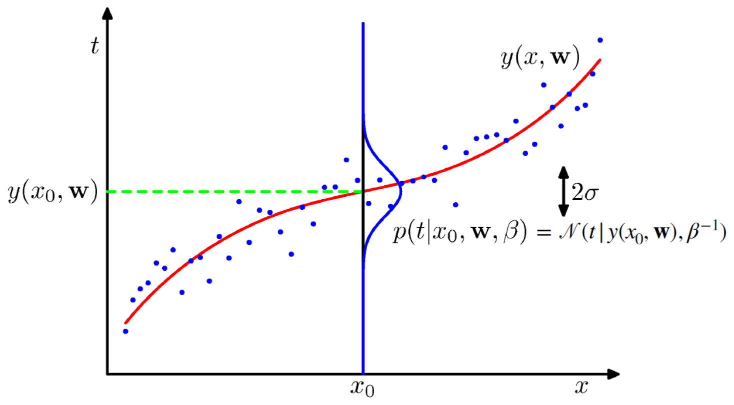
Advanced ML — Lecture 3: Linear Regression

Charalampos Giannakis



The model gives a single number at each input, but the data are noisy.
At the point x_0, real target values spread around the model's output.
Relying only on the single prediction ignores this uncertainty—that's the pitfall.

Imagine we know the joint probability of inputs and targets, $p(x, t)$.
What should our model do? The frequentist approach says: *we should minimize the expected loss*:

$$\mathbb{E}[L] = \iint L(t, y(x))\, p(x, t)\, dx\, dt$$

Now, what loss should we choose? A very natural choice is the **squared loss**, because it punishes large mistakes more strongly. That gives us:

$$\mathbb{E}[L] = \iint (y(x) - t)^2\, p(x, t)\, dx\, dt$$

In words: - Compare our prediction $y(x)$ to the true value t. - Square the difference so errors don't cancel. - Average over all possible data points, weighted by their probability $p(x, t)$. This is the standard frequentist regression goal: *find the curve that minimizes the average squared error over the data distribution.*

We now zoom in on a single input x.
If we use squared loss, the expected loss at this x is

$$\mathbb{E}[L(t, y(x))] = \int (y(x) - t)^2\, p(t \mid x)\, dt.$$

Think of $y(x)$ here as just a number we can tune.
Differentiate the expression with respect to $y(x)$ (move the derivative inside the integral):

$$\frac{d}{d\,y(x)}\mathbb{E}[L] = 2\int (y(x) - t)\,p(t \mid x)\,dt.$$

Set this derivative to zero to minimize the loss:

$$\int (y(x) - t)\,p(t \mid x)\,dt = 0 \;\Rightarrow\; y(x) = \int t\,p(t \mid x)\,dt = \mathbb{E}[t \mid x].$$

Message: under squared loss, the best point prediction at x is the **conditional mean** of t given x.

We start from the expected squared loss

$$\mathbb{E}[L] = \iint (y(x) - t)^2\,p(x,t)\,dx\,dt.$$

Insert and subtract the conditional mean $\mathbb{E}[t \mid x]$ inside the square and expand.
After taking expectations, the cross-term vanishes, yielding

$$\mathbb{E}[L] = \int \big(y(x) - \mathbb{E}[t \mid x]\big)^2 p(x)\,dx \;+\; \int \mathrm{var}[t \mid x]\;p(x)\,dx.$$

Meaning: the total error splits into (i) a part we can control—how far our predictor is from the conditional mean—and (ii) an **irreducible noise** term.

Derivation steps (algebra)

Let $\mu(x) := \mathbb{E}[t \mid x]$ and $\mathrm{Var}[t \mid x] := \mathbb{E}[(t - \mu(x))^2 \mid x]$.

1. **Start**
$$\mathbb{E}[L] = \iint (y(x) - t)^2\,p(x,t)\,dx\,dt.$$

2. **Condition on x**
$$\mathbb{E}[L] = \int \Big(\int (y(x) - t)^2\,p(t \mid x)\,dt \Big)\,p(x)\,dx.$$

3. **Add–subtract $\mu(x)$**
$$(y(x) - t)^2 = \big(y(x) - \mu(x) + \mu(x) - t\big)^2.$$

4. **Expand**

$$(y(x) - t)^2 = \big(y(x) - \mu(x)\big)^2 + 2(y(x) - \mu(x))(\mu(x) - t) + \big(\mu(x) - t\big)^2.$$

5. **Take** $\mathbb{E}[\cdot \mid x]$

$$\int (y(x)-t)^2 p(t \mid x)\,dt = \big(y(x)-\mu(x)\big)^2 + 2(y(x)-\mu(x)) \underbrace{\int (\mu(x)-t)p(t \mid x)\,dt}_{0}$$
$$+ \underbrace{\int (\mu(x)-t)^2 p(t \mid x)\,dt}_{\mathrm{Var}[t|x]}$$
$$= \big(y(x)-\mu(x)\big)^2 + \mathrm{Var}[t \mid x].$$

6. **Plug back and split**

$$\mathbb{E}[L] = \int \big(y(x)-\mu(x)\big)^2 p(x)\,dx + \int \mathrm{Var}[t \mid x]\, p(x)\,dx.$$

We restate the decomposition:

$$\mathbb{E}[L] = \int \big(y(x)-h(x)\big)^2 p(x)\,dx \;+\; \iint \big(h(x)-t\big)^2 p(x,t)\,dx\,dt, \qquad h(x) = \mathbb{E}[t \mid x].$$

- The **second term** is inherent noise: variability of t around its conditional mean—no model can remove it.

- The **first term** measures how far our predictor $y(x)$ is from the optimal $h(x)$; this is what learning aims to reduce.

Suppose we could repeat data collection many times.
Each dataset $\mathcal{D}$ (size N) yields a fitted function $y(x;\mathcal{D})$, which we compare to the ideal target $h(x) = \mathbb{E}[t \mid x]$.

Insert and subtract the **average predictor across datasets** $\mathbb{E}_{\mathcal{D}}[y(x;\mathcal{D})]$:

$$\{y(x;\mathcal{D}) - h(x)\}^2 = \{\, y(x;\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(x;\mathcal{D})] + \mathbb{E}_{\mathcal{D}}[y(x;\mathcal{D})] - h(x)\,\}^2.$$

Expand the square:

$$\begin{aligned}\{y(x;\mathcal{D}) - h(x)\}^2 &= \{y(x;\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(x;\mathcal{D})]\}^2 \\ &\quad + \{\mathbb{E}_{\mathcal{D}}[y(x;\mathcal{D})] - h(x)\}^2 \\ &\quad + 2\,\{y(x;\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(x;\mathcal{D})]\}\{\mathbb{E}_{\mathcal{D}}[y(x;\mathcal{D})] - h(x)\}.\end{aligned}$$

This sets up the bias–variance split: after averaging over datasets, the cross-term vanishes, leaving **variance** (first term) and **bias²** (second term).

Average the previous identity over all datasets $\mathcal{D}$.
The cross-term drops to zero, giving the **bias–variance split**:

$$\mathbb{E}_{\mathcal{D}}[(y(x;\mathcal{D}) - h(x))^2] = \underbrace{\big(\mathbb{E}_{\mathcal{D}}[y(x;\mathcal{D})] - h(x)\big)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}[(y(x;\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(x;\mathcal{D})])^2]}_{\text{variance}}.$$

Because

$$\mathbb{E}_{\mathcal{D}}[y - \mu_{\mathcal{D}}] = \mathbb{E}_{\mathcal{D}}[y] - \mu_{\mathcal{D}} = \mu_{\mathcal{D}} - \mu_{\mathcal{D}} = 0.$$

Interpretation:
- **Bias²**: how far the *average* learned model is from the truth $h(x)$.
- **Variance**: how much models fluctuate across different datasets.

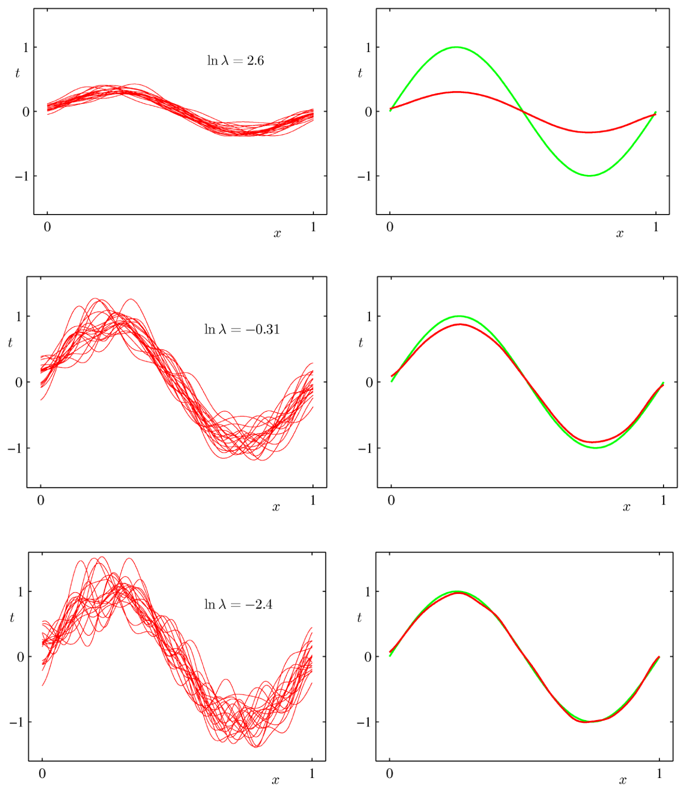
Total expected squared error at x = bias² + variance (noise will be added next).

Conclusion: the expected squared loss splits into three parts:

$$\text{expected loss} \;=\; \text{bias}^2 \;+\; \text{variance} \;+\; \text{noise}.$$

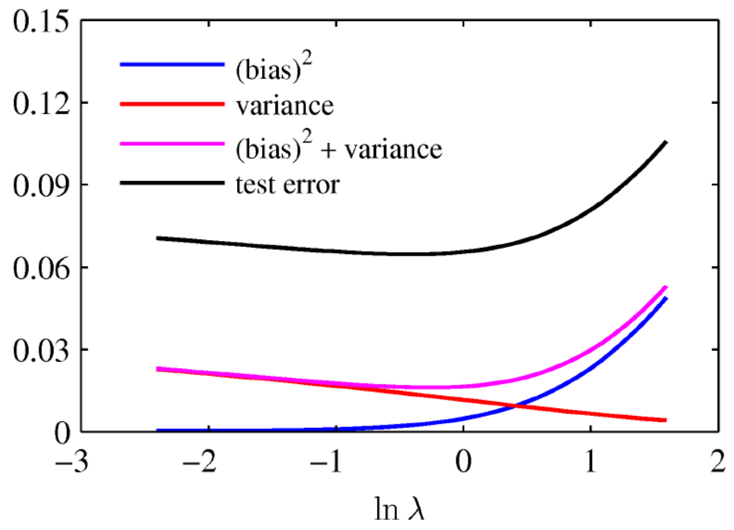
- **Bias²** — systematic error of the *average* learned model:
$$\int \left(\mathbb{E}_{\mathcal{D}}[y(x;\mathcal{D})] - h(x)\right)^2 p(x)\,dx.$$
- **Variance** — how much the learned model changes across datasets:
$$\int \mathbb{E}_{\mathcal{D}}\big[(y(x;\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(x;\mathcal{D})])^2\big]\; p(x)\,dx.$$
- **Noise** — irreducible randomness in the target around its conditional mean:
$$\iint (h(x) - t)^2\, p(x,t)\,dx\,dt.$$

Takeaway: learning can only trade off **bias** vs **variance**; the **noise** term is intrinsic to the data and cannot be reduced by the model.



Bias–variance on sinusoidal data (regularization sweep)
Imagine we repeatedly sample 100 different datasets from a noisy sinusoid (each with 25 points) and fit the same model each time, but we vary the regularization strength. When we overlay the fitted curves across datasets, two patterns jump out: with large λ (strong regularization), curves are very smooth and stay close to a simple shape—models barely move from run to run (low variance) but miss the wiggly truth (high bias). With small λ (weak regularization), curves chase the noise—each dataset yields a different, squiggly fit (high variance) that may pass near many training points but generalizes poorly. These panels visualize the bias–variance trade-off we derived earlier: average error = bias^2 + variance + noise; tuning λ slides us along that trade-off.

Train vs. test error as regularization changes (the sweet spot)



Reading this figure: $\ln\lambda$ on the x-axis; large $\lambda \to$ high bias (blue), small $\lambda \to$ high variance (red). The magenta curve bias^2 + variance is U-shaped, and the black test-error curve lies just above it due to irreducible noise, with its minimum near the middle (optimal $\lambda^\star$).

This plot shows the classic pattern when we sweep the regularization strength λ in ridge regression: $\mathcal{L}_\lambda(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(t_n - \mathbf{w}^\top\phi(x_n))^2 + \lambda\|\mathbf{w}\|^2$. As λ increases (stronger shrinkage), the model is rigid: training error rises slightly, variance falls, and bias grows. As λ decreases (weaker shrinkage), the model becomes flexible: training error keeps dropping, but variance explodes and test performance degrades. The test (or validation) error is therefore U-shaped: it first goes down (bias decreases faster than variance increases) and then goes up (variance dominates). The minimum marks the optimal $\lambda^\star$, where the decrease in bias is balanced by the increase in variance. The decomposition behind the plot is: $\mathbb{E}[(y(x)-t)^2] = (\mathbb{E}[y(x)] - h(x))^2 + \mathbb{V}[y(x)] + \mathbb{V}[t \mid x]$.

Practical recipe: pick $\lambda^\star$ by cross-validation and report test error at that setting—use enough flexibility to capture the signal (low bias) but enough shrinkage to keep the model stable across datasets (low variance).

Why bias–variance is mostly an *intuition* tool
The decomposition

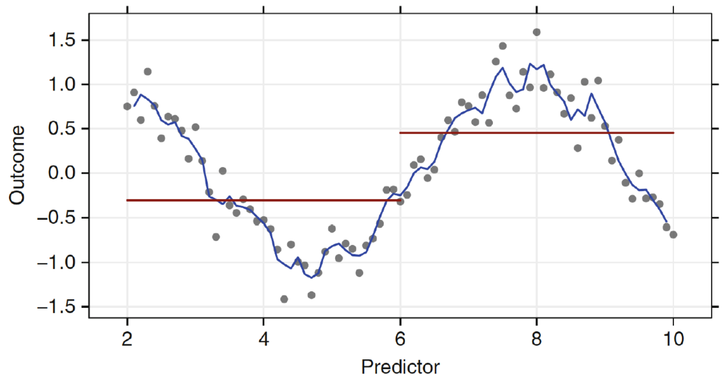
$$\mathbb{E}_{\mathcal{D}}[(y_{\mathcal{D}}(x)-t)^2] = (\mathbb{E}_{\mathcal{D}}[y_{\mathcal{D}}(x)] - h(x))^2 + \mathbb{V}_{\mathcal{D}}[y_{\mathcal{D}}(x)] + \mathbb{V}[t \mid x]$$

takes an expectation over datasets $\mathcal{D}$. In practice we have **one** dataset, and we don't know the true function $h(x)$, so we **can't measure** bias or variance directly. The curves are therefore

conceptual—useful for intuition about how λ affects bias and variance—but of **limited diagnostic value** on a single sample.

What we do instead: estimate generalization **empirically**—use a validation set, K-fold cross-validation, or bootstrap; pick λ at the validation minimum; report performance on a held-out **test** set.

Bias–variance tradeoff (visual intuition)



The grey dots are the data. The **blue curve** is a very **flexible** fit: it hugs the data tightly (low bias) but wiggles a lot from region to region (**high variance**). The **red step lines** are an **over-smoothed**/coarsely binned fit: it barely moves (low variance) but misses real structure (**high bias**). The goal is **neither** extreme—pick a model complexity (, bandwidth, k, depth, etc.) that captures the broad trend without chasing noise. In practice we tune that complexity by **cross-validation** and evaluate on a held-out test set.

Bayesian linear regression

So far we picked a single "best" weight vector $\mathbf{w}$. But one dataset rarely pins down the truth; many $\mathbf{w}$ explain the data almost as well. In the Bayesian view we **keep a distribution over weights** and let the data reshape it.

Bayes' rule glues the pieces together:
$p(\mathbf{w} \mid \mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})\, p(\mathbf{w})}{p(\mathcal{D})} \propto \underbrace{p(\mathcal{D} \mid \mathbf{w})}_{\text{likelihood}} \times \underbrace{p(\mathbf{w})}_{\text{prior}}.$

- **Prior** $p(\mathbf{w})$: what we believe before seeing data (e.g., "weights are small," often a zero-mean Gaussian).
- **Likelihood** $p(\mathcal{D} \mid \mathbf{w})$: the data model—linear regression with Gaussian noise.
- **Posterior** $p(\mathbf{w} \mid \mathcal{D})$: updated belief after seeing data; prior pulled toward regions where the model fits well.

This immediately connects to regularization: a Gaussian prior that favors small weights is the probabilistic version of **ridge** (L2) shrinkage. The payoff is in prediction: instead of plugging in one $\hat{\mathbf{w}}$, we **average over uncertainty**

$$p(t_* \mid x_*, \mathcal{D}) = \int p(t_* \mid x_*, \mathbf{w})\, p(\mathbf{w} \mid \mathcal{D})\, d\mathbf{w}$$

giving calibrated means **and** credible uncertainty bands that naturally temper overfitting.

Bayesian linear regression with a conjugate Gaussian prior — what's going on

We model targets as linear-in-features with Gaussian noise:

- Design matrix $\Phi \in \mathbb{R}^{N\times M}$ has rows $\phi(x_n)^\top$.
- Weights $\mathbf{w} \in \mathbb{R}^M$.
- Likelihood (noise precision $\beta = 1/\sigma^2$):

$$p(\mathbf{t} \mid \mathbf{w}) = \mathcal{N}(\mathbf{t} \mid \Phi\mathbf{w},\ \beta^{-1}\mathbf{I}).$$

Choose a **conjugate prior** on the weights:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{m}_0,\ \mathbf{S}_0).$$

Conjugacy $\Rightarrow$ the posterior is Gaussian:

$$p(\mathbf{w} \mid \mathbf{t}) = \mathcal{N}(\mathbf{w} \mid \mathbf{m}_N,\ \mathbf{S}_N), \qquad \mathbf{S}_N^{-1} = \mathbf{S}_0^{-1} + \beta\,\Phi^\top\Phi, \qquad \mathbf{m}_N = \mathbf{S}_N(\mathbf{S}_0^{-1}\mathbf{m}_0 + \beta\,\Phi^\top\mathbf{t}).$$

Where do those formulas come from? (quick derivation)
Start from Bayes:

$$\log p(\mathbf{w} \mid \mathbf{t}) = \log p(\mathbf{t} \mid \mathbf{w}) + \log p(\mathbf{w}) + C.$$

Plug in the Gaussian forms and drop constants:

$$-\tfrac{\beta}{2}\|\mathbf{t} - \Phi\mathbf{w}\|^2 - \tfrac{1}{2}(\mathbf{w} - \mathbf{m}_0)^\top\mathbf{S}_0^{-1}(\mathbf{w} - \mathbf{m}_0).$$

Expand and collect terms in $\mathbf{w}$:

$$-\tfrac{1}{2}\mathbf{w}^\top(\mathbf{S}_0^{-1} + \beta\Phi^\top\Phi)\mathbf{w} + \mathbf{w}^\top(\mathbf{S}_0^{-1}\mathbf{m}_0 + \beta\Phi^\top\mathbf{t}).$$

Completing the square yields the Gaussian with precision $\mathbf{S}_N^{-1}$ and mean $\mathbf{m}_N$ above.

Important special case
Isotropic zero-mean prior:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{0},\ \alpha^{-1}\mathbf{I}) \ \Rightarrow\ \mathbf{S}_N^{-1} = \alpha\mathbf{I} + \beta\Phi^\top\Phi, \qquad \mathbf{m}_N = \beta\,\mathbf{S}_N\Phi^\top\mathbf{t}.$$

Here α is **prior precision** (how strongly we shrink weights) and β is **noise precision** (data confidence).

Connection to ridge regression (MAP vs posterior)
The MAP equals the posterior mean for a Gaussian posterior:

$$\mathbf{w}_{\text{MAP}} = \mathbf{m}_N = \beta(\alpha\mathbf{I} + \beta\Phi^\top\Phi)^{-1}\Phi^\top\mathbf{t} = (\Phi^\top\Phi + \tfrac{\alpha}{\beta}\mathbf{I})^{-1}\Phi^\top\mathbf{t}.$$

This is **ridge regression** with penalty $\lambda = \alpha/\beta$.

Predictive distribution (the payoff)
For x_* with features $\phi_* = \phi(x_*)$:

$$p(t_* \mid x_*, \mathbf{t}) = \mathcal{N}\Big(\underbrace{\phi_*^\top\mathbf{m}_N}_{\text{predictive mean}},\ \underbrace{\beta^{-1}}_{\text{noise}} + \underbrace{\phi_*^\top\mathbf{S}_N\phi_*}_{\text{parameter uncertainty}}\Big).$$

Intuition / dynamics as data arrive

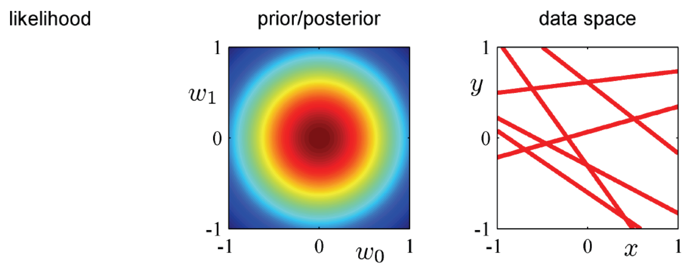
- Each observation adds $\beta\,\phi(x)\phi(x)^\top$ to the precision; $\mathbf{S}_N$ **shrinks** (certainty grows).
- **Large** α (strong prior): posterior stays near zero unless data are persuasive.
- **Large** β (low noise): data dominate; we recover least squares.

Key results (at a glance).

1. General conjugate update: $\mathbf{m}_N, \mathbf{S}_N$.
2. Specialize to $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \alpha^{-1}\mathbf{I})$:
$$\mathbf{m}_N = \beta\,\mathbf{S}_N\Phi^\top\mathbf{t}, \qquad \mathbf{S}_N^{-1} = \alpha\mathbf{I} + \beta\Phi^\top\Phi.$$
3. A simple line $y(x) = -0.3 + 0.5x$ makes it concrete: $\mathbf{m}_N$ moves toward the true slope/intercept and $\mathbf{S}_N$ tightens (credible intervals narrow) as data accumulate.

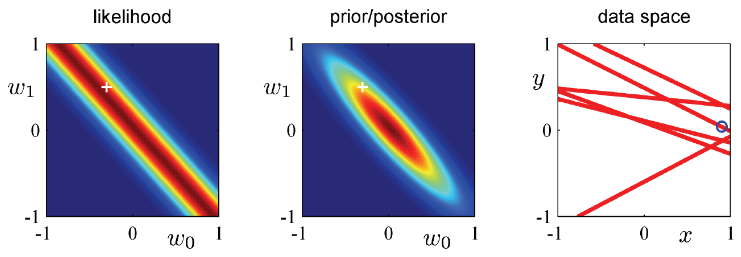
Bayesian learning in action — from prior to confident posterior (0 $\longrightarrow$ 20 points) We use the simple line $y = w_0 + w_1x + \varepsilon$ with a zero-mean Gaussian prior on $\mathbf{w} = (w_0, w_1)$ and Gaussian noise. Each slide shows three views: (1) the **likelihood** in (w_0, w_1)-space, (2) the **prior/posterior** over (w_0, w_1), (3) the **data space** (many red lines sampled from the current weight distribution).

- **0 points.**



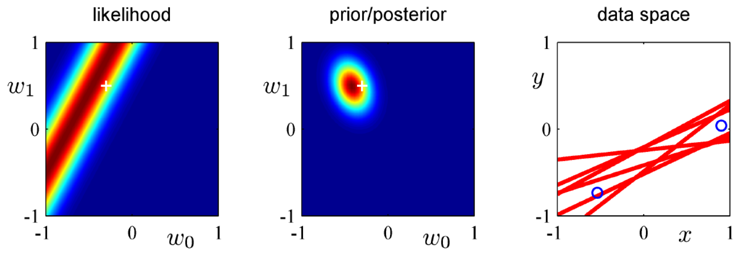
With no data, the likelihood is flat (uninformative). Only the **prior** remains: a round bump centered at 0, expressing that small weights are plausible. In data space this becomes **many possible lines** with all kinds of slopes and intercepts—complete uncertainty.

- **1 point.**



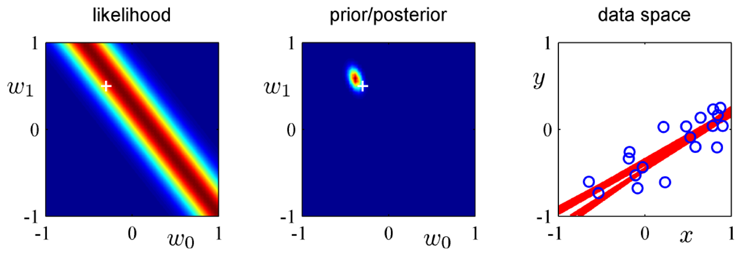
One observation (x_1, y_1) makes the likelihood a **diagonal ridge**: all (w_0, w_1) with $y_1 \approx w_0 + w_1x_1$ lie on that band. Multiplying by the prior yields an **elongated posterior ellipse** where that ridge overlaps high prior mass (white cross posterior mean). In data space, sampled lines now **pivot through the observed point**, but still fan out elsewhere.

- **2 points.**



Two distinct x's give two constraints; their ridges **intersect**, so the likelihood becomes a **thin needle** near that intersection. The posterior shrinks to a **tight ellipse**. In data space, sampled lines pass **close to both points**, with much less spread.

- **20 points.**



With many observations the likelihood concentrates sharply; the posterior is a **very tight blob**—strong agreement on slope and intercept. In data space, the red lines collapse into a **narrow ribbon** following the trend; remaining spread is mainly the observation noise.

What's happening under the hood (why the ellipses tighten):
With Gaussian prior/likelihood the posterior precision updates as $\mathbf{S}_N^{-1} = \alpha\mathbf{I} + \beta\,\Phi^\top\Phi$. Each new datapoint adds $\beta\,\phi(x)\phi(x)^\top$, so the covariance $\mathbf{S}_N$ **shrinks** monotonically. Predictions at a new x_* have variance
$\mathrm{Var}(t_* \mid x_*, \mathcal{D}) = \beta^{-1} + \phi_*^\top \mathbf{S}_N \phi_*$,
so uncertainty drops fastest **where we have data** and more slowly in extrapolation regions.

Bayesian linear regression with an isotropic Gaussian prior — all pieces together

We model
$y = \mathbf{w}^\top\phi(x) + \varepsilon$, with $\varepsilon \sim \mathcal{N}(0, \beta^{-1})$.
Collect the targets in $\mathbf{t} \in \mathbb{R}^N$ and features in $\Phi \in \mathbb{R}^{N\times M}$.

Prior, likelihood, posterior

- **Prior (common choice):** $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \alpha^{-1}\mathbf{I})$
  ("weights are probably small"; α is the *prior precision*).
- **Likelihood:** $p(\mathbf{t} \mid \mathbf{w}) = \mathcal{N}(\Phi\mathbf{w}, \beta^{-1}\mathbf{I})$
  (β is the *noise precision*).

Using Gaussian–Gaussian conjugacy,
$p(\mathbf{w} \mid \mathbf{t}) = \mathcal{N}(\mathbf{m}_N, \mathbf{S}_N), \quad \mathbf{S}_N^{-1} = \alpha\mathbf{I} + \beta\,\Phi^\top\Phi, \quad \mathbf{m}_N = \beta\,\mathbf{S}_N\Phi^\top\mathbf{t}.$

MAP = penalized least squares (why the formulas make sense)

Take logs and drop constants:
$\ln p(\mathbf{w} \mid \mathbf{t}) = -\frac{\beta}{2} \sum_{n=1}^{N} \left(t_n - \mathbf{w}^\top \phi(x_n)\right)^2 - \frac{\alpha}{2} \mathbf{w}^\top \mathbf{w} + \text{const}.$

Maximizing the posterior is **exactly** minimizing a **least-squares loss plus an L_2 penalty** on $\mathbf{w}$; the effective shrinkage level is $\lambda = \alpha/\beta$. This objective is strictly convex (unique minimizer) because $\alpha\mathbf{I}$ makes the quadratic form positive definite even when $\Phi^\top\Phi$ is ill-conditioned.

Edge cases that build intuition

- **No prior information ($\alpha \to 0$).**
  $\mathbf{S}_N^{-1} \to \beta\,\Phi^\top\Phi$ and
  $\mathbf{m}_N \to (\Phi^\top\Phi)^{-1}\Phi^\top\mathbf{t}$ (ordinary least squares), provided $\Phi^\top\Phi$ is invertible; otherwise one uses the Moore–Penrose pseudoinverse.
- **Extremely strong prior ($\alpha \to \infty$).**
  The penalty dominates: $\mathbf{m}_N \to \mathbf{0}$ (or to $\mathbf{m}_0$ if the prior mean is nonzero). Data are effectively ignored.
- **Infinite data ($N \to \infty$ with bounded α).**
  The data term $\beta\,\Phi^\top\Phi$ dominates; $\mathbf{m}_N$ converges to the least-squares solution, and $\mathbf{S}_N \to (\beta\,\Phi^\top\Phi)^{-1}$ so parameter uncertainty vanishes.

Posterior predictive — integrate out the weights
For a new input x, we average predictions over the posterior on $\mathbf{w}$: $p(t \mid x, \mathbf{t}, \alpha, \beta) = \int p(t \mid \mathbf{w}, \beta)\, p(\mathbf{w} \mid \mathbf{t}, \alpha, \beta)\, d\mathbf{w} = \mathcal{N}(t \mid \mathbf{m}_N^\top\phi(x),\ \sigma_N^2(x))$,
with predictive variance $\sigma_N^2(x) = \frac{1}{\beta} + \phi(x)^\top\mathbf{S}_N\phi(x)$.

How to read this:
- **Mean $\mathbf{m}_N^\top\phi(x)$**: plug the **posterior mean** of the weights into the model.
- **Variance $\sigma_N^2(x)$** splits into $\frac{1}{\beta}$ (aleatoric noise) + $\phi(x)^\top\mathbf{S}_N\phi(x)$ (epistemic uncertainty).

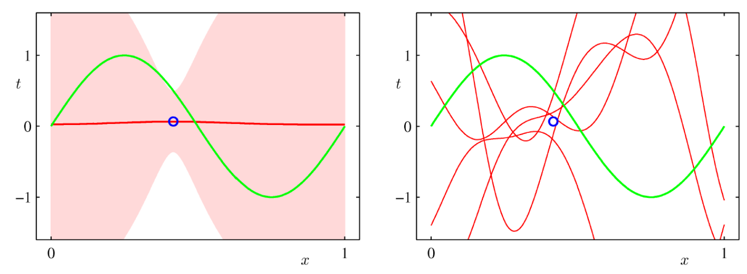
Consequences:
- Near regions dense in training inputs, $\phi(x)^\top\mathbf{S}_N\phi(x)$ is small tight intervals; in extrapolation it grows wider intervals.
- As data accumulate, $\mathbf{S}_N$ shrinks, so $\phi(x)^\top\mathbf{S}_N\phi(x)$ decreases; in the limit $\sigma_N^2(x) \to 1/\beta$.
- A 95% credible band at x is $\mathbf{m}_N^\top\phi(x) \pm 1.96\,\sigma_N(x)$.

(Recall: $\mathbf{S}_N^{-1} = \alpha\mathbf{I} + \beta\,\Phi^\top\Phi$ and $\mathbf{m}_N = \beta\,\mathbf{S}_N\Phi^\top\mathbf{t}$.)

Sinusoid with 9 Gaussian bases — watching the posterior learn

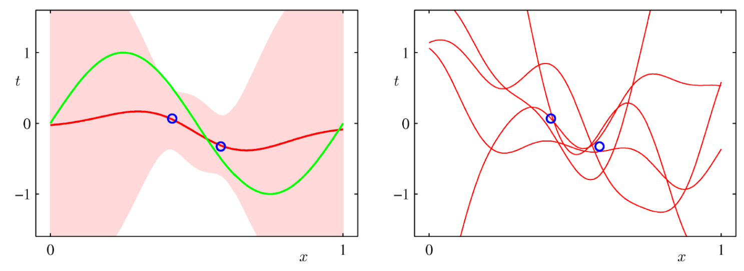
We model the unknown curve with features $\phi(x)$ made of **9 Gaussian basis functions**. We keep a Gaussian prior on the weights, observe data, and then predict with
$p(t \mid x, \mathcal{D}) = \mathcal{N}(\mathbf{m}_N^\top\phi(x),\ \sigma_N^2(x))$ where $\sigma_N^2(x) = \frac{1}{\beta} + \phi(x)^\top S_N\phi(x)$.
In the plots: **green** = true sinusoid, **red curve** = predictive mean, **pink band** = $\pm$ credible region, **red thin curves** (right panels) = samples from the posterior over functions, **blue circles** = observed points.

1 data point



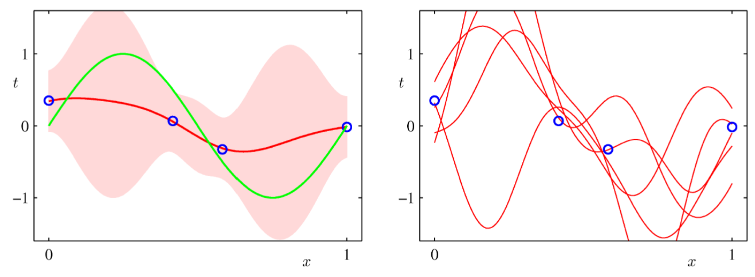
The single observation pins the function only **at that** x. The mean stays close to the **prior mean** elsewhere (nearly flat), and the band is huge because $\phi(x)^\top S_N \phi(x)$ is large almost everywhere. Posterior function samples swing wildly while passing near the one point.

2 data points



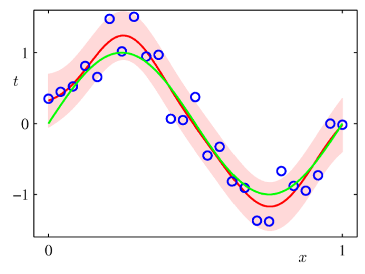
Two constraints reduce uncertainty to a **corridor** connecting them. The mean starts to bend but many shapes still fit; samples show large variability in amplitude and phase. Uncertainty is smallest **near the two** x**'s** and widest in between/at the edges.

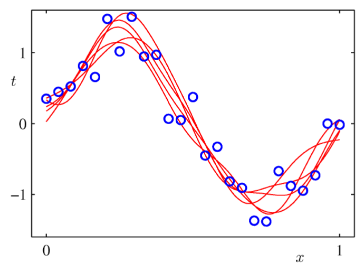
4 data points



Now the posterior has enough evidence to recover the **curvature**: the mean begins to trace the sinusoid and the credible band narrows, especially around observed x's. Samples are still diverse but concentrate around similar smooth shapes.

25 data points





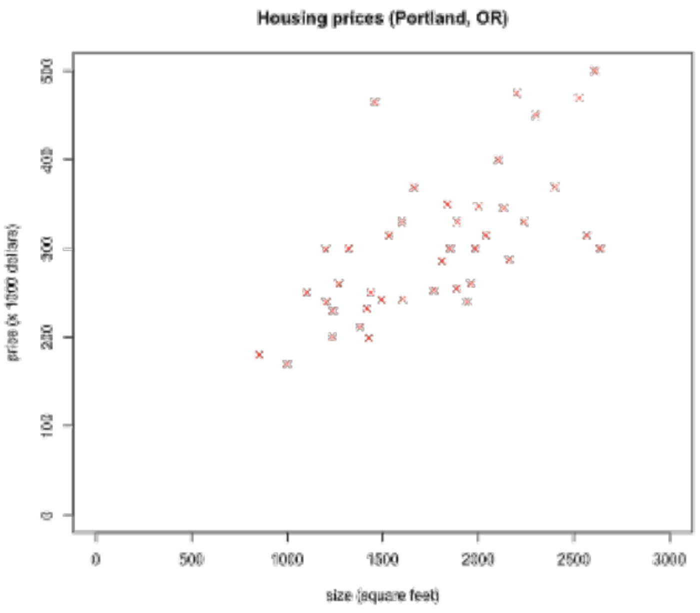
Data dominate the prior. The mean closely follows the green truth and the band shrinks to roughly the **noise level** (the $1/\beta$ term). Posterior samples cluster tightly—differences are now subtle and mostly in regions with fewer nearby points (band widens a bit near the boundaries).

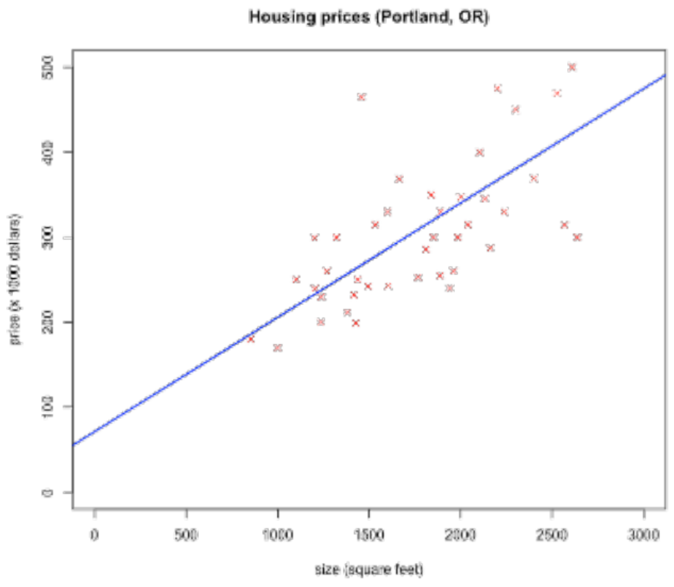
Key lesson. More data $\Rightarrow S_N$ shrinks $\Rightarrow \phi(x)^\top S_N \phi(x)$ drops, so uncertainty collapses first where we observe points. The 9 Gaussian bases give enough flexibility to capture the sinusoid, while the Bayesian update keeps that flexibility in check when data are scarce.

Conclusions — simple version

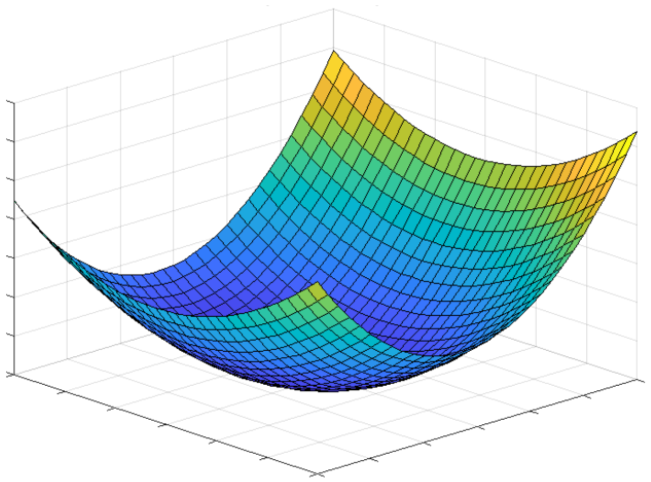
- Using **least squares / maximum likelihood** with a very flexible model and **little data** often **overfits** (it follows noise).
- The **Bayesian** way helps because it:
  - uses a **prior** to gently pull weights toward simple values when data are scarce (less overfitting);
  - gives **uncertainty bars** with every prediction, so you know how confident the model is.
- As you get **more data**, the prior matters less and the Bayesian and classical results become the same.

Linear model recap.
We model targets as a linear combination of features: $y(x, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \, \phi_j(x) = \mathbf{w}^\top \phi(x)$.
For the simple case $M = 2$ with $\phi_0(x) = 1, \ \phi_1(x) = x$, we get the straight line $y(x) = w_0 + w_1 x$. The least-squares solution can be written in matrix form as $\mathbf{w}_{\text{ML}} = (\Phi^\top \Phi)^{-1} \Phi^\top \mathbf{t}$ (when invertible).

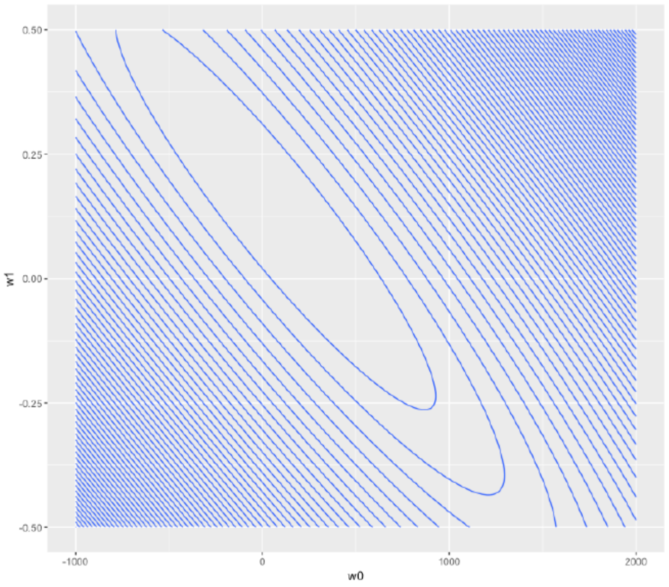




Objective.
We have $y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x$ We measure performance by the mean squared error $E(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} \left(y(x_n, \mathbf{w}) - t_n\right)^2$.
Our goal is to choose $\mathbf{w}$ to minimize this $E(\mathbf{w})$.



Intuition (the bowl).
Think of the loss $E(\mathbf{w})$ as a smooth bowl in weight space. Wherever we stand, the **gradient** $\nabla E(\mathbf{w})$ tells us the direction of steepest *increase*. So gradient descent just walks **downhill**: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla E(\mathbf{w})$. On a quadratic bowl (like least squares), every step lowers the loss and the slope flattens as we approach the unique bottom $\mathbf{w}^*$. With a reasonable step size α, this guarantees steady progress.



Contours (why it zig-zags).
Now view the same bowl from above: each ellipse is a level set of equal loss. The gradient at any point is **perpendicular** to the contour and points straight outward; gradient descent moves inward, crossing contours toward the center. If the ellipses are very **stretched** (features on different scales / ill-conditioning), the path **zig-zags** across the narrow valley and converges slowly. Standardizing features (or using a preconditioner/optimizer) makes the contours rounder, so steps head more directly to $\mathbf{w}^*$ and converge much faster.

Gradient descent algorithm.

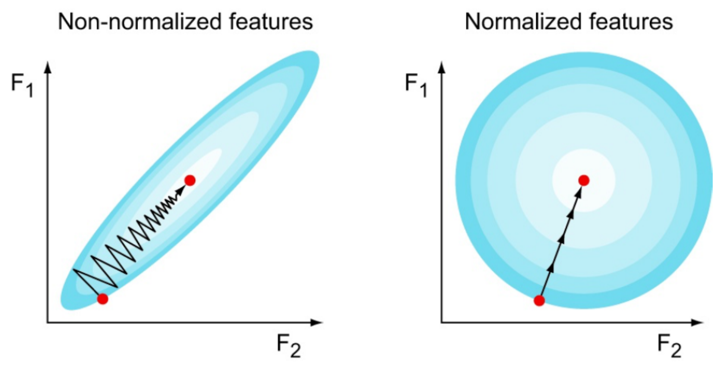
At iteration k with $\mathbf{w}^{(k)} = (w_0^{(k)}, w_1^{(k)})$, move downhill using the gradient **evaluated at the same point**: $w_j^{(k+1)} = w_j^{(k)} - \alpha \dfrac{\partial E}{\partial w_j}(\mathbf{w}^{(k)})$ for $j \in \{0,1\}$.

Incorrect update (sequential / mixed iteration).
Computing `temp0` and assigning w_0 **before** computing ‘temp1$ uses a *mixed* point $(w_0^{(k+1)}, w_1^{(k)})$ for the second derivative.
This is not the vector step $-\alpha\nabla E(\mathbf{w}^{(k)})$, can skew the direction, and may slow or destabilize convergence.

Correct update (simultaneous / synchronous).
Compute all tentative updates at $\mathbf{w}^{(k)}$ first, then assign together: $\text{temp0} := w_0^{(k)} - \alpha \dfrac{\partial E}{\partial w_0}(\mathbf{w}^{(k)})$,
$\text{temp1} := w_1^{(k)} - \alpha \dfrac{\partial E}{\partial w_1}(\mathbf{w}^{(k)})$,
$w_0^{(k+1)} := \text{temp0}, \quad w_1^{(k+1)} := \text{temp1}$.
Now the update equals $-\alpha\nabla E(\mathbf{w}^{(k)})$, the textbook GD step.

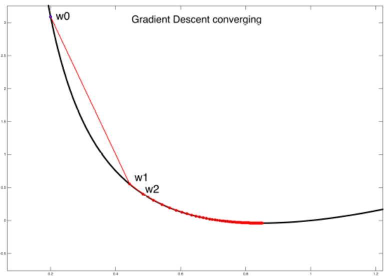
Gradient Descent — quick recipe

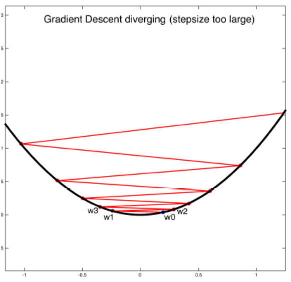
1. **Standardize features** (mean 0, std 1) to avoid stretched contours.
2. **Initialize $\mathbf{w} = \mathbf{0}$** (or small random).
3. **Choose step size α** (start small, e.g. 10^{-2} or 10^{-3}; reduce if E increases).
4. **Loop** until convergence:
   - Compute gradient at current weights: $\nabla E(\mathbf{w})$.
   - **Simultaneous update: $\mathbf{w} \leftarrow \mathbf{w} - \alpha\nabla E(\mathbf{w})$.**
5. **Stop** when relative change in E is small, $\|\nabla E\|$ is small, or max iters reached.



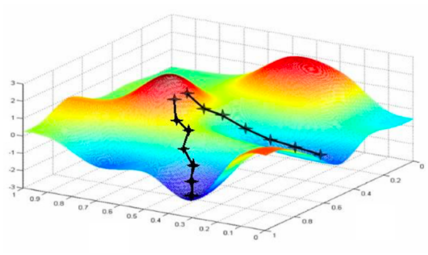
Feature scaling is important.
Contours show equal loss $E(\mathbf{w})$. With **non-normalized features**, one feature dominates the scale $\rightarrow$ the Hessian H is ill-conditioned (very uneven eigenvalues). Gradient steps, being perpendicular

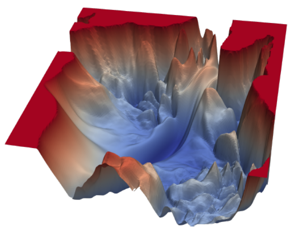
to contours, **zig-zag** across the thin direction and progress slowly.
Standardize each feature (except the intercept): $x'_j = (x_j - \mu_j)/\sigma_j$. This makes contours closer to circles (better conditioning) so steps aim more directly at the minimum and converge faster.





Step size controls convergence.
In 1D, $E(w) = \frac{1}{2}a(w - w^*)^2$ gives $w_{k+1} - w^* = (1 - \alpha a)(w_k - w^*) \to$ convergence iff $0 < \alpha < \frac{2}{a}$.
In multiple dimensions with Hessian H, a safe bound is $0 < \alpha < \dfrac{2}{\lambda_{\max}(H)}$.
- **Too small** α: slow approach.
- **Too large** α: overshoot and **diverge**.
Pick α by a small search; reduce it if the loss increases.





Convexity global optimality.
For **convex** losses (e.g., least squares), any local minimum is **global**; with a proper α, gradient descent reaches the optimum.
For **non-convex** surfaces (many basins/saddles), GD may get stuck or depend on initialization; restarts, good scaling, and regularization help but there's no global guarantee.

Gradient descent for linear regression — wrap-up.
General model: $y(x, \mathbf{w}) = \sum_{j=0}^{M-1} w_j\, \phi_j(x) = \mathbf{w}^\top \phi(x)$.

Repeat until convergence (simultaneous updates):
$$w_j \leftarrow w_j - \alpha\, \frac{1}{N} \sum_{n=1}^{N} \left(y(x_n, \mathbf{w}) - t_n\right) \phi_j(x_n), \quad j = 0, \dots, M-1.$$

Notes.
- Standardize features for faster, stabler convergence.
- Choose α small enough to avoid overshooting; reduce it if the loss increases.
- For least squares the loss is convex converges to the global optimum.
- Closed form exists, but GD scales better and accommodates extensions like regularization.