
Fitting Models with Local Information

David I. Inouye

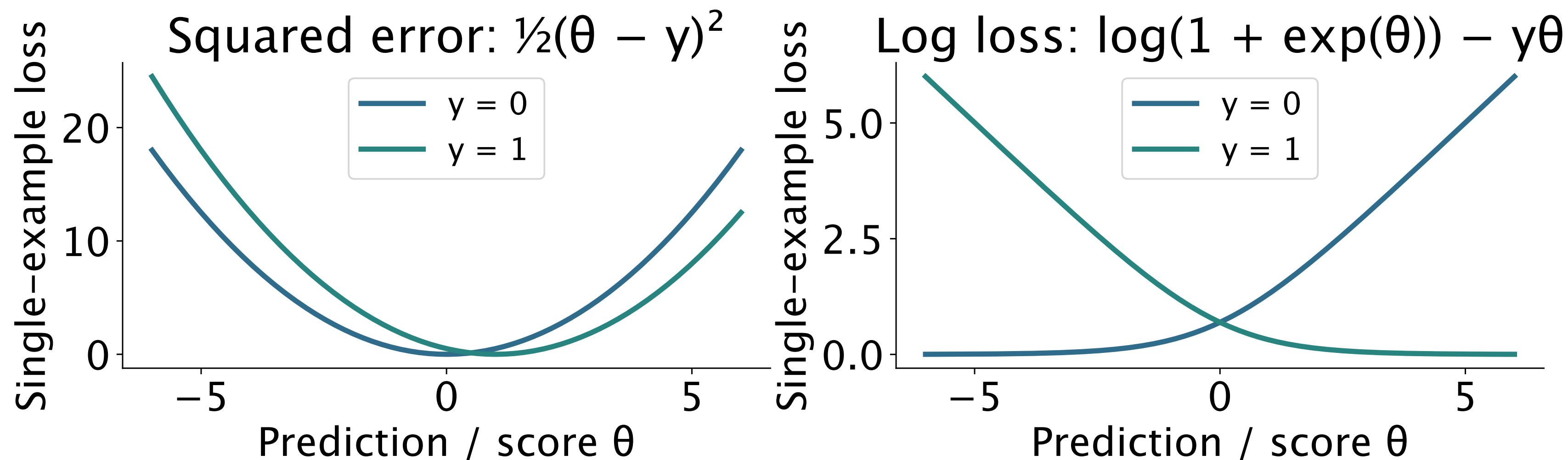
| A well-behaved shape still requires **a numerical route to the solution**

OLS: solve linear normal equations.

Logistic regression: repeatedly evaluate and improve the coefficients.

The objective is convex; our next task is to turn **local information into useful steps**.

| Logistic loss has flatter tails than squared error



- Each curve is the loss for **one example**, with its label fixed at $y = 0$ or $y = 1$.
- Squared error has constant curvature; log loss flattens toward a confident correct prediction.
- For $y = 1$, log loss decreases as the score grows; for $y = 0$, it decreases as the score falls.

| A derivative predicts the effect of a **small parameter change**

First-order Taylor expansion centered at θ , for a small change Δ :

$$J(\theta + \Delta) \approx J(\theta) + J'(\theta)\Delta.$$

- If $J'(\theta) > 0$, a small **negative** change reduces the objective.
- If $J'(\theta) < 0$, a small **positive** change reduces it.
- This is a **local approximation**, not a prediction for an arbitrary step.

| A gradient collects the **partial derivatives**

Bundle the parameters into $\boldsymbol{\theta} = (w_1, \dots, w_d, b)^\top$.

$$\nabla J(\boldsymbol{\theta}) = [\partial J / \partial w_1 \quad \dots \quad \partial J / \partial w_d \quad \partial J / \partial b]^\top.$$

For a small vector change $\boldsymbol{\Delta}$:

$$J(\boldsymbol{\theta} + \boldsymbol{\Delta}) \approx J(\boldsymbol{\theta}) + \nabla J(\boldsymbol{\theta})^\top \boldsymbol{\Delta}.$$

The gradient has **one component per parameter**.

| Moving against the gradient gives **local descent**

For $\alpha > 0$, choose $\Delta = -\alpha \nabla J(\theta)$.

$$\nabla J(\theta)^\top \Delta = -\alpha \|\nabla J(\theta)\|_2^2 \leq 0.$$

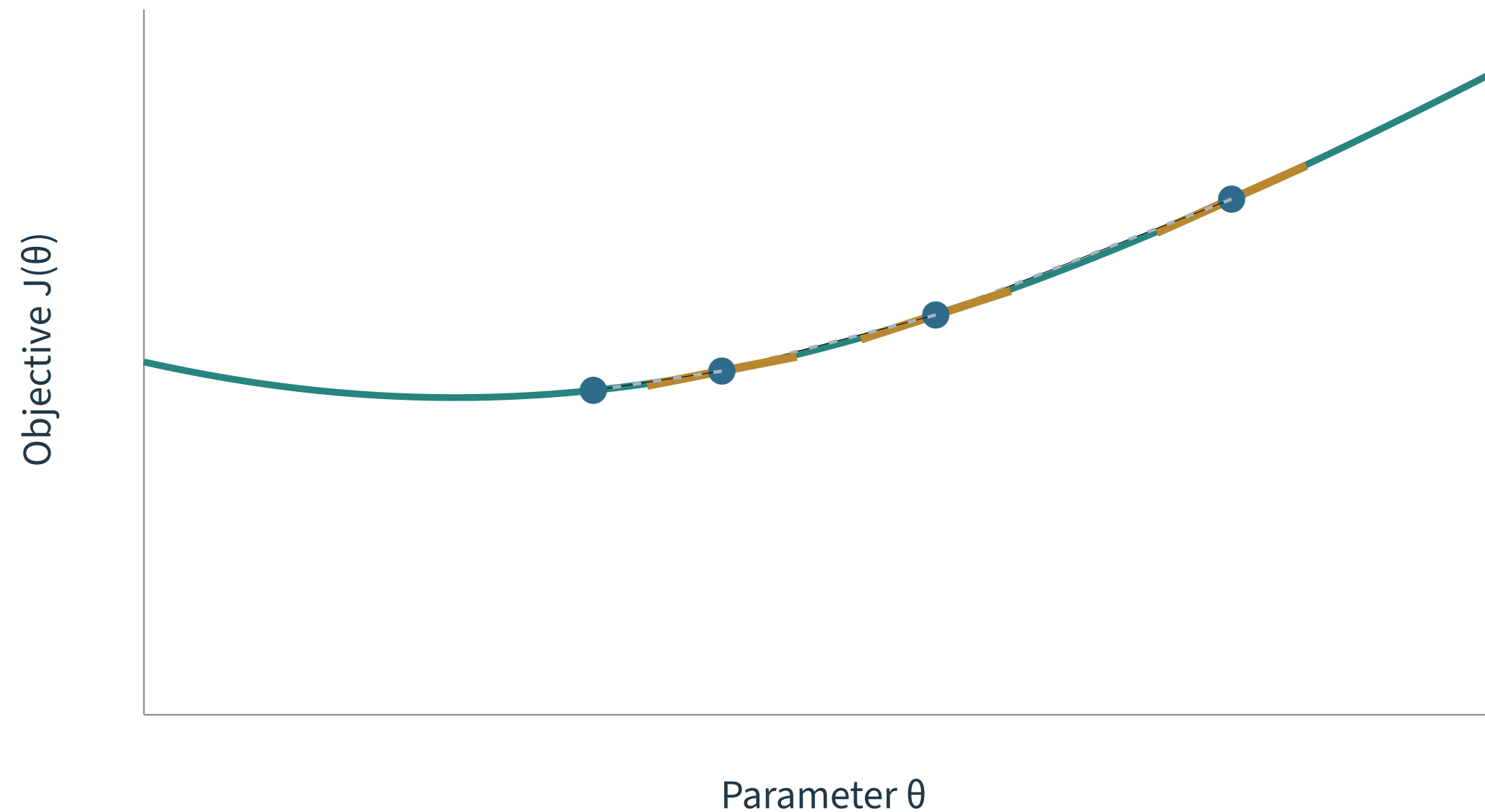
This motivates the repeated update

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t).$$

Next: inspect what these finite steps do when we can see only the local slope.

| Small steps follow the local downhill direction

Same objective, same starting point. **Learning rate:** $\alpha = 2.0$.



Full unregularized logistic loss — revealed last

The algorithm sees **this point and this slope**.

A negative-gradient step uses that local information.

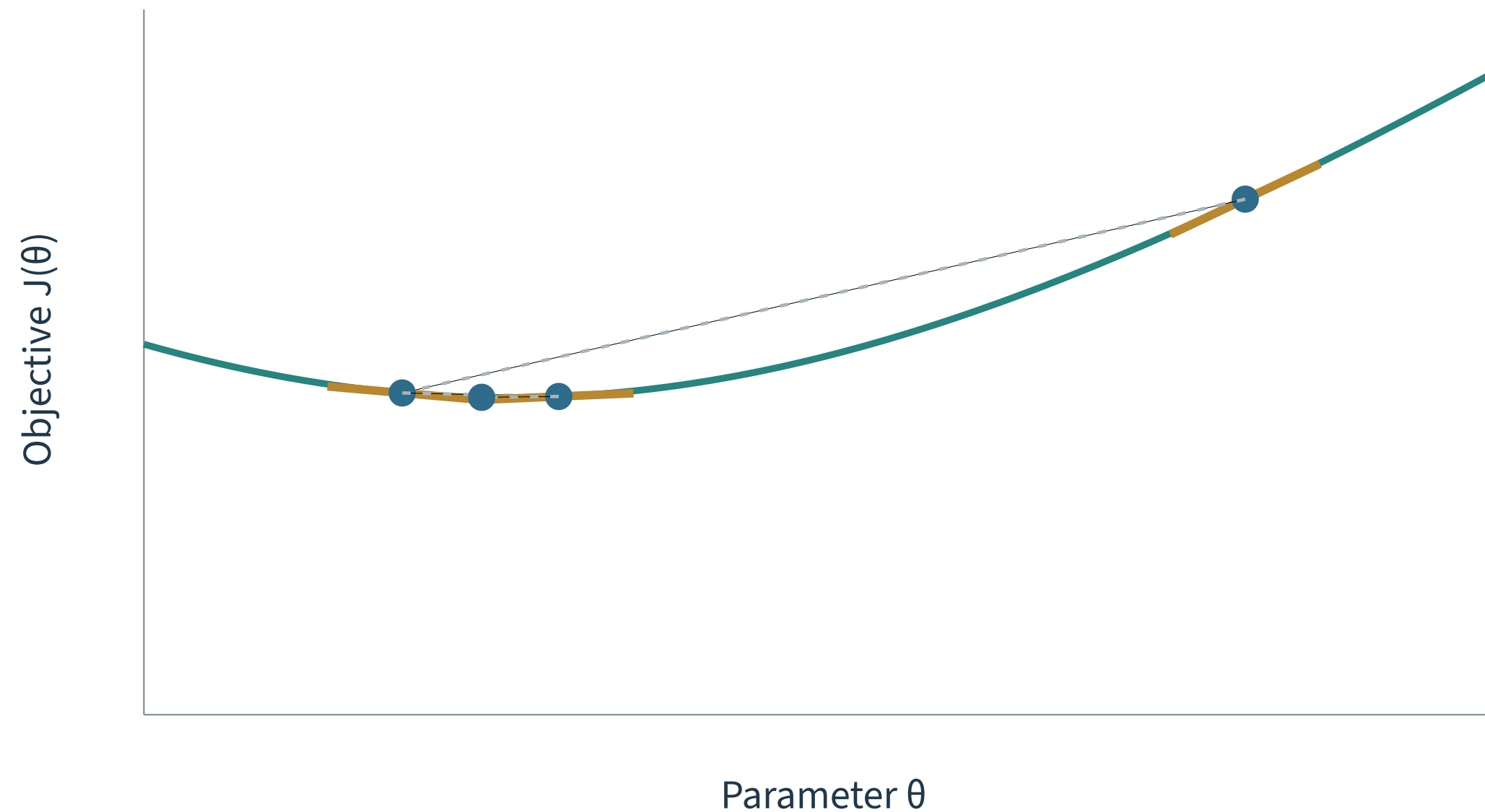
Gold segments show only the slopes we have evaluated.

The objective falls; the slope keeps its sign.

Three steps approach the minimum.

| What happens after this larger step?

Same objective, same starting point. **Learning rate:** $\alpha = 6.0$.



After the next update, what happens?

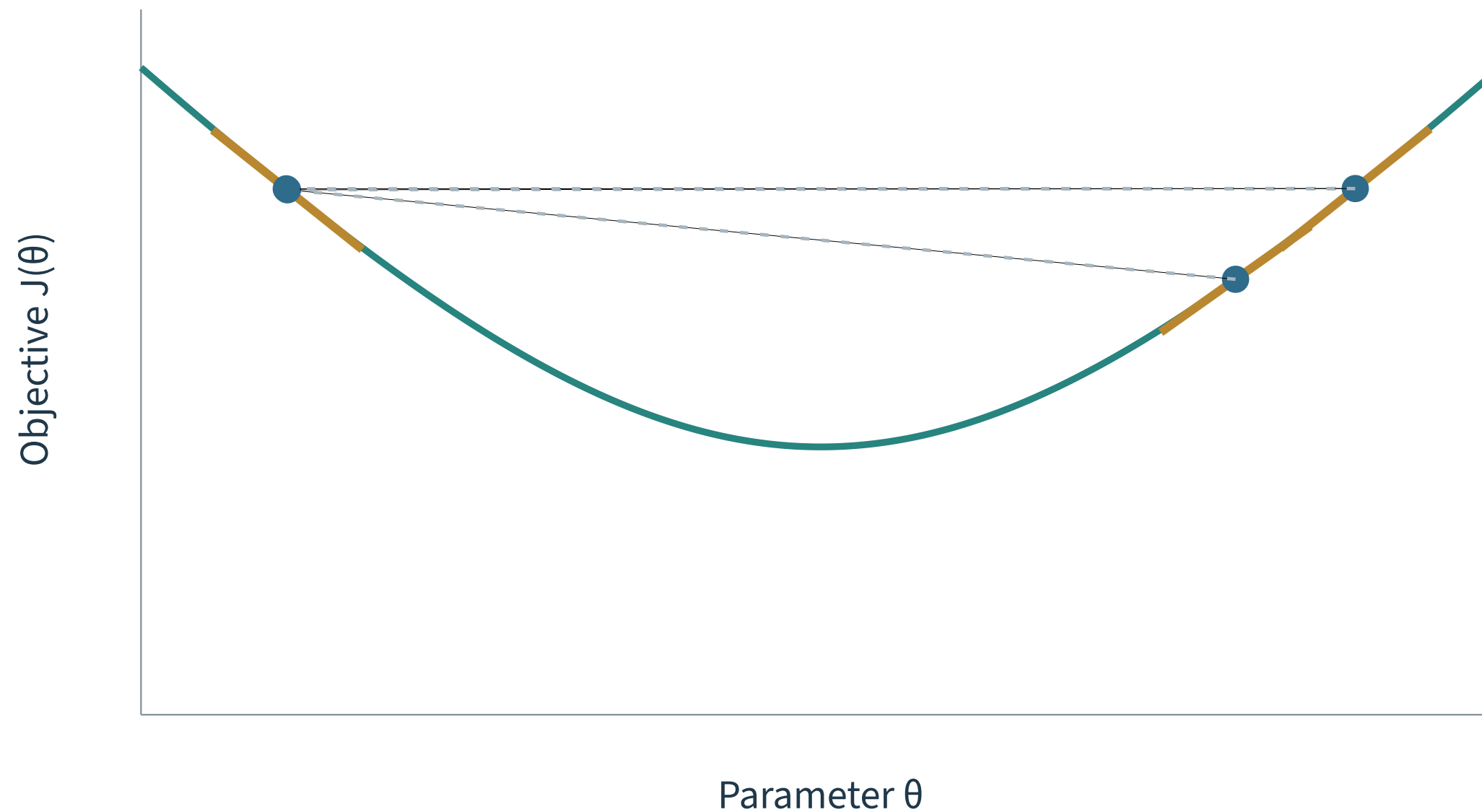
- A. Objective decreases; slope keeps its sign.
- B. Objective decreases; slope reverses sign.
- C. Objective increases; slope keeps its sign.
- D. Objective increases; slope reverses sign.

B. The objective falls, but the slope reverses: we jumped across the valley.

Three steps approach the minimum.

| What happens when we take an even larger step?

Same objective, same starting point. **Learning rate:** $\alpha = 12.0$.



Full unregularized logistic loss — revealed last

After the next update, what happens?

- A. Objective decreases; slope keeps its sign.
- B. Objective decreases; slope reverses sign.
- C. Objective increases; slope keeps its sign.
- D. Objective increases; slope reverses sign.

D. The objective rises and the slope reverses: we jumped too far.

Continuing these updates settles into an oscillation instead of reaching the minimum.

| The gradient is an **average of example contributions**

If $J(\boldsymbol{\theta}) = \frac{1}{n} \sum_i \ell_i(\boldsymbol{\theta})$, then

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{n} \sum_i \nabla \ell_i(\boldsymbol{\theta}).$$

Update	Examples used for one gradient estimate
Full-batch GD	All n examples
Single-example SGD	One randomly sampled example
Mini-batch SGD	A small randomly sampled group B

All use $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \mathbf{g}_t$.

| Sampling trades exact direction for **cheaper, noisy updates**

For uniform sampling at the current parameters:

$$\mathbf{g}_B = \frac{1}{|B|} \sum_{i \in B} \nabla \ell_i(\boldsymbol{\theta}), \quad \mathbb{E}_B[\mathbf{g}_B \mid \boldsymbol{\theta}] = \nabla J(\boldsymbol{\theta}).$$

- One stochastic step can increase the full training objective.
- Larger batches usually reduce gradient noise but cost more per update.
- Compare runs by **examples processed or time**, not only update count.

A fixed learning rate can leave persistent fluctuations near the minimum.

Adam averages directions and adapts the step size for each parameter

For the current gradient estimate $\mathbf{g}_t$:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t, \quad \mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2.$$

The hats correct the initial bias from $\mathbf{m}_0 = \mathbf{v}_0 = \mathbf{0}$:

$$\widehat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}, \quad \widehat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t}.$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \frac{\widehat{\mathbf{m}}_t}{\sqrt{\widehat{\mathbf{v}}_t} + \epsilon}.$$

Average the recent directions; reduce the step size for parameters with large recent squared gradients.

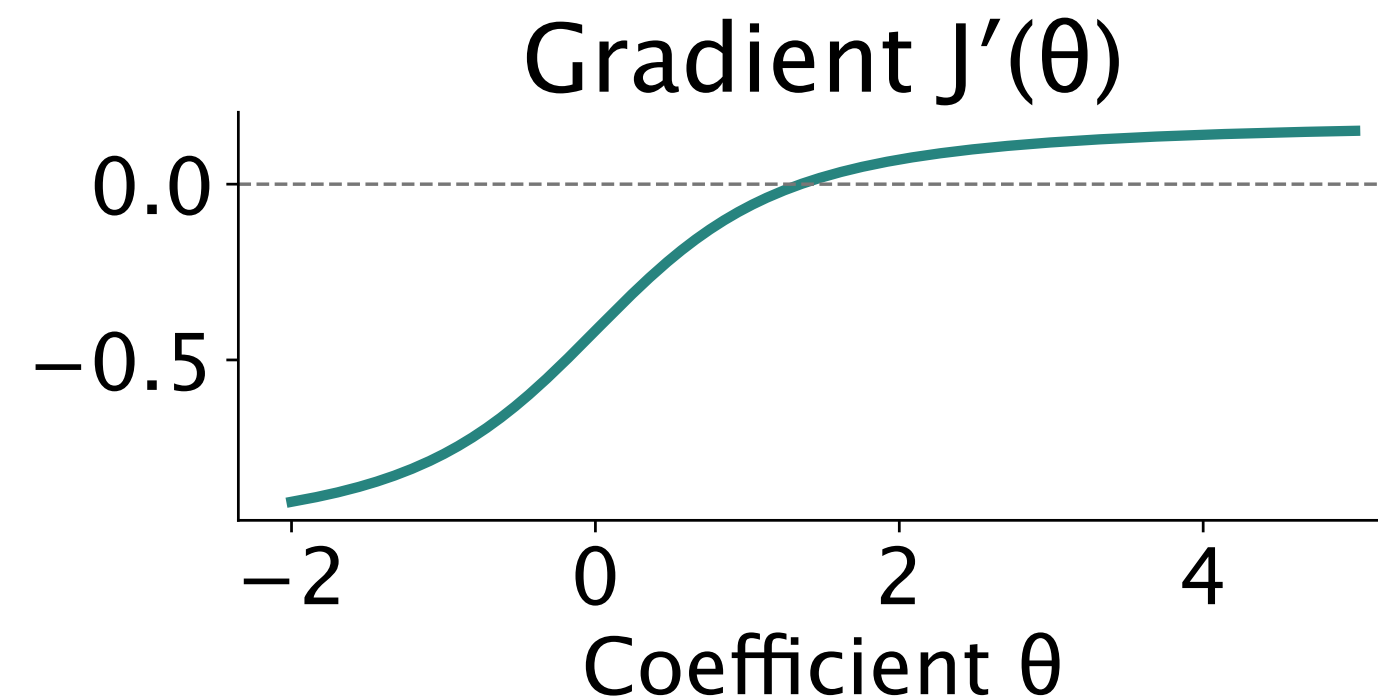
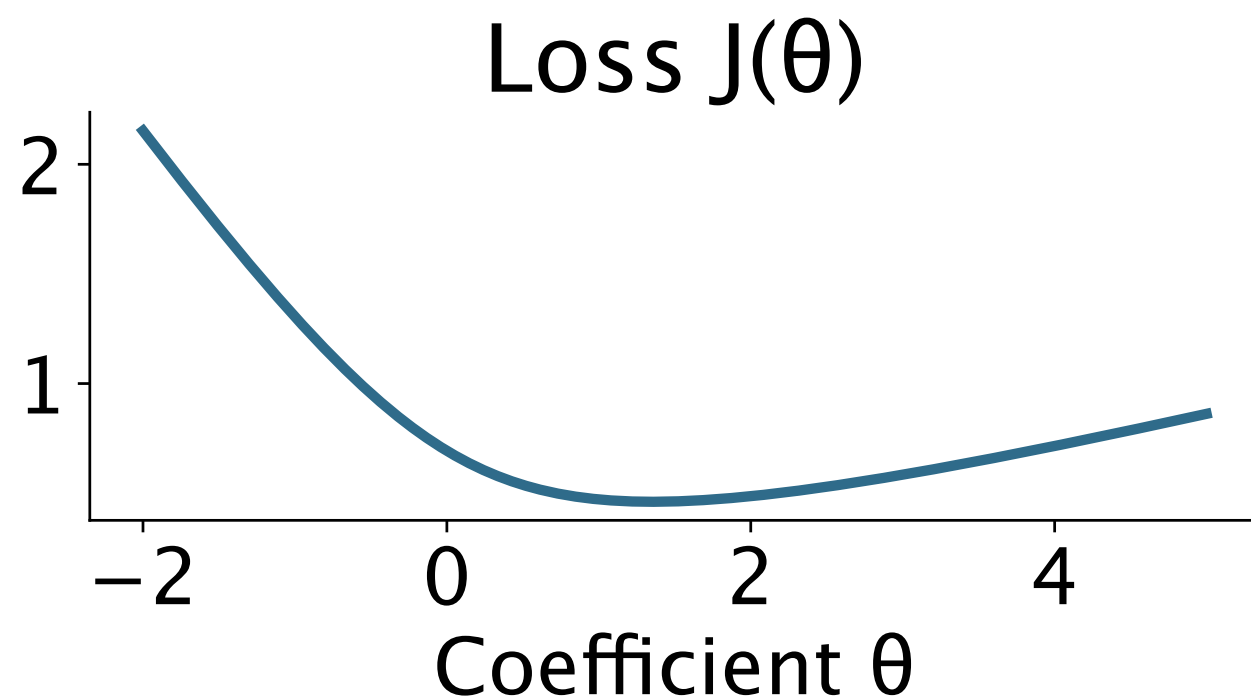
The base learning rate still matters. Adaptation is not a convergence guarantee.

The logistic objective has a directly computable gradient

Include the intercept in $\tilde{\mathbf{x}}_i = (\mathbf{x}_i^\top, 1)^\top$ and $\boldsymbol{\theta} = (\mathbf{w}^\top, b)^\top$; let $a_i = \sigma(\boldsymbol{\theta}^\top \tilde{\mathbf{x}}_i)$.

$$J(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^n [y_i \log a_i + (1 - y_i) \log(1 - a_i)], \quad \nabla_{\boldsymbol{\theta}} J = \frac{1}{n} \sum_{i=1}^n (a_i - y_i) \tilde{\mathbf{x}}_i.$$

One-coefficient example: $a_i = \sigma(\theta x_i)$, no intercept; $J'(\theta) = \frac{1}{n} \sum_i (\sigma(\theta x_i) - y_i) x_i$.



| Each example contributes **prediction error times its input**

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{n} \sum_i \underbrace{(a_i - y_i)}_{\text{probability residual}} \tilde{\mathbf{x}}_i, \quad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla J(\boldsymbol{\theta}_t).$$

- **Underpredict a positive label:** $a_i - y_i < 0$; that example's descent contribution raises its score.
- **Overpredict a negative label:** $a_i - y_i > 0$; its descent contribution lowers its score.
- Average these contributions: examples can pull in different directions; larger feature values amplify the corresponding gradient entry.

Now watch the full training objective change over repeated updates.

| Return to logistic regression and watch the fit evolve

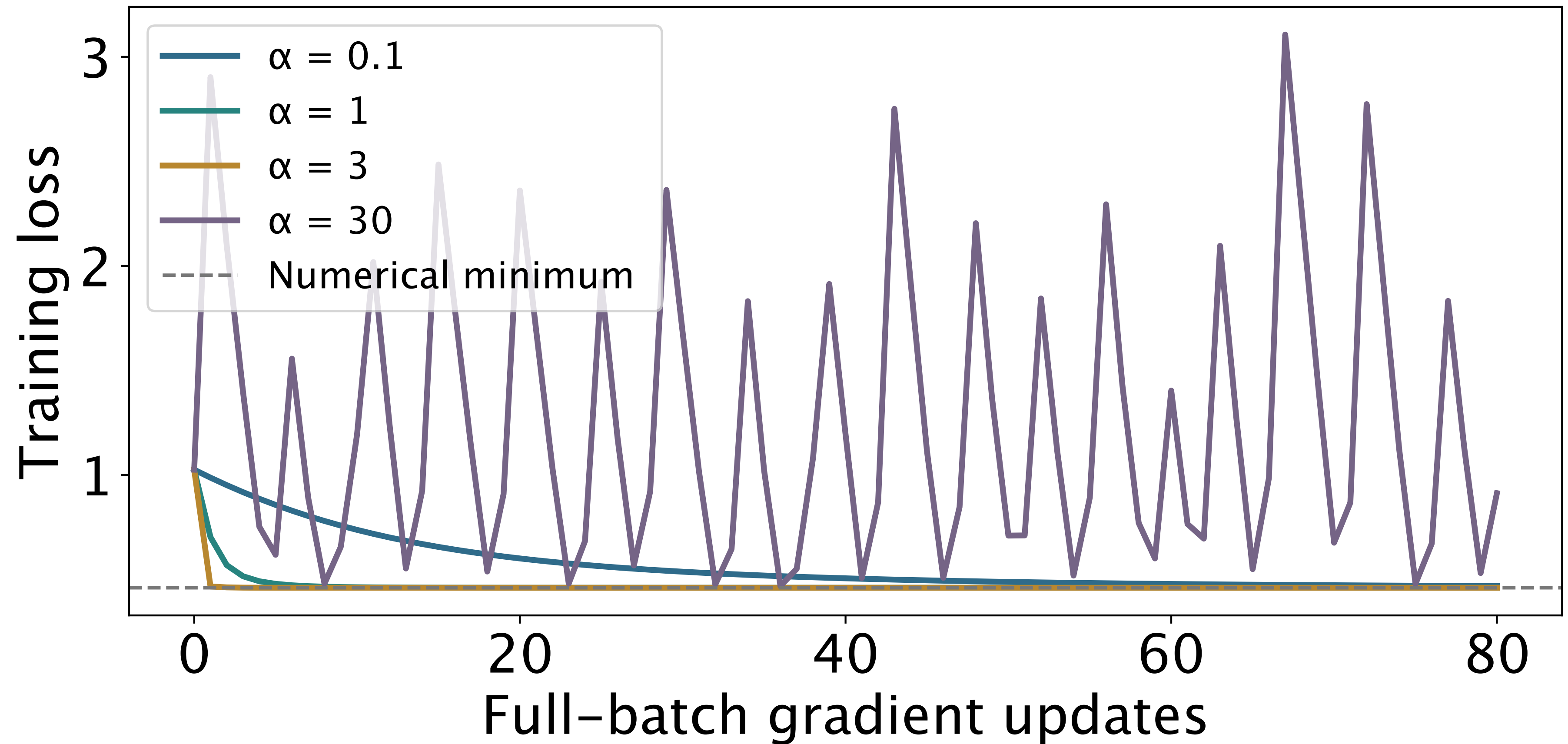
Small illustrative dataset: six inputs $x = (-2, -1, -0.5, 0.5, 1, 2)$ and labels $y = (0, 0, 1, 0, 1, 1)$.

$$a_i = \sigma(wx_i + b), \quad J(w, b) = \frac{1}{6} \sum_i \ell(y_i, a_i).$$

Start every run at $(w, b) = (-0.5, 0.8)$ and change **only the learning rate**.

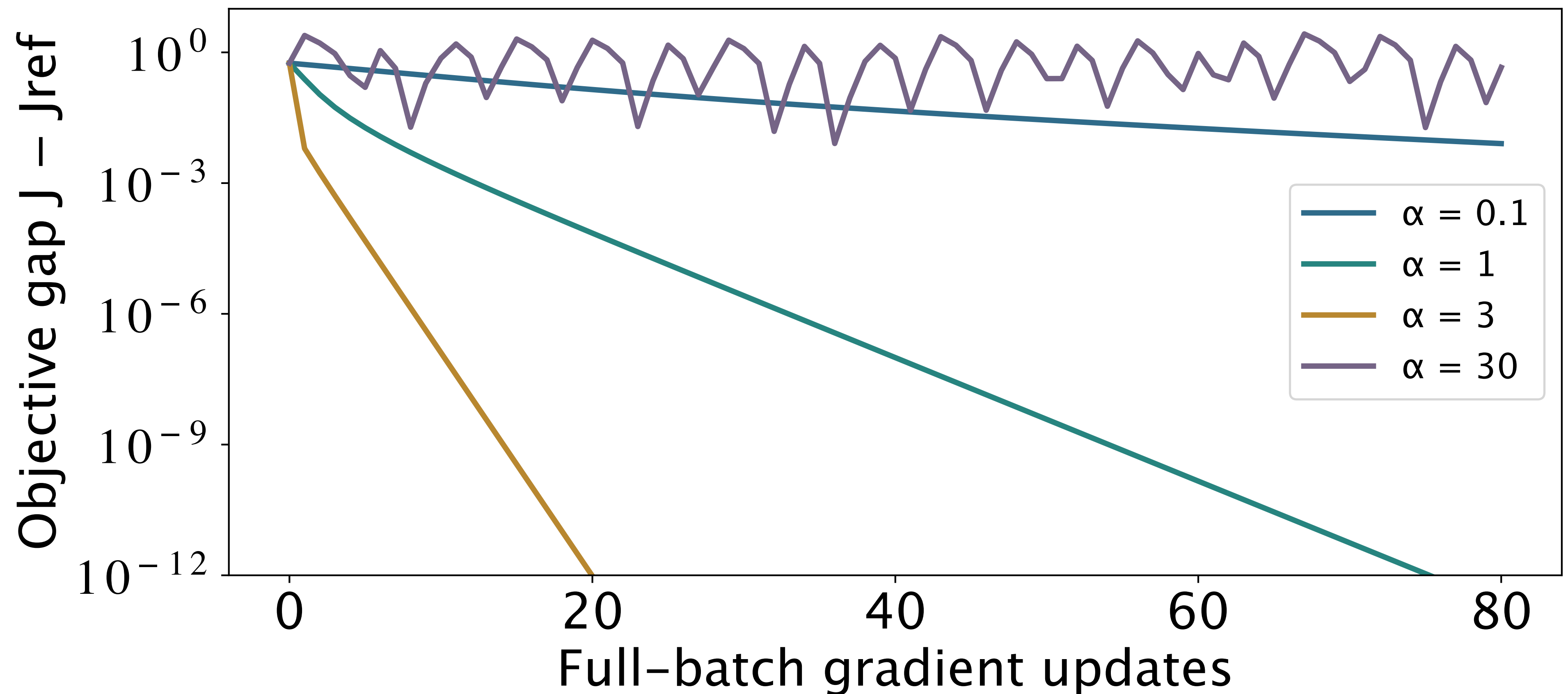
The overlapping labels give a finite minimum without a coefficient penalty.

| A linear loss axis shows **large early improvements**



Small steps are slow; larger useful steps improve quickly; oversized steps can keep oscillating.

| A log gap axis reveals **progress near the minimum**



Equal vertical distances now represent multiplicative changes in $J - J_{\text{ref}}$.

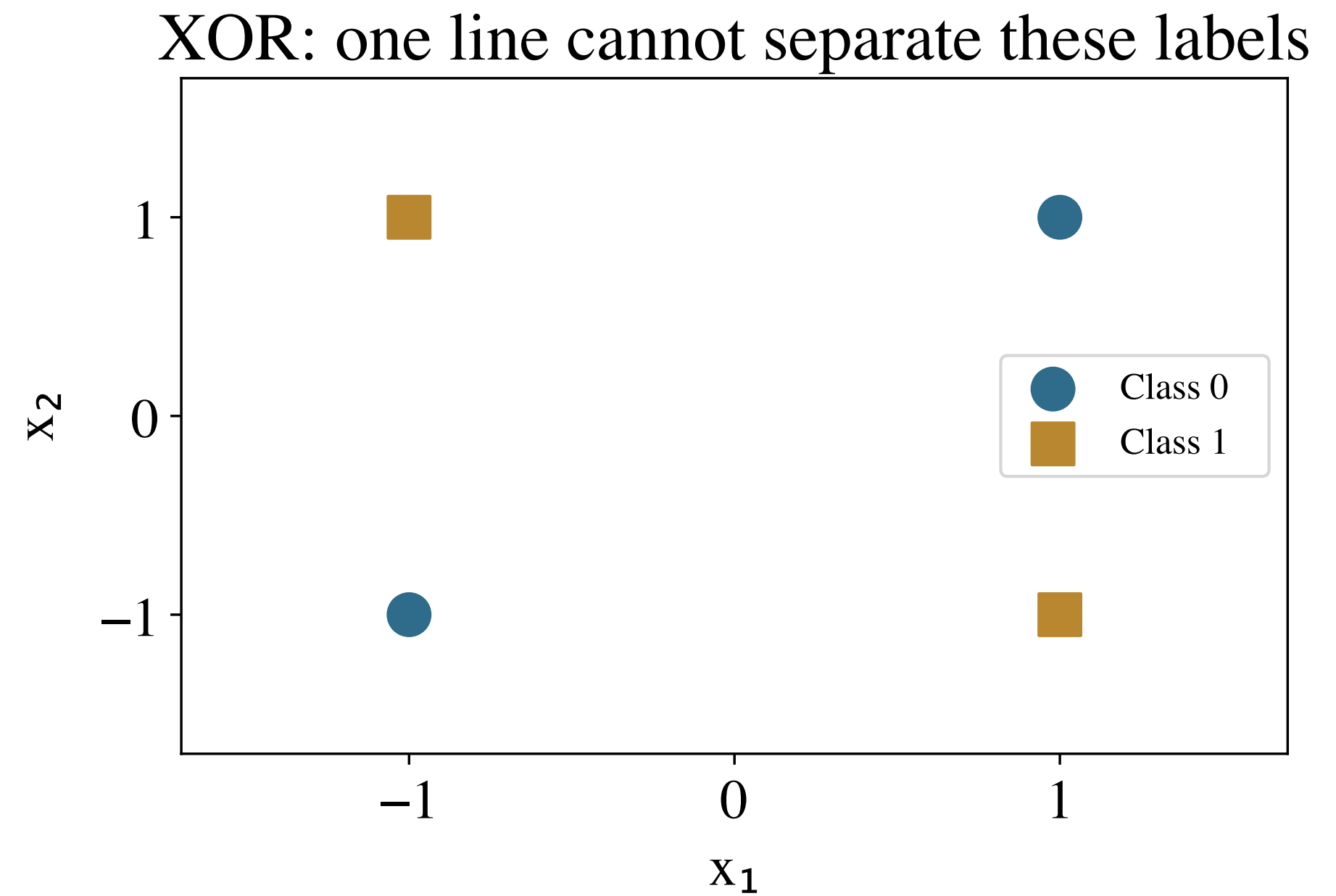
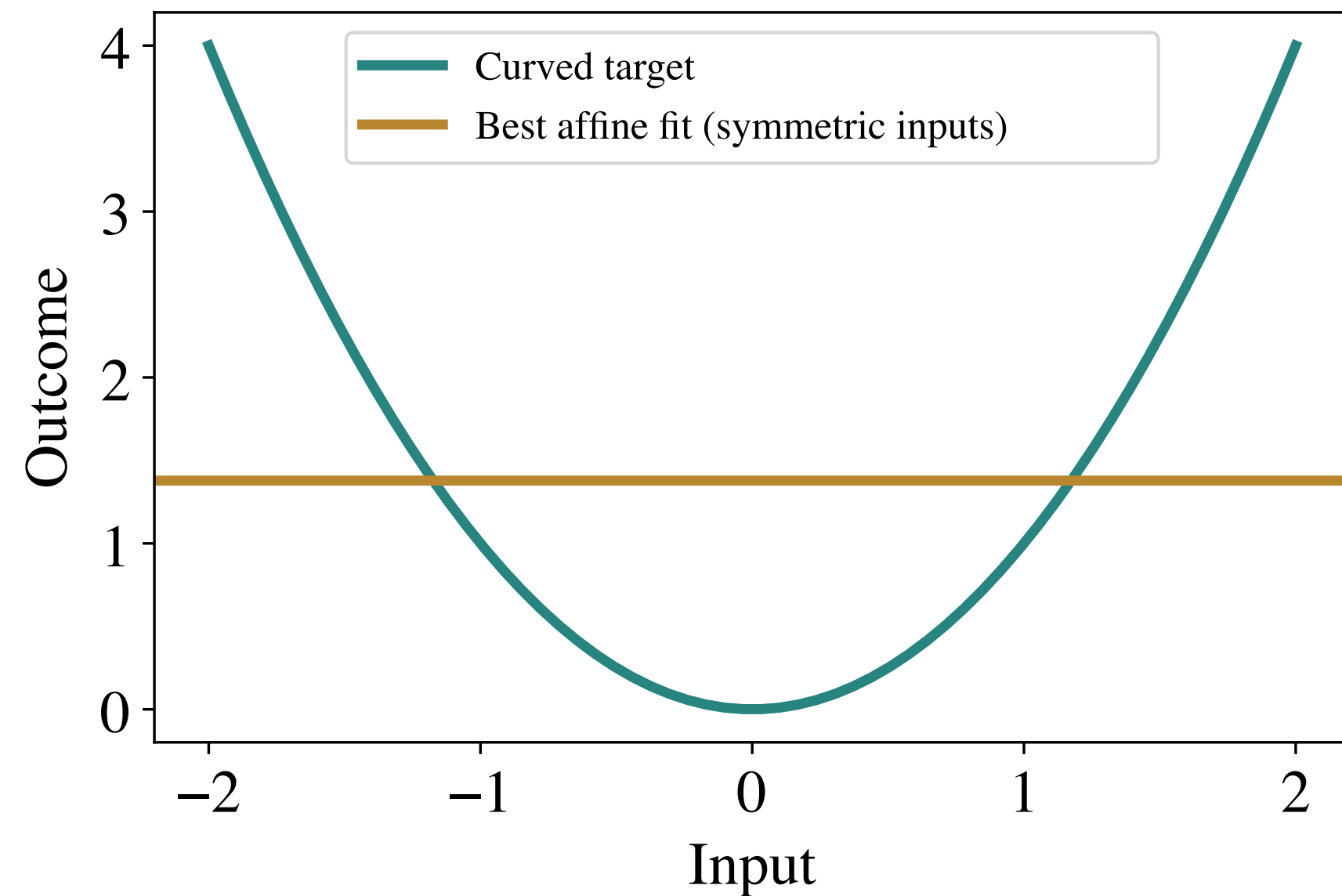
| Convergence means **little useful optimization remains**

Check both a small gradient and small sustained changes in the objective.

- A visually flat trace alone may reflect a tiny learning rate or poor scaling.
- With a finite convex optimum, solving to useful tolerance is a sensible stopping point.
- Optimization progress is measured on training loss; generalization needs held-out evidence.

Near-zero loss does not automatically mean “finished”: separable logistic fits and large deep models need more care.

| Better optimization cannot repair the wrong function class



A curved numerical pattern and XOR labels cannot be represented by one affine predictor.

| The next arc changes **the function we can represent**

We now have: models $\rightarrow$ objectives $\rightarrow$ exact or numerical fitting $\rightarrow$ evaluation.

The remaining limitation: a single affine map cannot express every useful pattern.

Next task: categorize real product images. Can learned nonlinear features improve the predictor?

We will use XOR to explain the mechanism, then return to the real images.