Kernel Methods, Double Descent, and Implicit Regularization

What happens once a model has more parameters than data?

CSCI 567 — Machine Learning • Lecture 04 • Sep 18, 2026

One loop left open, one boundary flagged

Lecture 02 flagged: if $d+1 > n$, $X^\top X$ is rank-deficient — infinitely many $\mathbf{w}$ fit the training data exactly. "We return to this in Lecture 04."

Lecture 03 built the classical bias-variance theory entirely on the *other* side of that boundary — and closed by handing us the tool to cross it: feature engineering, $\phi(\mathbf{x}) \in \mathbb{R}^{\tilde{d}}$ with $\tilde{d}$ our choice, plus its $O(\tilde{d})$ cost problem.

Today we cross that line on purpose.

This lecture: three steps past the $d + 1 > n$ line

1. **Kernel methods**: computing in high dimensions without paying for it — feature engineering stays tractable when the feature space is enormous, or infinite.
2. **Double descent**: past the interpolation threshold — the experiment classical theory has nothing to say about, and a real surprise.
3. **Implicit regularization**: why the surprise isn't chaos — which of many perfect fits gradient descent actually finds.
4. **Wrap-up**: one causal arc, closed.

feature engineering (Lecture 03) $\longrightarrow$ kernel methods $\longrightarrow$ double descent $\longrightarrow$ implicit regularization

Recap from Lecture 03: feature engineering, and n vs. $\tilde{d}+1$

Fix a map $\phi : \mathbb{R}^d \to \mathbb{R}^{\tilde{d}}$, fit a linear model on $\phi(\mathbf{x})$: $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x})$, still linear in $\mathbf{w}$ (same loss, convexity, closed form), nonlinear in $\mathbf{x}$.

$\Phi \in \mathbb{R}^{n \times \tilde{d}}$ (rows $\phi(\mathbf{x}_i)^\top$) plays X's role, so the invertibility question is n vs. $\tilde{d}+1$ — and $\tilde{d}$ is whatever we picked ϕ to produce

$M = 9$, $N = 15$:
$\tilde{d}+1 = 10 < 15$, underparameterized (Lecture 03's demo)

$M = 20$, same $N = 15$:
$\tilde{d}+1 = 21 > 15$, $\Phi^\top \Phi$ rank-deficient, on purpose

prerequisite double descent (Section 2) needs.

Recap: explicit features cost $O(\tilde{d})$, and the maps we want most are infinite

Forming $\phi(\mathbf{x})$ and computing $\mathbf{w}^\top \phi(\mathbf{x})$ costs $O(\tilde{d})$ per input.

dozens

fine (Lecture 03's polynomials)

millions

painful (all pairwise interactions of a few thousand raw features)

infinite

impossible: $\phi(\mathbf{x})$ cannot be written down at all

today's first job: compute everything the fit needs in feature space **without ever forming** $\phi(\mathbf{x})$ — Section 1 shows some of the most useful feature maps are exactly the infinite-dimensional ones

1. Kernel methods

Computing in high dimensions without paying for it

Ridge in feature space: the primal form costs $O(\tilde{d}^3)$, the dual $O(n^3)$

Lecture 03's ridge regression, replacing X with Φ everywhere, has two algebraically equivalent closed forms:

primal

$$\mathbf{w}^\star = \left(\Phi^\top \Phi + n\lambda I_{\tilde{d}}\right)^{-1} \Phi^\top \mathbf{y}$$

$O(\tilde{d}^3 + n\tilde{d}^2)$: form $\Phi^\top\Phi$ $(\tilde{d} \times \tilde{d})$, invert it

dual

$$\mathbf{w}^\star = \Phi^\top \alpha^\star, \qquad \alpha^\star = \left(\Phi\Phi^\top + n\lambda I_n\right)^{-1} \mathbf{y}$$

$O(n^3 + \tilde{d}n^2)$: form $\Phi\Phi^\top$ $(n \times n)$, invert it

The cheaper side depends on whether $\tilde{d} \gg n$ or $n \gg \tilde{d}$ — and the primal is unusable once $\tilde{d}$ is huge, meaningless once it is infinite.

The primal-dual identity, verified

The two forms agree because of the identity

$$(\Phi^\top \Phi + n\lambda I_{\tilde{d}})^{-1}\Phi^\top = \Phi^\top(\Phi\Phi^\top + n\lambda I_n)^{-1}$$

Verification. Multiply both sides on the left by $\Phi^\top\Phi + n\lambda I_{\tilde{d}}$ and on the right by $\Phi\Phi^\top + n\lambda I_n$:

both sides reduce to the same expression:

$$\Phi^\top\Phi\Phi^\top + n\lambda\Phi^\top$$

which confirms the identity.

$\mathbf{w}^\star$ always lies in the span of the $\phi(\mathbf{x}_i)$: the representer theorem

Whichever form is cheaper, the dual exposes the fact that matters for what follows, **always**, regardless of λ:

$$\mathbf{w}^\star = \Phi^\top \alpha^\star = \sum_{i=1}^{n} \alpha_i^\star \phi(\mathbf{x}_i) \;\in\; \operatorname{span}\{\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)\}$$

Special to squared loss? No — it holds for *any* loss:

> **Theorem** (Kimeldorf and Wahba, 1971; generalized by Schölkopf, Herbrich, Smola, 2001). For any loss ℓ and any $\lambda > 0$, every minimizer of $\hat{\mathcal{R}}(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 = \frac{1}{n}\sum_{i=1}^n \ell(\mathbf{w}^\top \phi(\mathbf{x}_i), \mathbf{y}_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$ can be written $\mathbf{w}^\star = \sum_{i=1}^n \alpha_i^\star \phi(\mathbf{x}_i)$ for some $\alpha^\star \in \mathbb{R}^n$.

So the same reduction works for *any* ERM-plus-ℓ_2-penalty problem: ridge, logistic regression with an ℓ_2 penalty, SVMs (Lecture 05).

Proof: split w into in-span and orthogonal parts; only the penalty sees the latter

Step 1: split any candidate w. Decompose $\mathbf{w} = \mathbf{w}_{\parallel} + \mathbf{w}_{\perp}$, where $\mathbf{w}_{\parallel} \in \text{span}\{\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)\}$ and $\mathbf{w}_{\perp}$ is orthogonal to that span.

Step 2: the loss term doesn't see $\mathbf{w}_{\perp}$. Since $\phi(\mathbf{x}_i)$ lies in the span and $\mathbf{w}_{\perp}$ is orthogonal to it,

$$\mathbf{w}^{\top}\phi(\mathbf{x}_i) = \mathbf{w}_{\parallel}^{\top}\phi(\mathbf{x}_i) + \mathbf{w}_{\perp}^{\top}\phi(\mathbf{x}_i) = \mathbf{w}_{\parallel}^{\top}\phi(\mathbf{x}_i),$$

so $\hat{\mathcal{R}}(\mathbf{w})$ depends only on $\mathbf{w}_{\parallel}$, never on $\mathbf{w}_{\perp}$.

Step 3: the penalty term only ever hurts. By the Pythagorean theorem, $\|\mathbf{w}\|_2^2 = \|\mathbf{w}_{\parallel}\|_2^2 + \|\mathbf{w}_{\perp}\|_2^2 \geq \|\mathbf{w}_{\parallel}\|_2^2$, with equality iff $\mathbf{w}_{\perp} = 0$.

Step 4: conclude. Replacing any minimizer $\mathbf{w}^{\star}$ by its in-span component $\mathbf{w}_{\parallel}^{\star}$ achieves an objective value no larger — so some minimizer has $\mathbf{w}_{\perp}^{\star} = 0$, i.e. lies entirely in $\text{span}\{\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)\}$ □

The dual computation only ever needs inner products: the kernel trick

The representer theorem says $\mathbf{w}^\star$ only ever needs $\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)$, combined linearly.

- $\Phi\Phi^\top$'s (i, j) entry is $\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$ — an inner product, not $\phi(\mathbf{x}_i)$ or $\phi(\mathbf{x}_j)$ individually
- A new prediction, $\phi(\mathbf{x})^\top \mathbf{w}^\star = \sum_{i=1}^n \alpha_i^\star \, \phi(\mathbf{x})^\top \phi(\mathbf{x}_i)$ — again, only inner products, never a bare $\phi(\mathbf{x})$

$$\kappa(\mathbf{x}, \mathbf{x}') := \phi(\mathbf{x})^\top \phi(\mathbf{x}')$$

The entire kernel trick: compute $\kappa(\mathbf{x}, \mathbf{x}')$ *directly*, as a function of $\mathbf{x}, \mathbf{x}'$, without ever forming $\phi(\mathbf{x})$ or $\phi(\mathbf{x}')$ as explicit vectors.

Kernel ridge regression: train and predict with κ only

Kernel matrix $K \in \mathbb{R}^{n \times n}$, $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$:

$$\alpha^\star = (K + n\lambda I_n)^{-1}\mathbf{y}, \qquad f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^\star \, \kappa(\mathbf{x}, \mathbf{x}_i)$$

train and predict, both using κ only — ϕ never appears explicitly. A new prediction is a weighted sum of kernel similarities to every training point

if κ can be evaluated cheaply, fitting and predicting cost nothing extra for large or infinite $\tilde{d}$ — Lecture 03's $O(\tilde{d})$ term disappears, replaced by whatever κ itself costs

Kernel logistic regression: same recipe, gradient descent on α (added Sep 17)

The representer theorem covers the ℓ_2-penalized logistic loss too. Substitute $\mathbf{w} = \sum_j \alpha_j \phi(\mathbf{x}_j)$: score on training point i is $\mathbf{w}^\top \phi(\mathbf{x}_i) = (K\alpha)_i$, penalty $\frac{\lambda}{2}\|\mathbf{w}\|^2 = \frac{\lambda}{2}\alpha^\top K\alpha$, so

$$J(\alpha) = \frac{1}{n}\sum_{i=1}^{n}\Big[-y_i \log\sigma\big((K\alpha)_i\big) - (1-y_i)\log\big(1-\sigma((K\alpha)_i)\big)\Big] + \frac{\lambda}{2}\alpha^\top K\alpha.$$

Chain rule through the scores $\mathbf{z} = K\alpha$, with Lecture 02's $\partial\ell/\partial z_i = \sigma(z_i) - y_i$ and K symmetric:

$$\nabla J(\alpha) = K\Big[\tfrac{1}{n}\big(\sigma(K\alpha) - \mathbf{y}\big) + \lambda\alpha\Big]$$

no closed form (as in Lecture 02): run GD on $\alpha \in \mathbb{R}^n$, $O(n^2)$ per step through K; predict with $\mathbb{P}(y = 1 \mid \mathbf{x}) = \sigma\big(\sum_i \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i)\big)$ — ϕ never appears. HW2 implements this with the RBF kernel

Worked examples: the linear kernel, and a polynomial kernel at half the cost

Linear kernel $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$: the trivial case, $\phi(\mathbf{x}) = \mathbf{x}$, plain linear regression recovered exactly.

Polynomial kernel, $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^3$, $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}')^2$. Expand directly:

$$(x_1 x_1' + x_2 x_2' + x_3 x_3')^2 = x_1^2 x_1'^2 + x_2^2 x_2'^2 + x_3^2 x_3'^2 + 2x_1 x_2 x_1' x_2' + 2x_1 x_3 x_1' x_3' + 2x_2 x_3 x_2' x_3'$$

Six terms, each a product of something in $\mathbf{x}$ and the same thing in $\mathbf{x}'$ — an inner product of two six-dimensional vectors, so κ is exactly $\phi(\mathbf{x})^\top \phi(\mathbf{x}')$ for

$$\phi(\mathbf{x}) = \left(x_1^2,\ x_2^2,\ x_3^2,\ \sqrt{2}x_1 x_2,\ \sqrt{2}x_1 x_3,\ \sqrt{2}x_2 x_3\right)$$

evaluating κ directly: $O(3)$ vs. forming ϕ and dotting: $O(6)$

A small saving here — but $\tilde{d}/d$ grows fast: degree-M polynomials in d raw dimensions give $\tilde{d} = \binom{d+M}{M}$.

Worked example: the RBF kernel has an infinite-dimensional ϕ, yet costs $O(1)$

Scalar case, for clarity: $\kappa(x, x') = e^{-(x-x')^2}$. Expand the exponent and use the Taylor series of exp:

$$e^{-(x-x')^2} = e^{-x^2} e^{-x'^2} e^{2xx'} = e^{-x^2} e^{-x'^2} \sum_{k=0}^{\infty} \frac{(2xx')^k}{k!}$$

$$= \sum_{k=0}^{\infty} \left(e^{-x^2} \sqrt{\tfrac{2^k}{k!}}\, x^k \right) \left(e^{-x'^2} \sqrt{\tfrac{2^k}{k!}}\, x'^k \right)$$

$$\text{so} \quad \phi(x) = \left(e^{-x^2},\ e^{-x^2}\sqrt{2}\,x,\ e^{-x^2}\sqrt{2}\,x^2/\sqrt{2!},\ \dots \right)$$

$\phi(x)$ is **infinite**-dimensional — one coordinate per power of x; you could never form it directly. Yet $\kappa(x, x')$ is a single exponential, $O(1)$: exactly the case Lecture 03 said feature engineering could not reach explicitly — reached anyway

Is a candidate κ a valid kernel? Mercer's condition, and the easy direction

Given some candidate similarity function κ: does some feature map ϕ with $\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$ actually exist?

Theorem (Mercer, 1909). κ is a valid kernel if and only if, for every finite set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, the Gram matrix K with $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ is positive semi-definite.

Forward direction (immediate). If $\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$, then for any $\mathbf{v} \in \mathbb{R}^n$:

$$\mathbf{v}^\top K \mathbf{v} = \sum_{i,j} v_i v_j \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) = \left\| \sum_i v_i \phi(\mathbf{x}_i) \right\|_2^2 \geq 0$$

The useful direction gives a checkable criterion; closure properties extend it

the reverse direction is the one that matters in practice — PSD-checking on the Gram matrix is a **sufficient** certificate, not just necessary (proved by constructing a feature space from κ itself)

The polynomial and RBF kernels are valid because their construction already exhibits a ϕ. Two closure properties (each provable by an explicit feature-map construction) extend validity to combinations, without exhibiting ϕ by hand each time:

if κ_1, κ_2 are valid kernels and $a, b \geq 0$: $a\kappa_1 + b\kappa_2$ is a valid kernel, and $\kappa_1\kappa_2$ (pointwise product) is a valid kernel

2. Double descent

Past the interpolation threshold

The classical prediction: past $\tilde{d} + 1 = n$, interpolation, then only worse

Lecture 03's bias-variance theory, as $\tilde{d}$ grows: bias keeps falling, variance keeps rising, and past some point variance dominates so completely that test error explodes.

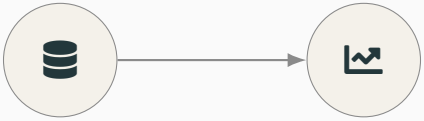
exactly what Lecture 03's own $M = 9$, $N = 15$
demo showed: test MSE 4.48, over 100× the noise floor

Once $\tilde{d} + 1$ exceeds n, the model **interpolates**: it fits every training point exactly, training error is 0.

the intuition: things can only get worse from here — more capacity, more freedom to chase noise, worse generalization, presumably forever

That's a prediction, not a proof. Let's actually run past the line and see.

Extending Lecture 03's demo past the interpolation threshold



$N = 20$ points now (more room to see both regimes)

sweep M well past $N-1 = 19$

$\mathbf{y} = \sin(2\pi x) + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, 0.04)$, $x \sim \text{Unif}[0, 1]$ — Lecture 03's exact setup.

interpolation threshold: the point where the number of parameters ($M+1$) first reaches the number of training points (N), so the fit can begin to pass through every point exactly. Here: $M = N - 1 = 19$

A numerical note: minimum-norm interpolants, in a well-conditioned basis

- Past the threshold, infinitely many $\mathbf{w}$ interpolate — we fit the minimum-ℓ_2-norm one, via the pseudoinverse (Section 3 explains why this is the natural choice, not an arbitrary one)
- Raw polynomial powers become too ill-conditioned past a few dozen degrees — this section's numbers use Legendre polynomials on rescaled $x \in [-1, 1]$, evaluated in this lecture's Colab notebook

A re-parameterization of the identical degree-M polynomial model class, not a different model class.

Up to the threshold: the classical U-shape, then a catastrophic spike at $M = 19$

M	$M{+}1$	**Train MSE**	**Test MSE**
5	6	0.0107	0.052
6	7	0.0106	**0.052 (best)**
12	13	0.0072	0.165
15	16	0.0054	295
18	19	0.0010	1.15×10^{7}
19	**20 = N**	**0.0000**	$\mathbf{2.07 \times 10^{9}}$

- Bottoms out at $M = 5, 6$: test MSE ≈ 0.052, close to the noise floor $\mathrm{Var}[\varepsilon] = 0.04$
- Degrades steadily past that, exactly as classical theory predicts
- Blows up catastrophically right at $M = 19 = N - 1$ — the interpolation threshold

the worst point on the entire curve, nine orders of magnitude above the good-fit regime — exactly where classical intuition says the story should end

Well past the threshold: test error falls more than nine orders of magnitude

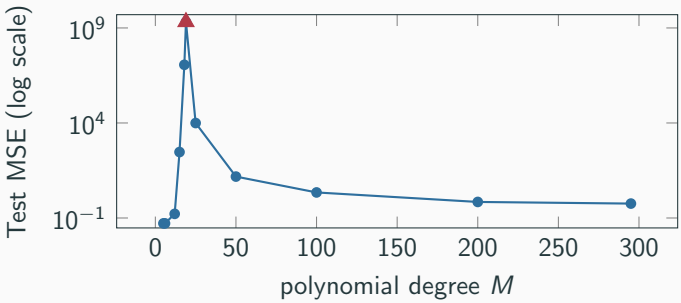
M	$M{+}1$	**Test MSE**
19 (threshold)	20	2.07×10^9
25	26	9908
50	51	15.1
100	101	2.21
200	201	0.699
295	296	0.573

Continue the same sweep, now with M ranging into the hundreds, $M + 1 \gg N = 20$.

test error falls **more than nine orders of magnitude** from the threshold's peak to $M = 295$

A real, substantial **second descent** — well past the point where classical theory predicted only catastrophe.

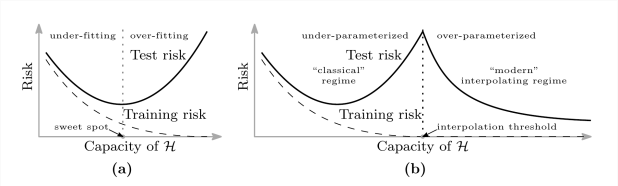
The whole curve: double descent (Belkin, Hsu, Ma, Mandal, 2019)



Real numbers from both tables above — not a stylized invention. Red triangle: the interpolation threshold, $M = 19$.

Double descent: test error, against model capacity, first follows the classical U-shape, spikes at the interpolation threshold, then descends a *second* time in the overparameterized regime beyond it.

The paper's own figure: the classical U-shape is the first of two descents



Belkin, Hsu, Ma, Mandal (2019), Figure 1. (a) the classical U-shaped risk curve. (b) the double descent risk curve, extending it past the interpolation threshold.

Panel (a), the "classical" regime, is exactly what we saw up through the threshold; panel (b) adds the second, "modern" regime beyond it, with the interpolation threshold as the join.

the double-descent curve does not *replace* the classical U-shape — it *contains* it, as the first of two descents

An honest caveat: how far it recovers depends on the feature family

in this experiment, the second descent's floor (≈ 0.573 at $M = 295$) sits **above** the classical regime's best point (≈ 0.052 at $M = 5, 6$) — it falls dramatically, but does not fully recover

- Belkin et al. and later work document settings (e.g. random-feature or neural-network model families) where the second descent *does* surpass the classical optimum
- The robust phenomenon: a genuine second descent, rather than monotonic degradation forever
- Exactly how far it descends is not universal — and should not be overclaimed from one demo

Why, a first-pass intuition: one wild interpolant at the threshold, many past it

At the interpolation threshold there is exactly *one* (or very few) way to fit the data exactly.

and that one way is typically a wild, high-norm function — exactly $M = 19$'s row, with its 9-orders-of-magnitude error spike

Past the threshold the picture changes:

there are *many* interpolating solutions, and among them are increasingly well-behaved, low-norm ones

Which one a fitting procedure actually lands on, among the many available, turns out to matter enormously.

So which interpolating solution does plain gradient descent actually find?

3. Implicit regularization

Why interpolation isn't chaos

The puzzle: infinitely many perfect fits, and gradient descent picks a good one

Past the threshold, *infinitely many* **w** achieve zero training error — any point in the affine space of exact interpolants.

- Some are catastrophic (huge norm, generalizes arbitrarily badly)
- Plain gradient descent on squared error, **no explicit penalty term**, reliably finds a well-behaved one
- Why?

> **implicit bias** (Gunasekar, Lee, Soudry, Srebro, 2018): a systematic preference among equally-good solutions, coming from *how* we optimize, with nothing added to the loss to cause it

The answer is a fact about the optimization dynamics themselves, not about the objective being minimized.

Gradient descent from zero: every gradient lies in row(Φ)

Plain (unregularized) squared error, starting from $\mathbf{w}_0 = 0$:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \, \nabla \hat{\mathcal{R}}(\mathbf{w}_t), \qquad \nabla \hat{\mathcal{R}}(\mathbf{w}_t) = \frac{1}{n} \Phi^\top (\Phi \mathbf{w}_t - \mathbf{y})$$

every gradient is of the form $\Phi^\top$(something) — a linear combination of the *rows* of Φ, i.e. of $\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)$

$$\text{row}(\Phi) := \text{span}\{\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)\}$$

The same span that showed up in Section 1's dual form and representer theorem.

Every iterate stays in row(Φ) forever, so the limit does too

$\mathbf{w}_0 = 0$ is trivially in row(Φ), and each update only ever adds another vector from that same span.

every iterate $\mathbf{w}_t$ stays in row(Φ) for all t — by induction on the update rule alone, regardless of how long it runs or how overparameterized the problem is

Now suppose gradient descent converges to *some* interpolating solution $\mathbf{w}_\infty$ (zero training error) — one of the puzzle's infinitely many candidates, and one that lies in row(Φ). Which one?

The only interpolant in row(Φ) is the minimum-norm one, so that is $\mathbf{w}_\infty$

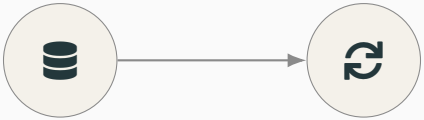
Among *all* interpolating solutions, exactly one lies in row(Φ) and is orthogonal to everything outside it — by the same Pythagorean argument as Section 1's representer-theorem proof, that point is exactly the **minimum-ℓ_2-norm** interpolating solution.

Every gradient-descent iterate lies in row(Φ), so $\mathbf{w}_\infty$ does too; and $\mathbf{w}_\infty$ interpolates, by assumption. Hence:

> **Gradient descent from $\mathbf{w}_0 = 0$ on an overparameterized least-squares problem converges to the minimum-norm interpolating solution among all of them.**

An implicit bias coming purely from the optimization trajectory — with no $\frac{\lambda}{2}\|\mathbf{w}\|_2^2$ term anywhere in the objective.

Verified numerically: two million GD steps land on the pseudoinverse solution



$M = 30, 31$ params, well past $N = 20$'s threshold

full-batch GD from $\mathbf{w}_0 = 0$, 2,000,000 iterations

	Train MSE	$\|\mathbf{w}\|_2$	Null-space component
GD, $t = 0$	0.215	1.11	2.0×10^{-16}
GD, $t = 10^4$	1.13×10^{-3}	6.78	1.7×10^{-13}
GD, $t = 10^6$	1.74×10^{-6}	42.33	4.6×10^{-12}
GD, $t = 2 \times 10^6$ (final)	2.5×10^{-9}	43.90	4.6×10^{-12}
Pseudoinverse min-norm $\mathbf{w}^\star$	0 (exact)	43.96	—

Reading the table: the null-space component is zero, and GD matches the exact solution

null-space component: the projection of $\mathbf{w}_t$ onto the orthogonal complement of row(Φ), which the row-space argument predicts is exactly 0 — it never exceeds 5×10^{-12} across two million iterations, floating-point noise, not a real component

the final weight vector matches the exact pseudoinverse minimum-norm solution to within $\|\mathbf{w}_{GD} - \mathbf{w}^\star\|_2 = 0.062$ (against norms of $\approx$ 44), and their test errors match almost exactly: 76.98 (GD) vs. 77.20 (pseudoinverse)

The row-space argument is not just a plausible story — it is what the numbers do.

Constructing worse interpolants on purpose: equally perfect fits, test error five orders apart

The training data admits interpolating solutions with *far* larger norm: $\mathbf{w}^\star + c\,\mathbf{v}$ with $\Phi\mathbf{v} = 0$, since adding a null-space vector never changes a training prediction, for *any* scalar c.

	$\|\mathbf{w}\|_2$	Train MSE	Test MSE
Min-norm $\mathbf{w}^\star$ (found by GD)	44.0	0 (exact)	77.2
$\mathbf{w}^\star + 100\mathbf{v}$	109	0 (exact)	555
$\mathbf{w}^\star + 1000\mathbf{v}$	1001	0 (exact)	47,843
$\mathbf{w}^\star + 10{,}000\mathbf{v}$	10,000	0 (exact)	4.78×10^6

every row fits the training data *exactly* — **equally perfect** fits — but test error grows **nearly five orders of magnitude** with the norm; gradient descent lands on the top row

Connecting back to double descent: the second descent is systematic, not luck

Section 2 left open why test error, past the interpolation threshold, *descends* rather than continuing to explode.

- Interpolation alone says nothing about generalization (the table above); gradient descent does not sample interpolants arbitrarily — it systematically prefers the minimum-norm one
- The *minimum achievable* norm among interpolators tends to *shrink* as parameters grow well past the threshold — more directions in $\text{row}(\Phi)$ to distribute the same interpolation requirement across

it is specifically gradient descent's **minimum-norm bias** that keeps an overparameterized fit sane: implicit regularization is the **mechanism** behind the second descent — a **systematic phenomenon**, not luck

4. Wrap-up

One causal arc, closed

The arc, in one line each; Lecture 02's oldest forward pointer closed

Feature engineering (Lecture 03): choose $\tilde{d}$ freely $\Rightarrow$ cross Lecture 02's line on purpose

Kernel methods: makes it tractable even at $\tilde{d} = \infty$ — ϕ never appears explicitly

Implicit regularization: why that second descent isn't chaos — gradient descent's minimum-norm bias

Double descent: the empirical payoff — test error descends a second time

$d+1 > n$ is not a failure mode to patch around — it is a regime with its own theory, its own empirical surprises, and its own reason (implicit bias, not luck) that things still work

Next time

Lecture 05 needs kernels directly —
SVMs are kernelized the same way

Lecture 07 revisits today's implicit bias under a new name, **inductive bias** — generalized from a property of the optimizer to a property of the model architecture itself.

HW2, covering Lectures 03–04, announced today.

References

- Kimeldorf, Wahba. *Some Results on Tchebycheffian Spline Functions.* Journal of Mathematical Analysis and Applications, 33(1), 1971.
- Mercer. *Functions of Positive and Negative Type, and Their Connection with the Theory of Integral Equations.* Philosophical Transactions of the Royal Society A, 209, 1909.
- Belkin, Hsu, Ma, Mandal. *Reconciling Modern Machine-Learning Practice and the Classical Bias-Variance Trade-off.* PNAS, 116(32), 2019.
- Gunasekar, Lee, Soudry, Srebro. *Characterizing Implicit Bias in Terms of Optimization Geometry.* ICML 2018 (PMLR 80).

Kernel methods content (Section 1) adapted from EPFL's `ML_course`, used with permission; double descent and implicit regularization written directly from the primary literature.

Full citation details: `references/sources/lecture04-representer-mercer-citations.md`, `lecture04-double-descent-implicit-regularization-citations.md`, `epfl-l04-feature-engineering-kernels-double-descent.md` (same folder)