of the problem.

(b) [4 points] Now, draw $m = 2000$ new samples $\widehat{x}_1, \ldots, \widehat{x}_n \in \mathbb{R}^d$ from the same distribution, and on this new set of points, which we will treat as a test set, report the average loss evaluated at $\widehat{\theta}$:

$$L_{\text{test}}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\langle \theta^*, \widehat{x}_i \rangle - \langle \theta, \widehat{x}_i \rangle\right)^2 \ .$$

Is this value small? How does this behavior change as we vary m? Can you explain why this happens?

(c) [4 points] We can also ask how well our solution $\widehat{\theta}$ approximates θ^*, something known as *parameter recovery.* Try inspecting the quality of your solutions, for multiple trials involving different training sets. Do you see any recurring patterns in how well your $\widehat{\theta}$ approximates θ^*? Can you offer any intuitive explanation for why this might be happening?

(EC) [4 points] Write the expected test loss as a deterministic expression involving the parameters θ and θ^*, and give a mathematical justification why (Hint: recall some of the properties of Gaussians we discussed in Lecture 4).

(d) [6 points] First, review the primer on gradient descent linked on the course webpage.

You will solve the same setup as in part (a) using gradient descent on the training loss function L_{train}. Recall that the gradient is a linear operator, so

$$\nabla L_{\text{train}}(\theta) = \sum_{i=1}^{n} \nabla L_i(\theta) \ , \tag{2}$$

where $L_i(\theta) = (\langle \theta, x_i \rangle - y_i)^2$. What is the expression for the gradient of ∇L_{train}?

Using this expression, run gradient descent to find a coefficient vector θ that approximately minimizes L_{train}. Run gradient descent three times, with step sizes $\alpha \in \{0.00007, 0.00035, 0.0007\}$, each time starting at initial value θ_0 being the all zeros vector. Plot the objective function value for 20 iterations of gradient descent for all three choices of step size on the same graph.

Which step size has the best final objective value? Discuss how the choice of step size seems to affect the behavior of gradient descent. Feel free to experiment with other choices of step size to support your argument.

(e) [3 points] When the objective function is linear (as is it for us, recall (2)), it's often the case (as it is for us) that each individual gradient ∇L_i can be evaluated much more efficiently than the whole gradient. In such settings, it's often better to use *stochastic* gradient descent (SGD for short).

In SGD, at iteration t, at iterate θ_t, instead of computing $\nabla L_{\text{train}}(\theta_t)$, we will compute a *random* value Z_t that will give the gradient in expectation, i.e.:

$$\mathbb{E}[Z_t] = \nabla L_{\text{train}}(\theta_t) \ .$$

When the objective function is linear, a straightforward way of doing this is to choose an index $i \in \{1, \ldots, n\}$ uniformly at random, and letting $Z_t = n \cdot \nabla L_i(\theta_t)$. In practice, we often omit the scaling with n, so in our experiments, we'll take $Z_t = \nabla L_i(\theta_t)$.

With these definitions, write down pseudocode for SGD with step size α. Your algorithm should crucially only compute the gradient of a single data point every iteration.

(f) [3 points] Now, run SGD with step size $\alpha \in \{0.0005, 0.005, 0.01\}$, each for $T = 20,000$ iterations. Plot the objective function value vs. the iteration number for all 3 step sizes on the same graph. Discuss how the step size appears to affect the convergence of SGD, and compare this to what you observed for standard gradient descent. Also report the step size that had the best final objective function value and the corresponding objective function value.

(g) [3 points] Compare the performance of the two methods. How do the best final objective function values compare? For a fixed amount of wall-clock runtime, which method seems to do better?

(h) [6 points: 2 for performance and 4 for discussion] You've seen the effect of step size on both methods, and you should see pros and cons for using larger and smaller step sizes. In practice, folks often use a variable step size as an attempt to get the "best of both worlds". That is, instead of fixing a step size for α throughout the entire algorithm, they choose different α_t for every iteration. Can you try to design a better step size schedule (other than a constant one) that is able to get this best of both worlds performance? Report the best performance you're able to achieve, the corresponding schedule, and explain why you chose this step size schedule.

Don't worry about getting the best possible performance! The goal of this problem is just so that you can get a sense of the general principles at play here. In practice, designing a good step size schedule for your specific task is often both very important and also very much black magic.

Problem 2: Regularization [24 points]

So far, we've been in a setting where we have way more data points than dimensions (i.e. $n \gg d$), and hence, you should've seen that the generalization error was pretty good (i.e. your test loss in (b) should've been pretty small....spoilers).

We'll now consider the setting where $d = 200$, the training set has size $n = 200$, and the test set has size 2000, and examine what changes regarding training and test loss. We can define the data matrices X_{train} and X_{test} analogously to how we defined X previously, and it will be convenient for us to work with the *normalized losses* here:

$$\widehat{L}_{\text{train}}(\theta) = \frac{\|X_{\text{train}}\theta - y\|_2}{\|y\|_2} \qquad \text{and} \qquad \widehat{L}_{\text{test}}(\theta) = \frac{\|X_{\text{test}}\theta - y\|_2}{\|y\|_2} . \tag{3}$$

(a) [2 points] As a baseline for comparison, compute the linear regression solution from Problem 1 without any regularization. Report the normalized training and test error averaged over 10 trials.

(b) [5 points] Now, consider what happens when you try an ℓ_2 regularization (known in the literature as *ridge regression*). Now, the training loss function is given by

$$L_{\text{ridge}}(\theta) = \sum_{i=1}^{n} \left(\langle \theta, x_i \rangle - y_i\right)^2 + \lambda \|\theta\|_2^2 ,$$

where λ is a regularization parameter. It turns out that this still has a closed form solution, given by

$$\theta_{\text{ridge}} = \left(X_{\text{train}}^T X_{\text{train}} + \lambda I\right)^{-1} X_{\text{train}} y .$$

Present a plot of the normalized training and test error for $\lambda \in \{0.0005, 0.005, 0.05, 0.5, 5, 50, 500\}$. As before, you should average over 10 trials. Explain the effect you observe empirically when you vary λ, and briefly justify why this should be the case.

(c) [4 points] Let's revisit the parameter recovery question from Problem 1(c). How does varying λ affect the quality of the error of the approximation between your solution and θ^*, on a coordinate-by-coordinate basis? Can you offer an intuitive explanation for why this might be the case?

(d) [5 points] Now, try running SGD on the *original* (not regularized!) objective function L_{train}, with initial guess being the all zeros vector. Run SGD for $1,000,000$ iterations for step size choices $\alpha \in \{0.00005, 0.0005, 0.005\}$. Report the normalized training error and test error for each of these settings, averaged over 10 trials.

How does the SGD solution compare with the solutions achieved via regularization and the unregularized solution you got for part (a)? Note that SGD is not doing any explicit regularization either!

(e) [8 points] For all the choices of α in the previous setting that didn't diverge, let's examine the behavior of SGD more closely. For each such α:

- Plot the normalized training error vs. iteration number.
- Plot the normalized test error vs. iteration number.
- Plot the ℓ_2 norm of the SGD solution vs iteration number.

Discuss the plots. What can you say about the generalization ability of SGD with different step sizes? How does this relate to the ℓ_2 norm of the solution?

Problem 3: Try anything [11 points]

Consider now the under-determined setting where $d > n$ and the traditional theory of generalization says that it should be much more difficult. Take the same setup as before, but now with $d = 300$, a training set of size $n = 200$, and a test set of size 2000.

[11 points: 4 for performance and 7 for analysis / discussion] Try to achieve the best test error you can, using whatever method you can think of...although we suggest some combination of the things discussed above. The only constraint is that the learning algorithm is only allowed to touch the training set, and cannot refer to θ^*. Briefly discuss the approach you used, the thought process that informed your decisions, and the extent to which you believe a better test error is achievable.

Your score will be based on a combination of the short discussion and the average test error you obtain. A paragraph of analysis is enough to earn full credit—you don't need to go too crazy.