
Fitting Models with Local Information (continued)

ECE 57000 — October 9, 2026

David I. Inouye

| Monday connected **objective shape and practical updates**

- Convexity makes every local minimum global; existence and uniqueness need more.
- Gradients give local information: step size controls whether an update helps.
- Mini-batches trade computation for noise; Adam averages and rescales gradients.

Today: fit logistic regression, diagnose convergence, and separate optimization from generalization.

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.

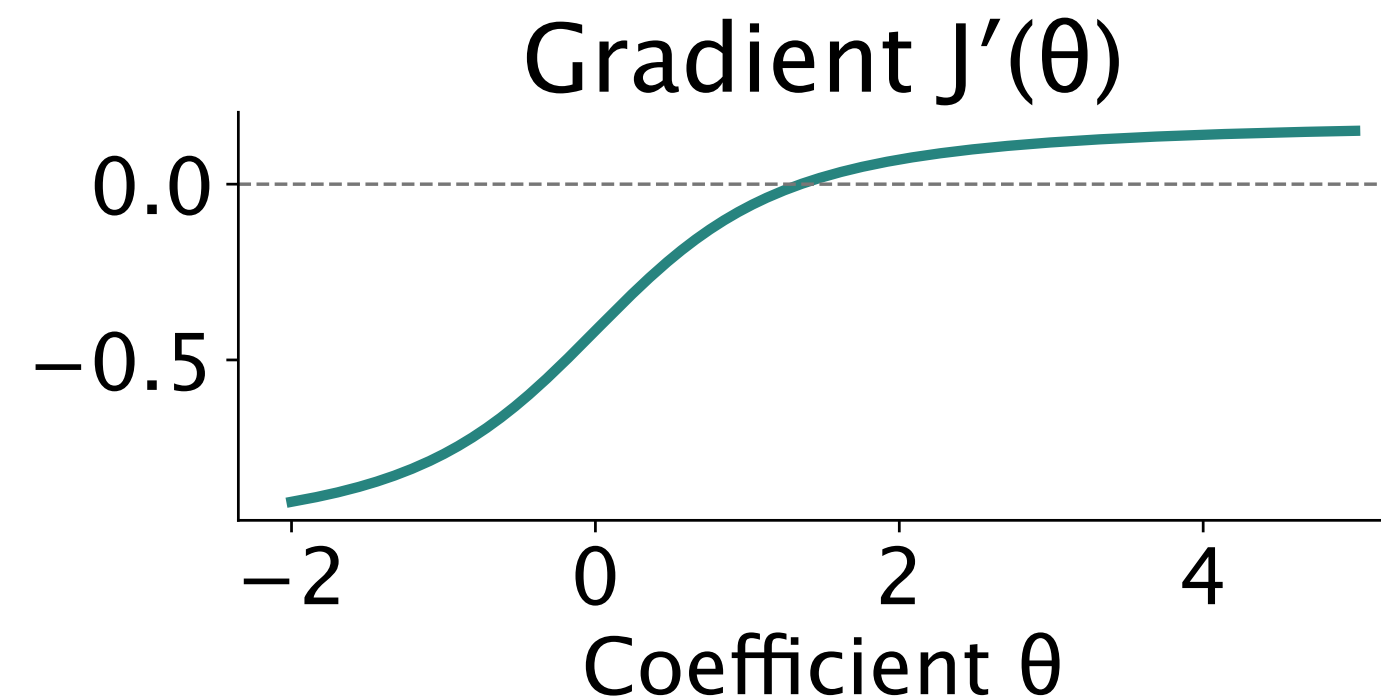
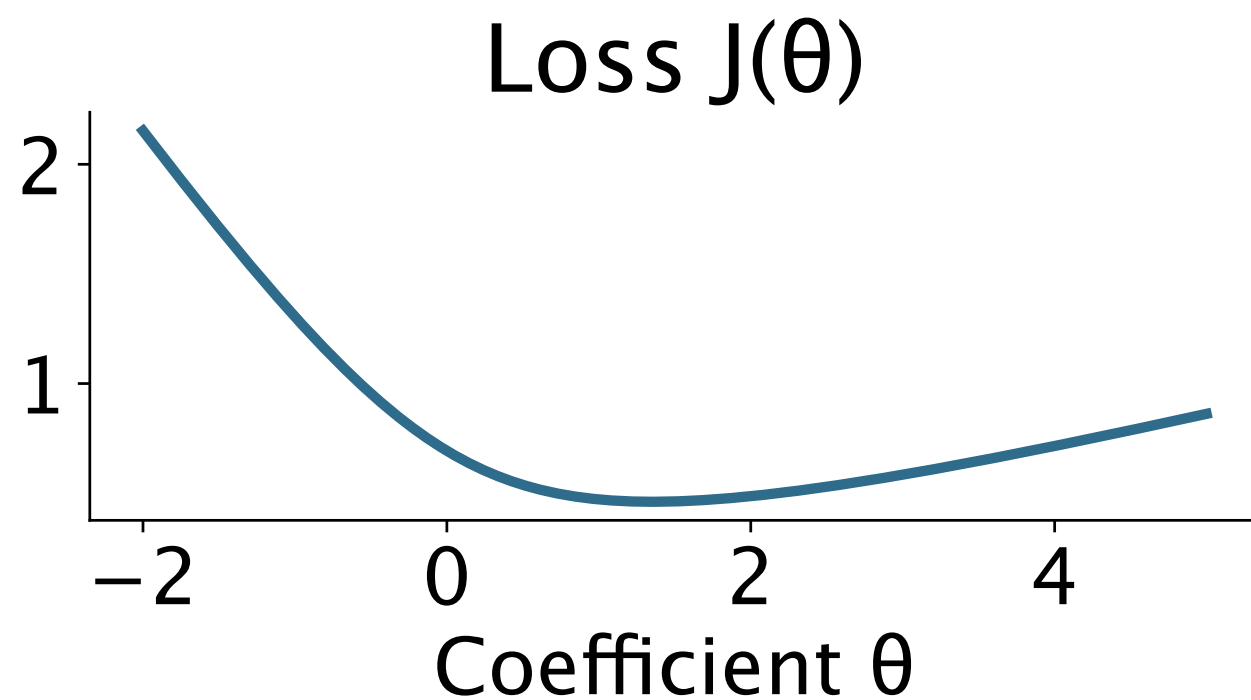
The base learning rate still matters. Adaptation is not a convergence guarantee.

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

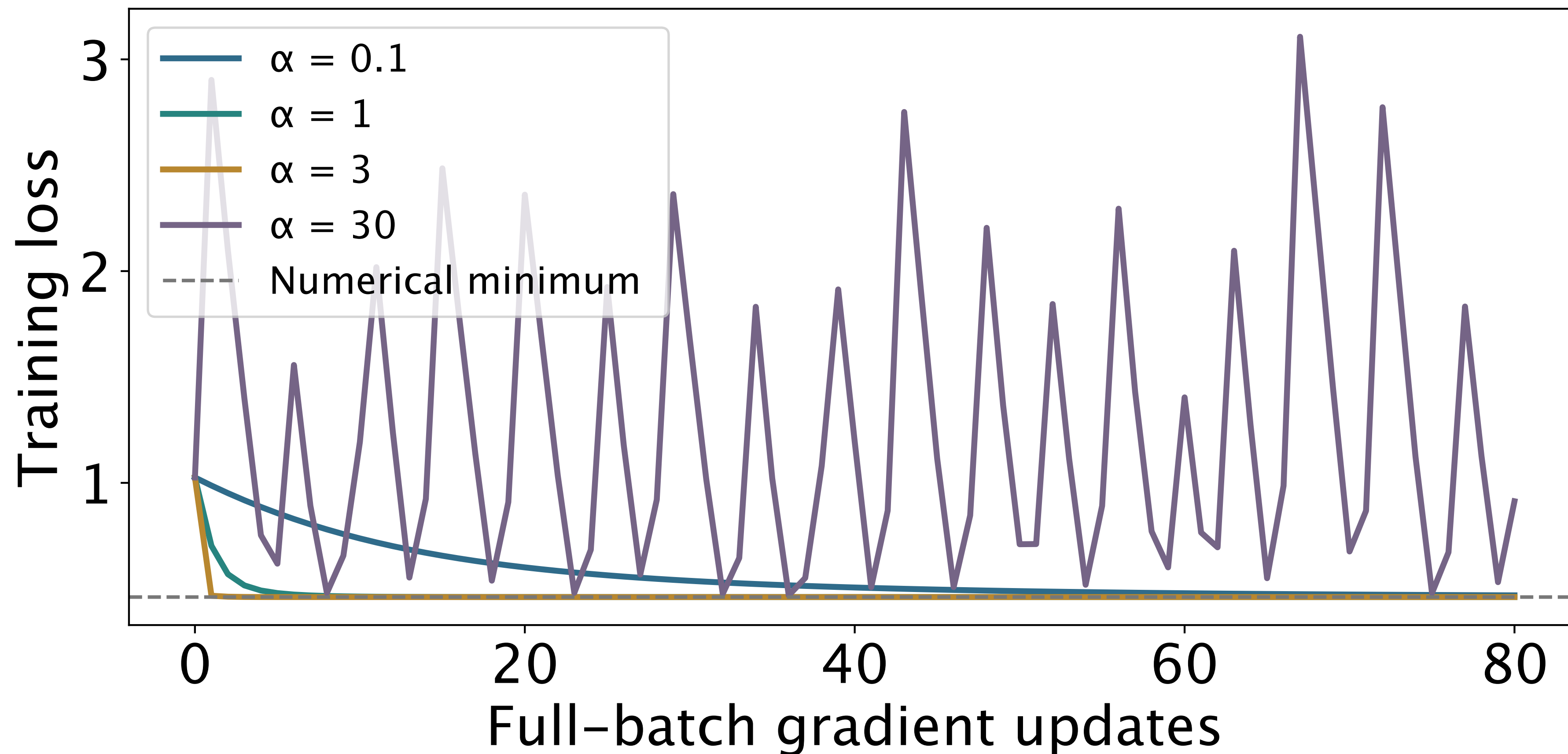
Small illustrative dataset: six inputs $x = (-2, -1, -0.5, 0.5, 1, 2)$ and labels $y = (0, 0, 1, 0, 1, 1)$.

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

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

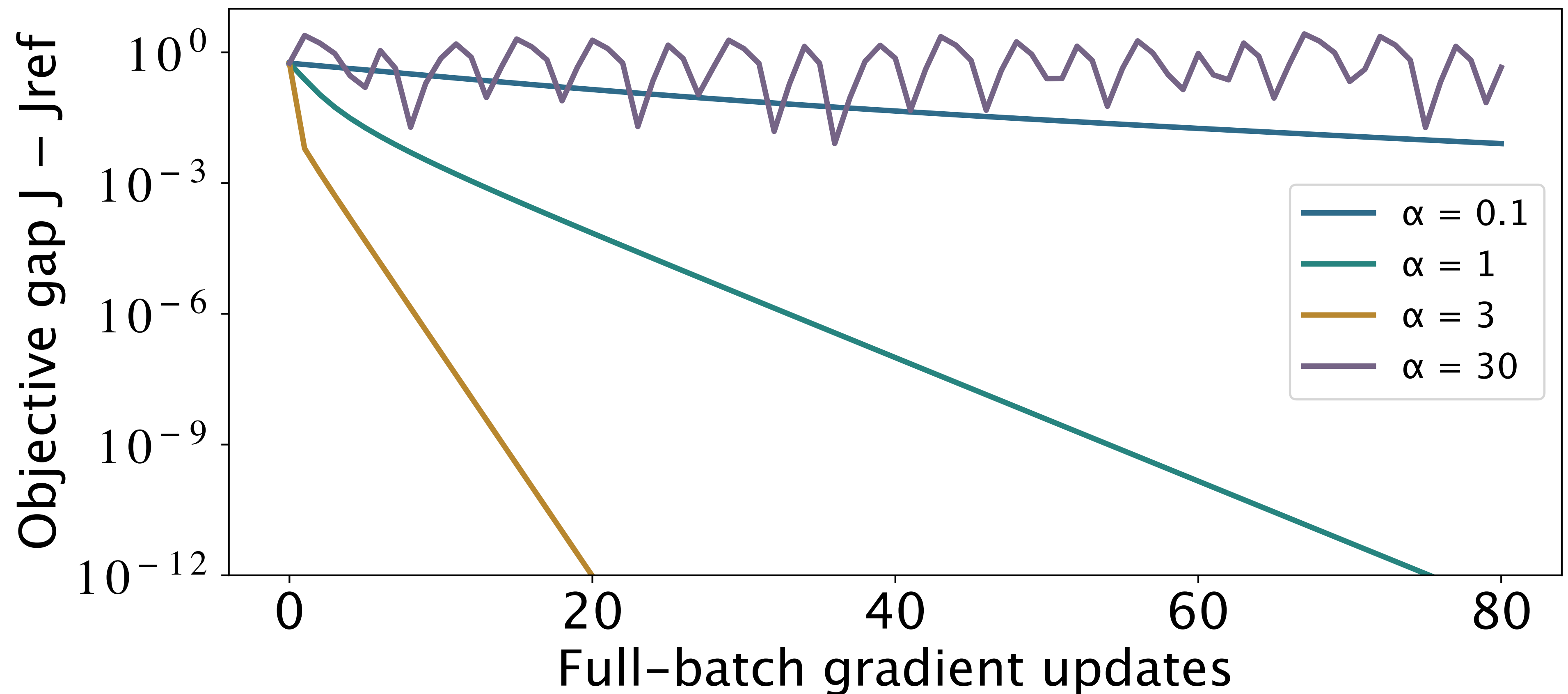
The overlapping labels give a finite minimum without a coefficient penalty.

| 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.

| Does a flat loss trace show **convergence**?

Training loss barely changes over 100 updates, but the gradient norm remains large.

What should you check next?

A. Whether the learning rate is tiny or the features are poorly scaled.

B. Whether a stronger penalty can certify convergence to the same objective.

C. Whether the flat trace alone is enough to stop optimizing.

A. Small updates can hide a substantial remaining descent direction.

Controlling model complexity for generalization

| Better training fit can mean worse prediction on new data

We have learned how to minimize a training objective.

The ML goal is still performance on new examples.

Regularization limits how freely a model can fit the training data, trading some flexibility for potentially better generalization.

Recall KNN: a tiny neighborhood can follow sample-specific fluctuations.

Larger KNN neighborhoods and coefficient penalties both control flexibility

Mechanism	Weaker restriction	Stronger restriction
KNN neighborhood size k	Small k : follow nearby examples closely	Large k : average over more examples
Coefficient penalty strength λ	Small λ : allow larger coefficients	Large λ : discourage large coefficients

For a linear model, one common choice is **L2 regularization**:

$$J_{\lambda}(\boldsymbol{w}, b) = J_{\text{train}}(\boldsymbol{w}, b) + \frac{\lambda}{2} \|\boldsymbol{w}\|^2.$$

Different mechanisms, same purpose: control sensitivity to the training sample.

Too much restriction can hide useful signal. k and λ have no numerical equivalence.

| Both mechanisms can improve **held-out prediction**

Ames KNN using living area only — validation RMSE:

$k = 1$	$k = 100$	$k = 1758$ (all training sales)
\$75,300	\$57,700	\$80,200

Ames linear models using 226 encoded features — test RMSE:

No penalty: OLS	L2 penalty: validation-selected ridge
\$26,300	\$25,600

Compare **within each row**: these use different representations and evaluation splits.
The restriction can help; the strongest restriction need not be best.

| Choose the restriction with validation; **assess the choice on test data**

1. **Fit:** train each candidate using only training observations.
2. **Choose:** compare candidates on validation data.
3. **Assess:** evaluate the selected procedure on untouched test data.

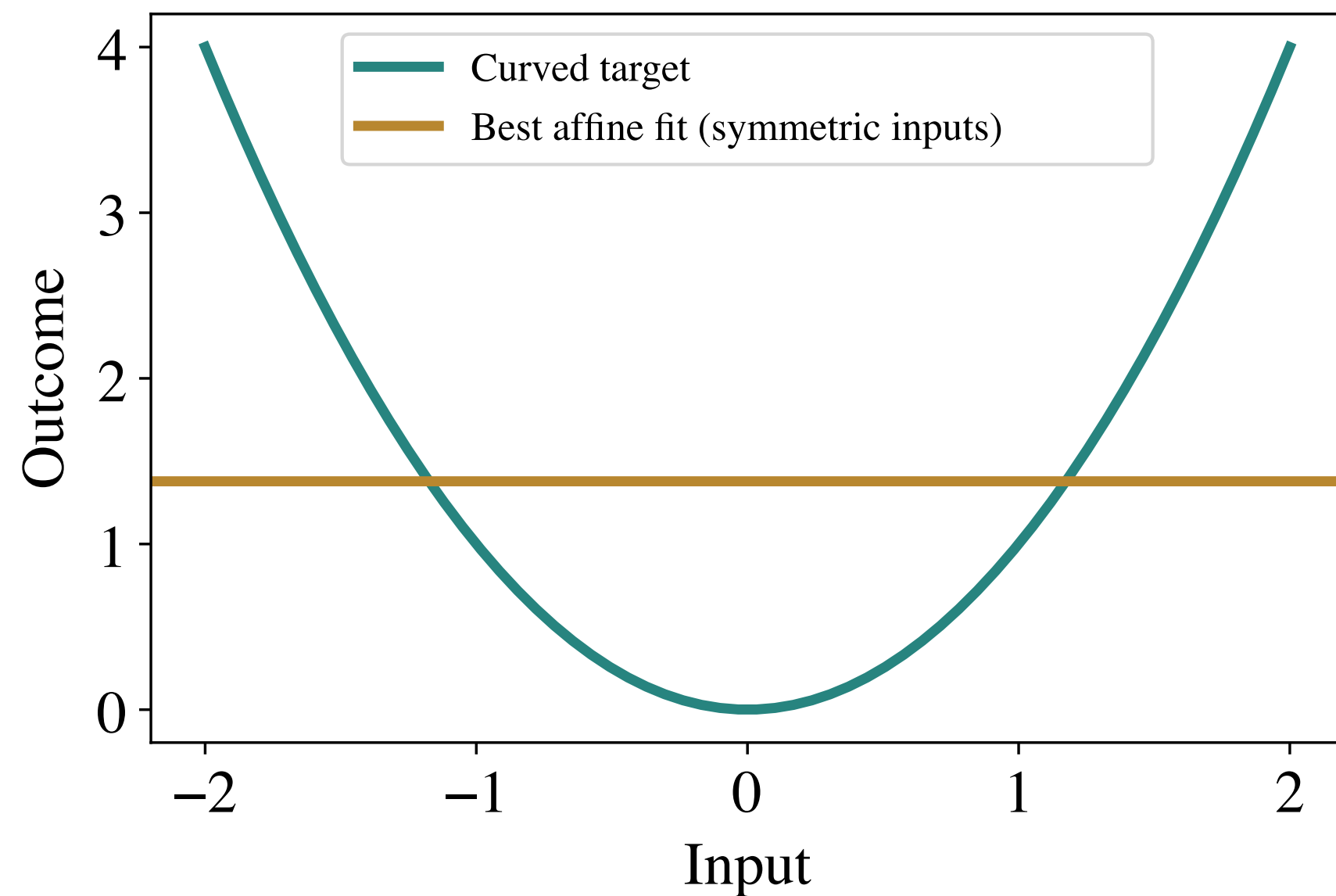
Search positive λ values on a log scale: multiply, rather than add.

Search	Example λ values
Broad search: factors of 10	$10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 100$
Refine near a promising $\lambda = 1$	0.1, 0.3, 1, 3, 10

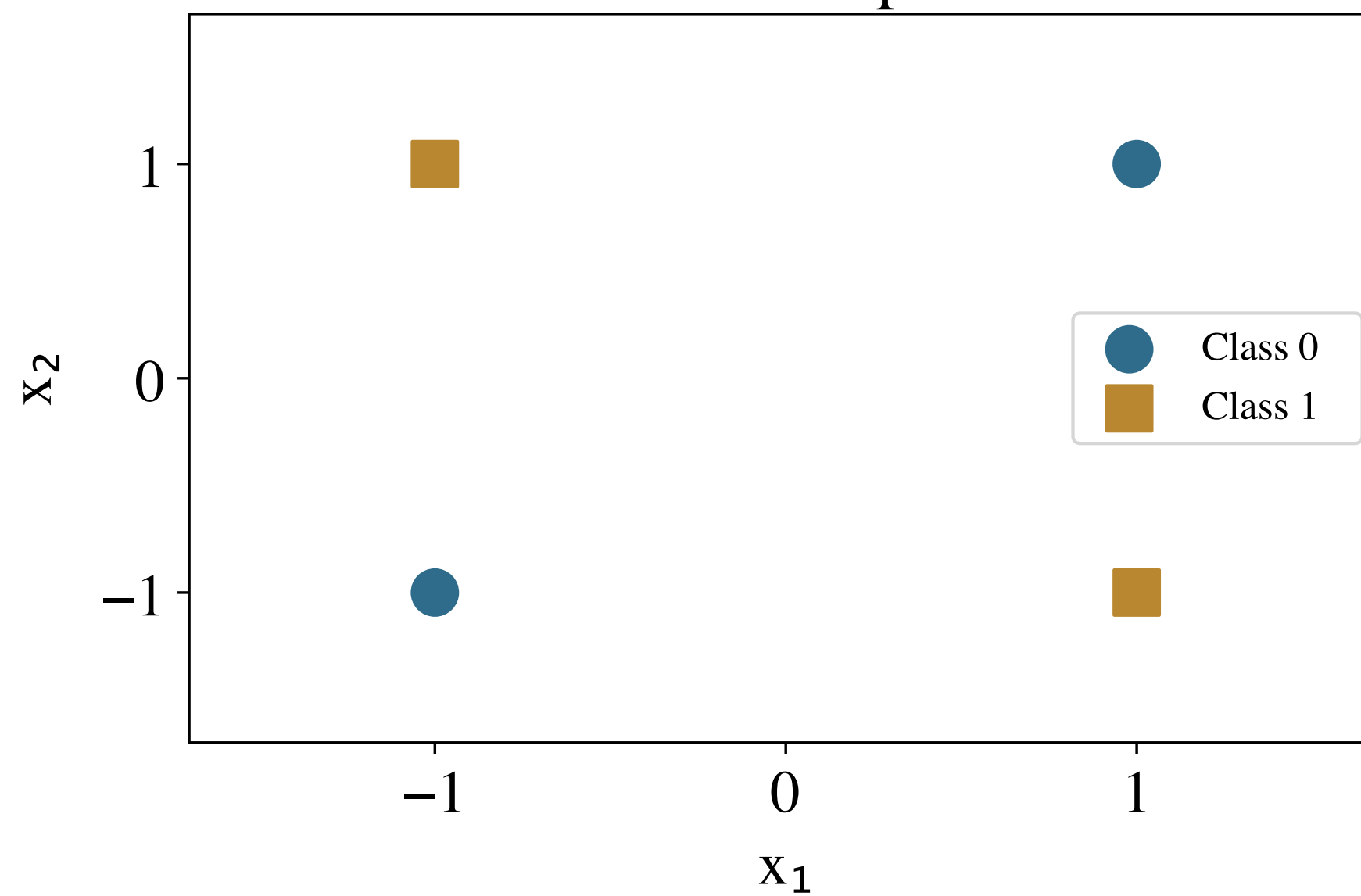
Common mistake: 0, 1, 2, 3, ... misses the entire range $0 < \lambda < 1$.

Include $\lambda = 0$ separately as the **no-penalty baseline**; zero has no logarithm.

| Better optimization cannot repair the wrong function class



XOR: one line cannot separate these labels



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?

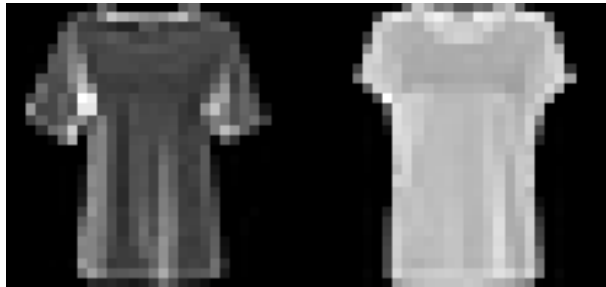
We will use XOR to explain the mechanism, then return to the real images.

Learning nonlinear representations

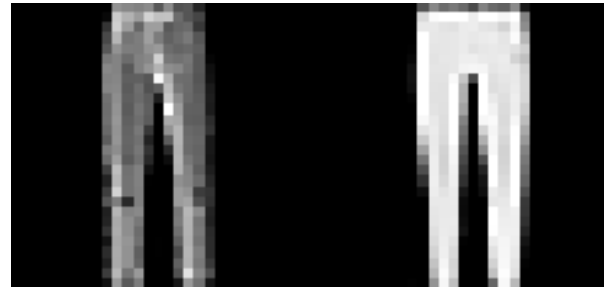
| A product image must become a **useful category**

A catalog task: assign a product image to a category such as shirt, bag, or shoe.

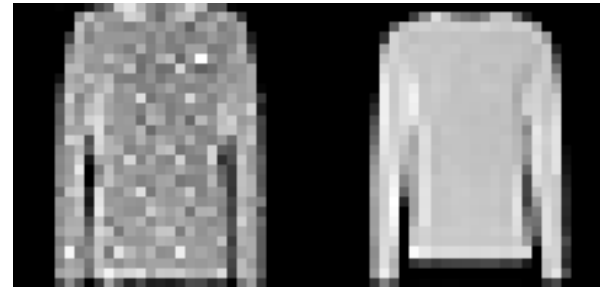
T-shirt/top



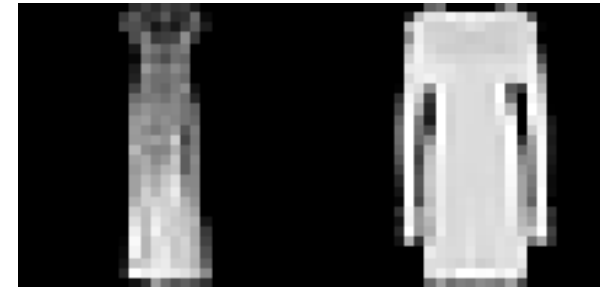
Trouser



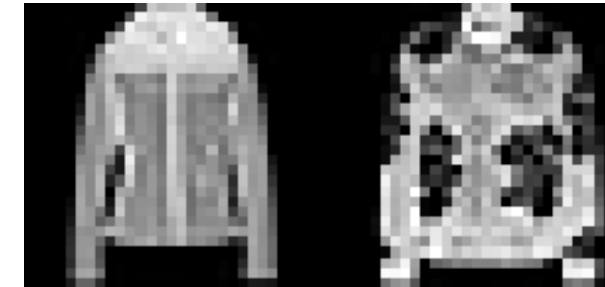
Pullover



Dress



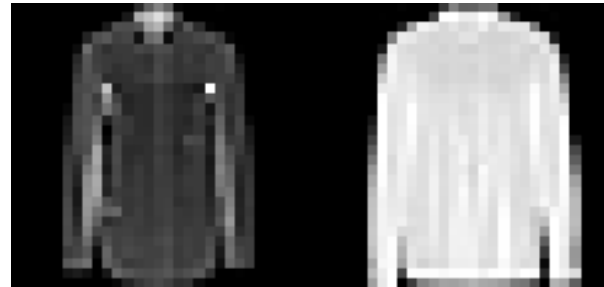
Coat



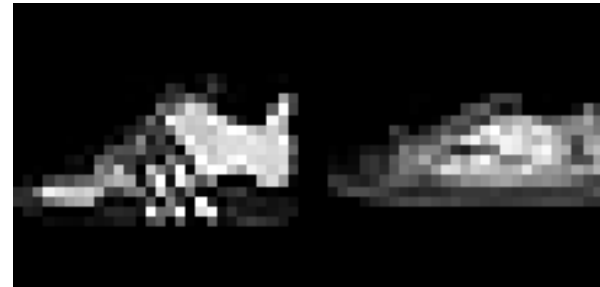
Sandal



Shirt



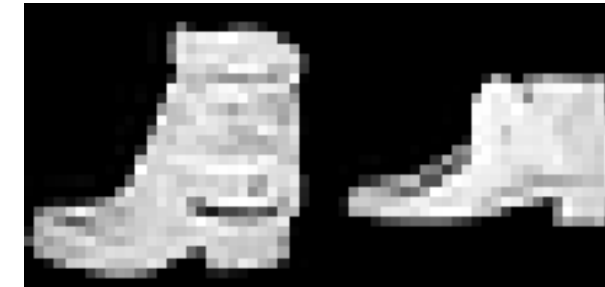
Sneaker



Bag



Ankle boot



Fashion-MNIST: 28×28 grayscale images, **784 pixel features**, ten categories.

Zalando Research, Fashion-MNIST

| Softmax and cross-entropy extend binary logistic regression

For image $x \in [0, 1]^{784}$ and category $c = 1, \dots, 10$:

- 1. Compute an affine score:** $s_c = w_c^\top x + b_c$.
- 2. Apply softmax** to turn the scores into probabilities that sum to one:

$$p_c = \frac{\exp(s_c)}{\sum_{j=1}^{10} \exp(s_j)}.$$

- 3. Use cross-entropy:** if the true category is y , penalize low p_y :

$$\ell(y, p) = -\log p_y, \quad J = \frac{1}{n} \sum_{i=1}^n -\log p_{i, y_i}.$$

Softmax generalizes sigmoid; cross-entropy generalizes binary log loss.

| Regularization helps, but a nonlinear model improves validation accuracy

Same 12,000 training and 3,000 validation images; same pixel features.

Model	Training accuracy	Validation accuracy
Linear, $C = 0.01$ (stronger penalty)	84.66%	82.37%
Linear, $C = 0.1$	88.70%	84.03%
Linear, $C = 1$	92.21%	82.73%
Linear, $C = 10$ (weaker penalty)*	94.90%	80.40%
Nonlinear model	94.10%	85.70%

C is inverse regularization strength. *This fit reached its iteration limit.

How do we go beyond linear functions, and why can that help?

A tiny image task will make the missing interaction explicit.

| An image label can depend on **relationships between regions**

To isolate the mechanism, use a toy image task: flag a mismatch between two regions.

Equal: $y = 0$



x_1

x_2

Different: $y = 1$



x_1

x_2

Different: $y = 1$



x_1

x_2

Equal: $y = 0$



x_1

x_2

Represent their brightness by $x = [x_1, x_2]^T \in \{0, 1\}^2$.

Predict $y = 1$ when the regions differ; otherwise predict $y = 0$.

| A raw affine score gives each pixel **one fixed contribution**

$$z = w_1x_1 + w_2x_2 + b, \quad a = \sigma(z).$$

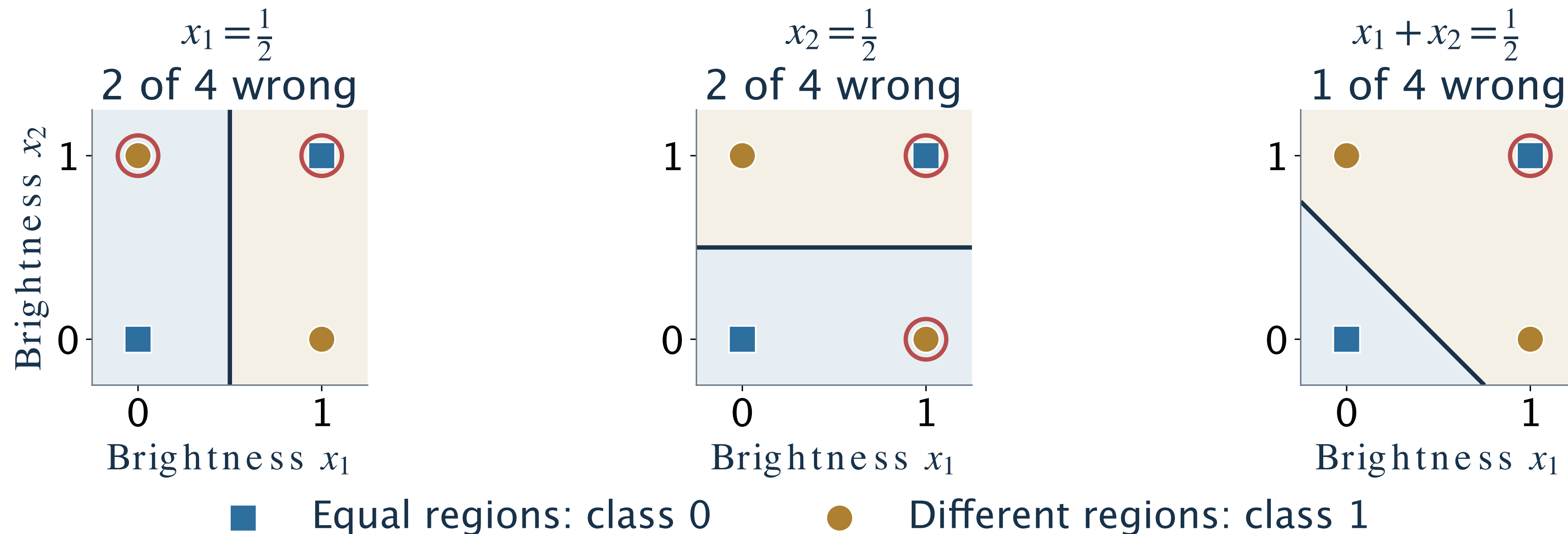
Changing x_1 from 0 to 1 changes the score by w_1 , whatever x_2 is.

Other region	What the mismatch task needs when $x_1 : 0 \rightarrow 1$
$x_2 = 0$	Class changes from 0 to 1
$x_2 = 1$	Class changes from 1 to 0

A monotone sigmoid cannot make the same score increment point both ways.

| XOR cannot be separated by **one straight decision line**

Toy image: two brightness features $x_1, x_2 \in \{0, 1\}$. Label **different regions 1**, equal regions 0.

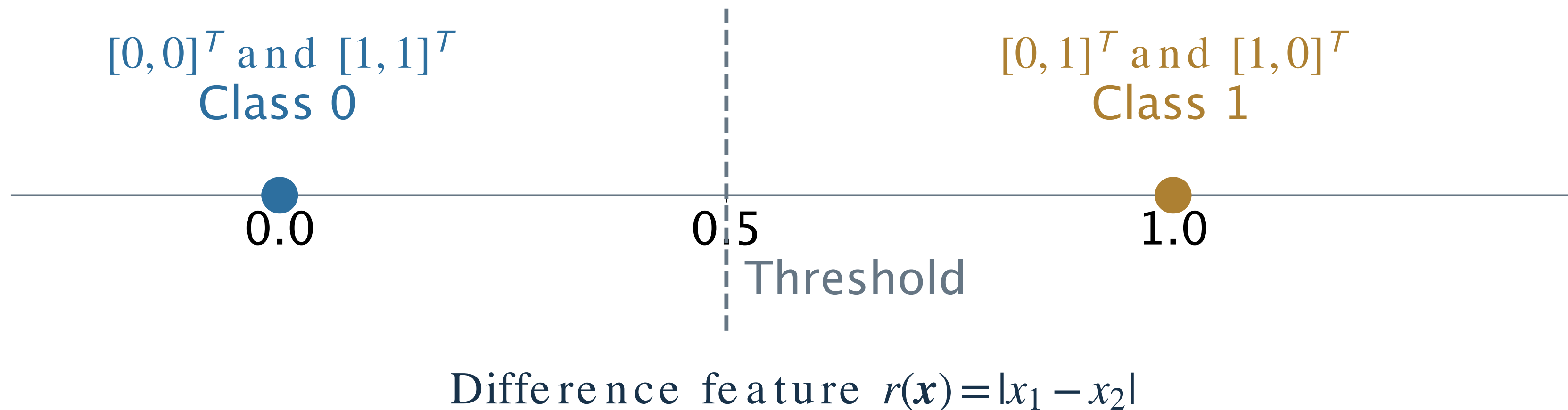


Shading shows predictions; markers show true classes. **Red rings mark errors.**

No single straight decision line separates the two classes.

| A difference feature makes **the same labels easy to separate**

Choose the feature $r(\mathbf{x}) = |x_1 - x_2|$.

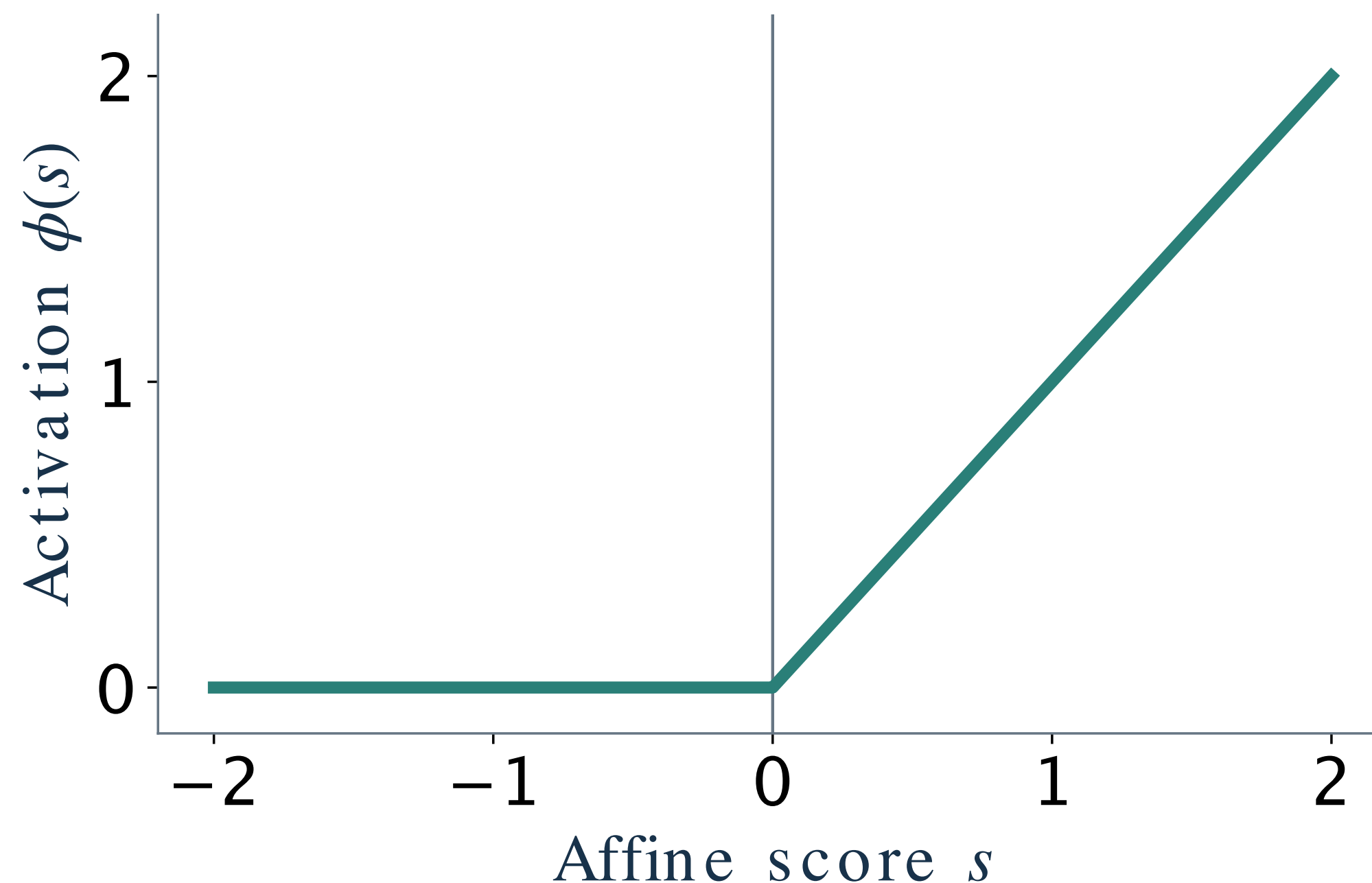


$$\hat{y} = \mathbb{1}\{r(\mathbf{x}) > \frac{1}{2}\}.$$

We changed the **representation**, not the four labels or the fitting algorithm.

| A neuron applies a **nonlinear function** to an affine score

$$s = \boldsymbol{w}^\top \boldsymbol{x} + b, \quad h = \phi(s).$$



Rectified linear unit (ReLU):

$$\phi(s) = \max(0, s).$$

For $\boldsymbol{w} = [1, -1]^\top$ and $b = 0$:

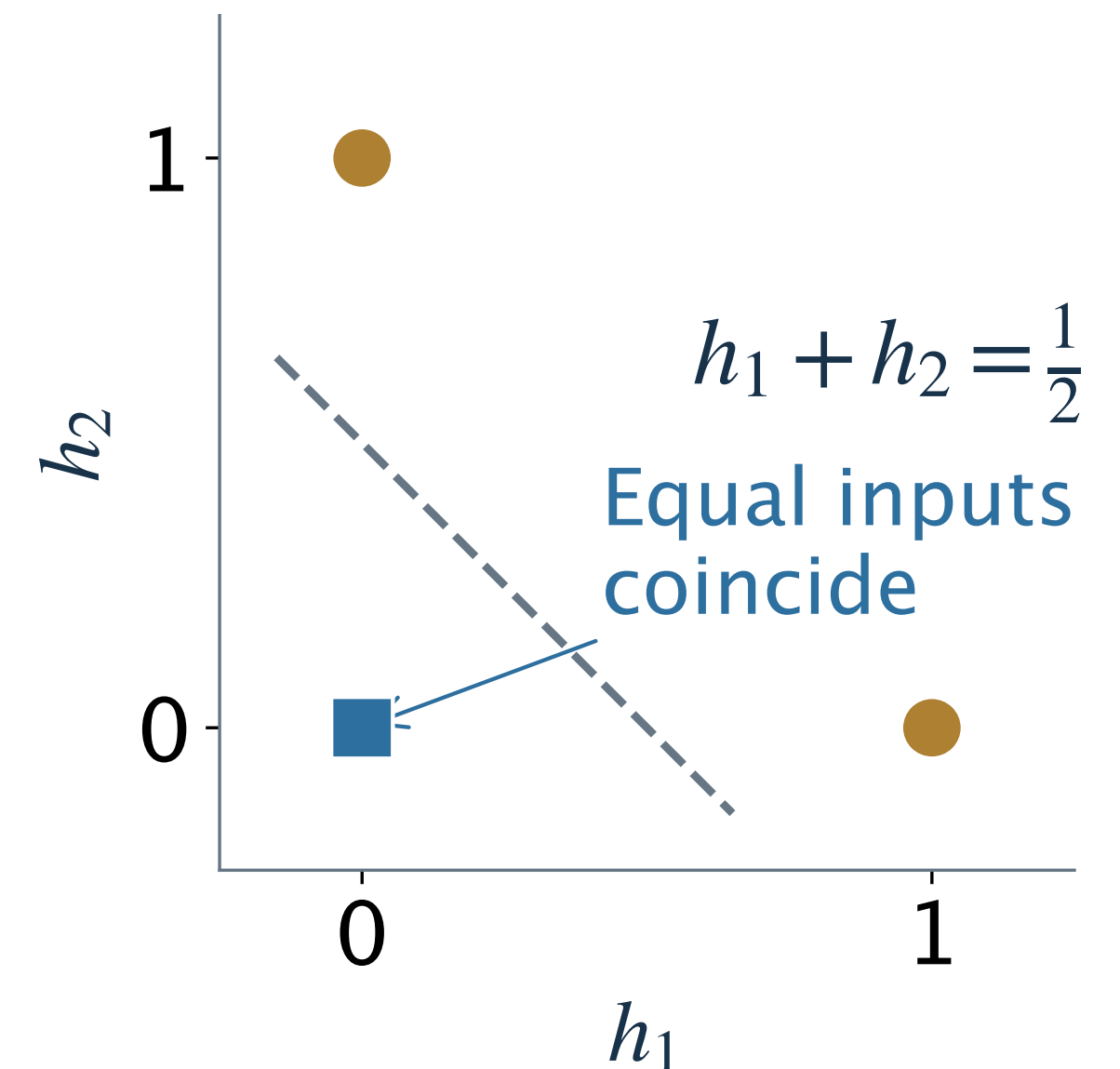
$$h = \max(0, x_1 - x_2).$$

It responds when region 1 is brighter.

| Two directional responses create a useful hidden representation

$$h_1 = \max(0, x_1 - x_2), \quad h_2 = \max(0, x_2 - x_1).$$

Input x	Hidden h	Class y
$[0, 0]^T$	$[0, 0]^T$	0
$[0, 1]^T$	$[0, 1]^T$	1
$[1, 0]^T$	$[1, 0]^T$	1
$[1, 1]^T$	$[0, 0]^T$	0



$$h_1 + h_2 = |x_1 - x_2|, \quad \hat{y} = \mathbb{1}\{h_1 + h_2 > \frac{1}{2}\}.$$

A linear decision in the hidden features solves the nonlinear input-space task.

| A hidden layer collects **several computed features into a vector**

$$h = [h_1, \dots, h_m]^T \in \mathbb{R}^m.$$

Input

x_1

x_2

Hidden representation

$$h_1 = \max(0, x_1 - x_2)$$

$$h_2 = \max(0, x_2 - x_1)$$

Decision

$$h_1 + h_2 > \frac{1}{2}$$

Each hidden unit has its own weights and bias; all see the same input.

The readout combines the resulting features.

| Stack the neuron weights to obtain **one matrix expression**

Each row of W is one neuron's $w_j^\top$:

$$\mathbf{h} = \phi(W\mathbf{x} + \mathbf{b}), \quad W \in \mathbb{R}^{m \times d}, \quad \mathbf{b} \in \mathbb{R}^m.$$

Apply ϕ separately to each feature.

For our two-region example:

$$W = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \mathbf{h} = \phi \left(\begin{bmatrix} x_1 - x_2 \\ x_2 - x_1 \end{bmatrix} \right).$$