CORNELL TECH | HOME OF THE JACOBS TECHNION-CORNELL INSTITUTE

[Fall 2025] ECE 5290/7290 and ORIE 5290
Distributed Optimization for Machine Learning and AI

Homework 2

Gradescope Due: September 28th at 5PM

Objective of This Assignment

The objective of this assignment is to deepen your understanding of fundamental gradient-based optimization algorithms and their convergence properties. You will analyze the behavior of these methods on a variety of problem classes, including ill-conditioned quadratic, constrained, and nonconvex functions. The problems will guide you through advanced, practical topics such as the acceleration provided by momentum (the Heavy-ball method), and the variance and trade-offs of Stochastic Gradient Descent (SGD).

Instruction of Homework Submission

This assignment includes both an analytical part and a coding part (Problem 5b). We will use Gradescope to check the correctness of your code. Therefore, you will see two separate assignments on Gradescope. For the analytical part, please follow the same instructions as Homework 1 and upload your work to **Homework 2**. For the coding part, please read the instructions below.

(a) A starter .py file has been provided for Problem 5b. Do not change the function names, the parameters of these functions, or the filename.

(b) When you are done, upload your completed .py file to **Homework 2 Coding** on Gradescope.

Question 1: The Impact of Condition Number on GD (20 points)

This problem explores the practical consequences of the condition number on the convergence of Gradient Descent (GD). Consider the quadratic objective $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x}$.

(a) **(5 points)** Consider two matrices:

$$\mathbf{Q}_1 = \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{Q}_2 = \begin{pmatrix} 5.5 & 4.5 \\ 4.5 & 5.5 \end{pmatrix}$$

Calculate the condition number $\kappa = \lambda_1/\lambda_n$ for both $\mathbf{Q}_1$ and $\mathbf{Q}_2$. Which one is better-conditioned?

(2 Points) For $Q_1 = \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix}$ the eigenvalues are $\{10, 1\}$, so

$$\kappa(Q_1) = \frac{\lambda_{\max}}{\lambda_{\min}} = \frac{10}{1} = 10.$$

(3 Points) For $Q_2 = \begin{pmatrix} 5.5 & 4.5 \\ 4.5 & 5.5 \end{pmatrix}$ The eigenvalues are also $\{10, 1\}$ so $\kappa(Q_2) = 10$

(b) **(5 points)** For both quadratic objectives defined by $\mathbf{Q}_1$ and $\mathbf{Q}_2$, write down the linear convergence rate for GD using the optimal constant stepsize $\eta = \frac{2}{\lambda_1+\lambda_n}$, in terms of the contraction factor $\rho = \frac{\kappa-1}{\kappa+1}$.

The step size is $\frac{2}{\lambda_1+\lambda_n} = \frac{2}{11}$

(3 Points) The contraction factor $\rho = \frac{9}{11}$.

(2 Point) Gradient Descent then enjoys the linear rate (for either Q_1 or Q_2)

$$\|x_k - x^*\|_2 \leq \rho^k \|x_0 - x^*\|_2.$$

(c) **(10 points)** Sketch the contour plots for both functions. On each plot, sketch the expected path of Gradient Descent starting from a point like $\mathbf{x}^0 = [1, 1]^\top$. Briefly explain how the different condition numbers lead to different convergence behaviors.

Answer. For Q_1, the contours of $f(x) = \frac{1}{2}x^\top Q_1 x$ are axis-aligned ellipses along the x_1-direction. Starting from $x^0 = [1, 1]^\top$, gradient descent shrinks the x_2-coordinate smoothly toward zero, while the x_1-coordinate alternates in sign each step. The trajectory therefore zig-zags across the vertical axis while contracting toward the origin.

For Q_2, the contours are ellipses rotated by $45°$, with principal axes along $[1, 1]^\top$ and $[1, -1]^\top$. Since the starting point $x^0 = [1, 1]^\top$ lies exactly along the eigenvector corresponding to the larger eigenvalue, all iterates remain on this line. Gradient descent therefore moves in a straight line along the $45°$ diagonal, alternating in sign but shrinking in magnitude each step. Thus, while both cases share the same linear rate $\rho = \frac{9}{11}$, the geometry of the contours leads to different trajectories.

Question 2: Stochastic Gradient Descent (25 points)

Consider the problem for linear regression: $F(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^N f_i(\boldsymbol{\theta})$, where $f_i(\boldsymbol{\theta}) = \frac{1}{2}(\mathbf{a}_i^\top \boldsymbol{\theta} - y_i)^2$. The stochastic gradient for a single sample i_t (chosen uniformly at random) is $g_{i_t}(\boldsymbol{\theta}) = \nabla f_{i_t}(\boldsymbol{\theta})$.

(a) **(5 points) Unbiased Estimator:** Show that the stochastic gradient is an unbiased estimator of the true gradient, i.e., $\mathbb{E}_{i_t}[g_{i_t}(\boldsymbol{\theta})] = \nabla F(\boldsymbol{\theta})$.

$$\mathbb{E}_i\big[g_i(\theta)\big] = \frac{1}{N}\sum_{i=1}^{N}\nabla f_i(\theta) = \nabla F(\theta).$$

Hence $g_i(\theta)$ is an unbiased estimator of the true gradient.

(b) **(10 points) Variance of the Stochastic Gradient:** Derive an expression for the variance of the stochastic gradient estimator, defined as $\sigma^2_{\boldsymbol{\theta}} = \mathbb{E}_{i_t}[||g_{i_t}(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta})||_2^2]$.

(5 Points) Using $\mathbb{E}\|X - \mathbb{E}X\|^2 = \mathbb{E}\|X\|^2 - \|\mathbb{E}X\|^2$ with $X = g_i(\theta)$,

$$\sigma^2_\theta \overset{\text{def}}{=} \mathbb{E}_i\big\|g_i(\theta) - \nabla F(\theta)\big\|^2 = \underbrace{\frac{1}{N}\sum_{i=1}^{N}\|g_i(\theta)\|^2}_{\text{first moment}} - \underbrace{\Big\|\frac{1}{N}\sum_{i=1}^{N} g_i(\theta)\Big\|^2}_{\|\nabla F(\theta)\|^2}.$$

Plugging $g_i(\theta) = (a_i^\top\theta - y_i)a_i$ gives the explicit form

$$\sigma^2_\theta = \frac{1}{N}\sum_{i=1}^{N}(a_i^\top\theta - y_i)^2\,\|a_i\|^2 - \left\|\frac{1}{N}\sum_{i=1}^{N}(a_i^\top\theta - y_i)\,a_i\right\|^2.$$

(Hint: You can use the identity $\mathbb{E}[||\mathbf{X} - \mathbb{E}[\mathbf{X}]||^2] = \mathbb{E}[||\mathbf{X}||^2] - ||\mathbb{E}[\mathbf{X}]||^2$. Your final expression will be in terms of the gradients of the individual component functions, $\nabla f_i(\boldsymbol{\theta})$).

(c) **(10 points) The Power of Mini-batching - For ECE/ORIE 5290 students:** Consider a mini-batch stochastic gradient as

$$g_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{B}\sum_{j\in\mathcal{B}} g_j(\boldsymbol{\theta}),$$

where $\mathcal{B}$ is a mini-batch of size B sampled uniformly *with replacement* from $\{1, ..., N\}$. [1]

i) Show that this mini-batch estimator is also unbiased.

ii) Show that the variance of the mini-batch estimator is reduced by a factor of B:

$$\mathrm{Var}(g_{\mathcal{B}}(\boldsymbol{\theta})) = \mathbb{E}_{\mathcal{B}}[||g_{\mathcal{B}}(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta})||_2^2] = \frac{1}{B}\sigma^2_{\boldsymbol{\theta}}$$

Let a mini-batch B of size $|B| = B$ be drawn i.i.d. with replacement and

$$g_B(\theta) = \frac{1}{B}\sum_{j\in B} g_j(\theta).$$

(5 Points) *Unbiasedness:*

$$\mathbb{E}\big[g_B(\theta)\big] = \frac{1}{B}\sum_{j\in B}\mathbb{E}[g_j(\theta)] = \frac{1}{B}\cdot B\,\nabla F(\theta) = \nabla F(\theta).$$

[1] Sampling *with replacement* means that after a data point is selected for the mini-batch, it is "put back" into the dataset and is eligible to be selected again for the same mini-batch. This is a common assumption that simplifies the mathematical analysis of variance, as it ensures every selection is an independent event.

(5 Points) *Variance reduction:* independence (from sampling with replacement) yields

$$\mathrm{Var}\big(g_B(\theta)\big) = \mathbb{E}\big\|g_B(\theta) - \nabla F(\theta)\big\|^2 = \frac{1}{B^2}\sum_{j\in B}\mathbb{E}\big\|g_j(\theta) - \nabla F(\theta)\big\|^2 = \frac{1}{B}\,\sigma_\theta^2.$$

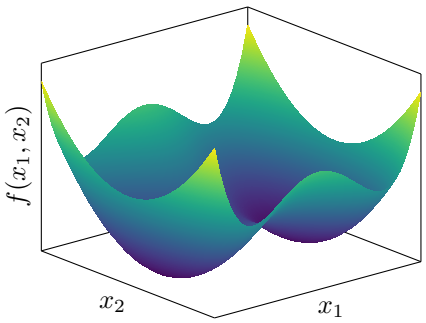
(d) **(10 points) Connection to convergence - For ECE 7290 students:** Look at the long-term error term in the SGD convergence from the lecture: $\frac{\eta L \sigma_g^2}{2\mu}$. Based on your result from part (c), if we use a mini-batch of size B, how does this error term change? What does this tell you about the trade-off between computational cost per iteration and convergence behavior when choosing a batch size?

The error term decreases by a factor of B. But the per-iteration computational cost increases by a factor of B.

Question 3: GD on Nonconvex Functions (25 points)

Consider the 2D nonconvex function below, which is a classic example of a surface with multiple minima and a saddle point. A plot of the function is provided for visualization.

$$f(x_1, x_2) = x_1^4 - 2x_1^2 + x_2^2$$



(a) **(5 points)** Find all stationary points of f by setting its gradient $\nabla f(x_1, x_2)$ to zero.

(2 Points)

$$\nabla f(x_1, x_2) = \begin{bmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \end{bmatrix} = \begin{bmatrix} 4x_1^3 - 4x_1 \\ 2x_2 \end{bmatrix} = \begin{bmatrix} 4x_1(x_1^2 - 1) \\ 2x_2 \end{bmatrix}.$$

(2 Points) Setting $\nabla f = 0$ gives $x_2 = 0$ and $x_1 \in \{-1, 0, 1\}$.

(1 Points)

Stationary points: $(-1, 0),\ (0, 0),\ (1, 0)$.

(b) **(5 points)** Compute the Hessian matrix $\nabla^2 f(x_1, x_2)$.

$$\nabla^2 f(x_1, x_2) = \begin{bmatrix} \partial^2 f/\partial x_1^2 & \partial^2 f/\partial x_1 \partial x_2 \\ \partial^2 f/\partial x_2 \partial x_1 & \partial^2 f/\partial x_2^2 \end{bmatrix} = \begin{bmatrix} 12x_1^2 - 4 & 0 \\ 0 & 2 \end{bmatrix}.$$

(c) **(5 points)** For each stationary point you found, evaluate the Hessian and use its eigenvalues to classify the point as a local minimum, local maximum, or a saddle point.

Stationary Point	$\nabla^2 f(\cdot)$	Eigenvalues / Type
$(0, 0)$	$\begin{bmatrix} -4 & 0 \\ 0 & 2 \end{bmatrix}$	$\{-4, 2\} \Rightarrow$ saddle
$(\pm 1, 0)$	$\begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix}$	$\{8, 2\} \Rightarrow$ strict local minima

(d) **(10 points) For ECE/ORIE 5290 students:** One of the stationary points is a saddle point. If you were to run GD and initialize it *exactly* at this saddle point, what would happen? Now, what would happen if you initialized it *very close* to the saddle point (e.g., with a tiny random perturbation)? Briefly explain your reasoning.

Gradient Descent (GD) with step size $\eta > 0$ updates $x^{k+1} = x^k - \eta \nabla f(x^k)$.

- If initialized *exactly* at the saddle $(0, 0)$, then $\nabla f(0, 0) = 0$ and GD does not move: $x^{k+1} = x^k = (0, 0)$.
- If initialized *very close* to the saddle, e.g. $x^0 = (\varepsilon_1, \varepsilon_2)$ with tiny ε_i, then $\nabla f(x^0) \approx (-4\varepsilon_1, 2\varepsilon_2)$, so the GD step $-\nabla f(x^0) \approx (4\varepsilon_1, -2\varepsilon_2)$ pushes the iterate away from $x_1 = 0$ (toward $x_1 = \pm 1$ depending on the sign of ε_1) while contracting x_2 toward 0. Hence GD escapes the saddle and converges to one of the minima $(\pm 1, 0)$ for typical small perturbations.

(d) **(10 points) For ECE 7290 students:** The stationary point at $(0, 0)$ is a saddle point. Let's analyze how Gradient Descent escapes it.

(1) Consider an initial point very close to the saddle, $\mathbf{x}^0 = [\epsilon, \epsilon]^\top$, for a very small $\epsilon > 0$. Compute the gradient $\nabla f(\mathbf{x}^0)$. For a very small ϵ, you can ignore higher-order terms (like ϵ^3). What is the approximate direction of the first GD step, $-\nabla f(\mathbf{x}^0)$?

(2) The Hessian at the saddle point, $\nabla^2 f(0, 0)$, has one negative eigenvalue. The corresponding eigenvector points in the direction of negative curvature—the direction the algorithm will "roll off" the saddle. What is this eigenvector?

(3) Compare your answers from (1) and (2). What do you notice about the relationship between the GD step direction and the eigenvector corresponding to the negative eigenvalue? Briefly explain why this ensures GD can escape saddle points in practice.

Question 4: Coding - Implementing and Comparing Optimizers (30 points)

In this problem, you will implement and compare different first-order optimization algorithms on a logistic regression problem. The dataset is specifically designed to be **ill-conditioned** to highlight the performance differences between the algorithms.

(a) Data Generation and Visualization (5 points)

Use the provided Python code snippet to generate and visualize the 2D synthetic dataset. The features have been intentionally scaled to create an ill-conditioned problem (notice the different scales on the x and y axes). Explain why this feature scaling leads to an ill-conditioned Hessian for the logistic regression loss.

```
import numpy as np
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt

# Generate a synthetic dataset
X, y = make_classification(n_samples=200, n_features=2, n_redundant=0,
                           n_informative=2, random_state=1, n_clusters_per_class=1)

# Induce ill-conditioning by scaling one feature
X[:, 1] = X[:, 1] * 20

# Plotting the data
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2 (scaled)")
plt.title("Ill-Conditioned Synthetic Dataset")
plt.show()
```

In logistic regression, the Hessian has the form

$$H(\theta) \;=\; X^{\top} W(\theta)\, X,$$

where $W(\theta)$ is a positive diagonal matrix depending on the current class probabilities. If one feature x_j is scaled by a large factor s, the j-th column of X is multiplied by s. This scales the curvature along that coordinate by approximately s^2 in $X^{\top}WX$, yielding widely separated eigenvalues of H. Hence the condition number $\kappa(H) = \lambda_{\max}(H)/\lambda_{\min}(H)$ becomes large and the problem is *ill-conditioned*. Geometrically, the loss surface becomes very steep in the scaled direction and relatively flat in others, which slows and destabilizes first-order methods unless features are normalized.

(b) Implementation (10 points)

Implement the following three algorithms to minimize the Binary Cross-Entropy loss (introduced in the class) on this dataset. You will also need to implement the sigmoid function and the BCE loss and its gradient.

(a) Batch Gradient Descent (GD)

(b) Heavy-ball Method (GD with Momentum)

(c) Stochastic Gradient Descent (SGD) with a mini-batch size of your choice (e.g., $B = 8$).

(c) Hyperparameter Tuning (5 points)

For each of the three algorithms, experiment to find a "good" set of hyperparameters (learning rate η and, for momentum methods, the momentum parameter β). There is no single "correct" answer, but you should demonstrate that you've tried a few values to get reasonable performance. Report the final hyperparameters you chose for each algorithm.

(d) Comparison and Analysis (10 points)

Using your best hyperparameters from part (c), run all four optimizers from the same initial point $\boldsymbol{\theta}^0 = \mathbf{0}$ for a fixed number of epochs (e.g., 50 epochs).

(a) **Plot 1:** Generate a single plot showing the **Loss vs. Epochs** for all four methods. The y-axis should be on a logarithmic scale.

(b) **Plot 2:** Generate a single plot showing the 2D data points and the final **decision boundary** learned by each of the four algorithms.

(c) **Written Analysis:** In a short paragraph, answer the following:

- Why does the path of standard GD likely show slow, zig-zagging behavior on this dataset?
- How does the Heavy-ball method improve upon this? What is the key difference you observe in their loss curves?
- Describe the SGD loss curve. Why is it noisy, and what is its main advantage in the early epochs compared to the batch methods?

(c) Written Analysis.

(a) *Why GD zig–zags and is slow.* Standard GD follows the steepest descent direction, which causes oscillations when the loss surface is ill-conditioned. The gradient alternates across the steep direction and makes little progress along the flat one. A small step size is required for stability, leading to slow convergence.

(b) *How Heavy-ball improves upon GD.* Heavy-ball adds a momentum term $\theta_{k+1} = \theta_k - \eta\nabla F(\theta_k) + \beta(\theta_k - \theta_{k-1})$ that averages gradients over iterations. This damps oscillations in steep directions and accelerates motion along flat ones, producing a smoother and faster-decaying loss curve than GD.

(c) *Why the SGD loss is noisy and its advantage.* SGD uses stochastic gradient estimates with variance σ_g^2/B, causing the loss curve to fluctuate. Despite the noise, each update is cheap and frequent, enabling rapid early progress.