Yale University

Iterative Methods for Linear Regression

S&DS 265 · Introductory Machine Learning · Lecture 5

Zhuoran Yang

Fall 2026

AI usage: figures were generated, and these slides polished, with the help of AI tools.

Outline

Learning Objectives

In this lecture you will learn:

1. Why iterative methods replace the exact least-squares solution at large scale;
2. How the least-squares gradient produces the gradient-descent update;
3. How feature scaling and curvature determine stable learning rates;
4. Why minibatches give useful but noisy gradient estimates;
5. How momentum and training diagnostics improve practical optimization.

Outline

Motivating Question: Fitting Without a Formula

Lecture 3 solved least squares exactly, by forming and inverting $X^\top X$. That works for the problems seen so far. It does not scale, and for most model classes—logistic regression, neural networks—no formula for the minimizer exists at all.

How do we fit a model when we cannot solve for the answer?

Why Not Solve the Normal Equations?

Even when a formula exists, its cost grows quickly. Forming $X^\top X$ takes $O(nd^2)$ operations and factoring it takes $O(d^3)$, while a single gradient

$$\nabla L(\theta) = \frac{1}{n} X^\top (X\theta - y)$$

costs only $O(nd)$: two passes over the data, no matrix to factor.

Why Not Solve the Normal Equations?

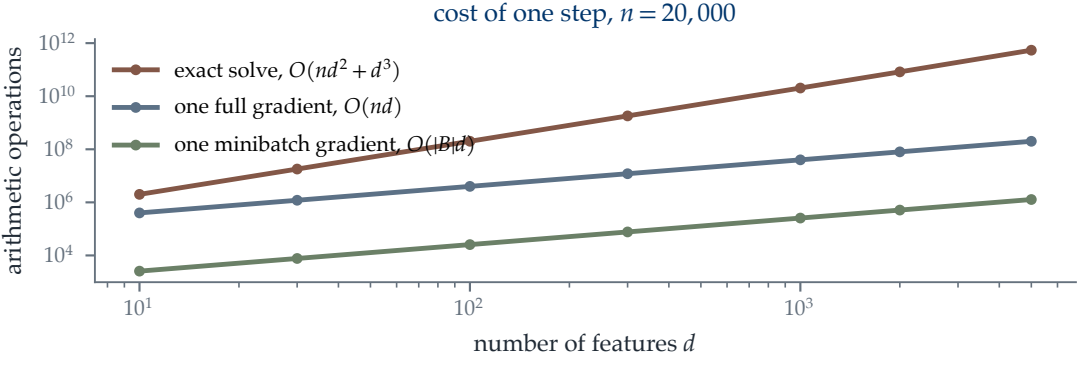
Even when a formula exists, its cost grows quickly. Forming $X^\top X$ takes $O(nd^2)$ operations and factoring it takes $O(d^3)$, while a single gradient

$$\nabla L(\theta) = \frac{1}{n} X^\top (X\theta - y)$$

costs only $O(nd)$: two passes over the data, no matrix to factor.

At $n = 20{,}000$ and $d = 2000$ the exact solve takes about 770 ms on a laptop; one full gradient takes 12 ms, and a gradient on 128 rows takes 0.1 ms.

The Cost of One Step



The exact solve is cubic in d and a gradient step is linear, so the gap widens without bound. Sampling a minibatch drops the cost again by the factor $n/|B|$, independent of d.

Operation counts, not timings: wall-clock on a laptop is dominated by threading and reads as noise.

A Simulated Regression Problem

This lecture uses simulated data. A real data set of teaching size would undercut the argument just made: at $n = 200$ and $d = 2$ the exact solve finishes in well under a millisecond.

A Simulated Regression Problem

This lecture uses simulated data. A real data set of teaching size would undercut the argument just made: at $n = 200$ and $d = 2$ the exact solve finishes in well under a millisecond.

Draw n rows independently and set

$$x_i \sim \mathcal{N}(0, \Sigma), \qquad y_i = x_i^\top \theta^\star + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}(0, \sigma^2),$$

with $\theta^\star$, Σ, and σ chosen by us.

A Simulated Regression Problem

This lecture uses simulated data. A real data set of teaching size would undercut the argument just made: at $n = 200$ and $d = 2$ the exact solve finishes in well under a millisecond.

Draw n rows independently and set

$$x_i \sim \mathcal{N}(0, \Sigma), \qquad y_i = x_i^\top \theta^\star + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}(0, \sigma^2),$$

with $\theta^\star$, Σ, and σ chosen by us.

Simulation buys something a real data set cannot: choosing Σ fixes the curvature of the objective, and curvature is what this lecture is about. The figures below use $n = 2000$, $d = 2$ so the iterates can be drawn, with the condition number set to 8.

Problem Setup

Training data. $\mathcal{D}_n = \{(x_i, y_i)\}_{i=1}^n$, drawn independently.

Features. $x_i \in \mathbb{R}^d$, with covariance Σ fixed by us.

Response. $y_i \in \mathbb{R}$, generated from a linear rule plus noise.

Goal. Approximate the least-squares solution without solving a large linear system.

The drawn examples use $n = 2000$ and $d = 2$ so that iterates can be plotted; nothing in the algorithm depends on either being small.

Data Assumption and Regression Target

We retain the statistical model from Lecture 3:

$$Y = m(X) + \varepsilon, \qquad \mathbb{E}[\varepsilon \mid X] = 0, \qquad \boxed{m(x) = \mathbb{E}[Y \mid X = x]}.$$

Squared prediction loss therefore targets the conditional mean m. The optimization algorithm does not change this target; it changes only how we search within the chosen model class.

Model Class, Objective, and Solution

Model class. Linear functions $\mathcal{F}_{\text{lin}} = \{f_\theta(x) = x^\top \theta : \theta \in \mathbb{R}^d\}$.

Empirical risk.

$$L(\theta) = \widehat{R}_n(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (x_i^\top \theta - y_i)^2 = \frac{1}{2n} \|X\theta - y\|_2^2.$$

Solution method. Begin at θ_0 and repeatedly use the gradient to reduce L, stopping when further computation is not useful.

The Least-Squares Gradient

derive

The iterative method needs the gradient of the average squared loss. Expand the objective:

$$L(\theta) = \frac{1}{2n}\left(\theta^\top X^\top X\theta - 2y^\top X\theta + y^\top y\right).$$

The Least-Squares Gradient

derive

The iterative method needs the gradient of the average squared loss. Expand the objective:

$$L(\theta) = \frac{1}{2n}\left(\theta^\top X^\top X\theta - 2y^\top X\theta + y^\top y\right).$$

Differentiating gives

$$\boxed{\nabla L(\theta) = \frac{1}{n}X^\top(X\theta - y)}.$$

A stationary point solves the normal equation $X^\top X\theta = X^\top y$.

The General Optimization Problem

Nothing from here on is special to least squares. Write the problem in general form,

$$\underset{\theta \in \mathbb{R}^d}{\text{minimize}} \; L(\theta),$$

and assume throughout that L is differentiable.

The General Optimization Problem

Nothing from here on is special to least squares. Write the problem in general form,

$$\underset{\theta \in \mathbb{R}^d}{\text{minimize}} \; L(\theta),$$

and assume throughout that L is differentiable.

Least squares is the instance $L(\theta) = \frac{1}{2n}\|X\theta - y\|_2^2$ whose gradient we just computed. Logistic regression and neural networks supply other instances later in the course, and the methods below carry over unchanged.

The General Optimization Problem

Nothing from here on is special to least squares. Write the problem in general form,

$$\underset{\theta \in \mathbb{R}^d}{\text{minimize}} \; L(\theta),$$

and assume throughout that L is differentiable.

Least squares is the instance $L(\theta) = \frac{1}{2n}\|X\theta - y\|_2^2$ whose gradient we just computed. Logistic regression and neural networks supply other instances later in the course, and the methods below carry over unchanged.

A closed form for the minimizer is the exception, not the rule. In general we can only search: start somewhere and improve repeatedly.

Iterative Methods and Local Information

Every method in this lecture has one shape. From the current iterate, choose a direction and how far to move along it:

$$\theta_{t+1} = \theta_t + \eta_t d_t, \qquad d_t \in \mathbb{R}^d, \quad \eta_t > 0.$$

The **direction** d_t says where to go, the **step size** η_t how far.

Iterative Methods and Local Information

Every method in this lecture has one shape. From the current iterate, choose a direction and how far to move along it:

$$\theta_{t+1} = \theta_t + \eta_t d_t, \qquad d_t \in \mathbb{R}^d, \quad \eta_t > 0.$$

The **direction** d_t says where to go, the **step size** η_t how far.

The direction can use only what is computable at θ_t. There are three levels of such local information:

- The value $L(\theta_t)$ alone — **zeroth order**;
- The gradient $\nabla L(\theta_t)$ — **first order**;
- The Hessian $\nabla^2 L(\theta_t)$ — **second order**.

This Lecture Uses First-Order Information

Richer information buys a better direction but costs more per step. For least squares with n rows and d features:

- $\nabla L(\theta) = \frac{1}{n} X^\top (X\theta - y)$ costs $O(nd)$, one pass over the data;
- $\nabla^2 L(\theta) = \frac{1}{n} X^\top X$ costs $O(nd^2)$ to form and $O(d^3)$ to solve with — exactly the cost we set out to avoid.

This Lecture Uses First-Order Information

Richer information buys a better direction but costs more per step. For least squares with n rows and d features:

- $\nabla L(\theta) = \frac{1}{n} X^\top (X\theta - y)$ costs $O(nd)$, one pass over the data;
- $\nabla^2 L(\theta) = \frac{1}{n} X^\top X$ costs $O(nd^2)$ to form and $O(d^3)$ to solve with — exactly the cost we set out to avoid.

Second-order methods are unaffordable at scale — in deep learning d runs to billions. So d_t comes from $\nabla L(\theta_t)$.

This Lecture Uses First-Order Information

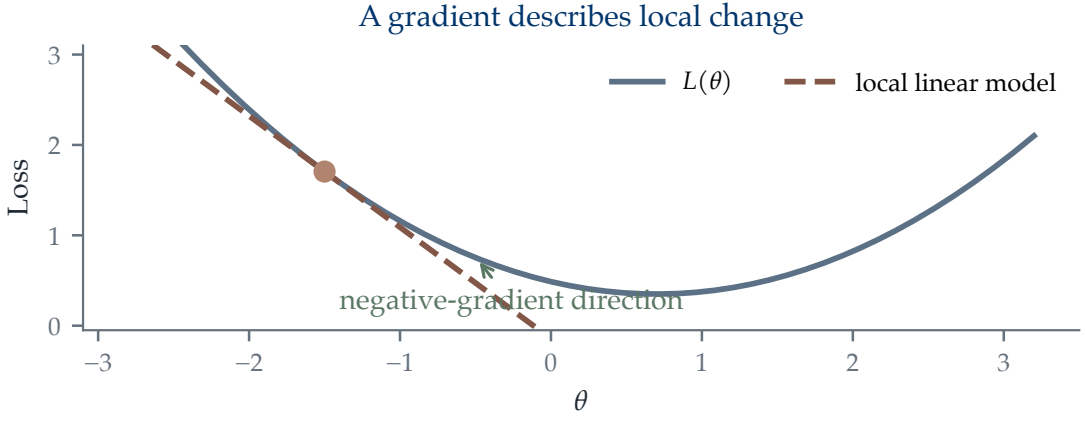
Richer information buys a better direction but costs more per step. For least squares with n rows and d features:

- $\nabla L(\theta) = \frac{1}{n} X^\top (X\theta - y)$ costs $O(nd)$, one pass over the data;
- $\nabla^2 L(\theta) = \frac{1}{n} X^\top X$ costs $O(nd^2)$ to form and $O(d^3)$ to solve with — exactly the cost we set out to avoid.

Second-order methods are unaffordable at scale — in deep learning d runs to billions. So d_t comes from $\nabla L(\theta_t)$.

That leaves one question: given the gradient, which direction should we move?

Local Linearization of a Loss



Course calculation on a declared analytic quadratic.

The Negative Gradient Is Steepest Descent

derive

The gradient determines the best local direction under a Euclidean step budget. For a small step Δ,

$$L(\theta + \Delta) = L(\theta) + \nabla L(\theta)^\top \Delta + o(\|\Delta\|).$$

Among steps with $\|\Delta\|_2 \le \eta$, Cauchy–Schwarz gives

$$\nabla L(\theta)^\top \Delta \ge -\|\nabla L(\theta)\|_2 \|\Delta\|_2 \ge -\eta \|\nabla L(\theta)\|_2.$$

Equality holds at $\Delta = -\eta \nabla L(\theta) / \|\nabla L(\theta)\|_2$.

The Negative Gradient Is Steepest Descent

derive

The gradient determines the best local direction under a Euclidean step budget. For a small step Δ,

$$L(\theta + \Delta) = L(\theta) + \nabla L(\theta)^\top \Delta + o(\|\Delta\|).$$

Among steps with $\|\Delta\|_2 \le \eta$, Cauchy–Schwarz gives

$$\nabla L(\theta)^\top \Delta \ge -\|\nabla L(\theta)\|_2 \|\Delta\|_2 \ge -\eta \|\nabla L(\theta)\|_2.$$

Equality holds at $\Delta = -\eta \nabla L(\theta) / \|\nabla L(\theta)\|_2$.

Conclusion: $-\nabla L(\theta)$ decreases $L(\cdot)$ at the fastest local rate.

The Gradient-Descent Iteration

Gradient descent uses the gradient direction with a learning rate $\eta_t > 0$:

$$\boxed{\theta_{t+1} = \theta_t - \eta_t \nabla L(\theta_t).}$$

Each iteration performs three operations:

1. Evaluate the current loss and gradient;
2. Choose a step size or schedule;
3. Update parameters and repeat until a stopping condition.

Examples of stopping condition: $\|\nabla L(\theta_t)\|_2 \leq \epsilon$.

Outline

How Large Should the Step Be?

The direction is settled: move along $-\nabla L(\theta_t)$. The update still carries a free parameter,

$$\theta_{t+1} = \theta_t - \eta_t \nabla L(\theta_t),$$

and nothing so far says what η_t should be.

How Large Should the Step Be?

The direction is settled: move along $-\nabla L(\theta_t)$. The update still carries a free parameter,

$$\theta_{t+1} = \theta_t - \eta_t \nabla L(\theta_t),$$

and nothing so far says what η_t should be.

The gradient is a local statement. It describes L near θ_t and says nothing about a point far away, so a long step can leave the region where the linear model is any good and the loss can rise instead of fall.

How Large Should the Step Be?

The direction is settled: move along $-\nabla L(\theta_t)$. The update still carries a free parameter,

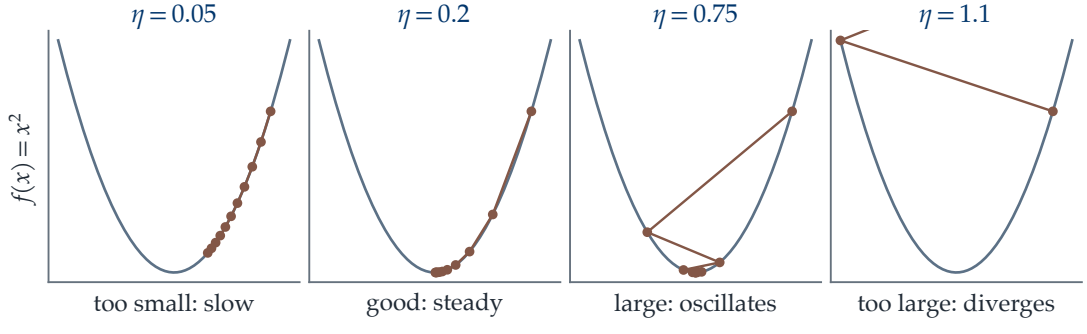
$$\theta_{t+1} = \theta_t - \eta_t \nabla L(\theta_t),$$

and nothing so far says what η_t should be.

The gradient is a local statement. It describes L near θ_t and says nothing about a point far away, so a long step can leave the region where the linear model is any good and the loss can rise instead of fall.

The question is not whether the direction is right, but how far we may trust it. We answer it on a quadratic first, where every quantity is computable, and then in general.

Four Learning Rates on One Quadratic



Ten steps on $f(x) = x^2$ from $x_0 = 10$. Too small and progress crawls; too large and the iterates overshoot and grow. In between, one range descends monotonically and another still converges while oscillating across the minimum.

Example following *Dive into Deep Learning* §12.3.

Exact Stability for a One-Dimensional Quadratic

For $L(\theta) = \frac{1}{2} a \theta^2$ with $a > 0$,

$$\theta_{t+1} = (1 - \eta a)\theta_t, \qquad \theta_t = (1 - \eta a)^t \theta_0.$$

Convergence requires

$$|1 - \eta a| < 1 \quad \Longleftrightarrow \quad \boxed{0 < \eta < \frac{2}{a}}.$$

Motion is monotone for $0 < \eta \leq 1/a$ and oscillatory for $1/a < \eta < 2/a$.

From the Quadratic to a General Loss

That threshold was exact because $\frac{1}{2}a\theta^2$ has the same curvature a everywhere. A general L has no single curvature, so the argument cannot be repeated verbatim.

From the Quadratic to a General Loss

That threshold was exact because $\frac{1}{2}a\theta^2$ has the same curvature a everywhere. A general L has no single curvature, so the argument cannot be repeated verbatim.

What we can ask instead is that the curvature be bounded. When it is, the tangent line at u plus a quadratic term lies above L everywhere and touches it at u: a surrogate that is easy to minimize and never promises more than L delivers.

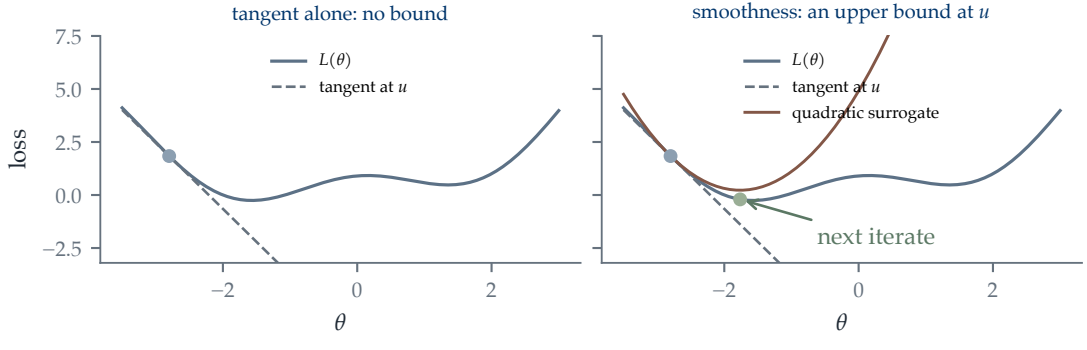
From the Quadratic to a General Loss

That threshold was exact because $\frac{1}{2}a\theta^2$ has the same curvature a everywhere. A general L has no single curvature, so the argument cannot be repeated verbatim.

What we can ask instead is that the curvature be bounded. When it is, the tangent line at u plus a quadratic term lies above L everywhere and touches it at u: a surrogate that is easy to minimize and never promises more than L delivers.

Minimizing that surrogate turns out to be exactly a gradient step, and the curvature bound will play the role a played above.

A Quadratic Surrogate Above the Loss



The tangent alone bounds nothing: L can fall away from it or rise above it. Adding $\frac{M}{2}\|\theta - u\|_2^2$ lifts it to an upper bound, and the minimizer of that bound is the next iterate.

Course illustration on a declared analytic function.

Smoothness Controls the Error of a Gradient Step

A differentiable function is M-smooth when

$$\|\nabla L(u) - \nabla L(v)\|_2 \leq M\|u - v\|_2.$$

Smoothness implies the descent lemma

$$L(v) \leq L(u) + \nabla L(u)^\top (v - u) + \frac{M}{2}\|v - u\|_2^2.$$

The right-hand side is the surrogate from the previous figure: the tangent at u lifted by a quadratic. Smoothness is exactly the condition making it an upper bound, and M is the curvature bound we needed.

The descent lemma follows from the definition by integrating the gradient along the segment from u to v; the appendix carries this out.

A Stable Step Decreases a Smooth Objective

Set $v = u - \eta \nabla L(u)$ in the descent lemma:

$$\begin{aligned} L(u - \eta \nabla L(u)) &\leq L(u) - \eta \|\nabla L(u)\|_2^2 + \frac{M\eta^2}{2} \|\nabla L(u)\|_2^2 \\ &= L(u) - \eta \left(1 - \frac{M\eta}{2}\right) \|\nabla L(u)\|_2^2. \end{aligned}$$

A Stable Step Decreases a Smooth Objective

Set $v = u - \eta \nabla L(u)$ in the descent lemma:

$$\begin{aligned} L(u - \eta \nabla L(u)) &\leq L(u) - \eta \|\nabla L(u)\|_2^2 + \frac{M\eta^2}{2} \|\nabla L(u)\|_2^2 \\ &= L(u) - \eta \left(1 - \frac{M\eta}{2}\right) \|\nabla L(u)\|_2^2. \end{aligned}$$

Thus any $0 < \eta < 2/M$ guarantees descent unless the gradient is zero.

Same form as the quadratic, with the global curvature a replaced by the bound M: $0 < \eta < 2/a$ becomes $0 < \eta < 2/M$.

Outline

Descent Is Not Enough

So far we know that a step with $0 < \eta < 2/M$ decreases L. That settles whether gradient descent makes progress; it says nothing about how much progress, or how many steps we should expect to pay for.

Descent Is Not Enough

So far we know that a step with $0 < \eta < 2/M$ decreases L. That settles whether gradient descent makes progress; it says nothing about how much progress, or how many steps we should expect to pay for.

A method that decreases the objective by a vanishing amount each step is useless in practice. The question we actually care about is the rate: how does the error after t steps shrink with t?

Descent Is Not Enough

So far we know that a step with $0 < \eta < 2/M$ decreases L. That settles whether gradient descent makes progress; it says nothing about how much progress, or how many steps we should expect to pay for.

A method that decreases the objective by a vanishing amount each step is useless in practice. The question we actually care about is the rate: how does the error after t steps shrink with t?

We answer it first for a quadratic, where the iteration can be solved in closed form, and then state the general result.

Quadratic Dynamics Separate into Eigendirections

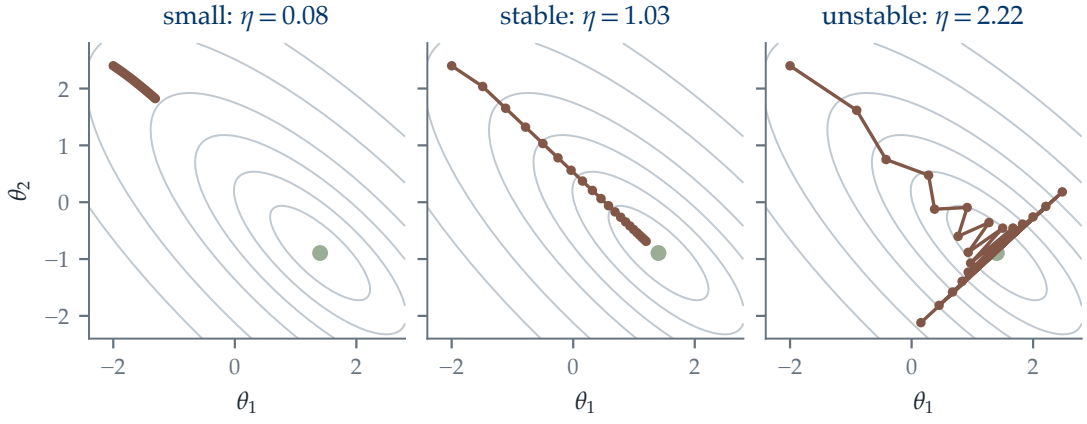
For $L(\theta) = \frac{1}{2}\theta^\top A\theta$ with $Aq_j = \lambda_j q_j$, expand $\theta_t = \sum_j c_{j,t} q_j$. Then

$$\begin{aligned} \theta_{t+1} &= (I - \eta A)\theta_t, \\ c_{j,t+1} &= (1 - \eta\lambda_j)c_{j,t}, \\ c_{j,t} &= (1 - \eta\lambda_j)^t c_{j,0}. \end{aligned}$$

Every coordinate needs $|1 - \eta\lambda_j| < 1$, so the largest eigenvalue $M = \lambda_{\max}$ sets the stable range. Taking the largest safe step $\eta = 1/M$, coordinate j contracts by $1 - \lambda_j/M$ each step, so the slowest is the smallest eigenvalue $m = \lambda_{\min}$:

$$1 - \frac{m}{M} = 1 - \frac{1}{\kappa}, \qquad \kappa = \frac{M}{m}.$$

Learning Rates on the Simulated Objective



Simulated least-squares objective, $n = 2000$, $d = 2$, condition number 7.7.

The Condition Number Is the Iteration Count

A contraction of $1 - 1/\kappa$ per step means the error after t steps is $(1 - 1/\kappa)^t$ times its initial value. Since $(1 - 1/\kappa)^t \le e^{-t/\kappa}$, driving it below ε needs

$$\boxed{t \;\gtrsim\; \kappa \log(1/\varepsilon)}$$

iterations.

The Condition Number Is the Iteration Count

A contraction of $1 - 1/\kappa$ per step means the error after t steps is $(1 - 1/\kappa)^t$ times its initial value. Since $(1 - 1/\kappa)^t \le e^{-t/\kappa}$, driving it below ε needs

$$\boxed{t \gtrsim \kappa \log(1/\varepsilon)}$$

iterations.

So κ characterizes the hardness of minimizing a quadratic function. Doubling the condition number doubles the work, and a problem with $\kappa = 10^4$ needs ten thousand times the steps of a perfectly conditioned one for the same accuracy.

The Same Rate Holds Beyond Quadratics

The quadratic answered exactly because its curvature is constant. A general L needs curvature bounded on both sides: smoothness gives the upper bound, and **m-strong convexity** gives the lower one,

$$L(v) \;\geq\; L(u) + \nabla L(u)^\top (v - u) + \frac{m}{2}\|v - u\|_2^2,$$

The Same Rate Holds Beyond Quadratics

The quadratic answered exactly because its curvature is constant. A general L needs curvature bounded on both sides: smoothness gives the upper bound, and **m-strong convexity** gives the lower one,

$$L(v) \;\geq\; L(u) + \nabla L(u)^{\top}(v-u) + \frac{m}{2}\|v-u\|_2^2,$$

Theorem. If L is M-smooth and m-strongly convex, gradient descent with $\eta = 1/M$ satisfies

$$L(\theta_t) - L^{\star} \;\leq\; \left(1 - \tfrac{1}{\kappa}\right)^{t} (L(\theta_0) - L^{\star}), \qquad \kappa = \frac{M}{m}.$$

The Same Rate Holds Beyond Quadratics

The quadratic answered exactly because its curvature is constant. A general L needs curvature bounded on both sides: smoothness gives the upper bound, and **m-strong convexity** gives the lower one,

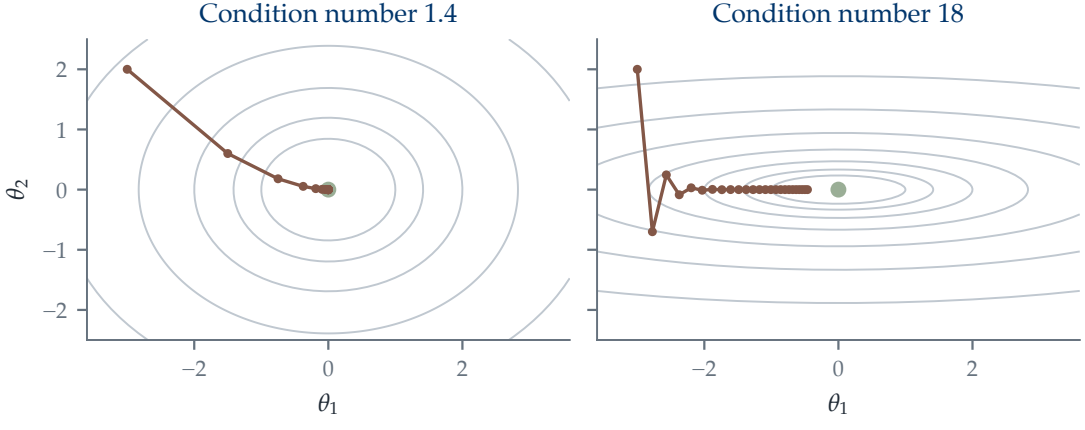
$$L(v) \;\geq\; L(u) + \nabla L(u)^{\top}(v-u) + \frac{m}{2}\|v-u\|_2^2,$$

Theorem. If L is M-smooth and m-strongly convex, gradient descent with $\eta = 1/M$ satisfies

$$L(\theta_t) - L^{\star} \;\leq\; \left(1 - \tfrac{1}{\kappa}\right)^{t} (L(\theta_0) - L^{\star}), \qquad \kappa = \frac{M}{m}.$$

The error falls by a constant factor each step, so $O(\kappa \log(1/\varepsilon))$ steps suffice. Proved in Appendix A.

Conditioning and Zig-Zagging



Trajectories of GD for quadratic objectives with the same stepsize.

Feature Scaling Changes Optimization Geometry

If a new coordinate is $z = D^{-1}x$, then predictions satisfy

$$x^\top \theta = z^\top (D\theta),$$

so the representable linear functions are unchanged. But the Hessian becomes

$$\nabla^2 L_z = \frac{1}{n} Z^\top Z = D^{-1} \left(\frac{1}{n} X^\top X \right) D^{-1}.$$

Standardization can reduce scale-driven ill-conditioning, allowing one learning rate to make useful progress across coordinates.

Outline

Empirical Risk Is a Finite Sum

Machine-learning objectives commonly have the form

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta), \qquad \nabla L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(\theta).$$

Empirical Risk Is a Finite Sum

Machine-learning objectives commonly have the form

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta), \qquad \nabla L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(\theta).$$

A full gradient touches all n observations before one update. For a very large dataset, cheaper approximate gradients can make more progress per unit of computation.

A Uniform Example Gradient Is Unbiased

At step t draw an index I_t uniformly from $\{1, \dots, n\}$ and use the single-example gradient $g_t = \nabla \ell_{I_t}(\theta_t)$ in place of $\nabla L(\theta_t)$. Writing $g(\theta) = \nabla \ell_I(\theta)$ for one such draw,

A Uniform Example Gradient Is Unbiased

At step t draw an index I_t uniformly from $\{1, \dots, n\}$ and use the single-example gradient $g_t = \nabla \ell_{I_t}(\theta_t)$ in place of $\nabla L(\theta_t)$. Writing $g(\theta) = \nabla \ell_I(\theta)$ for one such draw,

$$\begin{aligned} \mathbb{E}_I[g(\theta)] &= \sum_{i=1}^{n} \mathbb{P}(I = i) \nabla \ell_i(\theta) \\ &= \frac{1}{n} \sum_{i=1}^{n} \nabla \ell_i(\theta) \\ &= \boxed{\nabla L(\theta)}. \end{aligned}$$

The noise $\xi_t = g_t - \nabla L(\theta_t)$ has conditional mean zero, not zero magnitude.

Minibatch Averaging Reduces Variance

For independent example gradients with covariance Σ, a batch average

$$g_B = \frac{1}{B} \sum_{b=1}^{B} g_{I_b}$$

Minibatch Averaging Reduces Variance

For independent example gradients with covariance Σ, a batch average

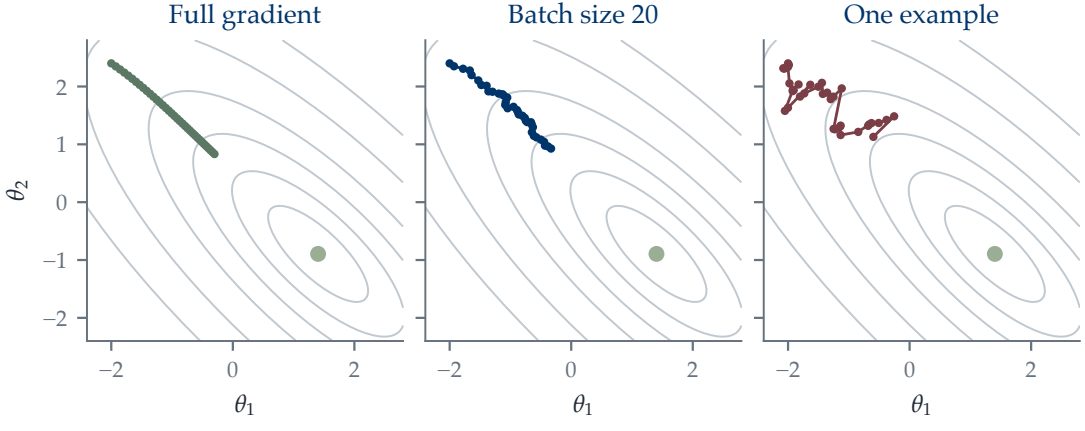
$$g_B = \frac{1}{B} \sum_{b=1}^{B} g_{I_b}$$

satisfies

$$\mathbb{E}[g_B] = \nabla L(\theta), \qquad \mathrm{Cov}(g_B) = \frac{\Sigma}{B}.$$

Hence root-mean-square estimation error decreases like $1/\sqrt{B}$, while computation per update grows roughly with B.

Stochastic Paths on the Simulated Objective



GD trajectories with full, minibatch, and single-example gradients on the same objective.

Updates, Batches, and Epochs

With n observations and batch size B,

$$\text{updates per epoch} \approx \left\lceil \frac{n}{B} \right\rceil, \qquad \text{examples processed after } T \text{ updates} = TB.$$

For $n = 10{,}000$ and $B = 100$, one epoch is 100 updates; 350 updates process about 35,000 example presentations.

Compare optimizers by processed examples, wall-clock time, or compute—not iteration count alone.

A PyTorch Minibatch Update

```
for xb, yb in loader:
    optimizer.zero_grad()
    prediction = model(xb)
    loss = loss_fn(prediction, yb)
    loss.backward()
    optimizer.step()
```

Backpropagation computes gradients; the optimizer consumes them and updates parameter state.

Outline

What the Zig-Zag Wastes

Look again at the ill-conditioned path. Along the steep direction consecutive gradients point in opposite directions: the iterate crosses the valley, overshoots, and comes back. Along the flat direction every gradient points the same way, and the iterate creeps.

What the Zig-Zag Wastes

Look again at the ill-conditioned path. Along the steep direction consecutive gradients point in opposite directions: the iterate crosses the valley, overshoots, and comes back. Along the flat direction every gradient points the same way, and the iterate creeps.

So the gradient sequence carries two kinds of component: one that keeps reversing sign and one that persists. Averaging recent gradients would cancel the first and accumulate the second — damping the crossing while building speed along the valley.

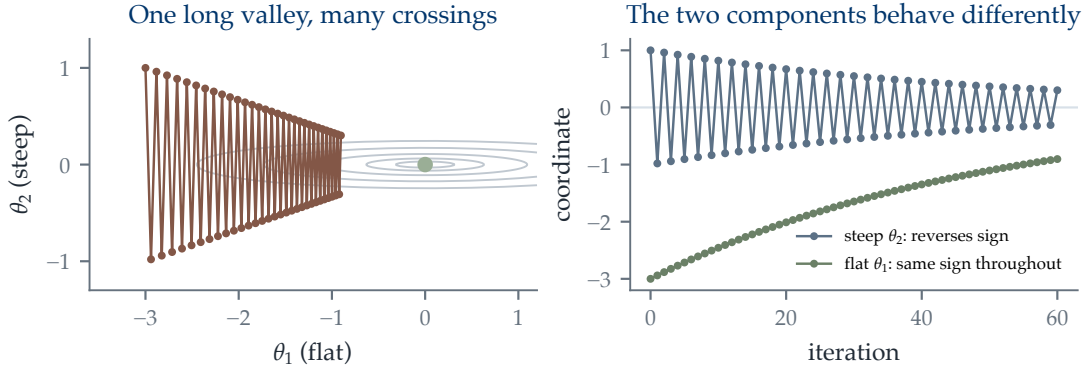
What the Zig-Zag Wastes

Look again at the ill-conditioned path. Along the steep direction consecutive gradients point in opposite directions: the iterate crosses the valley, overshoots, and comes back. Along the flat direction every gradient points the same way, and the iterate creeps.

So the gradient sequence carries two kinds of component: one that keeps reversing sign and one that persists. Averaging recent gradients would cancel the first and accumulate the second — damping the crossing while building speed along the valley.

That is exactly what momentum does.

The Two Components of the Zig-Zag



Gradient descent on $\kappa = 100$ at its best constant step. The steep coordinate reverses sign on every one of the 60 steps, so recent values nearly cancel; the flat coordinate keeps the same sign throughout, so recent values add up.

Momentum Introduces a Velocity State

Heavy-ball momentum updates

$$v_{t+1} = \beta v_t + g_t, \qquad \boxed{\theta_{t+1} = \theta_t - \eta v_{t+1}},$$

where $0 \leq \beta < 1$. The velocity remembers persistent gradient directions and can average away alternating components in a narrow valley.

Momentum Introduces a Velocity State

Heavy-ball momentum updates

$$v_{t+1} = \beta v_t + g_t, \qquad \boxed{\theta_{t+1} = \theta_t - \eta v_{t+1}},$$

where $0 \leq \beta < 1$. The velocity remembers persistent gradient directions and can average away alternating components in a narrow valley.

The update has two knobs, but in practice only one is searched: β is fixed at a large constant in $(0, 1)$ — 0.9 is the usual default — and the learning rate η is tuned. Setting $\beta = 0$ recovers plain gradient descent.

Unrolling the Momentum Memory

Starting from $v_0 = 0$,

$$\begin{aligned} v_{t+1} &= g_t + \beta v_t \\ &= g_t + \beta g_{t-1} + \beta^2 v_{t-1} \\ &= \sum_{j=0}^{t} \beta^j g_{t-j}. \end{aligned}$$

Unrolling the Momentum Memory

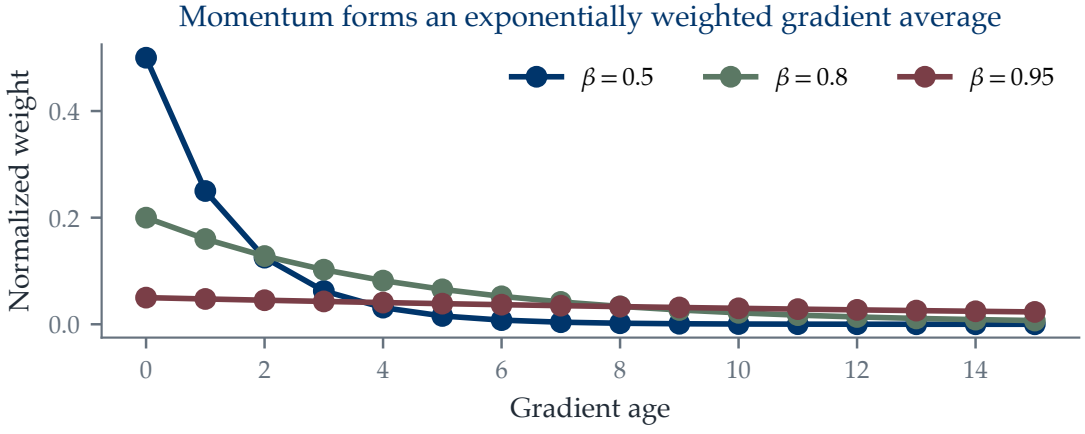
Starting from $v_0 = 0$,

$$\begin{aligned} v_{t+1} &= g_t + \beta v_t \\ &= g_t + \beta g_{t-1} + \beta^2 v_{t-1} \\ &= \sum_{j=0}^{t} \beta^j g_{t-j}. \end{aligned}$$

The velocity is a sum over all past gradients, weighted by β^j: it remembers with **exponential forgetting**, discounting each step further back by another factor of β. Multiplying by $1 - \beta$ normalizes the weights, and the effective memory spans roughly the last $1/(1-\beta)$ gradients.

At $\beta = 0.9$ that is about ten gradients; at $\beta = 0.99$, a hundred.

Momentum Memory Weights



A gradient j steps old receives normalized weight $(1-\beta)\beta^j$. Larger β makes the weights decay more slowly, so momentum remembers a longer history — roughly $1/(1-\beta)$ updates.

Heavy-Ball Dynamics in One Eigendirection

Eliminate velocity using $\theta_t - \theta_{t-1} = -\eta v_t$. For $L(c) = \frac{1}{2}\lambda c^2$,

$$\begin{aligned} c_{t+1} &= c_t - \eta(\beta v_t + \lambda c_t) \\ &= (1 + \beta - \eta\lambda)c_t - \beta c_{t-1}. \end{aligned}$$

The characteristic polynomial is

$$r^2 - (1 + \beta - \eta\lambda)r + \beta = 0.$$

Its roots determine contraction and oscillation in that eigendirection.

On a Quadratic, Momentum Replaces κ by $\sqrt{\kappa}$

The roots of $r^2 - (1 + \beta - \eta\lambda)r + \beta = 0$ can be made complex with modulus $\sqrt{\beta}$ in every eigendirection at once. Choosing

$$\beta^\star = \Big(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\Big)^2, \qquad \eta^\star = \frac{4}{(\sqrt{M} + \sqrt{m})^2}$$

makes the error contract by $\sqrt{\beta^\star}$ per step, so

$$\boxed{t \gtrsim \sqrt{\kappa}\,\log(1/\varepsilon)}$$

iterations suffice, against $\kappa \log(1/\varepsilon)$ for gradient descent.

On a Quadratic, Momentum Replaces κ by $\sqrt{\kappa}$

The roots of $r^2 - (1 + \beta - \eta\lambda)r + \beta = 0$ can be made complex with modulus $\sqrt{\beta}$ in every eigendirection at once. Choosing

$$\beta^\star = \Big(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\Big)^2, \qquad \eta^\star = \frac{4}{(\sqrt{M} + \sqrt{m})^2}$$

makes the error contract by $\sqrt{\beta^\star}$ per step, so

$$\boxed{t \;\gtrsim\; \sqrt{\kappa}\,\log(1/\varepsilon)}$$

iterations suffice, against $\kappa \log(1/\varepsilon)$ for gradient descent.

At $\kappa = 10^4$ that is a hundredfold reduction in steps, at no extra cost per step.

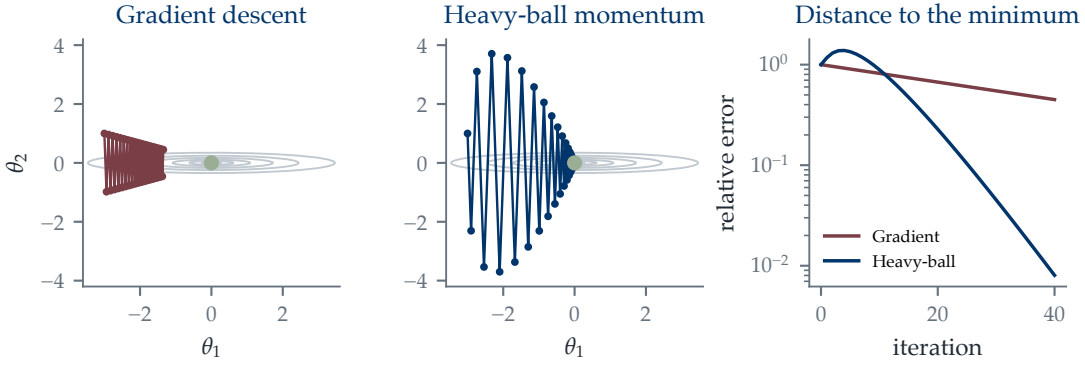
Choosing β in Practice

Inverting $\beta^\star$ shows what a given β assumes about the problem: $\sqrt{\kappa} = (1 + \sqrt{\beta})/(1 - \sqrt{\beta})$.

β	0.9	0.95	0.99
implied κ	1400	6100	158,000
memory $1/(1-\beta)$	10	20	100

The deep-learning default is $\beta = 0.9$, with 0.95 and above used for large-batch training. These are far more aggressive than a mildly ill-conditioned quadratic would justify, which is a fair signal that real objectives are badly conditioned and their gradients noisy.

Gradient Descent and Heavy-Ball Momentum



Left and centre: the iterate paths at $\kappa = 100$, each method run at its own optimal constant parameters. Both oscillate, so the paths alone do not say which is better. Right: the distance to the minimum on a log scale, where the answer is plain — 0.98 per step against 0.82, leaving 0.45 against 0.008 after 40 steps.

Momentum Trades Acceleration for Overshoot Risk

Helpful	Risky
persistent gradients, minibatch noise, and ill-conditioned valleys	large learning rate, sharp turns, and stale velocity near a minimum

Learning rate and momentum must be considered jointly because both affect the characteristic roots.

Optimizer State Is Part of a Checkpoint

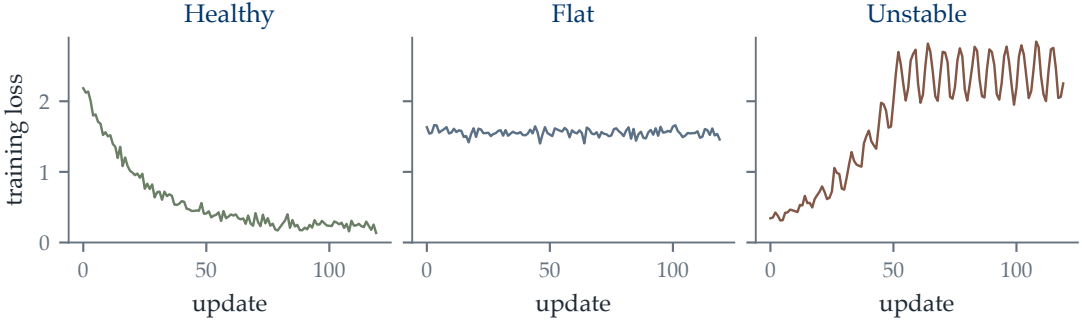
A reproducible training restart saves

- Parameters θ_t;
- Momentum or adaptive buffers such as v_t;
- Learning-rate schedule position;
- Random-number generator and data-loader state when exact replay matters.

Restoring weights without optimizer state does not resume the same algorithmic trajectory.

Outline

Reading a Training Curve



Minibatch gradients make every curve noisy, so jitter is not a failure. What matters is the trend: a healthy run decays onto a plateau, a flat run never moves, and an unstable run grows and oscillates.

Diagnostic shapes rather than empirical results.

When the Loss Becomes NaN

A non-finite loss almost always means a number grew until it overflowed. The usual culprits, in the order worth checking:

- The learning rate is too large, so the iterates diverge as in the right-hand panel;
- The initialization is too large, so gradients start big and compound through the layers;
- Unscaled features, which act like a learning rate that is too large in some coordinates;
- log of something that reached zero, or division by it.

When the Loss Becomes NaN

A non-finite loss almost always means a number grew until it overflowed. The usual culprits, in the order worth checking:

- The learning rate is too large, so the iterates diverge as in the right-hand panel;
- The initialization is too large, so gradients start big and compound through the layers;
- Unscaled features, which act like a learning rate that is too large in some coordinates;
- log of something that reached zero, or division by it.

Halve the learning rate first: if the run survives, the step size was the cause.

Summary: Gradient and Stochastic Descent

$$\theta_{t+1} = \theta_t - \eta_t g_t, \qquad \mathbb{E}[g_t \mid \theta_t] = \nabla L(\theta_t).$$

- Smoothness and curvature determine stable learning rates.
- Condition number governs the mismatch between fast and slow eigendirections.
- Minibatches exchange gradient precision for cheaper, more frequent updates.

Summary: Momentum and Diagnostics

$$v_{t+1} = \beta v_t + g_t, \qquad v_{t+1} = \sum_{j=0}^{t} \beta^j g_{t-j}.$$

- Momentum accumulates persistent directions and suppresses alternating ones.
- Its state, schedule, and random state belong in a checkpoint.
- Curves, gradient norms, updates, and tiny-batch tests localize failures.

Next Lecture

Iterative optimization can fit a large linear model, but a constant slope still cannot represent a relationship that bends with the input.

Lecture 6 — Nonlinear Regression and the Bias–Variance Tradeoff defines parametric and nonparametric function classes, fits polynomial, kernel, and neural-network regressors, and explains their flexibility and dimensionality costs.

Reading for this lecture: D2L Chapter 12, Optimization Algorithms; DL Chapter 4.

Appendix

Two self-contained parts, neither presented in the lecture hour.

Part A. Why gradient descent converges. The descent lemma from smoothness, and the linear rate under strong convexity.

Part B. The lasso. Why a gradient is not enough, the soft-thresholding operator, and the proximal gradient method.

Part B supplies the computational method that Lecture 4 left outstanding for the lasso.

Part A

Why Gradient Descent Converges

A.1 The Theorem and the Plan

Theorem. If L is M-smooth and m-strongly convex, gradient descent with $\eta = 1/M$ satisfies, for every t,

$$L(\theta_t) - L^\star \;\le\; \Big(1 - \frac{1}{\kappa}\Big)^t (L(\theta_0) - L^\star), \qquad \kappa = \frac{M}{m}.$$

A.1 The Theorem and the Plan

Theorem. If L is M-smooth and m-strongly convex, gradient descent with $\eta = 1/M$ satisfies, for every t,

$$L(\theta_t) - L^\star \;\le\; \Big(1 - \frac{1}{\kappa}\Big)^t (L(\theta_0) - L^\star), \qquad \kappa = \frac{M}{m}.$$

The proof is four steps, and it is worth naming what each one buys:

1. Smoothness ⇒ the descent lemma;
2. The descent lemma ⇒ each step decreases L by at least $\|\nabla L\|_2^2/2M$;
3. Strong convexity ⇒ a small gradient forces a small gap $L - L^\star$;
4. Combining 2 and 3 contracts the gap by a constant factor.

A.2 Both Assumptions Are Quadratic Bounds

Both assumptions say the same kind of thing: L is trapped between two quadratics that touch it at u.

M-smooth. $\|\nabla L(u) - \nabla L(v)\|_2 \leq M\|u - v\|_2$, which gives the upper bound

$$L(v) \leq L(u) + \nabla L(u)^\top (v - u) + \frac{M}{2}\|v - u\|_2^2.$$

m-strongly convex. By definition the lower bound

$$L(v) \geq L(u) + \nabla L(u)^\top (v - u) + \frac{m}{2}\|v - u\|_2^2.$$

A.2 Both Assumptions Are Quadratic Bounds

Both assumptions say the same kind of thing: L is trapped between two quadratics that touch it at u.

M-smooth. $\|\nabla L(u) - \nabla L(v)\|_2 \leq M\|u - v\|_2$, which gives the upper bound

$$L(v) \leq L(u) + \nabla L(u)^\top (v - u) + \frac{M}{2}\|v - u\|_2^2.$$

m-strongly convex. By definition the lower bound

$$L(v) \geq L(u) + \nabla L(u)^\top (v - u) + \frac{m}{2}\|v - u\|_2^2.$$

The upper bound tells us a step cannot overshoot; the lower bound tells us a flat gradient means we are already close. Neither alone gives a rate.

A.3 Step 1: Smoothness Gives the Descent Lemma

derive

Track L along the segment from u to v: with $g(t) = L(u + t(v - u))$, the chain rule and $L(v) - L(u) = g(1) - g(0)$ give

$$L(v) - L(u) = \int_0^1 \nabla L(u + t(v - u))^\top (v - u)\, dt.$$

A.3 Step 1: Smoothness Gives the Descent Lemma

derive

Track L along the segment from u to v: with $g(t) = L(u + t(v - u))$, the chain rule and $L(v) - L(u) = g(1) - g(0)$ give

$$L(v) - L(u) = \int_0^1 \nabla L(u + t(v - u))^\top (v - u)\, dt.$$

Since $\nabla L(u)^\top (v - u) = \int_0^1 \nabla L(u)^\top (v - u)\, dt$, subtracting the linear term gives

$$L(v) - L(u) - \nabla L(u)^\top (v - u) = \int_0^1 [\nabla L(u + t(v - u)) - \nabla L(u)]^\top (v - u)\, dt.$$

A.3 Step 1: Smoothness Gives the Descent Lemma

derive

Track L along the segment from u to v: with $g(t) = L(u + t(v - u))$, the chain rule and $L(v) - L(u) = g(1) - g(0)$ give

$$L(v) - L(u) = \int_0^1 \nabla L(u + t(v - u))^\top (v - u)\, dt.$$

Since $\nabla L(u)^\top (v - u) = \int_0^1 \nabla L(u)^\top (v - u)\, dt$, subtracting the linear term gives

$$L(v) - L(u) - \nabla L(u)^\top (v - u) = \int_0^1 [\nabla L(u + t(v - u)) - \nabla L(u)]^\top (v - u)\, dt.$$

The two gradient arguments differ by $t(v - u)$, so Cauchy–Schwarz and M-smoothness bound the integrand by $Mt\|v - u\|_2^2$. As $\int_0^1 t\, dt = \frac{1}{2}$, the right side is at most $\frac{M}{2}\|v - u\|_2^2$.

A.4 Step 2: One Step Decreases the Objective

derive

Put $v = \theta_t - \eta \nabla L(\theta_t)$ into the descent lemma:

$$L(\theta_{t+1}) \leq L(\theta_t) - \eta\Big(1 - \frac{M\eta}{2}\Big)\|\nabla L(\theta_t)\|_2^2.$$

A.4 Step 2: One Step Decreases the Objective

derive

Put $v = \theta_t - \eta \nabla L(\theta_t)$ into the descent lemma:

$$L(\theta_{t+1}) \le L(\theta_t) - \eta\Big(1 - \frac{M\eta}{2}\Big)\|\nabla L(\theta_t)\|_2^2.$$

At the choice $\eta = 1/M$ the bracket is $\frac{1}{2}$, giving

$$\boxed{L(\theta_{t+1}) \;\le\; L(\theta_t) - \frac{1}{2M}\|\nabla L(\theta_t)\|_2^2}$$

A.4 Step 2: One Step Decreases the Objective

derive

Put $v = \theta_t - \eta \nabla L(\theta_t)$ into the descent lemma:

$$L(\theta_{t+1}) \le L(\theta_t) - \eta\Big(1 - \frac{M\eta}{2}\Big)\|\nabla L(\theta_t)\|_2^2.$$

At the choice $\eta = 1/M$ the bracket is $\frac{1}{2}$, giving

$$\boxed{L(\theta_{t+1}) \;\le\; L(\theta_t) - \frac{1}{2M}\|\nabla L(\theta_t)\|_2^2}$$

This is not yet a rate. It says the loss falls by an amount tied to the gradient, but a small gradient could still sit far above $L^\star$. Ruling that out is what strong convexity is for.

A.5 Step 3: Strong Convexity Bounds the Gap

derive

The lower bound holds for every v, so it holds for the v that minimizes its right-hand side. That right-hand side is a quadratic in v, smallest at $v = u - \nabla L(u)/m$, where its value is

$$L(u) - \frac{1}{2m}\|\nabla L(u)\|_2^2.$$

A.5 Step 3: Strong Convexity Bounds the Gap

derive

The lower bound holds for every v, so it holds for the v that minimizes its right-hand side. That right-hand side is a quadratic in v, smallest at $v = u - \nabla L(u)/m$, where its value is

$$L(u) - \frac{1}{2m}\|\nabla L(u)\|_2^2.$$

Every $L(v)$ exceeds that number, and in particular $L^\star$ does:

$$L^\star \;\geq\; L(u) - \frac{1}{2m}\|\nabla L(u)\|_2^2 \quad\Longleftrightarrow\quad \boxed{\|\nabla L(u)\|_2^2 \;\geq\; 2m(L(u) - L^\star)}$$

A large gap forces a large gradient, so progress cannot stall while we are still far from the minimum.

A.6 Step 4: Combine and Iterate

derive

Subtract $L^\star$ from Step 2 and substitute Step 3:

$$\begin{aligned} L(\theta_{t+1}) - L^\star &\le (L(\theta_t) - L^\star) - \frac{1}{2M}\|\nabla L(\theta_t)\|_2^2 \\ &\le (L(\theta_t) - L^\star) - \frac{2m}{2M}(L(\theta_t) - L^\star) \;=\; \Big(1 - \frac{m}{M}\Big)(L(\theta_t) - L^\star). \end{aligned}$$

A.6 Step 4: Combine and Iterate

derive

Subtract $L^\star$ from Step 2 and substitute Step 3:

$$\begin{aligned} L(\theta_{t+1}) - L^\star &\le (L(\theta_t) - L^\star) - \frac{1}{2M}\|\nabla L(\theta_t)\|_2^2 \\ &\le (L(\theta_t) - L^\star) - \frac{2m}{2M}(L(\theta_t) - L^\star) \;=\; \Big(1 - \frac{m}{M}\Big)(L(\theta_t) - L^\star). \end{aligned}$$

Applying this t times proves the theorem. Since $(1 - 1/\kappa)^t \le e^{-t/\kappa}$, reaching accuracy ε needs

$$t \;\ge\; \kappa \log \frac{L(\theta_0) - L^\star}{\varepsilon}$$

iterations — the $\kappa \log(1/\varepsilon)$ count the quadratic predicted.

Part B

Solving the Lasso by Proximal Gradient

B.1 The Lasso Needs More Than a Gradient

The lasso objective splits into a smooth part and a non-smooth one:

$$\underbrace{\frac{1}{2n}\|X\theta - y\|_2^2}_{f,\ \text{smooth}} + \underbrace{\lambda\|\theta\|_1}_{g,\ \text{kinked at 0}} .$$

B.1 The Lasso Needs More Than a Gradient

The lasso objective splits into a smooth part and a non-smooth one:

$$\underbrace{\frac{1}{2n}\|X\theta - y\|_2^2}_{f,\ \text{smooth}} + \underbrace{\lambda\|\theta\|_1}_{g,\ \text{kinked at 0}} .$$

Gradient descent does not apply: $|\theta_j|$ has no derivative at zero, and zero is exactly where the solution wants to sit.

B.1 The Lasso Needs More Than a Gradient

The lasso objective splits into a smooth part and a non-smooth one:

$$\underbrace{\frac{1}{2n}\|X\theta - y\|_2^2}_{f,\ \text{smooth}} + \underbrace{\lambda\|\theta\|_1}_{g,\ \text{kinked at 0}}.$$

Gradient descent does not apply: $|\theta_j|$ has no derivative at zero, and zero is exactly where the solution wants to sit.

Recall the surrogate view: a gradient step minimizes the quadratic upper bound on f. Keep that, and carry g along exactly:

$$\theta_{t+1} = \arg\min_u \Big\{ f(\theta_t) + \nabla f(\theta_t)^\top (u - \theta_t) + \frac{1}{2\eta}\|u - \theta_t\|_2^2 + \lambda\|u\|_1 \Big\}.$$

B.2 The Step Becomes One Dimensional

derive

Dropping constants and completing the square, the minimization above is

$$\theta_{t+1} = \arg\min_u \left\{ \tfrac{1}{2}\|u - z_t\|_2^2 + \eta\lambda\|u\|_1 \right\}, \qquad z_t = \theta_t - \eta\nabla f(\theta_t),$$

an ordinary gradient step z_t followed by a correction.

B.2 The Step Becomes One Dimensional

derive

Dropping constants and completing the square, the minimization above is

$$\theta_{t+1} = \arg\min_{u} \left\{ \tfrac{1}{2}\|u - z_t\|_2^2 + \eta\lambda\|u\|_1 \right\}, \qquad z_t = \theta_t - \eta\nabla f(\theta_t),$$

an ordinary gradient step z_t followed by a correction.

Both terms are separable across coordinates, so this is d copies of one scalar problem:

$$\min_{u \in \mathbb{R}} \ \tfrac{1}{2}(u - z)^2 + \gamma|u|, \qquad \gamma = \eta\lambda.$$

B.3 Soft Thresholding

derive

Solve the scalar problem by cases.

- $u > 0$: stationarity gives $u - z + \gamma = 0$, so $u = z - \gamma$, valid when $z > \gamma$;
- $u < 0$: gives $u = z + \gamma$, valid when $z < -\gamma$;
- $u = 0$: optimal exactly when $0 \in -z + \gamma[-1, 1]$, i.e. $|z| \leq \gamma$.

B.3 Soft Thresholding

derive

Solve the scalar problem by cases.

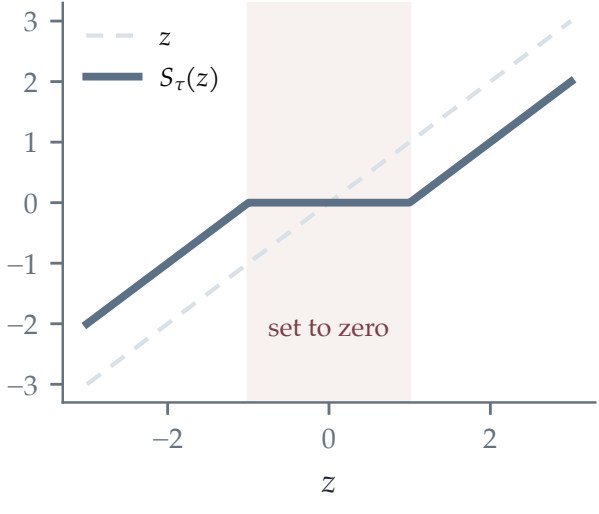
- $u > 0$: stationarity gives $u - z + \gamma = 0$, so $u = z - \gamma$, valid when $z > \gamma$;
- $u < 0$: gives $u = z + \gamma$, valid when $z < -\gamma$;
- $u = 0$: optimal exactly when $0 \in -z + \gamma[-1, 1]$, i.e. $|z| \leq \gamma$.

Combining the three cases defines the **soft-thresholding** operator

$$\boxed{S_\gamma(z) = \operatorname{sign}(z)\,(|z| - \gamma)_+}$$

It shrinks every coordinate toward zero by γ and sets to exactly zero anything smaller than γ.

B.4 The Soft-Thresholding Operator



A dead zone of width 2γ around the origin, and a unit slope outside it.

B.5 Proximal Gradient for the Lasso

Assembling the pieces gives one line per iteration:

$$\boxed{\theta_{t+1} = S_{\eta\lambda}(\theta_t - \eta \nabla f(\theta_t))}$$

a gradient step on the smooth part, then a soft threshold.

B.5 Proximal Gradient for the Lasso

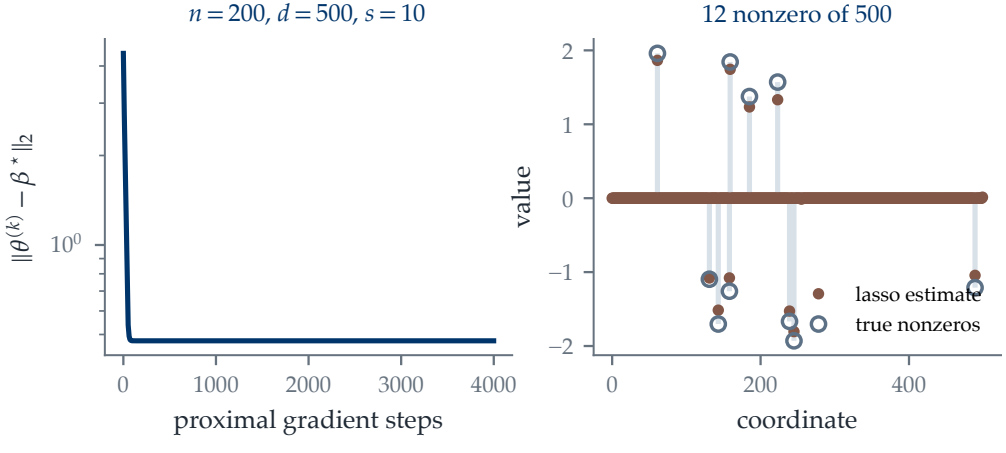
Assembling the pieces gives one line per iteration:

$$\boxed{\theta_{t+1} = S_{\eta\lambda}(\theta_t - \eta \nabla f(\theta_t))}$$

a gradient step on the smooth part, then a soft threshold.

- The threshold is what produces exact zeros; plain gradient descent would only make coefficients small.
- The same $0 < \eta < 2/M$ analysis applies, with M the smoothness constant of f alone.
- Any method from this lecture — minibatches, momentum — can supply the gradient step, with the threshold applied afterwards.

B.6 Recovering a Sparse Signal



A simulation with $d = 500$ features of which only 10 are real. Proximal gradient returns a solution with most coordinates set exactly to zero.

Course simulation.