




Machine Learning for Physical Sensing & Perception

CMSC 818V : Fall 2026

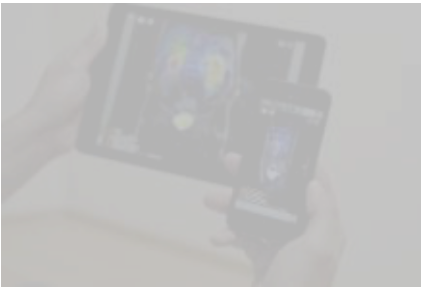
Lecture 4: Learning in Inverse problems

Nirupam Roy

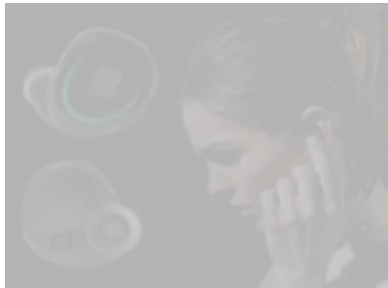
COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

Tu-Th 12:30 – 1:45pm

IRB 2107







Where Lecture 3 left off

$$y = f(x, \theta) + n$$

- A physical forward model turns a state x into a measurement y.
- The classical inverse recovers x by fitting the data and penalizing implausible solutions.

$$\hat{x} = \arg\min_x \; \mathbf{D(y, f(x,\theta))} \; + \; \lambda \mathbf{R(x)}$$

- data term — stay consistent with what the sensor actually measured
- regularizer — prefer plausible states among those consistent with the data

Lecture 4 is not a switch from physics to generic deep learning — it is the next question forced by the inverse problem itself.

The central lecture question

Which component of the inference chain is still missing?

When the classical inverse is inaccurate, non-unique, computationally expensive, or based on incomplete physics — should learning...

- replace the inverse outright,
- supply the missing prior,
- accelerate / unroll the solver,
- repair the forward model,
- discover the dynamics, or
- only enforce a known physical law?

Six answers, one per section of this lecture

1

Learn the Inverse
the map itself is too complex

2

Learn the Prior
data alone doesn't pick one state

3

Unroll the Optimizer
the objective is right; iteration is slow

4

Learn Missing Physics
the model is incomplete, biased

5

Learn the Dynamics
the transition law itself is unknown

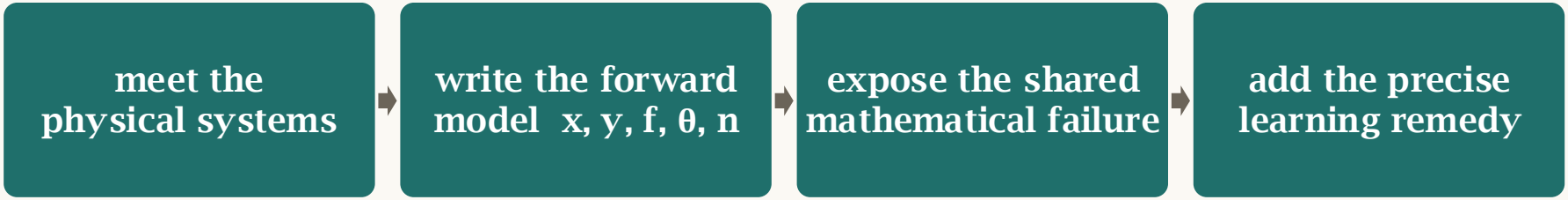
6

Constrain with Physics
the law is known; the field is sparse

Same rhythm every time: meet the physical systems, expose the shared failure mode, add exactly the learning that failure demands.

Every section follows the same rhythm

Two or three very different sensing systems will turn out to share exactly one mathematical failure — that repetition, not the sensor hardware, is the actual lesson.



Notation stays fixed throughout: physical state x, forward physics f(x,θ), measurement y, noise n, learned component φ. New vocabulary is the sensor's — not the mathematics'.

Watch for the moment each section closes: what the new method still cannot guarantee becomes the transition question into the next one.

1 Learn the Inverse Directly

When a useful inverse mapping exists statistically, even though an analytic inverse is unavailable

A formal definition: the direct learned inverse

Use this to establish the simplest possible relationship between Machine Learning and inverse problems: the network is an approximator of the map from sensor measurements to a desired latent physical state.

$$y = f(x, \theta) + n \quad \rightarrow \quad \hat{x} = g_{\varphi}(y)$$

- Given paired data $\{(y_i, x_i)\}$, $g\varphi$ is fit directly by empirical risk minimization — no explicit optimization runs at test time.
- This can be useful even when the analytic inverse is hard to write down or too slow to invert online.

Supervised training and a physical consistency check

$$\hat{\varphi} = \arg\min_{\varphi} \; (1/N) \Sigma_i \; L(g_{\varphi}(\mathbf{y}_i), \mathbf{x}_i)$$

For squared error, the predictor is driven toward the conditional mean E[x | y]. This can be useful even when the analytic inverse is hard — but it does not ensure that the predicted state actually reproduces the observed measurement.

physical consistency check: $f(x, \theta) \;\; ?\approx \;\; y$

A direct learned inverse can be accurate statistically without being an explicit model of the sensing physics. It may exploit reproducible correlations that are useful in-distribution but brittle when the sensing environment changes.

Example A — sparse-view CT: what a detector measures



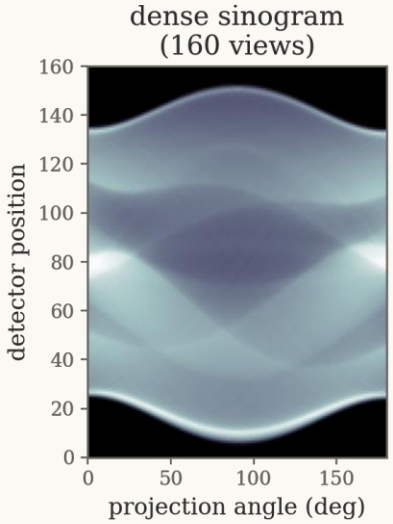
- An X-ray source and detector rotate around the body; each detector value is produced after X-rays pass through tissue.
- With only a small number of projection angles, the scanner records incomplete angular information.

$y \approx Ax + n$ *(after Beer–Lambert log transform; A encodes ray–pixel intersection lengths)*

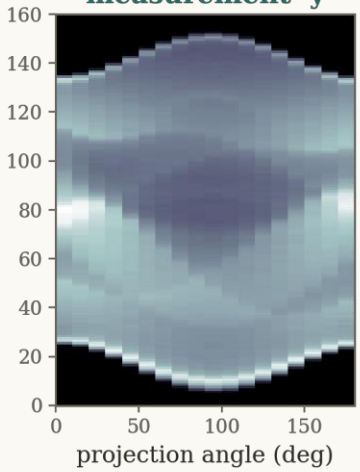
CT measurement formation: dense to sparse

ground truth x

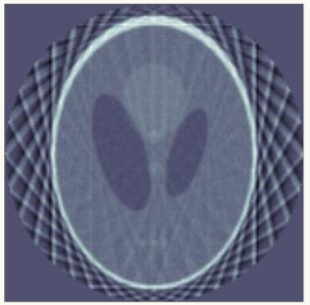




sparse sinogram (18 views)
measurement y



filtered backprojection
from 18 views



Filtered backprojection assumes sufficiently dense angular sampling. With few views, the reconstruction shows streaks and large null / near-null directions — several images become almost equally consistent with the data.

Learned direct reconstruction



A network g_φ can be trained to map sparse projection data directly to the image, or to map an initial artifact-laden reconstruction to a corrected image.

What the network has learned that is not in the sparse sinogram?

[Ref] Zhu et al., AUTOMAP, Nature 2018 — a canonical learned-reconstruction example.

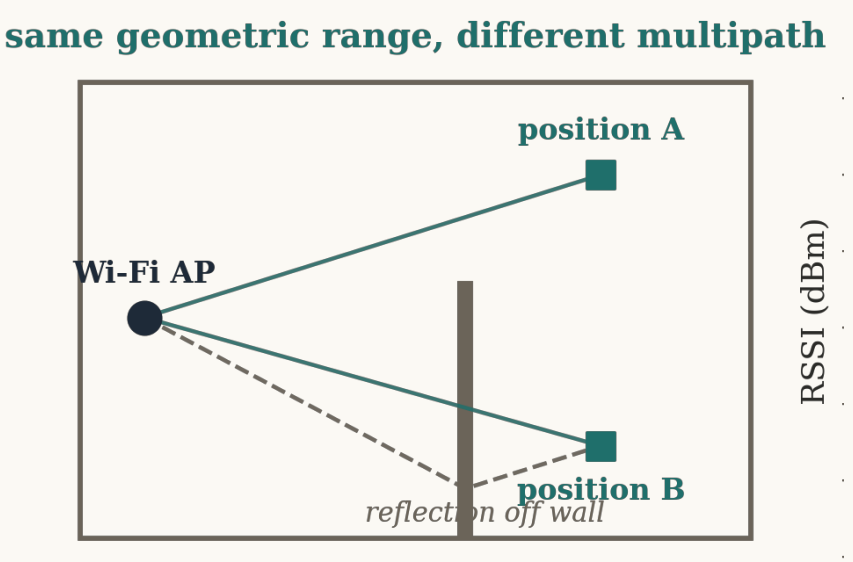
Example B — RF fingerprint localization: the physical environment

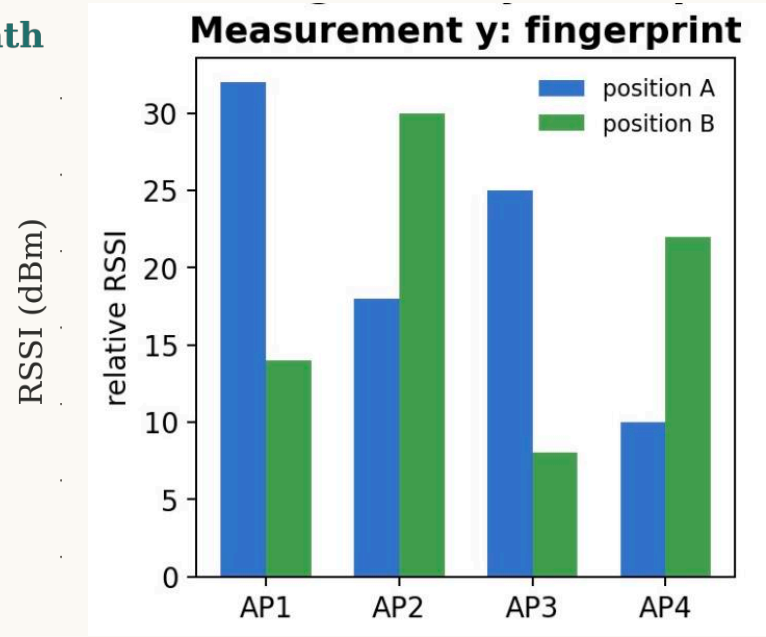
- A device receives Wi-Fi / UWB / cellular signals from several fixed transmitters (access points).
- Walls, furniture, and people create site-specific reflection, diffraction, and shadowing.
- Two locations with similar geometric ranges can therefore have very different channel signatures.

$$y = f_{site}(x) + n \qquad f_{site} \text{ unknown / too costly to model}$$

A complete electromagnetic / multipath forward model exists in principle but is too costly and too environment-specific for routine inversion. Classical triangulation's simplified distance-to-signal model is what multipath violates.

From fingerprint to position: a site-specific inverse





gφ learns the empirical inverse from fingerprint vector to position. Contrast "geometry-based localization" with "fingerprint-based inverse inference" — what happens when furniture or AP placement changes?

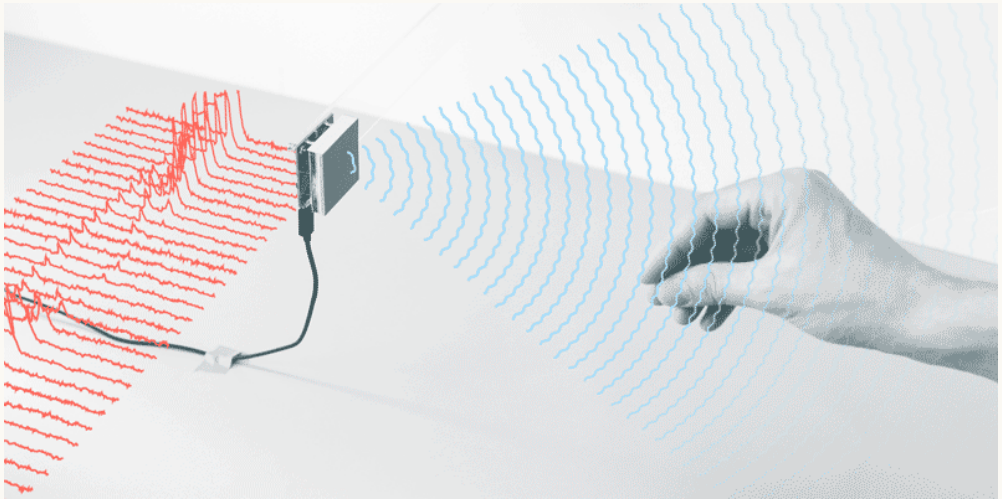
[Ref] Wang et al., DeepFi, MobiCom 2015.

Example C — radar gesture sensing: the physical measurement

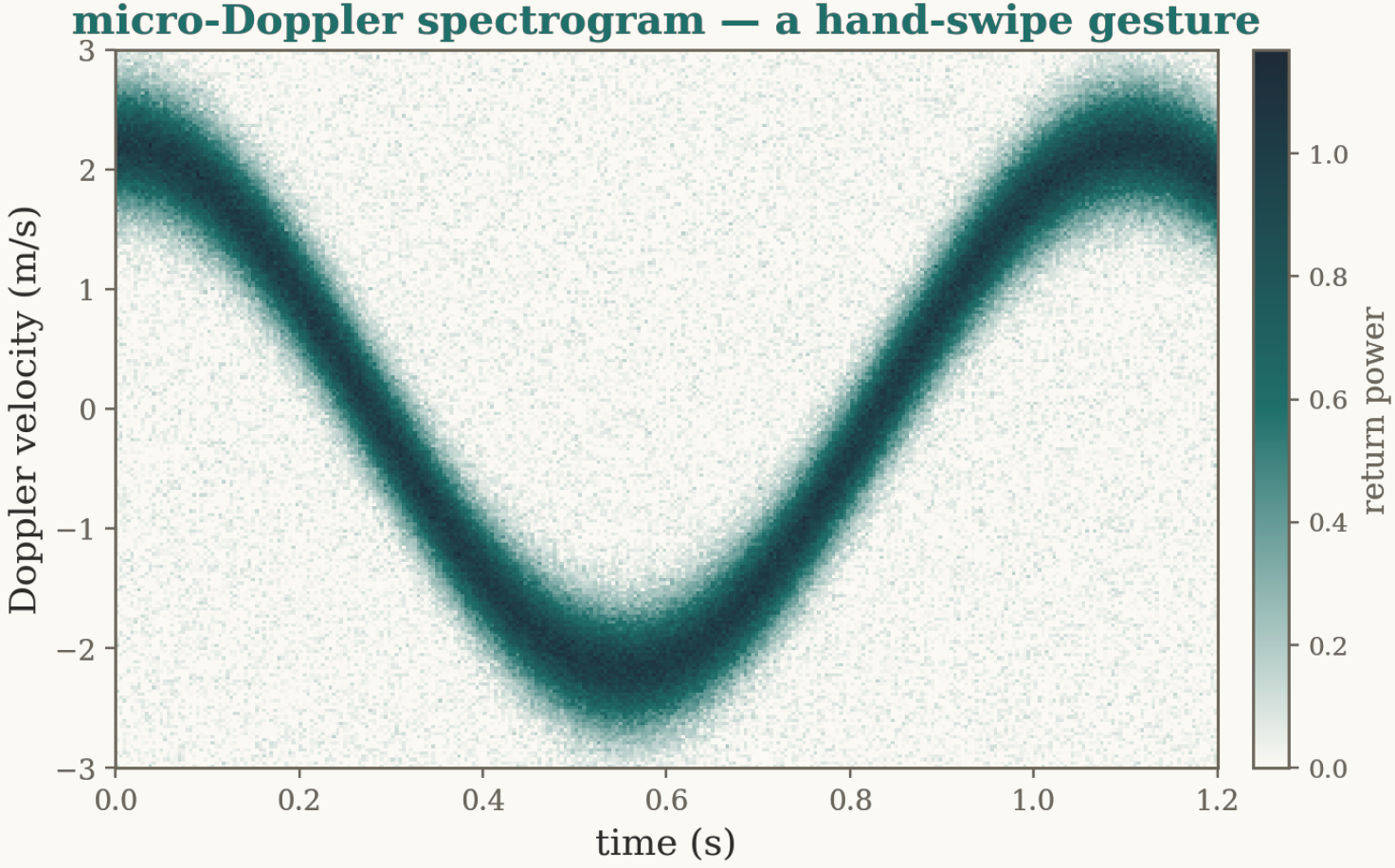
FMCW radar (chirps) → moving scatterers (hand / body) → range / Doppler / angle features

- Different body parts create time-varying reflections with different range and radial velocity.
- Raw beat signals are transformed by known radar signal processing into range / Doppler / angle features — exact inversion from distributed scattering to semantics is not analytically defined.

Broadens "inverse problem": the target x can be a low-dimensional state or label, not only a pixelwise image.



Radar gesture sensing: inverse inference need not mean image reconstruction



The full physical reflectivity field is more detailed than the task requires — gφ maps the structured radar measurement directly to a semantic, task-relevant state, not a full reflectivity image.

Three systems, one learned-inverse abstraction

System	measurement y	target x	why direct learning helps	what it may ignore
CT	sparse projections	attenuation image	learn difficult inverse	known projector A / consistency
RF localization	RSSI / CSI	position	site-specific inverse	causal propagation / domain shift
Radar gesture	radar tensor	activity / gesture	task-level inverse	full scene physics

Why direct learning is not enough

- A direct inverse is trained on a particular distribution of states and environments — performance can degrade sharply out of distribution.
- Nothing in the training objective enforces that $f(x_{\square}, \theta)$ actually reproduces the observed measurement y.
- Measurement consistency and generalization are both left entirely to the data.

If the forward operator A is already known and the real difficulty is that the measurements are insufficient, why learn the entire inverse? Can we keep measurement physics exact and learn only what makes a state plausible?

2 Learn the Prior, Not the Entire Inverse

When many physical states explain the same measurements equally well

Two sources of information in reconstruction

$$\hat{x} = \arg\min_x \; D(y, Ax) + \lambda R(x)$$

sensor physics A
D(y, Ax): does x explain what the sensor measured?

prior knowledge R(x)
is x plausible, independent of this measurement?



reconstruction $\hat{x}$

The measurement term knows the experiment; the prior knows the world.

Ambiguity lives in the null space

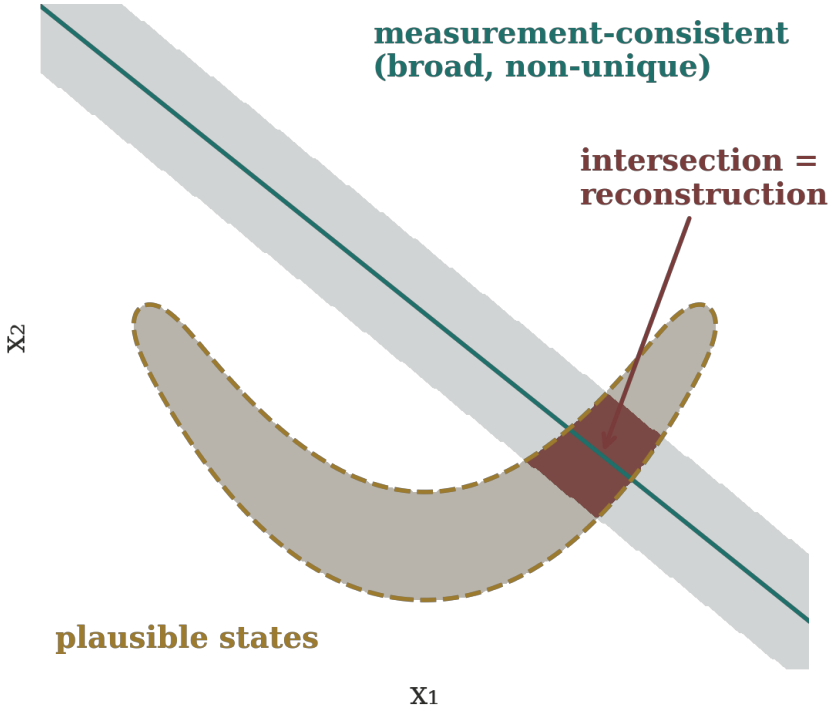
$$y = Ax + n$$

With limited or weak measurements, A can be rank-deficient or ill-conditioned. If z lies in the null space, then:

$$Az = 0 \quad \Rightarrow \quad A(x + z) = Ax$$

- Even when the null space is only approximate, small singular values create directions that are weakly observed and therefore unstable under noise.
- The likelihood / data term can tell us whether a candidate explains y — never whether it is a plausible physical state.

Why is a prior needed at all?



- Undersampling, null spaces, ill-conditioning, noise, missing frequencies, limited views, occlusion…
- …all let many states x explain the same y.
- Data consistency narrows the set; it may not select one stable answer.
- Reconstruction seeks the intersection with the plausible set.

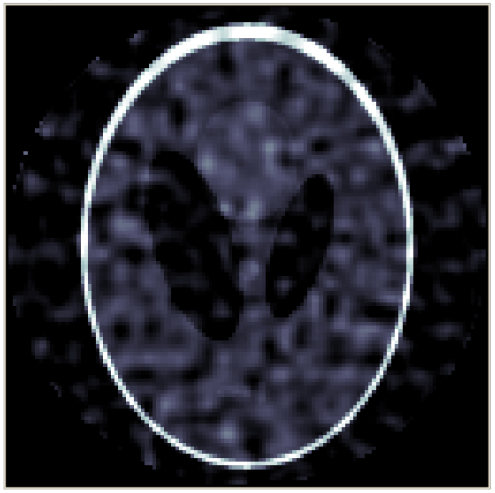
Without a prior, "the" solution of an inverse problem is often not even defined.

Sparse-view CT: fitting data is not enough

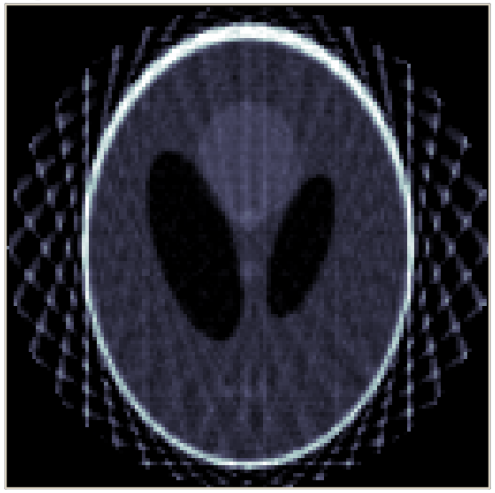
true image



alternative image
projection mismatch 0.3%



filtered backprojection
projection mismatch 10.0%



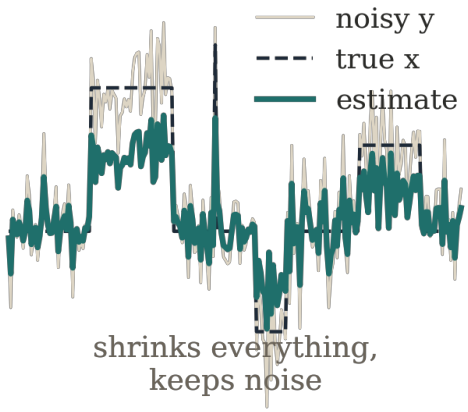
18 projection angles. The middle image was built by adding a (near) null-space pattern of A to the truth: its projections differ by only 0.3%, yet the anatomy is visibly different. FBP matches the data worse (10%) and shows streaks.

The sensor cannot tell these images apart — something else must.

Classical priors are assumptions

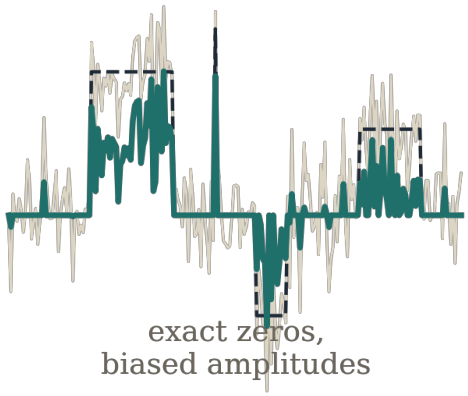
$R(x) = \|x\|_2^2$ $R(x) = \|x\|_1$ $R(x) = TV(x)$

$\|x\|^2_2$: small energy



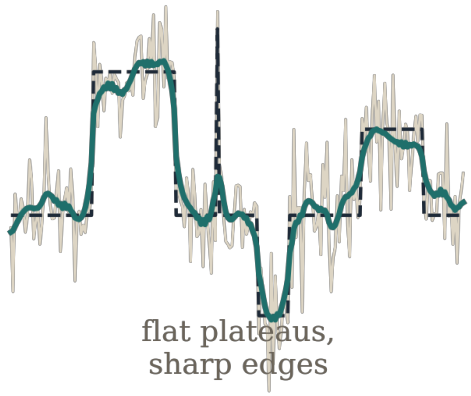
shrinks everything,
keeps noise

$\|x\|_1$: sparsity



exact zeros,
biased amplitudes

TV(x) : piecewise smooth



flat plateaus,
sharp edges

- ☐ Hand-designed: small energy (Tikhonov), sparsity, piecewise smoothness.
- ☐ Powerful because they encode structure — intentionally simple.
- ☐ Who decided “sparse”, “smooth”, or “piecewise smooth”? We did.

Real anatomy, RF occupancy patterns and room impulse responses carry richer structure than these can express.

Learn the prior, not the whole inverse

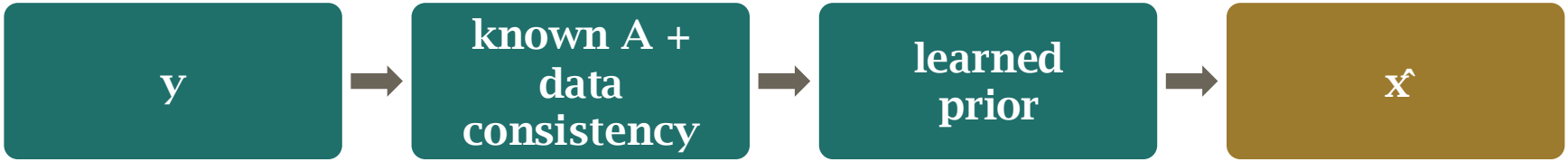
$$\textit{direct inverse:}\quad \hat{x} = g_{\varphi}(y)$$

$$\textit{learned prior:}\quad \hat{x} = \arg\min_{x} D(y, Ax) + \lambda\, \boldsymbol{R_{\varphi}(x)}$$

labels teach the entire $y \rightarrow x$ map



physics stays explicit; data only teach what x looks like



- Smaller sample burden: the network need not rediscover A.
- More interpretable, and measurement consistency stays explicit.

Three technical forms of learned prior

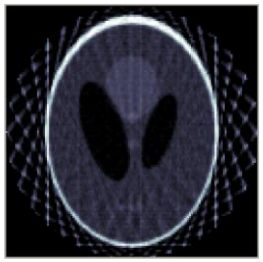
1 Explicit regularizer	2 Learned proximal	3 Generative / score
$R_\varphi(x)$	$D_\varphi(x)$	$G_\varphi(z)$, $s_\varphi(x, t)$
outputs a scalar	outputs a corrected x	outputs x, or a direction
how implausible is x?	*how should x be improved?*	*where do plausible x live?*

All three keep A. They differ in what the network outputs, and so in how the prior enters the solver.

Form 1: an explicit learned regularizer

$$\hat{x} = \arg\min_x \; D(y, Ax) + \lambda R_\varphi(x)$$

$$R_\varphi : R^n \to R$$



candidate x

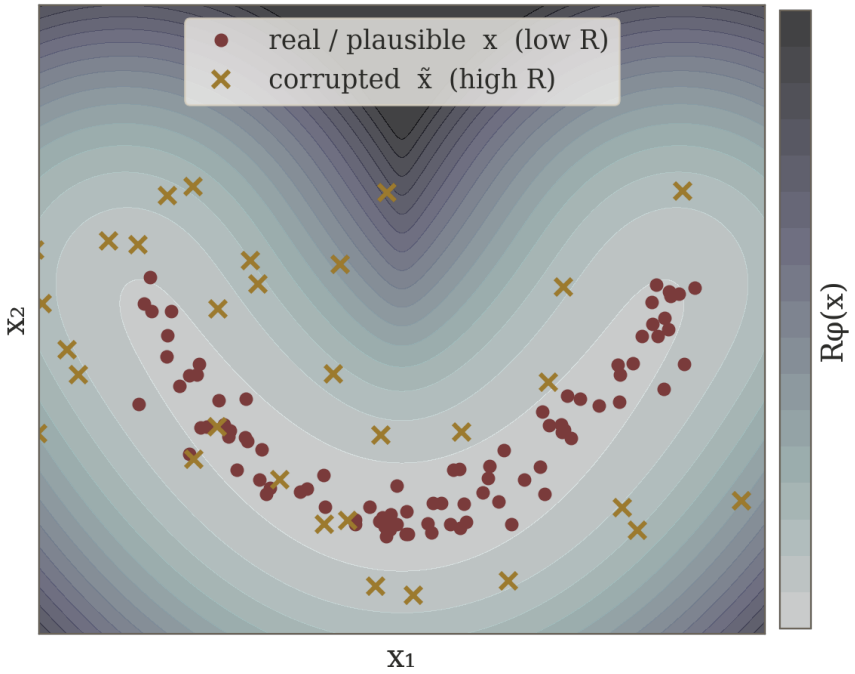
→ **network Rφ** →



one scalar "energy"

- Input: a candidate state or image x. Output: one number.
- Low Rφ(x): plausible. High Rφ(x): implausible.
- The optimizer stays explicit; only the prior term is learned.

How can the explicit regularizer be trained?



$$R_\varphi(x_{real}) < R_\varphi(\tilde{x}_{bad})$$

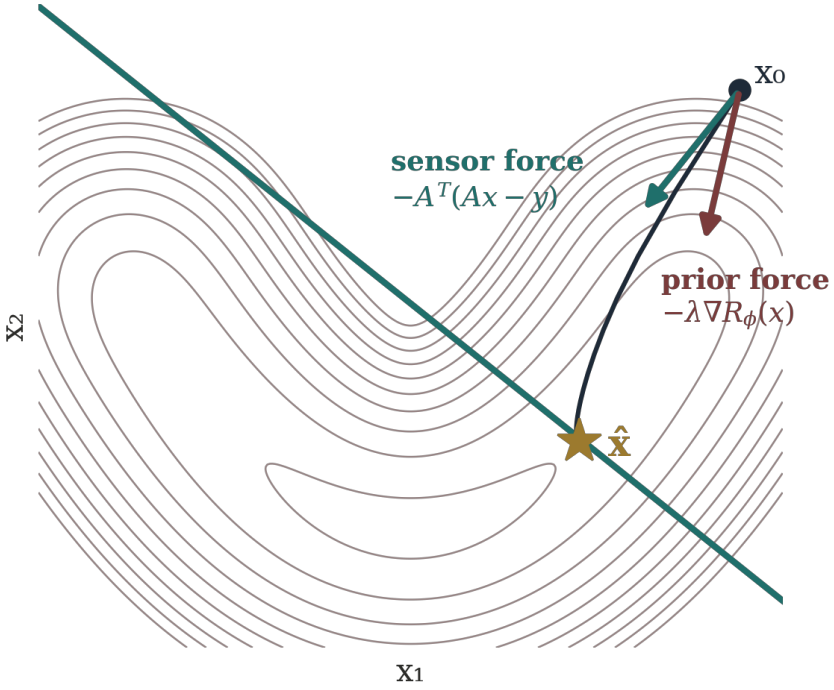
$$\text{view: } R_\varphi(x) \approx -\log p_\varphi(x)$$

- Separate: train Rφ on real vs. corrupted / unrealistic states (ranking or energy loss), freeze, plug in.
- End-to-end: unroll the reconstruction and train Rφ for final reconstruction quality.

Once Rφ is frozen, the inverse solver only needs its gradient with respect to x.

What does the optimizer do with a scalar?

$$x_{k+1} = x_k - \eta \left[A^T(Ax_k - y) + \lambda \nabla_x R_\varphi(x_k) \right]$$

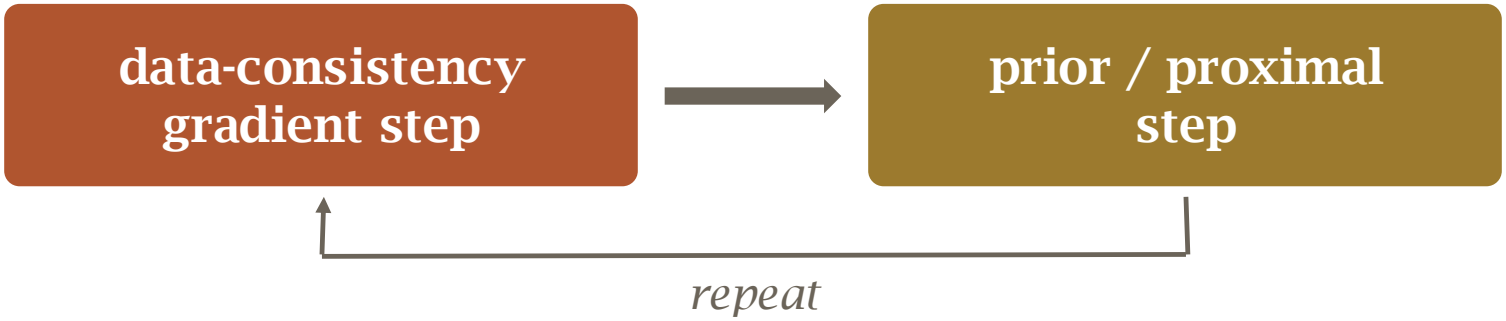


- Sensor force: pull toward the measured data.
- Prior force: the direction that reduces learned implausibility.
- The network never outputs the reconstruction — it shapes the landscape.

Sparse-view CT: one force pushes toward the projections, the other toward plausible anatomy.

Form 2: Learn the Proximal/Denoiser (proximal is a mechanism, not "sparsity")

$$z_k = x_k + \eta\, A^T(y - Ax_k), \qquad x_{k+1} = prox_{\eta\lambda R}(z_k)$$



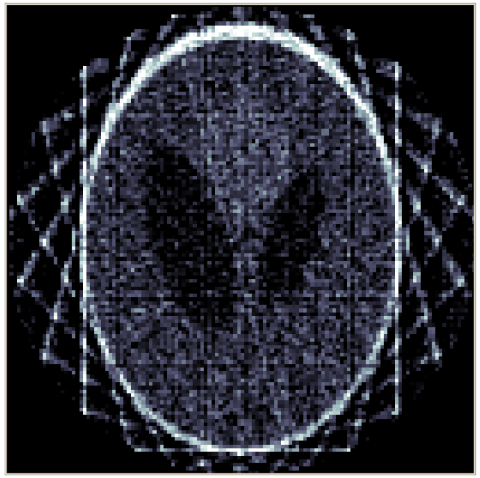
Prior R(x)	Its proximal step
$\|x\|_1$ (sparsity)	soft-thresholding (ISTA)
TV(x)	a TV-denoising problem
other priors	a different correction operator

"Proximal" names how a prior is enforced — broader than sparsity or any activation function.

Learn the proximal action instead of R(x)

$$x_{k+1} = \boldsymbol{D_{\varphi}}(x_k + \eta\, A^T(y - Ax_k))$$

streaky estimate (FBP)
PSNR 14.1 dB



data step + denoiser, ×150
PSNR 25.6 dB



ground truth

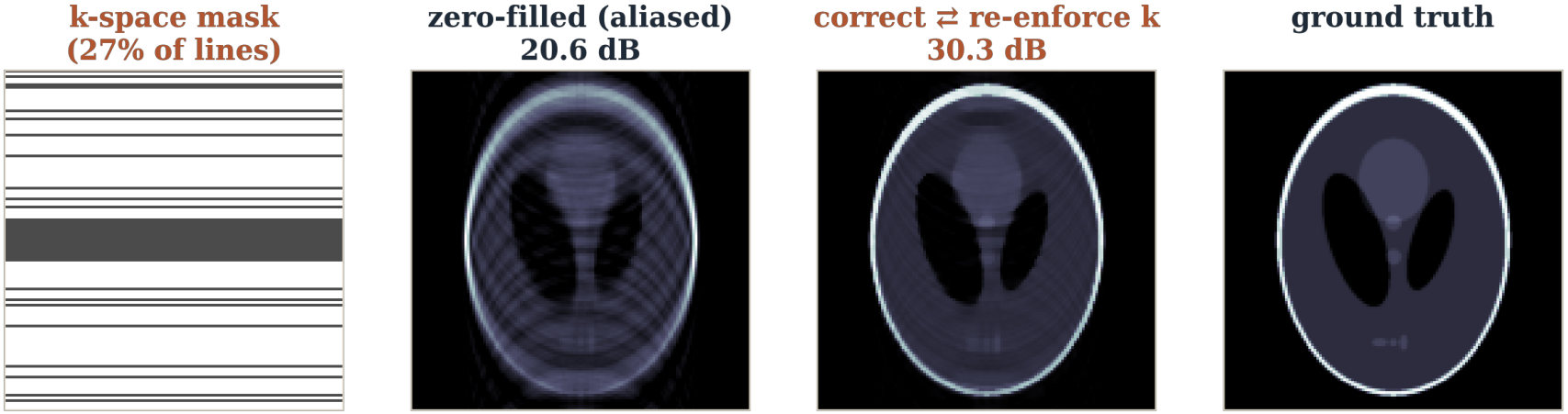


- Input: a measurement-corrected but imperfect z_k. Output: a corrected state of the same size.
- Dφ may be a CNN, U-Net, ResNet or transformer.
- "Denoising" includes noise, aliasing, streaks, ringing — any artifact.

How is a learned denoiser trained?

$$\textit{separate:} \quad \min_{\varphi} \|D_{\varphi}(\tilde{x}) - x^*\|^2$$

$$\textit{end-to-end:} \quad \min_{\varphi} \|x_K(y;\varphi) - x^*\|^2$$



- Separate: make corrupted / clean pairs, train Dφ, freeze it, insert it in the loop.
- End-to-end: unroll K steps and train Dφ for this inverse task — task-specific correction, not generic denoising.

MRI loop: undersampled k-space → data consistency → learned correction → repeat.

Explicit regularizer vs. learned proximal

$R_\varphi(x)$

"How bad is this x?"

outputs a scalar

enters through ∇R_φ in the objective

$D_\varphi(x)$

"How should I improve this x?"

outputs a corrected signal / image

replaces the proximal step directly

Both are learned priors. The denoiser route is often easier to deploy: it learns the correction action directly.

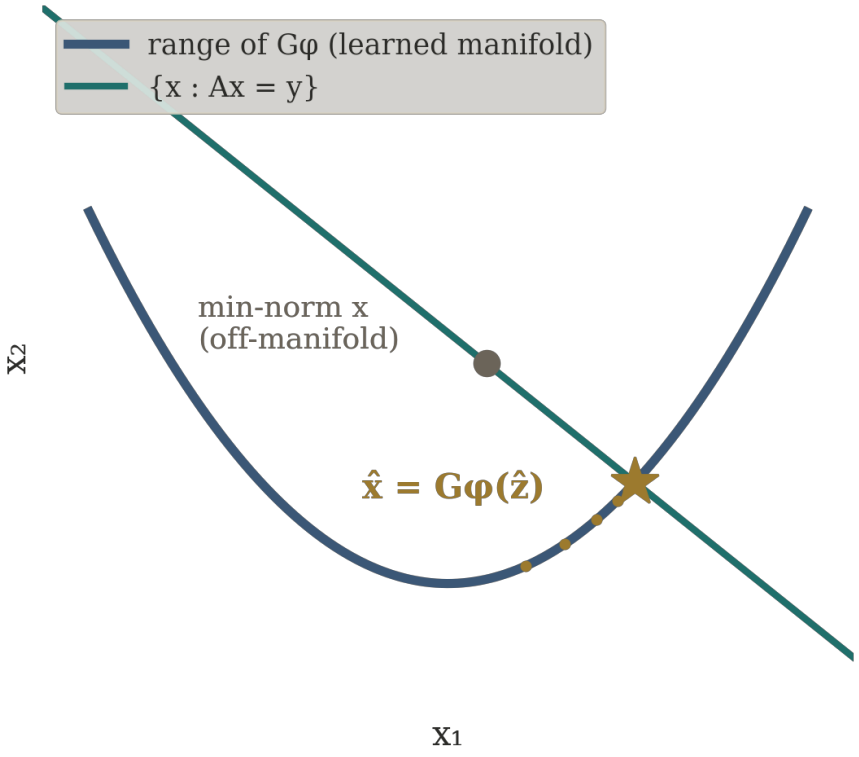
Form 3: a generative prior

$$z \sim p(z), \qquad x = G_\varphi(z)$$

$$\hat{z} = \arg\min_z D(y, AG_\varphi(z))$$

$$\hat{x} = G_\varphi(\hat{z})$$

- G is trained on examples of plausible states.
- Reconstruction is restricted to what G can produce.
- Far stronger than “smooth” or “sparse”.



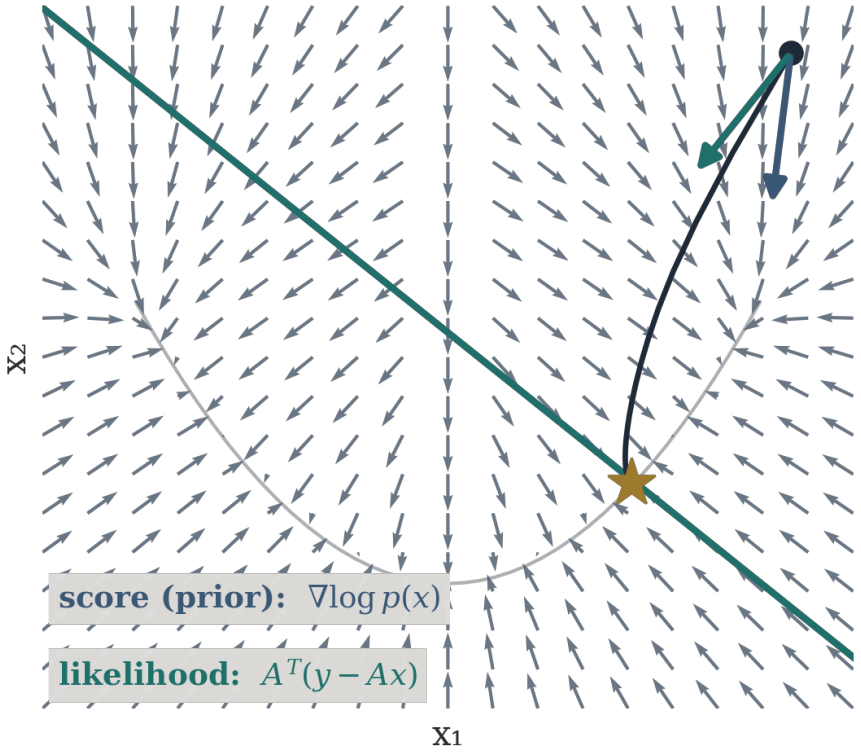
CT: search only among anatomically plausible images, then pick the one that best explains the projections.

Score prior: a direction toward probability

$$s_\varphi(x, t) \approx \nabla_x \log p_t(x)$$

$$\nabla \log p(x|y) = \nabla \log p(y|x) + \nabla \log p(x)$$

- Input: noisy state x_t and noise level t.
- Output: a vector the size of x, pointing to more probable states.
- Gaussian noise: likelihood direction $\propto A^T(y - Ax)$.



Posterior direction = measurement consistency + learned plausibility — the same two forces, now as vectors.

Training input / output summary

Model	Input	Output	Training signal
Explicit R_φ	state x	scalar $R_\varphi(x)$	plausible < implausible
Learned proximal D_φ	imperfect $\tilde{x}$	corrected x	corrupted→clean, or end-to-end
Generator G_φ	latent z	plausible x	distribution of real x
Score s_φ	noisy x_t, t	direction	clean x + controlled noise

In every row the forward model A is not learned — it stays in the solver.

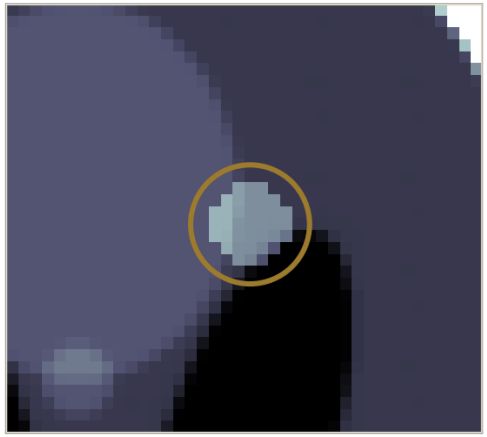
How each form enters the inverse

Form	How it is used	Intuition
Explicit regularizer	add to objective, differentiate	score plausibility
Learned proximal	replace the proximal step	correct toward plausible
Generative prior	search the generator's range	stay on learned manifold
Score prior	add to likelihood direction	climb toward high probability

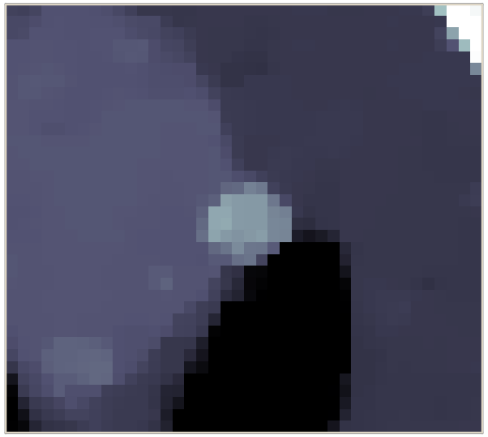
Spend most class time on the first two rows — they build directly on MAP, regularization, ISTA and LISTA.

Plausibility must not overpower evidence

truth with a small, unusual lesion



balanced prior data mismatch 1.4%



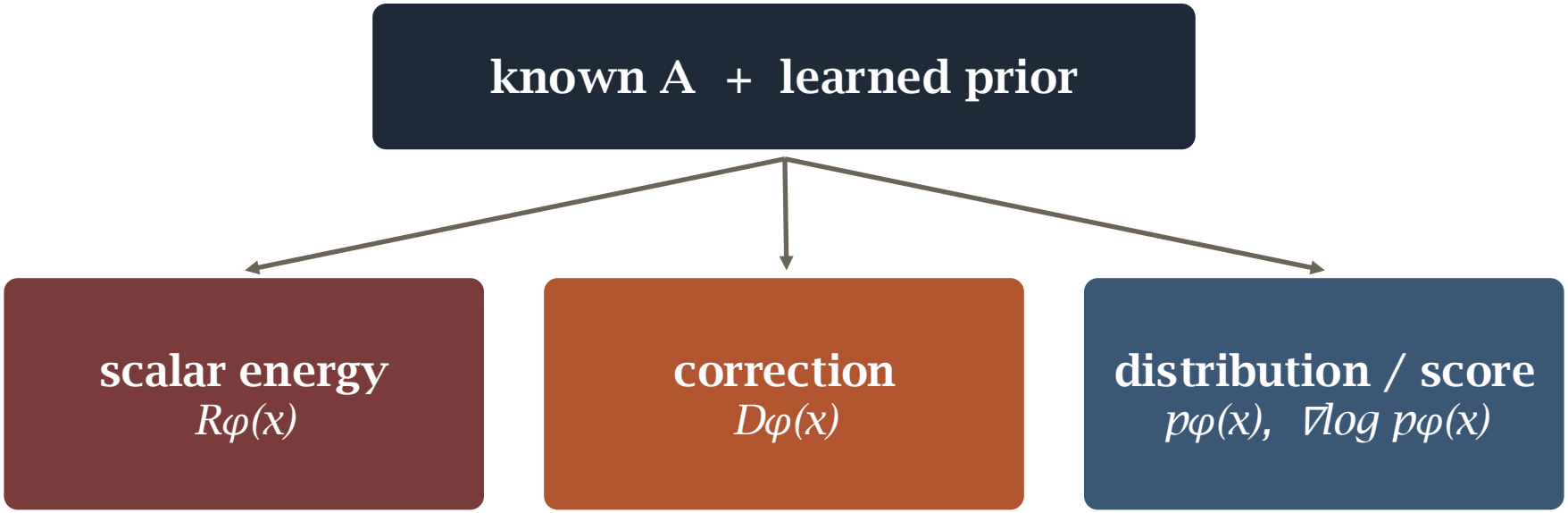
prior too strong data mismatch 7.9%



- Too large a λ or too strong a prior hides unusual but real structure (here a small lesion).
- A strong prior can give a plausible-looking answer that no longer matches the measurements (1.4% → 7.9% mismatch).
- Keep measurement consistency explicit whenever the forward model is trustworthy.

Known physics → what can explain y. Learned prior → which explanations are plausible.

Summary: three ways to learn the prior



Next: learned iterative reconstruction — train the prior block and parts of the optimizer together, while the sensing physics stays explicit.

3 Learn / Unroll the Optimizer

When the objective is principled but repeated numerical optimization is too slow

Keep the objective. Learn the solver.

The purpose of this section: show that a neural-network architecture can be derived from a known numerical algorithm — the most concrete bridge from optimization to deep learning in this course.

principled objective, slow generic solver → **unroll K iterations into K trainable stages**

The architecture inherits the inverse algorithm rather than being chosen arbitrarily.

Where does this sit relative to LISTA?



LISTA: learn the solver

learned prior: learn x

- LISTA learns how to optimize a sparse problem faster.
- Learned priors learn what plausible x looks like.
- Unrolled methods (MoDL) do both: A fixed, prior block learned, steps optionally learned.

Two independent axes: "what is learned?" — the solver, the prior, or both.

3 Learn / Unroll the Optimizer

a **Gradient descent:** local slope gives the direction

b **Curvature:** L sets the step scale; least squares L = БАБ$_2$²

c **Backprop:** differentiates a composed graph w.r.t. parameters

d **Unrolling:** K iterations м K-stage graph; tying is separate

e **ISTA:** gradient on the data term, prox on the sparse prior

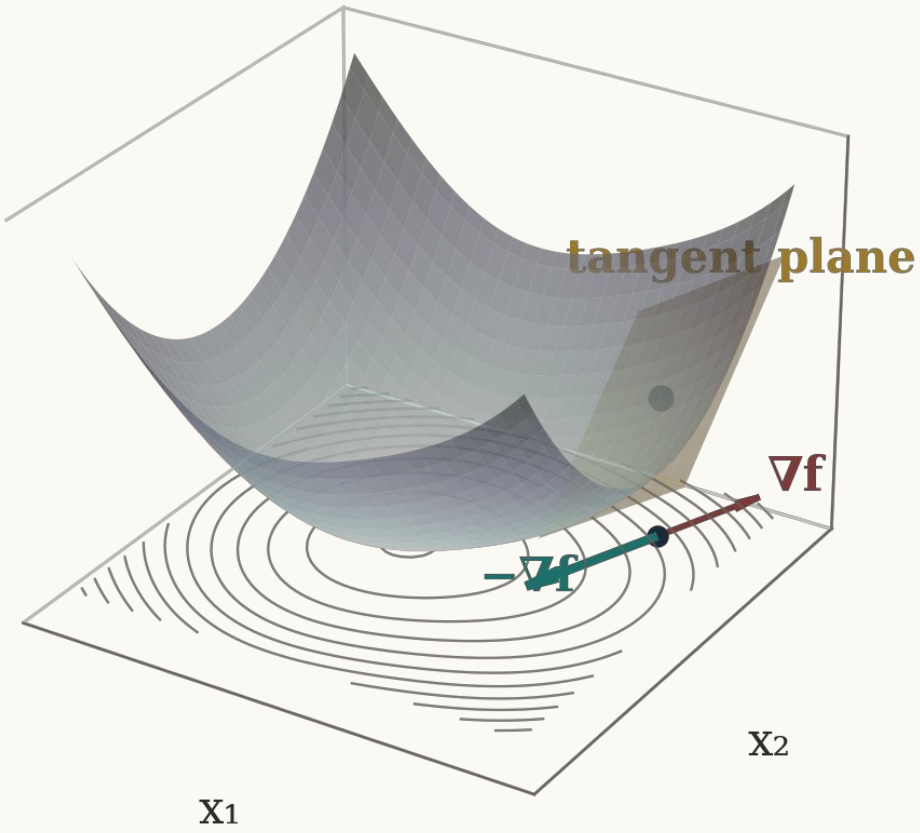
f **LISTA:** the ISTA layer as a trainable finite-depth solver

g **Radar DOA:** A = array physics, A□ (Ax − y) = backprojected error

3.a) Gradient Descent and Backpropagation

Start with geometry, not the update equation

Gradients



$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x$$

$$\Delta x = -\alpha \nabla f(x)$$

$$f(x + \Delta x) \approx f(x) - \alpha \| \nabla f \|_2^2$$

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

The tangent plane is a local linear model; $-\nabla f$ is its steepest descent.

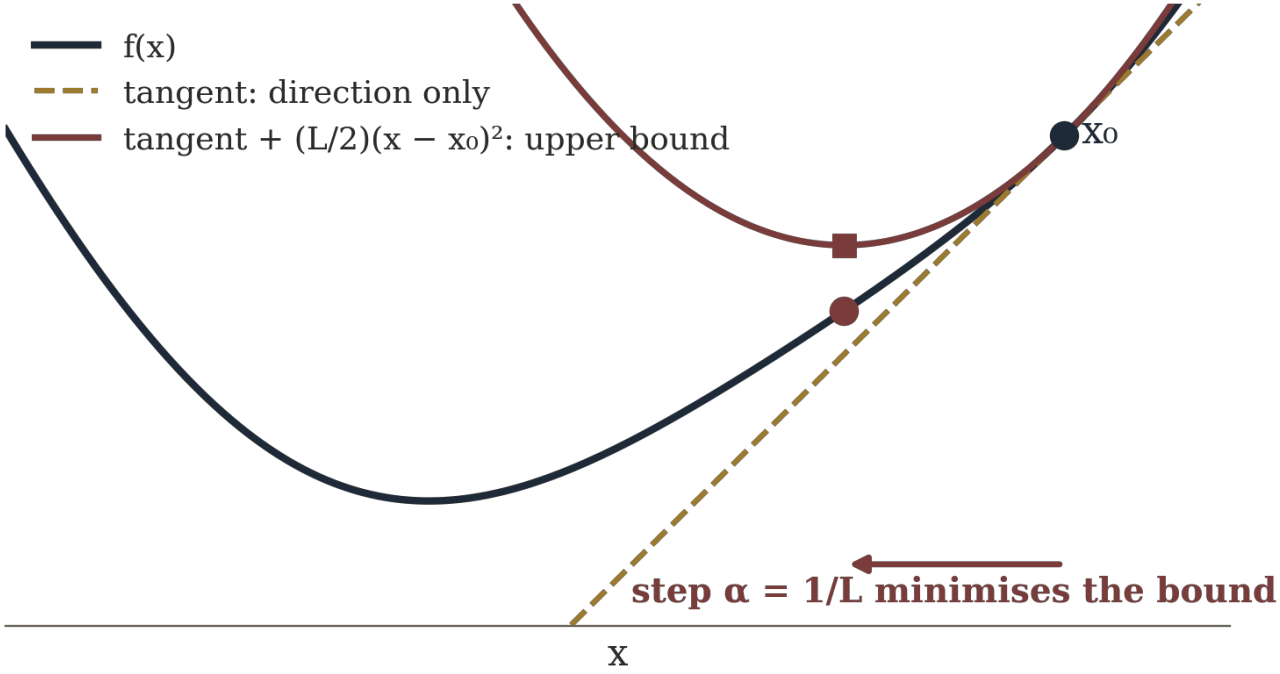
"The first-order picture gives the direction — not how far the tangent model stays trustworthy."

3.b) Step Size from Curvature

The learning rate is not born empirically

Tangent -> direction. Curvature -> distance.

$$f(x - \alpha g) - f(x) \approx -\alpha \|g\|^2 + (\alpha^2/2)\, g^T H g, \qquad H = \nabla^2 f$$



Large curvature makes the quadratic term grow fast: a step can overshoot even when it points the right way.

Lipschitz gradient: a global curvature bound

$$\|\nabla f(x) - \nabla f(z)\| \le L\|x - z\|$$

$$f(x + \Delta) \le f(x) + \nabla f(x)^T\Delta + (L/2)\|\Delta\|^2 \qquad \textit{(descent lemma)}$$

$$f(x_{k+1}) \le f(x_k) - \alpha(1 - \alpha L/2)\|\nabla f(x_k)\|^2$$

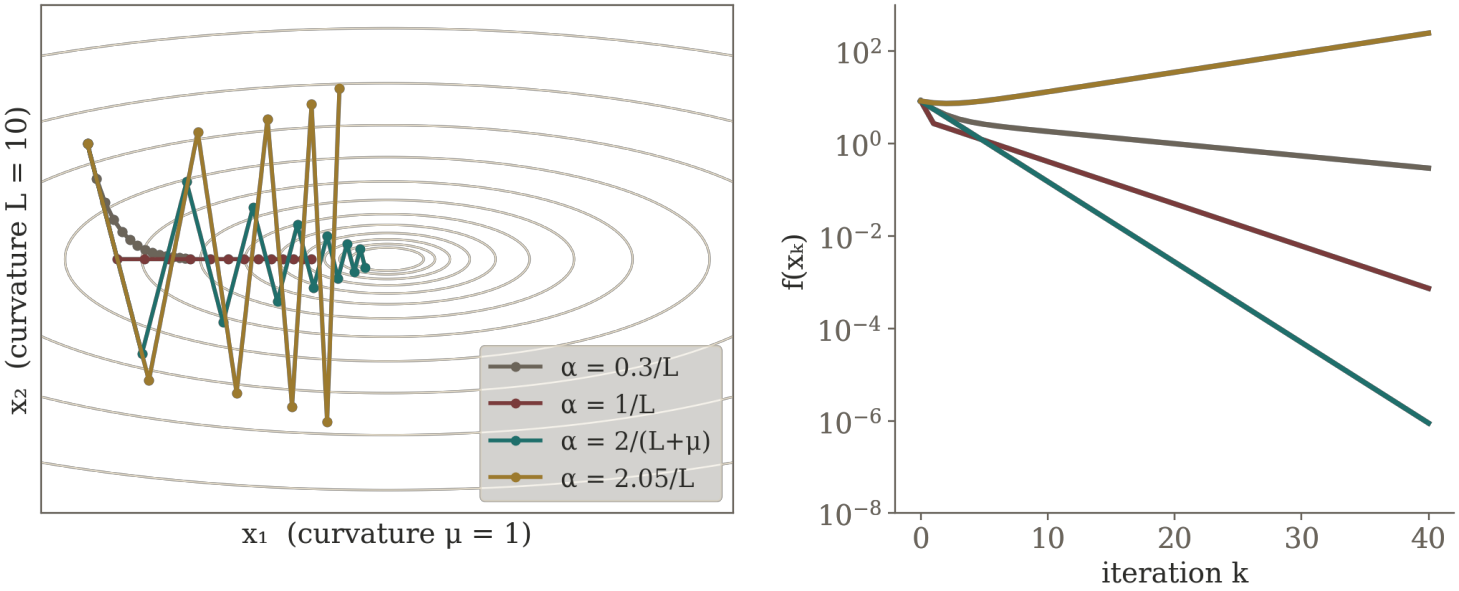
Step size	What the bound gives
$0 < \alpha < 2/L$	guaranteed decrease (smooth GD)
$\alpha \le 1/L$	standard conservative choice
$0 < \alpha \le 1/L$	canonical safe step for ISTA

When L is unknown: backtracking line search.

Least squares: the bound becomes concrete

$$f(x) = \tfrac{1}{2}\|Ax - y\|^2, \quad \nabla f = A^T(Ax - y), \quad H = A^TA$$

$$L = \lambda_{max}(A^TA) = \|A\|_2^2 \quad \Rightarrow \quad \alpha \le 1/\|A\|_2^2$$



Same quadratic (μ = 1, L = 10, κ = 10); only α changes. The largest-curvature direction sets the stability limit; 2/(L+μ) is the best fixed step.

A common misconception about 1/L

misconception	correction
"$\alpha = 1/L$ is always the fastest possible step."	$1/L$ is a standard safe choice. With known μ and L, $2/(L+\mu)$ is the optimal fixed step (worst case). Momentum, preconditioning, line search — or learned layer-wise steps — can do better.

"The learning rate can still be tuned empirically, but it is not born empirically. Analysis gives the scale; tuning chooses within it."

$\kappa = L/\mu$ decides how badly an elongated valley slows first-order descent.

Two gradients, two jobs

inside an inverse solver

$\nabla_x f(x)$ tells the algorithm how to update the current estimate x_k.

The state x changes every iteration, for every new measurement y.

while training a network

$\nabla\theta L$ tells backprop how to update trainable parameters θ.

θ changes during training, then is frozen for inference.

$$\min_\theta L(\theta) = (1/N)\,\Sigma_i\, \ell(f_\theta(x_i), y_i), \qquad \theta_{t+1} = \theta_t - \eta\,\nabla_\theta L$$

LISTA contains both: at inference only the state update remains; training shaped its parameters.

What is x? What is θ?

x unknown quantity → *sensor:* $y = 2x$ → *measured* $y = 10$

$$f_y(x) = \tfrac{1}{2}(2x - y)^2$$

how badly does x explain y?

$$x_{k+1} = x_k - \alpha\, \partial f/\partial x$$

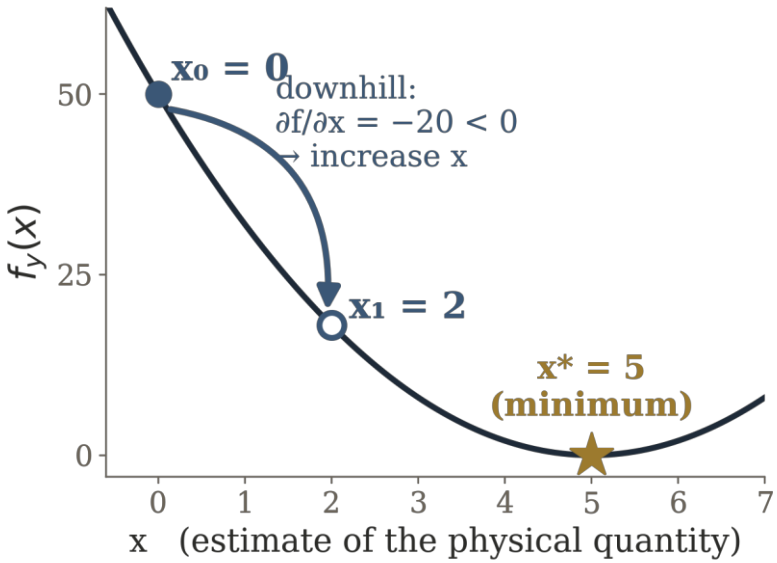
$$\partial f/\partial x = 2(2x - y) = 4x - 2y$$

$$x_0 = 0, \quad y = 10, \quad \alpha = 0.1$$

$\partial f/\partial x$ at $x_0 = 0$: $2(0 - 10) = -20$

$$x_1 = 0 - 0.1(-20) = 2$$

x is the answer we are trying to compute for THIS measurement.



$$x : 0 \rightarrow 2$$

α = 0.1: a setting we chose by hand. Nothing learned yet.

$\nabla_x f$ *changes the* **CURRENT SOLUTION.**

Now make the solver itself trainable

$$x_1 = x_0 - \alpha\, \partial f/\partial x(x_0)$$

Why must I choose α by hand?

$\alpha \rightarrow \theta$ trainable

$$x_1(\theta) = x_0 - \theta\, \partial f/\partial x(x_0) = 20\theta$$

with $\theta = 0.1$: $x_1 = 2$

TRAINING: we know the answer $x^* = 5$

$$L(\theta) = \tfrac{1}{2}\,[\, x_1(\theta) - x^* \,]^2$$

$$= \tfrac{1}{2}\,(20\theta - 5)^2$$

$$\partial L/\partial\theta = (20\theta - 5)\cdot 20 = -60 \quad \text{at } \theta = 0.1$$

$$\theta_{new} = \theta - \beta\, \partial L/\partial\theta \qquad (\beta = 0.001)$$

$$= 0.1 - 0.001(-60) = 0.16$$

forward: θ decides x_1 decides L



backward: $L \rightarrow x_1 \rightarrow \theta$

θ : 0.10 → 0.16

now $x_1 = 3.2$, closer to $x^* = 5$

x changed because we were **solving a sensing problem.**

θ changed because we were **learning how the solver should behave.**

$\nabla_x f$ *improves the* **ANSWER**

$\nabla_\theta L$ *improves the* **SOLVER**

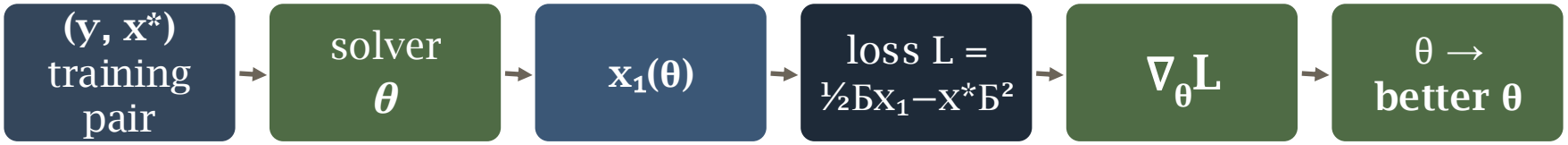
Two gradients, two jobs

STATE UPDATE — solving one problem



For a particular measurement, move x toward the solution.

PARAMETER UPDATE — learning the solver



Across many training examples, change the algorithm that produces x.

x	θ
current solution	solver parameter
depends on y	shared by all problems
changes at inference	frozen at inference
updated by $\nabla_x f$	updated by $\nabla\theta L$

$$\theta \rightarrow x_1(\theta) \rightarrow L$$

$$\partial L/\partial \theta = \partial L/\partial x_1 \cdot \partial x_1/\partial \theta$$

$$= (2 - 5) \cdot 20 = -60$$

Backprop: how does the error change if θ changes?

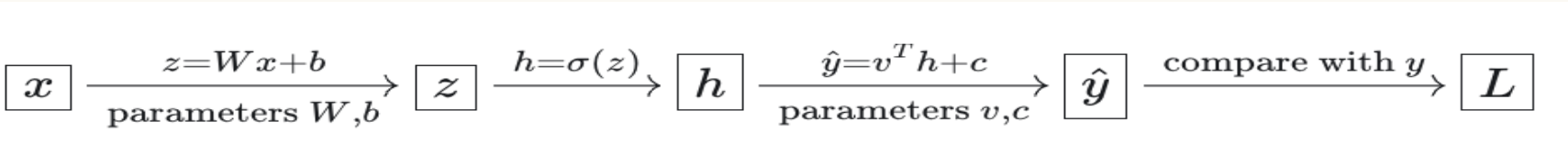
x is the answer. θ describes how the algorithm produces the answer.

$\nabla_x f$: solve the problem · $\nabla\theta L$: learn the solver

Backpropagation on a tiny network

$$z = Wx + b, \quad h = \sigma(z), \quad \hat{y} = v^T h + c, \quad L = \tfrac{1}{2}(\hat{y} - y)^2$$

forward: store values →



← backward: multiply local derivatives

$$\delta\hat{y} = \hat{y} - y \qquad \partial L/\partial v = \delta\hat{y}\, h, \quad \partial L/\partial c = \delta\hat{y}$$

$$\delta h = \delta\hat{y}\, v$$

$$\delta z = \delta h \odot \sigma'(z)$$

$$\partial L/\partial W = \delta z\, x^T, \quad \partial L/\partial b = \delta z$$

Board: write δŷ, then δh, then δz — ask which local derivative each arrow multiplies. Only then write ∂L/∂W.

Nondifferentiable is not the end

Backprop needs local derivatives along the graph. ReLU (kink at 0) and soft-thresholding (kinks at $\pm\tau$) are differentiable almost everywhere; autodiff uses a valid subgradient at isolated kinks.

backprop through a kink

fine: piecewise-differentiable operators train normally (ReLU networks, unrolled ISTA).

optimising a nonsmooth objective

different issue: БхБ_1 has no ordinary gradient at 0 — proximal methods solve that term exactly.

Ask: if ReLU is nondifferentiable at zero, why can we train ReLU nets? Then: should we just take a gradient of $\|x\|_1$ at zero? (No.)

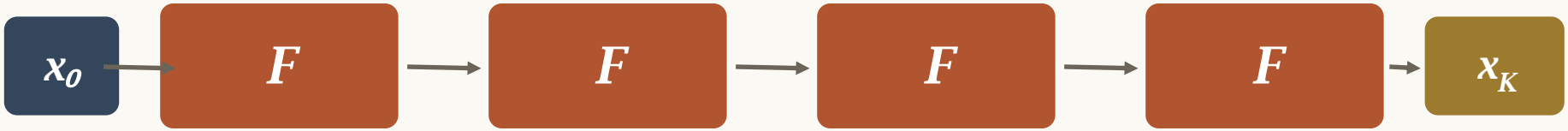
3.c) Unrolling

Turn an iterative algorithm into a finite computation graph

Unrolling

$$x_{k+1} = F(x_k, y; \theta), \qquad GD: \; F(x_k, y; \alpha) = x_k - \alpha \nabla_x f(x_k; y)$$

$$x_K = F(\, F(\, \cdots F(x_0, y; \theta) \cdots, y; \theta), y; \theta)$$

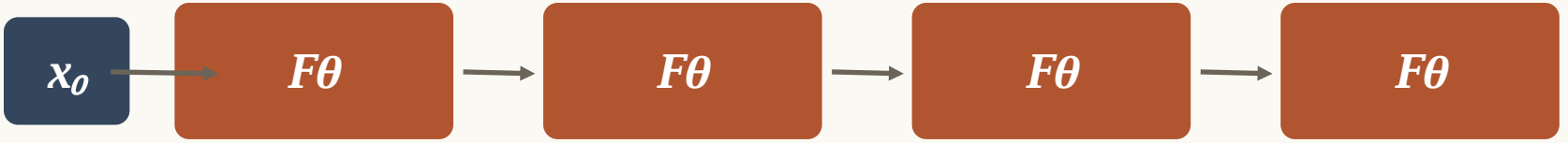


The nested expression is already a depth-K computation graph; y enters every stage.

Unrolling is a representation before it is a learning method. Learning enters only when we expose parameters and fit them to data.

Tied vs. untied unrolling

tied: $\theta_1 = \theta_2 = \cdots = \theta K$ (one recurrent cell, applied K times)



untied: each stage has its own θ_k



Two independent choices	Consequence
unroll or not?	explicit K-stage graph, end-to-end gradients
untie?	more expressive, more parameters
tie?	closer to the algorithm; extrapolates to more K

Repeated application of one layer = a tied unrolled graph.

What need leads to unrolling?

Not “solve to arbitrary precision”, but:

Given this distribution of sensing problems, produce a sufficiently accurate answer in exactly K stages, with predictable latency and memory.

what data can tune inside the fixed skeleton

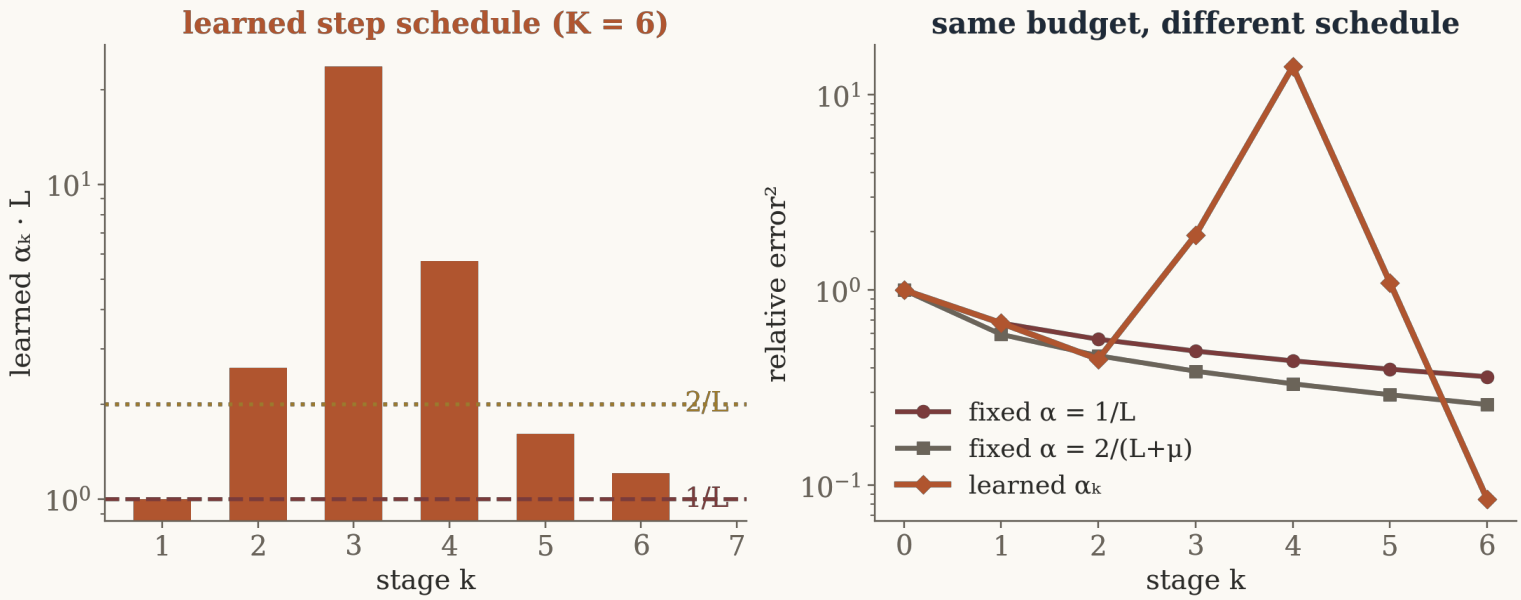
step sizes → **precon-ditioners** → **thresholds** → **transforms**

- Learning decides how to spend a fixed computational budget.
- The algorithmic skeleton keeps its model-based structure (forward/adjoint of A).

Warm-up: learned gradient descent

$$x_{k+1} = x_k - \alpha_k A^T(Ax_k - y) \qquad x_{k+1} = x_k - P_k A^T(Ax_k - y)$$

learn how far · learn scale + rotation



Real run: least squares with $\kappa(A^TA) = 100$, $K = 6$ learned α_k. Error after 6 stages: 0.36 ($\alpha = 1/L$), 0.26 ($2/(L+\mu)$), 0.085 (learned). Some learned steps exceed $2/L$ — unstable alone, but the 6-step product is not.

3.d) ISTA

Gradient descent plus an exact step for sparsity

The problem plain GD does not handle cleanly

$$\min_x \ \tfrac{1}{2}\|Ax - y\|_2^2 \ + \ \lambda\|x\|_1$$

f(x): smooth data fidelity	g(x) = λ‖x‖₁: nonsmooth
take an ordinary gradient step	$\|x_i\|$ has a kink at 0 — solve its step exactly

Smoothing |x| changes the problem and weakens exact sparsity.

ISTA's contribution is not "add thresholding": it is the proximal-gradient split — gradient on f, exact proximal step on g.

Derivation of the proximal-gradient step

quadratic upper model ($\alpha = 1/L$)

$$f(x) \le f(x_k) + \nabla f(x_k)^T (x - x_k) + (L/2)\|x - x_k\|^2$$

add g, complete the square

$$x_{k+1} = \arg\min_x \, (1/2\alpha)\|x - (x_k - \alpha \nabla f(x_k))\|^2 + \lambda \|x\|_1$$

which is a proximal operator

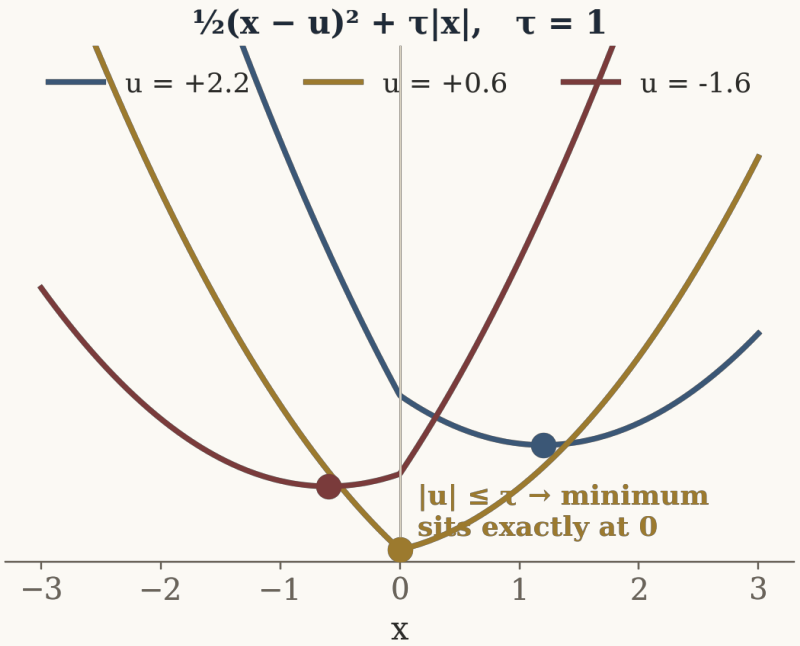
$$= \mathrm{prox}_{\alpha\lambda\|\cdot\|_1}(x_k - \alpha \nabla f(x_k))$$

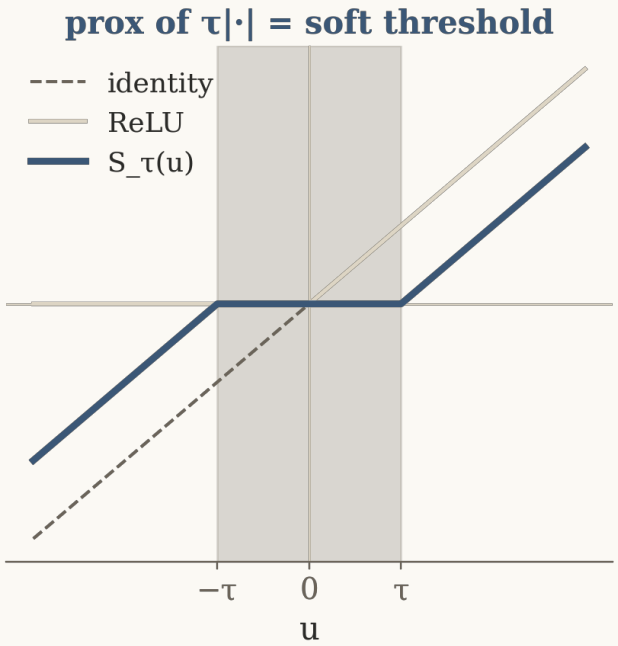
for $f = \tfrac{1}{2}\|Ax - y\|^2$:

$$z_k = x_k - \alpha A^T (A x_k - y), \qquad x_{k+1} = S_{\alpha\lambda}(z_k)$$

Why the prox becomes soft-thresholding

$$prox_{\tau|\cdot|}(u) = arg\ min_x\ \tfrac{1}{2}(x-u)^2 + \tau|x| \;=\; sign(u)\ max(|u| - \tau,\ 0)$$





Separable across coordinates. By cases: $x > 0 \Rightarrow x = u - \tau$; $x < 0 \Rightarrow x = u + \tau$; if $|u| \le \tau$ neither is consistent, so the optimum sits exactly at 0.

The "dead zone" is derived, not chosen as an activation function.

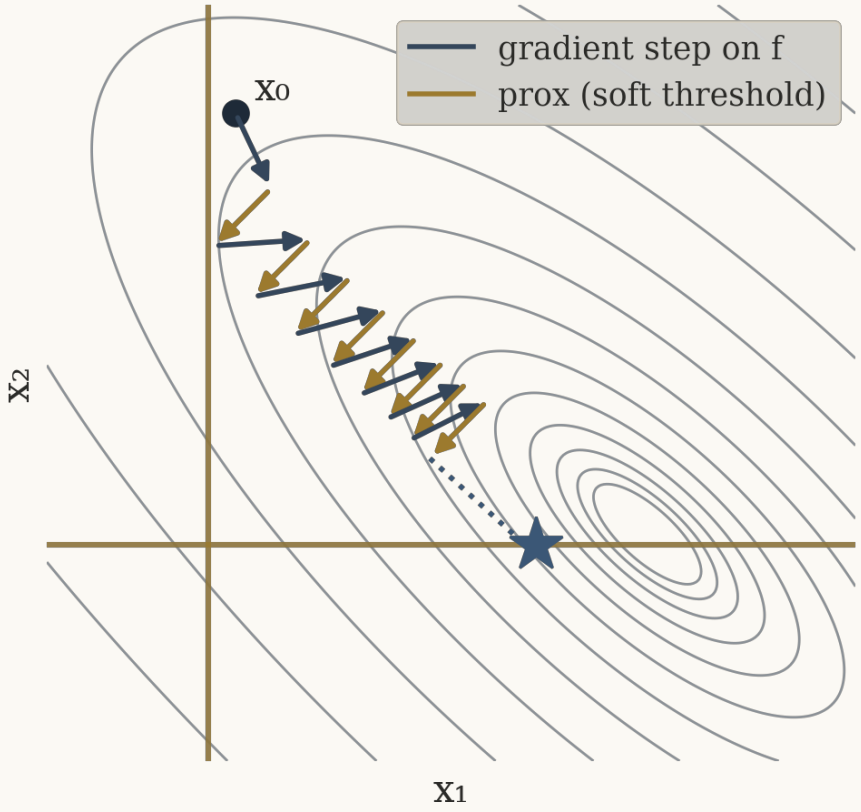
ISTA, line by line, in sensing language

Step	Math	Sensing meaning
1 residual	$r_k = Ax_k - y$	predicted vs. measured data
2 adjoint	$g_k = A^T r_k = \nabla f(x_k)$	backproject the mismatch
3 smooth step	$z_k = x_k - \alpha g_k$	data-consistency correction
4 prox step	$x_{k+1} = S_{\alpha\lambda}(z_k)$	apply the sparse prior

Input: A, y, λ, step $0 < \alpha \leq 1/\|A\|_2^2$. Initialise $x_0 = 0$. Repeat K times.

A is the forward physics; A^T maps measurement error back into the scene; shrinkage applies the prior.

What ISTA contributes — and what it does not



- Combines a smooth forward-model loss with a nonsmooth convex prior.
- Convergence guarantees under standard assumptions.
- But it can need many iterations.
- If A and the problem distribution recur, can data learn a better finite-depth solver?

That question is exactly the opening for LISTA.

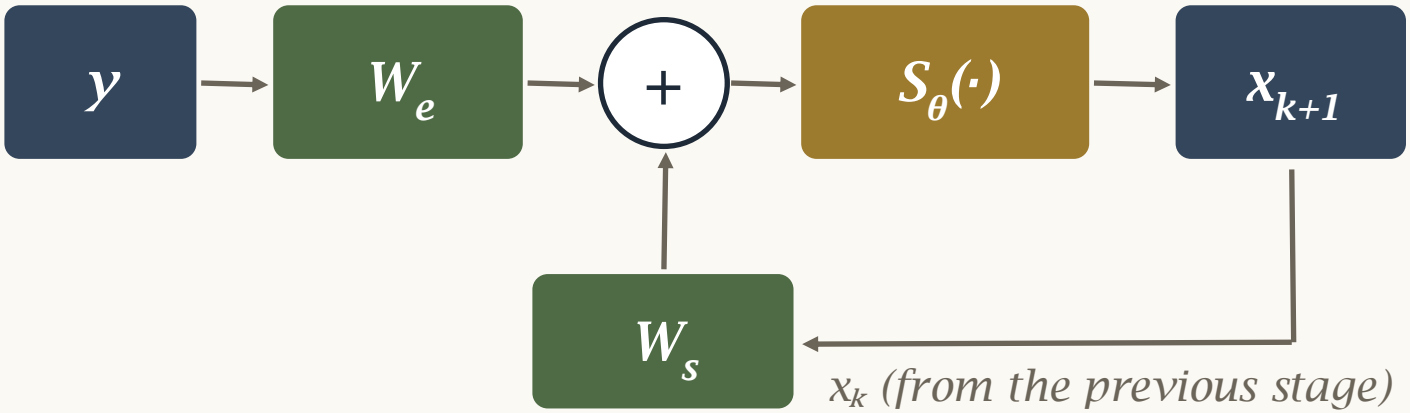
3.e) LISTA

Learn a finite-depth version of ISTA

Rewrite ISTA as a neural-network layer

$$x_{k+1} = S_{\alpha\lambda}\left((I - \alpha A^T A)\, x_k + \alpha A^T y \right)$$

$$x_{k+1} = S_\theta\left(W_e y + W_s x_k \right), \quad W_e = \alpha A^T, \;\; W_s = I - \alpha A^T A, \;\; \theta = \alpha\lambda$$



Linear maps + an elementwise nonlinearity: a single ISTA iteration already is a neural-network layer.

The LISTA move (Gregor & LeCun, 2010)

$$x_{k+1} = S_{\theta k}(W_e^{(k)} y + W_s^{(k)} x_k), \qquad k = 0, \ldots, K-1$$

Train a fixed-depth feed-forward predictor that approximates sparse codes much faster than running the optimizer to convergence.

ISTA supplies a physically meaningful initialisation:

$$W_e^{(k)} \leftarrow \alpha A^T, \qquad W_s^{(k)} \leftarrow I - \alpha A^T A, \qquad \theta_k \leftarrow \alpha\lambda$$

Training then adapts the finite sequence of updates to the distribution of problems seen in data.

What is actually learned?

Variant	Learned	Interpretation
Step-LISTA	α_k (and θ_k)	how far to move per stage
Threshold-LISTA	θ_k	sparsity schedule
Structured LISTA	constrained W_e, W_s	keep the link to A
Full LISTA	$W_e^{(k)}$, $W_s^{(k)}$, θ_k	most expressive
Tied LISTA	one W_e, W_s, θ	recurrent, fewer params
Untied LISTA	per-layer params	stage specialisation

Misconception: *"LISTA learns A^{-1}."* It learns the parameters of an iterative sparse-inference architecture — still alternating a measurement-driven linear correction with a sparsity nonlinearity.

Training LISTA by backpropagation

Pairs (y□ , x□ *): ground-truth sparse scenes, or high-accuracy offline solutions. Run the K layers forward, then:

$$L(\Theta) = (1/N)\,\Sigma_i \,\|x_K(y_i;\,\Theta) - x_i^*\|^2, \qquad \Theta = \{W_e^{(k)},\, W_s^{(k)},\, \theta_k\}$$

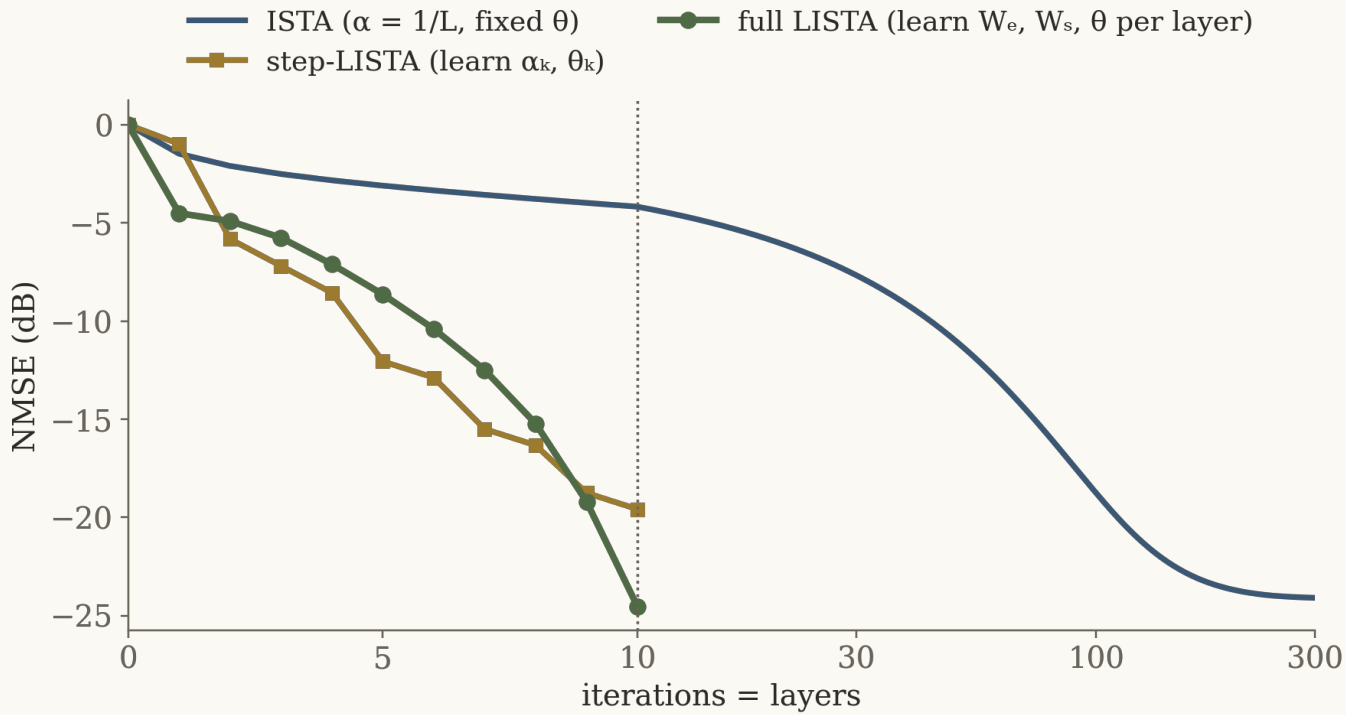
optional deep supervision:

$$L(\Theta) = \Sigma_k\, \omega_k\, \|x_k - x^*\|^2$$

forward: K layers → **loss at last layer** → **backward: ∇Θ L** → **update Θ**

Soft-thresholding is differentiable almost everywhere — backprop runs straight through the unrolled stages.

Why can LISTA be faster at a fixed depth?



Real run: A is 32 × 64 Gaussian, 4-sparse x, σ = 0.01, 10 layers, numpy-trained. At 10 stages — ISTA −4.2 dB, step-LISTA −19.6 dB, full LISTA −24.5 dB; ISTA needs ~300 iterations to reach −24 dB.

Ask: "If ISTA gets 10,000 iterations, must LISTA win?" No — LISTA's edge is under a fixed inference budget.

What is preserved, and what is sacrificed?

preserved / encouraged

- known forward-model structure
- meaningful intermediate states
- fixed compute and latency
- less data than a generic network
- interpretable steps and thresholds

at risk as learning grows

- exact optimization interpretation
- global convergence guarantees
- robustness outside training data
- easy transfer when A / hardware changes

The more of the layer we learn, the more speed we buy — and the more physics-derived guarantees we spend.