CS 170: Gradient Descent — Lecture Notes

Course: CS 170 — Efficient Algorithms and Intractable Problems, Spring 2026 **Instructors:** Lijie Chen, Umesh V. Vazirani **Lecture 11** (follows Backpropagation, precedes FFT)

Part 1: Why Gradient Descent?

In the previous lecture we saw how **backpropagation** computes the gradient $\nabla f(\mathbf{w})$ of a loss function with respect to all parameters in a single backward pass over a computational graph. But computing the gradient is only half the story — we still need to **use** it to find good parameters. This is the job of **gradient descent**.

1.1 The Optimization Problem

We want to solve:

$$\min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w})$$

where f is a differentiable loss function (e.g., mean squared error, cross-entropy) and $\mathbf{w}$ is a vector of parameters (weights and biases of a neural network, coefficients of a model, etc.).

1.2 What Does the Gradient Mean?

Before diving into the algorithm, let's build intuition for the gradient in multiple dimensions.

For a function $f(\mathbf{w})$ of d variables, the **partial derivative** $\frac{\partial f}{\partial w_i}$ tells us: if we perturb only the i-th coordinate by a tiny amount ε, the function value changes by approximately:

$$f(w_1, w_2, \ldots, w_i + \varepsilon, \ldots, w_d) \approx f(\mathbf{w}) + \varepsilon \frac{\partial f}{\partial w_i}(\mathbf{w})$$

Now suppose we perturb **all** coordinates simultaneously, each by a small amount ε_i. Writing $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_d)$, the overall change is (to first order) a sum of the individual effects:

$$f(\mathbf{w} + \boldsymbol{\varepsilon}) \approx f(\mathbf{w}) + \sum_{i=1}^{d} \varepsilon_i \frac{\partial f}{\partial w_i}(\mathbf{w}) = f(\mathbf{w}) + \nabla f(\mathbf{w})^\top \boldsymbol{\varepsilon}$$

The key point: as long as the perturbations are small, the change in f is **linear** in $\boldsymbol{\varepsilon}$. The gradient $\nabla f(\mathbf{w}) = \left(\frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_d}\right)$ packages all these partial derivatives into a single vector that completely characterizes the local first-order behavior of f.

1.3 The Gradient Descent Update Rule

The **gradient** $\nabla f(\mathbf{w})$ points in the direction of steepest **increase** of f. To decrease f, we step in the **opposite** direction:

$$\boxed{\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla f(\mathbf{w}_t)}$$

where $\eta > 0$ is the **learning rate** (step size) and $t = 0, 1, 2, \ldots$ counts the iterations.

1.4 Why the Gradient Direction?

Why does GD step along $-\nabla f(\mathbf{w})$ specifically? The idea is that we want to keep each step **short** — if we move too far, the linear approximation from Section 1.2 breaks down and we can no longer trust it to predict the change in f.

So suppose we fix the step length: we require $\|\boldsymbol{\varepsilon}\| = \delta$ for some small $\delta > 0$. Among all such steps, which one decreases f the most? The linear approximation says:

$$f(\mathbf{w} + \boldsymbol{\varepsilon}) \approx f(\mathbf{w}) + \nabla f(\mathbf{w})^\top \boldsymbol{\varepsilon}$$

We want to **minimize** $\nabla f(\mathbf{w})^\top \boldsymbol{\varepsilon}$ subject to $\|\boldsymbol{\varepsilon}\| = \delta$. By the Cauchy–Schwarz inequality:

$$\nabla f(\mathbf{w})^\top \boldsymbol{\varepsilon} \geq -\|\nabla f(\mathbf{w})\| \cdot \|\boldsymbol{\varepsilon}\| = -\delta \|\nabla f(\mathbf{w})\|$$

with equality when $\boldsymbol{\varepsilon} = -\delta \frac{\nabla f(\mathbf{w})}{\|\nabla f(\mathbf{w})\|}$, i.e., pointing in the **negative gradient** direction. So the gradient descent update $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f(\mathbf{w})$ moves in the direction that gives the greatest decrease in f per unit step, according to the linear approximation.

A natural question: How large should η be? If η is too small, we make very slow progress. If η is too large, we move so far that the linear approximation becomes inaccurate and we might actually *increase* the loss.

The intuition is: if f is relatively **smooth** — meaning the gradient does not change much as we move around — then the linear approximation stays accurate over larger steps, so η can be large. On the other hand, if f is **not smooth** — the gradient changes rapidly — then even a small step can land us somewhere the linear approximation badly misrepresents f, so η must be small. We will make this precise in Part 2 with a concrete example and in Part 3 with general theory.

Part 2: A Simple Example — $f(w) = 3w^2 + 1$

Let's now look at gradient descent in a very simple toy setting to build concrete intuition for how large η should be.

2.1 Setup

Consider the one-dimensional function:

$$f(w) = 3w^2 + 1$$

Its gradient is $f'(w) = 6w$, and the unique minimum is at $w^* = 0$ with $f(w^*) = 1$.

2.2 GD as a Geometric Sequence

The gradient descent update becomes:

$$w_{t+1} = w_t - \eta \cdot 6w_t = (1 - 6\eta)\, w_t$$

This is a **geometric sequence** with ratio $r = 1 - 6\eta$. Starting from w_0:

$$w_t = (1 - 6\eta)^t\, w_0$$

The behavior depends entirely on whether $|r| = |1 - 6\eta|$ is less than, equal to, or greater than 1.

2.3 Three Regimes

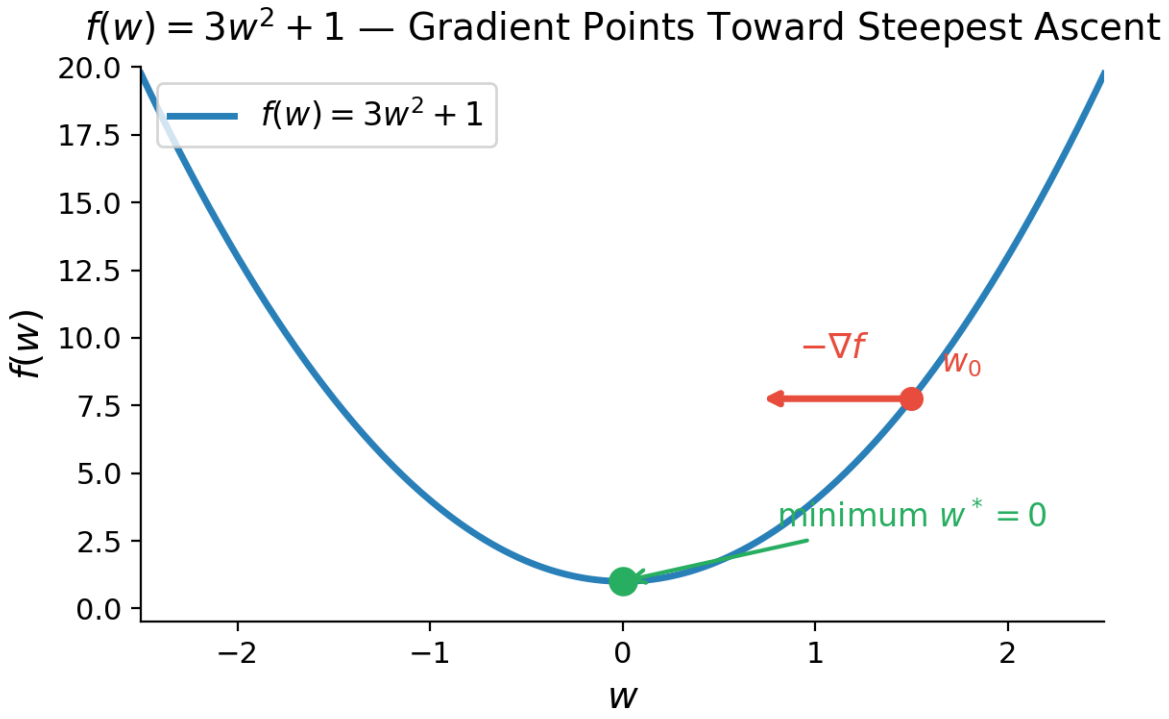

Figure 1: Quadratic function with gradient arrow

Regime	Condition on η	Ratio $r = 1 - 6\eta$	Behavior
Convergence	$0 < \eta < 1/3$	$\lvert r\rvert < 1$	$w_t \to 0$ exponentially
Monotone convergence	$0 < \eta < 1/6$	$0 < r < 1$	Approaches minimum from one side
Oscillatory convergence	$1/6 < \eta < 1/3$	$-1 < r < 0$	Bounces around minimum, still converges
Divergence	$\eta > 1/3$	$\lvert r\rvert > 1$	$\lvert w_t\rvert \to \infty$

Observation: The convergence threshold $\eta < 1/3$ depends on the constant 6 in $f'(w) = 6w$ — the steeper the function, the smaller the step size must be. This is not a coincidence: in Part 3 we will see that for general functions, the maximum safe step size is determined by how fast the gradient changes, and this threshold generalizes cleanly.

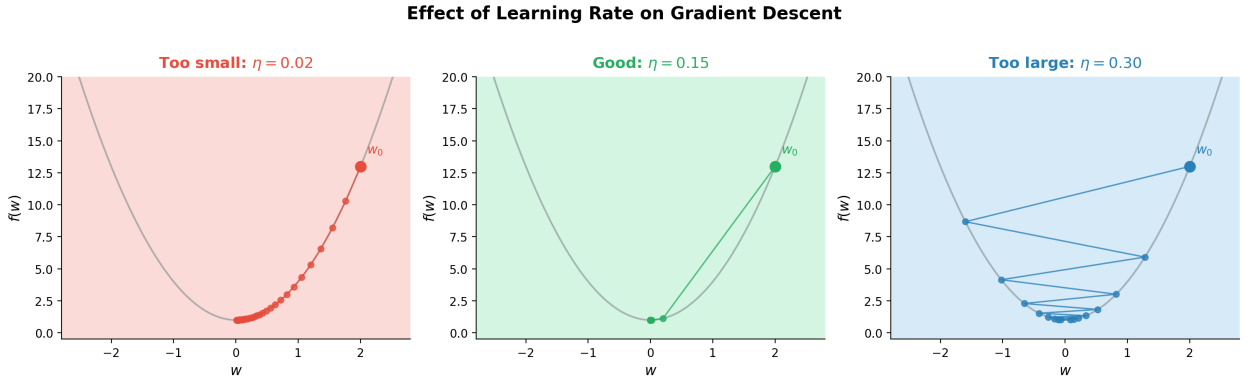

Figure 2: Learning rate comparison

Equivalently (for differentiable f): f is convex if and only if

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) \qquad \text{for all } \mathbf{x}, \mathbf{y}$$

Interpretation: The tangent line (or tangent plane) at any point lies **below** the function. Convexity guarantees that every local minimum is a **global** minimum.

3.3 The Descent Lemma

The descent lemma is the fundamental building block for analyzing gradient descent.

> **Lemma (Descent Lemma).** If f is L-smooth and we run gradient descent with step size $\eta = 1/L$, then:
>
> $$f(\mathbf{w}_{t+1}) \leq f(\mathbf{w}_t) - \frac{1}{2L}\|\nabla f(\mathbf{w}_t)\|^2$$

Proof. By L-smoothness, we have the **quadratic upper bound**: for any $\mathbf{x}, \mathbf{y}$,

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) + \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2$$

(This follows from integrating the Lipschitz gradient condition along the line segment from $\mathbf{x}$ to $\mathbf{y}$.)

Set $\mathbf{x} = \mathbf{w}_t$ and $\mathbf{y} = \mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{L}\nabla f(\mathbf{w}_t)$. Then $\mathbf{y} - \mathbf{x} = -\frac{1}{L}\nabla f(\mathbf{w}_t)$, so:

$$f(\mathbf{w}_{t+1}) \leq f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^\top \left(-\frac{1}{L}\nabla f(\mathbf{w}_t)\right) + \frac{L}{2}\left\|\frac{1}{L}\nabla f(\mathbf{w}_t)\right\|^2$$

$$= f(\mathbf{w}_t) - \frac{1}{L}\|\nabla f(\mathbf{w}_t)\|^2 + \frac{1}{2L}\|\nabla f(\mathbf{w}_t)\|^2$$

$$= f(\mathbf{w}_t) - \frac{1}{2L}\|\nabla f(\mathbf{w}_t)\|^2 \qquad \blacksquare$$

What this says: Every step of GD **decreases** the loss by at least $\frac{1}{2L}\|\nabla f(\mathbf{w}_t)\|^2$. The larger the gradient, the more progress we make.

3.4 Convergence Theorem (Smooth + Convex)

> **Theorem.** Let f be convex and L-smooth with minimizer $\mathbf{w}^*$. Gradient descent with step size $\eta = 1/L$ satisfies:
>
> $$f(\mathbf{w}_T) - f(\mathbf{w}^*) \leq \frac{L\|\mathbf{w}_0 - \mathbf{w}^*\|^2}{2T}$$

This is a $O(1/T)$ **convergence rate**: to reach $f(\mathbf{w}_T) - f(\mathbf{w}^*) \leq \varepsilon$, we need $T = O(L\|\mathbf{w}_0 - \mathbf{w}^*\|^2/\varepsilon)$ iterations.

Proof. By convexity:

$$f(\mathbf{w}^*) \geq f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^\top (\mathbf{w}^* - \mathbf{w}_t)$$

Rearranging: $f(\mathbf{w}_t) - f(\mathbf{w}^*) \leq \nabla f(\mathbf{w}_t)^\top (\mathbf{w}_t - \mathbf{w}^*)$.

By the descent lemma, $f(\mathbf{w}_{t+1}) \leq f(\mathbf{w}_t) - \frac{1}{2L}\|\nabla f(\mathbf{w}_t)\|^2$. Combining:

$$f(\mathbf{w}_t) - f(\mathbf{w}^*) \leq \nabla f(\mathbf{w}_t)^\top (\mathbf{w}_t - \mathbf{w}^*) \leq \frac{L}{2}\|\mathbf{w}_t - \mathbf{w}^*\|^2 - \frac{L}{2}\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2$$

where the second inequality uses the identity $\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 = \|\mathbf{w}_t - \frac{1}{L}\nabla f(\mathbf{w}_t) - \mathbf{w}^*\|^2 = \|\mathbf{w}_t - \mathbf{w}^*\|^2 - \frac{2}{L}\nabla f(\mathbf{w}_t)^\top(\mathbf{w}_t - \mathbf{w}^*) + \frac{1}{L^2}\|\nabla f(\mathbf{w}_t)\|^2$.

Summing from $t = 0$ to $T - 1$ and using $f(\mathbf{w}_T) \le f(\mathbf{w}_t)$ for all t (descent lemma):

$$T\big(f(\mathbf{w}_T) - f(\mathbf{w}^*)\big) \le \sum_{t=0}^{T-1}\big(f(\mathbf{w}_t) - f(\mathbf{w}^*)\big) \le \frac{L}{2}\|\mathbf{w}_0 - \mathbf{w}^*\|^2 - \frac{L}{2}\|\mathbf{w}_T - \mathbf{w}^*\|^2 \le \frac{L}{2}\|\mathbf{w}_0 - \mathbf{w}^*\|^2$$

Dividing by T: $f(\mathbf{w}_T) - f(\mathbf{w}^*) \le \dfrac{L\|\mathbf{w}_0 - \mathbf{w}^*\|^2}{2T}$. ■

Part 4: Stochastic Gradient Descent

4.1 Motivation

Recall from earlier lectures that in supervised machine learning we have a dataset of n **training examples** $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where each x_i is an input (e.g., an image, a sentence, a feature vector) and y_i is the desired output (e.g., a label, a score). Our model, parameterized by weights $\mathbf{w}$, makes a prediction on input x_i, and the **loss** $f_i(\mathbf{w})$ measures how far off that prediction is from the true output y_i. For example, $f_i(\mathbf{w})$ could be the squared error $(\text{prediction}_i - y_i)^2$.

The overall loss function is the **average** loss over all training examples:

$$f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w})$$

We want to find weights $\mathbf{w}$ that minimize this average loss — this is exactly the optimization problem from Part 1. The full gradient is a corresponding average of per-example gradients:

$$\nabla f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{w})$$

Problem: Each gradient descent step requires computing $\nabla f_i(\mathbf{w})$ for *every* training example — that's $O(n)$ gradient evaluations per step. When n is in the millions or billions, this is extremely expensive.

4.2 The SGD Update Rule

Stochastic gradient descent (SGD) replaces the full gradient with a **single-sample estimate**:

$$\boxed{\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla f_{i_t}(\mathbf{w}_t)}$$

where i_t is sampled uniformly at random from $\{1, 2, \ldots, n\}$.

Why this works: The stochastic gradient is an **unbiased estimator** of the true gradient:

$$\mathbb{E}_{i_t}\big[\nabla f_{i_t}(\mathbf{w}_t)\big] = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{w}_t) = \nabla f(\mathbf{w}_t)$$

On average, SGD points in the right direction — but any individual step is noisy.

4.3 Mini-Batch SGD

In practice, we use a **mini-batch** of size B: sample a random subset $\mathcal{B}_t \subset \{1, \ldots, n\}$ with $|\mathcal{B}_t| = B$ and update:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \cdot \frac{1}{B} \sum_{i \in \mathcal{B}_t} \nabla f_i(\mathbf{w}_t)$$

This reduces the **variance** of the gradient estimate by a factor of B while keeping the cost per step at $O(B)$ instead of $O(n)$. Typical values: $B = 32, 64, 128, 256$.

Part 5: Further Reading and Video Lectures

Video Lectures

- **Gilbert Strang (MIT 18.065, Lecture 22):** "Gradient Descent: Pair $S = A^\top A$." Clear treatment of gradient descent for quadratics, condition numbers, and convergence. MIT OCW
- **Andrew Ng (Stanford CS229):** Machine learning course with excellent coverage of gradient descent and SGD in the context of linear regression and neural networks. YouTube
- **Kilian Weinberger (Cornell CS4780):** "Machine Learning for Intelligent Systems" — thorough treatment of optimization for ML, including convergence proofs. YouTube

Notes and Textbooks

- **Stanford CS231n:** Optimization notes — practical guide to SGD, momentum, Adam, and learning rate schedules. cs231n.github.io
- **MIT 6.390 (Introduction to Machine Learning):** Clear exposition of gradient descent with convergence analysis. mit.edu
- **Ryan Tibshirani (CMU):** Convex Optimization course notes — rigorous treatment of the descent lemma, convergence rates, and strong convexity. stat.cmu.edu
- **Google ML Crash Course:** Practical introduction to gradient descent with interactive visualizations. developers.google.com