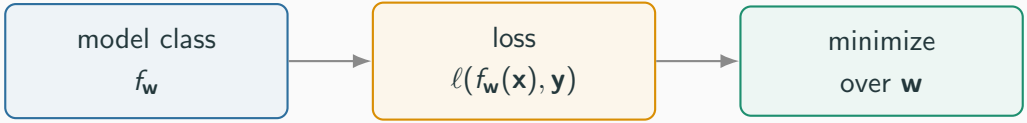
Optimization, Regression, and the Perceptron

How do we actually find the best model in a family?

CSCI 567 — Machine Learning • Lecture 02 • Sep 4, 2026

The template: pick a model, pick a loss, minimize



Linear regression, the Perceptron, logistic regression — three different answers to "which model, which loss." One recipe (gradient descent) solves the "minimize" step for **all three** — even the Perceptron, as we'll see.

This lecture: three models, one optimizer

- Introduces **three** models: linear regression, the Perceptron, logistic regression.
- Gives **one** general-purpose optimizer (gradient descent / SGD) that fits all three — including the Perceptron, whose classic update turns out to be a special case.
- Along the way: convexity, why it matters, and why every loss we build today has it.

this session is a pre-recorded video and fully **self-contained** — every symbol is defined from scratch, even where it echoes Lecture 01; it is deliberately dense, and Lecture 03 has room to revisit anything that feels rushed

Outline

1. Setup: empirical risk minimization
2. Iterative optimization and convexity, in general
3. Linear regression: the MLE view and the closed form
4. Gradient descent and stochastic gradient descent
5. Linear classifiers: why we need a surrogate loss
6. The Perceptron: mistake-driven learning, recovered from SGD
7. Logistic regression: from probabilities to a loss
8. Wrap-up: three models, one template

1. Setup

Empirical risk minimization

The ingredients: sample, model class, loss

training sample

$S = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$

model class

$f_{\mathbf{w}}$, indexed by $\mathbf{w} \in \mathbb{R}^d$

Each input $\mathbf{x}_i \in \mathbb{R}^d$; each label $\mathbf{y}_i$ is real-valued (regression) or a class (classification) — the exact type depends on which model we build. A choice of $\mathbf{w}$ is scored by a **loss** $\ell(f_{\mathbf{w}}(\mathbf{x}), \mathbf{y})$: small when $f_{\mathbf{w}}(\mathbf{x})$ predicts $\mathbf{y}$ well, large when it does not.

Training = empirical risk minimization (ERM)

Averaging the loss over the sample gives the **empirical risk**, and training minimizes it:

$$\hat{\mathcal{R}}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(f_{\mathbf{w}}(\mathbf{x}_i), \mathbf{y}_i), \qquad \mathbf{w}^\star = \arg\min_{\mathbf{w}} \hat{\mathcal{R}}(\mathbf{w})$$

$\hat{\mathcal{R}}$ echoes Lecture 01's true risk $\mathcal{R}(f) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}[\ell(f(\mathbf{x}), \mathbf{y})]$ — the same quantity, averaged over the finite sample S instead of the whole distribution $\mathcal{D}$: the one we can actually compute. For each model below, two questions: **which loss**, and **how do we solve the** arg min — answered three times over.

One simplification: folding the bias into w

Every model below is linear in its parameters, with a bias term: $f_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$.

append a constant 1 to every input: $\mathbf{x} \leftarrow (\mathbf{x}, 1)$, $\mathbf{w} \leftarrow (\mathbf{w}, b)$

Then every model in this lecture is just $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$, with the last coordinate of $\mathbf{w}$ playing the role of b. We drop the tildes and write $\mathbf{x}, \mathbf{w} \in \mathbb{R}^{d+1}$.

2. Iterative optimization and convexity

The two tools every model below will use

Why iterative optimization: no closed form, and search explodes

One model today, linear regression, will turn out to have a **closed-form** solution: pure algebra, no searching. That is the exception — most losses in this course, starting with the very next one we meet, have **no closed form** at all.

The naive alternative — lay a grid over parameter space and evaluate $\hat{\mathcal{R}}$ everywhere — explodes:

> $d + 1$ parameters, 10 candidate values each $\Rightarrow 10^{d+1}$ grid points — at $d + 1 = 20$ that is 10^{20} evaluations, roughly **3,000 years** at one per nanosecond

We need to use the structure of $\hat{\mathcal{R}}$ — its gradient, cheap to evaluate and full of local information — instead of blindly searching.

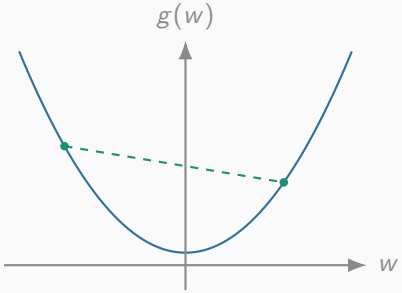
Convexity: the definition

If $\hat{\mathcal{R}}$ has many local minima, an iterative search could get stuck at a bad one — whether a zero-gradient point actually solves the arg min turns on one property.

$g : \mathbb{R}^m \to \mathbb{R}$ is **convex** if,
for all $\mathbf{w}, \mathbf{w}'$ and $t \in [0, 1]$:

$$g(t\mathbf{w} + (1-t)\mathbf{w}') \leq t\, g(\mathbf{w}) + (1-t)\, g(\mathbf{w}')$$

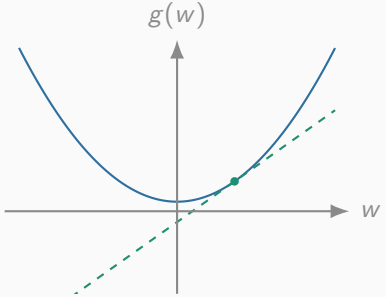
Geometrically: the chord between any two points on the graph lies on or above the graph itself.



Two equivalent characterizations (differentiable g)

first-order: $g(\mathbf{w}') \geq g(\mathbf{w}) + \langle \nabla g(x), y - x \rangle$ for all $\mathbf{w}, \mathbf{w}'$ — the tangent plane at any point lies entirely below g

second-order: the Hessian $\nabla^2 g(\mathbf{w})$ is positive semi-definite (PSD) everywhere



the tangent line sits below the curve everywhere

The payoff: zero gradient means global minimum

if g is convex and $\nabla g(\mathbf{w}^\star) = 0$, then $\mathbf{w}^\star$ is a **global** minimum — not just a local one

From the first-order condition: $g(\mathbf{w}') \geq g(\mathbf{w}^\star) + \langle 0, \mathbf{w}' - \mathbf{w}^\star \rangle = g(\mathbf{w}^\star)$ for every $\mathbf{w}'$. So for a convex $\hat{\mathcal{R}}$, finding *any* zero-gradient point solves ERM exactly, with no risk of a bad local minimum — this is what makes gradient descent trustworthy, and we will check convexity for each loss as we meet it.

Non-convex objectives (e.g. neural networks, Lecture 07) do not offer this guarantee.

3. Linear regression

The MLE view, and the closed form

Model and loss

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}, \qquad \mathbf{y} \in \mathbb{R}$$

squared error: $\ell(f_{\mathbf{w}}(\mathbf{x}), \mathbf{y}) = \frac{1}{2}(f_{\mathbf{w}}(\mathbf{x}) - \mathbf{y})^2$

giving the empirical risk (**mean squared error**, MSE):

$$\hat{\mathcal{R}}(\mathbf{w}) = \frac{1}{2n} \sum_{i=1}^{n} (\mathbf{w}^\top \mathbf{x}_i - \mathbf{y}_i)^2$$

Why the $\frac{1}{2}$, and an alternative loss

The $\frac{1}{2}$ is a convenience: it cancels the 2 that calculus produces when differentiating a square, so the gradient comes out clean. It does not change which $\mathbf{w}$ minimizes $\hat{\mathcal{R}}$.

mean absolute error (MAE), $\ell = |f_{\mathbf{w}}(\mathbf{x}) - \mathbf{y}|$:
more robust to outliers, non-differentiable at 0

Statistical robustness vs. optimization convenience — a running tradeoff. We use MSE here; Lecture 03 returns to robustness.

Why squared error? The maximum-likelihood view

Assume each label is signal plus Gaussian noise:

$$\mathbf{y}_i = \mathbf{w}^\top \mathbf{x}_i + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}(0, \mathrm{Var}[\varepsilon]) \text{ i.i.d.}$$

so conditional on $\mathbf{x}_i$, $\mathbf{y}_i \sim \mathcal{N}(\mathbf{w}^\top \mathbf{x}_i, \mathrm{Var}[\varepsilon])$, with density

$$p(\mathbf{y}_i \mid \mathbf{x}_i, \mathbf{w}) \propto \exp\left(-\frac{(\mathbf{y}_i - \mathbf{w}^\top \mathbf{x}_i)^2}{2\mathrm{Var}[\varepsilon]} \right)$$

MLE: pick $\mathbf{w}$ to maximize the probability the model assigns to the labels we actually observed

Gaussian MLE is exactly least squares

Taking the log of the (independent) sample likelihood and flipping the sign:

$$-\log p(\mathbf{y} \mid X, \mathbf{w}) = \frac{1}{2\mathrm{Var}[\varepsilon]} \sum_{i=1}^{n} (\mathbf{y}_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \text{const}$$

Minimizing this over $\mathbf{w}$ is the same as minimizing $\sum_i (\mathbf{y}_i - \mathbf{w}^\top \mathbf{x}_i)^2$ — the MSE objective, up to constants that don't change the arg min.

MLE under Gaussian noise and least-squares ERM are the same optimization problem.

MSE is convex: check the Hessian

By the second-order condition from Section 2, with $X \in \mathbb{R}^{n \times (d+1)}$ stacking the $\mathbf{x}_i^\top$ as rows:

$$\nabla^2 \hat{\mathcal{R}}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^\top = \frac{1}{n} X^\top X$$

always PSD — for any $\mathbf{v}$:

$$\mathbf{v}^\top (X^\top X) \mathbf{v} = \|X\mathbf{v}\|^2 \geq 0$$

MSE is convex for every dataset — no assumptions on X needed. By Section 2's payoff: the stationary point we find next is the **global** minimum.

The closed form: normal equations

MSE is a quadratic in $\mathbf{w}$, so instead of searching for a zero-gradient point we can solve $\nabla\hat{\mathcal{R}}(\mathbf{w}) = 0$ algebraically:

$$\nabla\hat{\mathcal{R}}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}^\top\mathbf{x}_i - \mathbf{y}_i)\,\mathbf{x}_i = \frac{1}{n}X^\top(X\mathbf{w} - \mathbf{y})$$

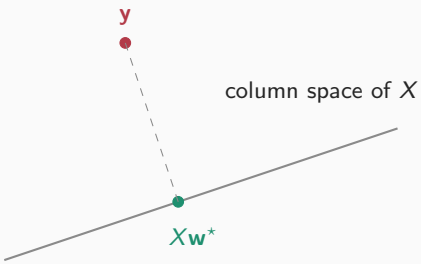
set to zero:

$$X^\top X\mathbf{w} = X^\top\mathbf{y}$$

when $X^\top X$ is invertible:

$$\mathbf{w}^\star = (X^\top X)^{-1}X^\top\mathbf{y}$$

Geometric picture: projection onto the column space



$X\mathbf{w}^\star$ is the **orthogonal projection** of $\mathbf{y}$ onto the column space of X — the closest point, in Euclidean distance, that a linear combination of the features can reach.

The dashed residual $\mathbf{y} - X\mathbf{w}^\star$ is perpendicular to every direction in the column space — exactly what $X^\top(X\mathbf{w} - \mathbf{y}) = 0$ says.

Two caveats about the closed form

rank deficiency: $X^\top X$ is invertible only when X has full column rank ($n \geq d + 1$, no exact linear dependence among features); if $d + 1 > n$, it is *always* rank-deficient and infinitely many $\mathbf{w}$ achieve zero training error — that overparameterized regime is Lecture 04

ill-conditioning: even when technically invertible, near-collinear features make $X^\top X$ close to singular and $\mathbf{w}^\star$ numerically unstable, with huge magnitude — regularization (Lecture 03) is the standard fix

4. Gradient descent and SGD

A general-purpose optimizer

Gradient descent

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \hat{\mathcal{R}}(\mathbf{w})$$

$\nabla \hat{\mathcal{R}}(\mathbf{w})$ points toward steepest *increase* — step in the opposite direction. $\eta > 0$ is the **learning rate** (step size). Since the update only needs $\hat{\mathcal{R}}$ to be differentiable, it carries over unchanged to every loss in this lecture.

Why a step decreases $\hat{\mathcal{R}}$ (to first order)

First-order Taylor expansion around $\mathbf{w}^{(t)}$:

$$\hat{\mathcal{R}}(\mathbf{w}) \approx \hat{\mathcal{R}}(\mathbf{w}^{(t)}) + \left\langle \nabla\hat{\mathcal{R}}(\mathbf{w}^{(t)}), \mathbf{w} - \mathbf{w}^{(t)} \right\rangle$$

plug in the GD step:

$$\hat{\mathcal{R}}(\mathbf{w}^{(t+1)}) \approx \hat{\mathcal{R}}(\mathbf{w}^{(t)}) - \eta \left\| \nabla\hat{\mathcal{R}}(\mathbf{w}^{(t)}) \right\|^2 \leq \hat{\mathcal{R}}(\mathbf{w}^{(t)})$$

A squared norm is never negative and $\eta > 0$, so the step provably decreases $\hat{\mathcal{R}}$ **to first order**.

if η is too large, the linear approximation itself breaks down — a step can *increase* $\hat{\mathcal{R}}$ instead: the "overshoot" failure mode

Step size, convergence, and what iterating buys

- Each step decreases $\hat{\mathcal{R}}$, for η small enough (just shown).
- Iterating drives $\mathbf{w}$ toward a stationary point, $\nabla\hat{\mathcal{R}}(\mathbf{w}) = 0$.
- Too large η: overshoots, can diverge. Too small: wastes iterations.

by Section 2's convexity argument: for convex $\hat{\mathcal{R}}$, that stationary point **is** the global minimum

Formal convergence rates (how many iterations T to guarantee $\hat{\mathcal{R}}(\mathbf{w}^{(T)}) - \hat{\mathcal{R}}(\mathbf{w}^\star) \leq \epsilon$) are a core topic of convex optimization, beyond this course — the qualitative point: more iterations buys more accuracy, provably, whenever $\hat{\mathcal{R}}$ is convex.

Worked example: two GD steps by hand

$f(x) = wx$ (no bias), three points $(x_i, y_i) \in \{(1,2),(2,4),(4,8)\}$, starting $w_0 = 0$, $\eta = 0.05$; the MSE gradient in one dimension is $\nabla\hat{\mathcal{R}}(w) = \frac{1}{3}\sum_i x_i(wx_i - y_i)$.

step 1: $\quad \nabla\hat{\mathcal{R}}(0) = \frac{1}{3}\big[1(-2) + 2(-4) + 4(-8)\big] = -14 \;\Rightarrow\; w_1 = 0 - 0.05(-14) = 0.7$

step 2: $\quad \nabla\hat{\mathcal{R}}(0.7) = \frac{1}{3}\big[1(-1.3) + 2(-2.6) + 4(-5.2)\big] = -9.1 \Rightarrow w_2 = 0.7 - 0.05(-9.1) = 1.155$

two steps move w from 0 to 1.155, heading toward the data's true slope of 2 — run in the Colab notebook, it converges fully in a couple hundred iterations, matching the closed form

Stochastic gradient descent (SGD)

Each full GD step needs $\nabla\hat{\mathcal{R}}(\mathbf{w}) = \frac{1}{n}\sum_i \nabla\ell(f_{\mathbf{w}}(\mathbf{x}_i), \mathbf{y}_i)$ — a sum over *every* training example: with n in the millions, one step is a full pass over the data, expensive and wasteful if most examples carry similar gradient information.

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\, \nabla\ell(f_{\mathbf{w}}(\mathbf{x}_i), \mathbf{y}_i), \qquad i \sim \text{Unif}\{1, \ldots, n\}$$

Replace the full gradient with the gradient of a single, randomly chosen example.

SGD's step is unbiased in expectation

Since i is drawn uniformly:

$$\mathbb{E}_i\big[\nabla \ell(f_{\mathbf{w}}(\mathbf{x}_i), \mathbf{y}_i)\big] = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(f_{\mathbf{w}}(\mathbf{x}_i), \mathbf{y}_i) = \nabla \hat{\mathcal{R}}(\mathbf{w})$$

Each SGD step is an **unbiased estimate** of the true gradient direction, even though any single step is noisier than a full GD step. Many cheap, noisy steps often reach a good $\mathbf{w}$ faster in wall-clock time than few, expensive, exact ones.

Mini-batches: average the gradient over, say, 32 or 256 examples — trades some noise for steps that parallelize well on a GPU. GD's and SGD's convergence are compared directly on the same dataset in the Colab notebook.

5. Linear classifiers

Why we need a surrogate loss

Classification setup, and the margin

Now the label is a class, not a real number. Binary case: $\mathbf{y} \in \{-1, +1\}$ (logistic regression will use an equivalent $\{0, 1\}$ coding later), predicting $\hat{\mathbf{y}} = \text{sign}(\mathbf{w}^\top \mathbf{x})$ — which still makes sense when the data is only *almost* linearly separable.

the **margin**:

$$z_i = \mathbf{y}_i \left(\mathbf{w}^\top \mathbf{x}_i\right)$$

Positive exactly when $\mathbf{x}_i$ is correctly classified — and larger in magnitude the more confidently correct (or wrong) the prediction is. We use z throughout this section.

The natural loss is the 0-1 loss — and it gives GD nothing

$\ell_{01}(f_{\mathbf{w}}(\mathbf{x}), \mathbf{y}) = \mathbb{1}[z \leq 0]$ makes $\hat{\mathcal{R}}(\mathbf{w})$ exactly the **misclassification rate**.

Flat everywhere except a jump at $z = 0$: zero (or undefined) gradient gives GD no signal — the flat region looks identical whether $\mathbf{w}$ is a thousand mistakes away or one nudge away.

Even offline, minimizing 0-1 loss is computationally hard

Setting optimization aside entirely: finding the linear classifier with the *fewest* misclassifications on a fixed dataset is intractable in the worst case.

for halfspaces, exactly maximizing agreement with the labels cannot even be well *approximated* in polynomial time unless $\mathrm{P} = \mathrm{NP}$ (Ben-David et al., 2003)

The fix: a convex surrogate, starting with the Perceptron loss

Replace ℓ_{01} with a **convex surrogate loss** $\ell(z)$: convex (so GD/SGD apply), large when z is very negative (penalizing confident mistakes).

$$\ell_{\text{perc}}(z) = \max(0, -z)$$

Zero once classified correctly with *any* positive margin; grows linearly with $|z|$ on a mistake. A hinge at zero, closely related to the **hinge loss** SVMs use (Lecture 05).

Loss is convex (prove!).

Subgradients: convexity without differentiability

ℓ_{perc} has a kink at $z = 0$ — not differentiable there.

a **subgradient**: any slope s with $\ell(z') \geq \ell(z) + s(z' - z)$ for all z' — the first-order condition, applied where the ordinary derivative doesn't exist

per example: $$\partial_{\mathbf{w}} \ell_{\text{perc}}(f_{\mathbf{w}}(\mathbf{x}), \mathbf{y}) = \begin{cases} -\mathbf{y}\mathbf{x} & \mathbf{y}(\mathbf{w}^\top \mathbf{x}) < 0 \\ 0 & \mathbf{y}(\mathbf{w}^\top \mathbf{x}) > 0 \end{cases}$$

(misclassified vs. correct). At $z = 0$, any $s \in [-1, 0]$ is a valid subgradient — enough for SGD to make sense on a loss that isn't smooth.

SGD with $\eta = 1$ recovers the Perceptron algorithm

No closed form here — but Section 4's solver applies unchanged to any convex loss with a subgradient. Take an SGD step with $\eta = 1$:

$$\mathbf{w} \leftarrow \mathbf{w} - 1 \cdot \partial_{\mathbf{w}} \ell_{\text{perc}} = \begin{cases} \mathbf{w} + \mathbf{y}_i \mathbf{x}_i & \text{misclassified} \\ \mathbf{w} & \text{correct} \end{cases}$$

this is exactly Frank Rosenblatt's 1958 mistake-driven update — recovered here as SGD on a specific convex loss, decades before that connection was made explicit

6. The Perceptron

The mistake-driven view

Rosenblatt's original, operational view

Section 5 derived the algorithm from first principles (SGD, $\eta = 1$, Perceptron loss). Rosenblatt's own 1958 presentation, decades before that loss-based view existed, was purely **operational** and needs no loss function at all:

> look at one training example at a time, and only update **w** when the current model gets that example **wrong**

Labels are coded $\mathbf{y} \in \{-1, +1\}$ (not $\{0, 1\}$ as in logistic regression) — this ± 1 coding is what makes the update rule so compact. The model predicts $\hat{\mathbf{y}} = \text{sign}(\mathbf{w}^\top \mathbf{x})$.

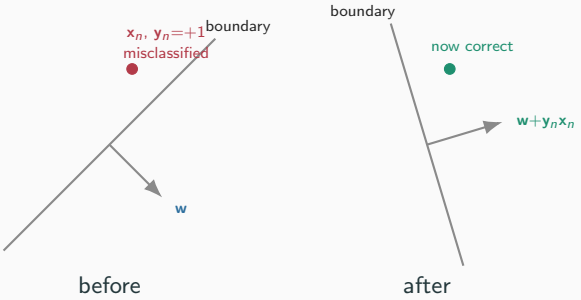
The Perceptron algorithm

✅ $y_n(\mathbf{w}^\top \mathbf{x}_n) > 0$: correctly classified — do nothing

❌ $y_n(\mathbf{w}^\top \mathbf{x}_n) \leq 0$: misclassified (or on the boundary) — update $\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$

Sweep through the training set repeatedly (each full pass is an **epoch**), applying this rule example by example, until an entire epoch produces zero mistakes.

Why the update makes sense, geometrically



Case $y_n = +1$ but $\mathbf{w}^\top \mathbf{x}_n \leq 0$: adding $\mathbf{x}_n$ to $\mathbf{w}$ increases $\mathbf{w}^\top \mathbf{x}_n$ by $\|\mathbf{x}_n\|^2 > 0$ — the boundary rotates toward correctly classifying this point.

Case $y_n = -1$ but $\mathbf{w}^\top \mathbf{x}_n \geq 0$: adding $y_n \mathbf{x}_n = -\mathbf{x}_n$ decreases $\mathbf{w}^\top \mathbf{x}_n$, symmetrically.

Either way: a nudge in exactly the direction that would have fixed this one mistake — geometrically confirming what Section 5 showed algebraically.

Worked example: convergence in two mistakes

Two points (augmented, last coordinate $= 1$ for the bias): $\mathbf{x}_1 = (1, 1, 1)$, $\mathbf{y}_1 = +1$; $\mathbf{x}_2 = (0, -1, 1)$, $\mathbf{y}_2 = -1$; start at $\mathbf{w}_0 = (0, 0, 0)$.

check $\mathbf{x}_1$: $\mathbf{w}_0^\top \mathbf{x}_1 = 0$, $\mathbf{y}_1 \cdot 0 \le 0 \Rightarrow$ mistake: $\mathbf{w}_1 = \mathbf{w}_0 + \mathbf{y}_1 \mathbf{x}_1 = (1, 1, 1)$

check $\mathbf{x}_2$: $\mathbf{w}_1^\top \mathbf{x}_2 = 0$, $\mathbf{y}_2 \cdot 0 \le 0 \Rightarrow$ mistake: $\mathbf{w}_2 = \mathbf{w}_1 + \mathbf{y}_2 \mathbf{x}_2 = (1, 2, 0)$

recheck: $\mathbf{w}_2^\top \mathbf{x}_1 = 3 > 0$ ✓ $\mathbf{w}_2^\top \mathbf{x}_2 = -2$, $\mathbf{y}_2 \cdot (-2) = 2 > 0$ ✓

an entire pass with zero mistakes: converged, after exactly 2 mistakes, to $\mathbf{w} = (1, 2, 0)$ — weight vector $(1, 2)$, bias 0

Does this always terminate?

not for arbitrary data — if no linear separator exists, the Perceptron cycles forever, always finding some misclassified point to correct

But when a separator *does* exist, Rosenblatt's algorithm provably finds one in a bounded number of mistakes — proved rigorously by Novikoff (1962).

Novikoff's convergence theorem

Definition: the data is linearly separable with margin $\gamma > 0$ if there exists a unit vector $\mathbf{u}$ ($\|\mathbf{u}\| = 1$) such that $\mathbf{y}_n \langle \mathbf{u}, \mathbf{x}_n \rangle \geq \gamma$ for every training example n

Theorem (Novikoff, 1962). Suppose the training data is linearly separable with margin γ, and $\|\mathbf{x}_n\| \leq R$ for every n. Then the Perceptron makes at most $(R/\gamma)^2$ mistakes before converging to a separator.

Reading the bound

- Does **not** depend on n (the number of training examples) at all — only on the geometry of the data.
- R: how spread out the points are. γ: how comfortably they are separated.
- A worst-case bound on *total* mistakes across all epochs, not per epoch.

Sanity check: the bound on the worked example

Take $\mathbf{u} = \mathbf{w}_2 / \|\mathbf{w}_2\|$ with $\mathbf{w}_2 = (1, 2, 0)$: $\|\mathbf{w}_2\| = \sqrt{5} \approx 2.236$, $\mathbf{u} \approx (0.447, 0.894, 0)$, giving margins

$$\mathbf{y}_1 \langle \mathbf{u}, \mathbf{x}_1 \rangle \approx 1.341, \qquad \mathbf{y}_2 \langle \mathbf{u}, \mathbf{x}_2 \rangle \approx 0.894$$

so this $\mathbf{u}$ witnesses separability with $\gamma \approx 0.894$ (any valid witness gives a true bound, not only the largest possible margin), and $R = \max(\sqrt{3}, \sqrt{2}) \approx 1.732$:

$$(R/\gamma)^2 \approx (1.732/0.894)^2 \approx 3.75$$

The theorem predicts at most 3.75 mistakes; the algorithm actually made 2 — consistent, as guaranteed: the bound is an **upper** bound, not an exact count.

7. Logistic regression

From probabilities to a loss

A third route: predict a probability

The Perceptron predicts the sign of $\mathbf{w}^\top \mathbf{x}$ directly, fit by mistake-driven updates. **Logistic regression** predicts a *probability* instead — spam vs. not spam, $\mathbf{y} \in \{0, 1\}$, and we want $\mathbb{P}(\mathbf{y} = 1 \mid \mathbf{x})$.

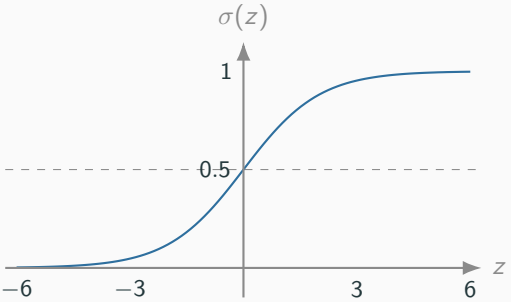
linear regression's raw output $\mathbf{w}^\top \mathbf{x}$ ranges over all of $\mathbb{R}$ — it cannot be read directly as a probability

The sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}} \in (0, 1)$$

Logistic regression squashes the linear score through σ:

$$\mathbb{P}(\mathbf{y} = 1 \mid \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$$



monotonically increasing, $\sigma(0) = \frac{1}{2}$, $\sigma(z) \to 1$ as $z \to \infty$, $\sigma(z) \to 0$ as $z \to -\infty$

A clean derivative, used directly when we differentiate the loss: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$ — from the quotient rule, $\sigma'(z) = e^{-z}/(1 + e^{-z})^2$ and $e^{-z}/(1 + e^{-z}) = 1 - \sigma(z)$.

Choosing w by maximum likelihood

Treat each label as a coin flip whose bias depends on $\mathbf{x}$:

$$\mathbf{y}_i \mid \mathbf{x}_i \sim \text{Bernoulli}(\sigma(\mathbf{w}^\top \mathbf{x}_i))$$

Write $p_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$; the Bernoulli probability mass function is exactly $p_i^{\mathbf{y}_i}(1-p_i)^{1-\mathbf{y}_i}$ — it evaluates to p_i when $\mathbf{y}_i = 1$, and $1 - p_i$ when $\mathbf{y}_i = 0$.

assuming independence, the sample likelihood:

$$\prod_{i=1}^{n} p_i^{\mathbf{y}_i}(1-p_i)^{1-\mathbf{y}_i}$$

MLE: pick $\mathbf{w}$ to maximize the probability the model assigns to the labels we actually observed.

The logistic loss

Maximizing a product of many small numbers is numerically unpleasant — maximize its **logarithm** instead (monotonic, same maximizer) and flip the sign to minimize:

$$-\log\text{-likelihood} = -\sum_{i=1}^{n} \Big[\mathbf{y}_i \log p_i + (1-\mathbf{y}_i)\log(1-p_i)\Big]$$

Dividing by n, this is exactly an empirical risk with per-example loss:

$$\ell(f_{\mathbf{w}}(\mathbf{x}), \mathbf{y}) = -\mathbf{y} \log \sigma(\mathbf{w}^\top \mathbf{x}) - (1-\mathbf{y}) \log\left(1 - \sigma(\mathbf{w}^\top \mathbf{x})\right)$$

the **logistic loss** (equivalently, binary cross-entropy).

MLE on a Bernoulli model and ERM with logistic loss are the same optimization problem.

The gradient

Chain rule through σ, using $\sigma' = \sigma(1-\sigma)$ to cancel the p_i and $1-p_i$ denominators:

per example $\nabla_{\mathbf{w}} \ell(f_{\mathbf{w}}(\mathbf{x}_i), \mathbf{y}_i) = \big(\sigma(\mathbf{w}^\top \mathbf{x}_i) - \mathbf{y}_i\big)\,\mathbf{x}_i$, in full:

$$\nabla \hat{\mathcal{R}}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \big(\sigma(\mathbf{w}^\top \mathbf{x}_i) - \mathbf{y}_i\big)\,\mathbf{x}_i = \frac{1}{n} X^\top \big(\sigma(X\mathbf{w}) - \mathbf{y}\big)$$

($\sigma(X\mathbf{w})$ applies σ coordinate-wise.) *Exactly* the same shape as linear regression's gradient $\frac{1}{n} X^\top (X\mathbf{w} - \mathbf{y})$: "(prediction − label) × feature," with σ inserted to turn the raw score into a probability. GD and SGD apply unchanged — but there is **no closed form**: setting this gradient to zero has no algebraic solution.

Worked example: one SGD step

One example, $\mathbf{x} = (1, 1)$ (feature 1, bias fold-in 1), $\mathbf{y} = 1$, starting $\mathbf{w} = (0, 0)$, so $\sigma(\mathbf{w}^\top \mathbf{x}) = \sigma(0) = 0.5$: the model is maximally unsure.

$$\nabla_{\mathbf{w}} \ell = (\sigma(0) - 1) \cdot (1, 1) = (-0.5, -0.5)$$

$$\text{with } \eta = 0.1: \qquad \mathbf{w} \leftarrow (0, 0) - 0.1 \cdot (-0.5, -0.5) = (0.05, 0.05)$$

new prediction: $\sigma(0.05 + 0.05) = \sigma(0.1) \approx 0.525$ — nudged toward the true label $\mathbf{y} = 1$, exactly as it should be

Two update rules, side by side

The Perceptron, $\{-1, +1\}$ coding, full step exactly on a mistake, nothing otherwise:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta\, \mathbf{y}_i \mathbf{x}_i\, \mathbb{1}\big[\mathbf{y}_i(\mathbf{w}^\top \mathbf{x}_i) \leq 0\big]$$

Logistic regression, $\{0, 1\}$ coding, from the gradient above:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta\, \big(\mathbf{y}_i - \sigma(\mathbf{w}^\top \mathbf{x}_i)\big)\, \mathbf{x}_i$$

Hard vs. soft: a "soft" Perceptron

Both have the same shape: (prediction error) $\times$ feature. But the error is qualitatively different.

Perceptron's error is **hard**: exactly ± 1 on a mistake, exactly 0 on a correct point, no matter how close to the boundary

logistic regression's error is **soft**: $\mathbf{y}_i - \sigma(\mathbf{w}^\top \mathbf{x}_i) \in (-1, 1)$, never exactly 0, scaling continuously with uncertainty

Logistic regression's SGD update is a smoothed, confidence-weighted Perceptron.

Logistic loss is convex: the same Hessian-PSD argument

$$\nabla^2 \hat{\mathcal{R}}(\mathbf{w}) = \frac{1}{n} X^\top S X, \qquad S = \operatorname{diag}\big(\sigma_i(1-\sigma_i)\big)_{i=1}^n, \quad \sigma_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$

for any $\mathbf{v}$:

$$\mathbf{v}^\top X^\top S X \mathbf{v} = \sum_i \sigma_i(1-\sigma_i)\,(\mathbf{x}_i^\top \mathbf{v})^2 \geq 0$$

S is diagonal with nonnegative entries ($\sigma_i(1-\sigma_i) \geq 0$ for any probability), so the quadratic form is a sum of nonnegative terms: $\nabla^2 \hat{\mathcal{R}}(\mathbf{w}) \succeq 0$ everywhere, **logistic loss is convex**, and gradient descent is safe from bad local minima, exactly as before.

Separable data: the loss has no finite minimizer

If the training data is **perfectly linearly separable** (some $\mathbf{w}$ classifies every point correctly), scaling that $\mathbf{w}$ up pushes every $\sigma(\mathbf{w}^\top \mathbf{x}_i)$ closer to the extreme (0 or 1) it should be.

the likelihood keeps improving as $\|\mathbf{w}\| \to \infty$: the loss approaches 0 but never reaches it at any finite $\mathbf{w}$ — gradient descent does not converge, it drifts toward ever larger weights

Not a local-minima problem — the loss is still convex; the minimizer just sits at infinity. This shows up concretely in the Colab notebook; the fix — ℓ_2 regularization, penalizing large $\|\mathbf{w}\|$ — is exactly Lecture 03's topic.

8. Wrap-up

Three models, one template

What we built

Three models under one template — pick a model class, pick a loss, minimize the empirical risk — and two general-purpose ways to do the minimizing.

closed form when it exists (linear regression only)

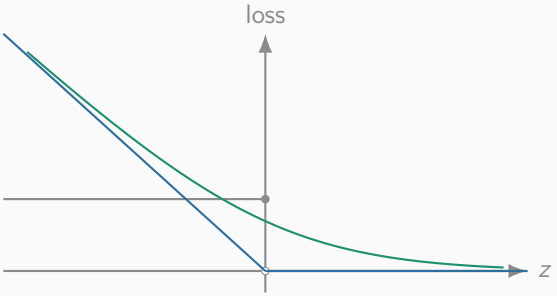
gradient descent / SGD otherwise — *every* loss we built, trustworthy because every one turned out convex

the Perceptron's mistake-driven update is **not** a different paradigm — it is literally SGD with $\eta = 1$ on a convex loss, a special case rather than an exception

Three models, side by side

Model	Loss	How it's fit
Linear regression	MSE (neg. log-lik., Gaussian)	closed form, or GD/SGD
Perceptron	hinge-at-zero	GD/SGD, $\eta = 1$ (no closed form)
Logistic regression	Logistic loss (neg. log-lik., Bernoulli)	GD/SGD (no closed form)

Three losses, one axis



0/1 (hard step) Perceptron: $\max(0, -z)$ Logistic: $\log(1 + e^{-z})$

All three are functions of the same margin $z = \mathbf{y}(\mathbf{w}^\top \mathbf{x})$ from Section 5: the Perceptron loss and the logistic loss are two different smooth-enough replacements for the same hard-to-optimize step, motivated from two different directions (mistake-driven updates; probability and MLE), related exactly as the "soft Perceptron" comparison showed.

Next time: Lecture 03

how well does a model fit this way **generalize** beyond its training sample — and what do we do about overfitting?

All three models return: bias-variance, regularization (ℓ_2 fix for both ill-conditioning and separable-data divergence). HW1, covering this lecture and Lecture 03, is announced today.

References

- Rosenblatt. *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.* Psychological Review, 65(6), 1958.
- Novikoff. *On Convergence Proofs on Perceptrons.* Proc. Symposium on the Mathematical Theory of Automata, 1962.
- Ben-David, Eiron, Long. *On the Difficulty of Approximately Maximizing Agreements.* JCSS, 66(3), 2003.

Optimization/regression/logistic-regression content adapted from EPFL's `ML_course`; narrative order follows Vatsal Sharan's Spring 2026 CSCI 567 Lecture 2 — both credited in `references/sources/lecture02-perceptron-citations.md`.

the pattern this lecture kept repeating

pick a model, pick a loss, minimize it by GD/SGD — even "just count mistakes" turns out to be exactly this, with $\eta = 1$

Questions?