Deep Learning in Economics: A Geometric Interpretation

Jesús Fernández-Villaverde[1]

November 13, 2025

[1]University of Pennsylvania

Primary Question

What is the source of the unreasonable effectiveness of deep learning in economics?

Secondary question

Why do we care?

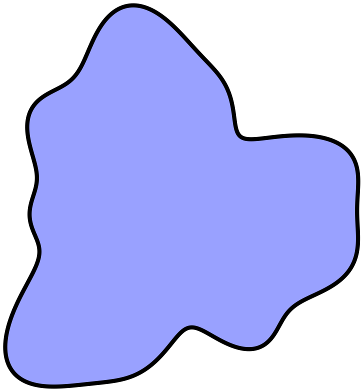
Answer to the primary question

- Deep learning is just a sequence of geometric transformations that progressively warp and flatten data manifolds until the underlying function approximation problem becomes simple in the final representation space.
- One can show that, through these geometric transformations, deep neural networks find the most regular function that interpolates the data within a dense functional space.
  - For example, while solving DSGE models, the policy function with the lowest Sobolev semi-norm in a Banach space of possible policy functions that interpolates all grid points.
- In other words, deep learning provides us with a constructive procedure to work with a better geometrical representation of the underlying function approximation problem without having to rely on domain knowledge.

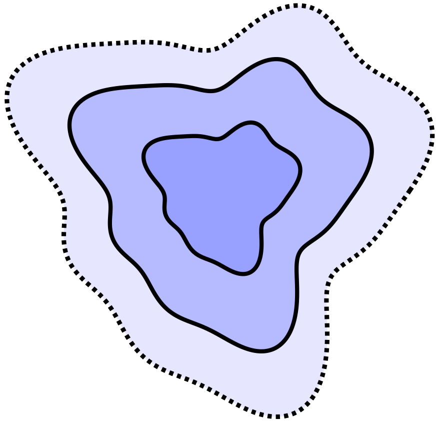
Answer to the secondary question

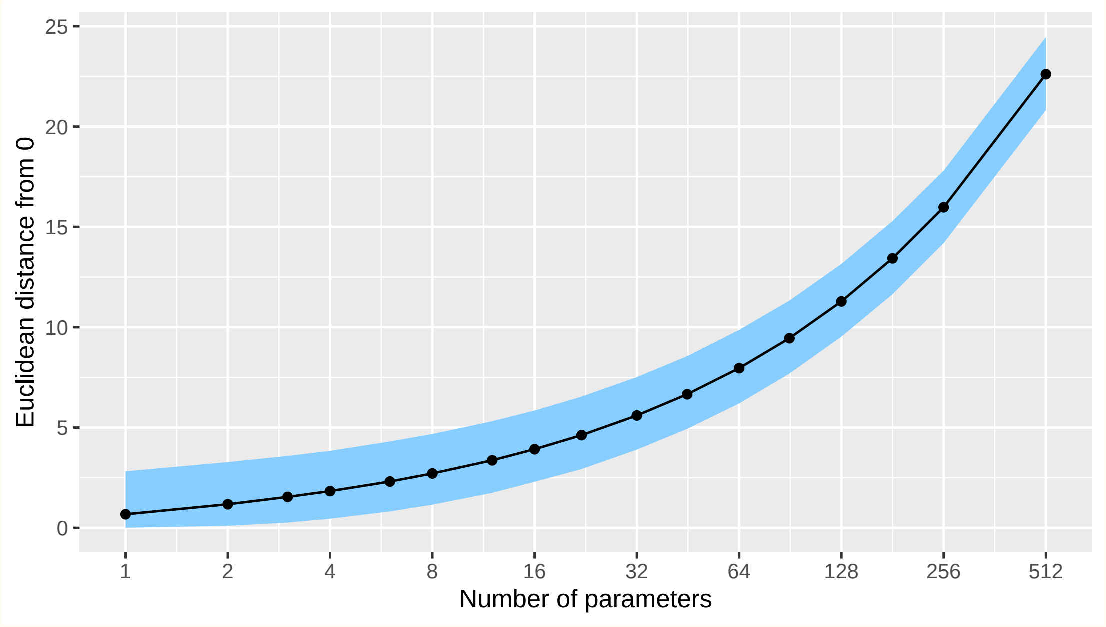
- Three key implications:
  1. Rather than restricting the solutions a model can represent, we want to specify a preference for certain solutions over others through a soft inductive bias.
  2. The double descent phenomenon teaches us that counting parameters is *not* the right metric of complexity of a model with a class.
  3. The architecture of our model class should not depend on the amount of data available.
- It is probably also connected with grokking and mode connectivity, but we still do not understand these two phenomena well enough.
- Challenge: Basic results of high-dimensional geometry can clash with our 3-D natural intuition.
  - Think, for instance, about the location of a typical set in a high-dimensional Gaussian distribution.

Restriction Bias



Soft Inductive Bias




Euclidean distance from 0
25
20
15
10
5
0
1
2
4
8
16
32
64
128
256
512
Number of parameters

The geometry of function approximation

Ultimately, all we do is approximate unknown functions

- In economics, we want to approximate ("learn") unknown functions: a value function, a policy function, a pricing kernel, a best response, a conditional expectation, a classifier, ...
- Formally, we approximate $f : \mathcal{X} \to \mathbb{R}$:
$$y = f(X)$$
where $y \in \mathbb{R}$ is a scalar and $X = \{x_1, x_2, ..., x_N\} \in \mathcal{X}$ a vector.
- The elements of X are called the features of the data (observations, points on the grid, ...) and belong to a feature space $\mathcal{X}$.
- N can be large (possibly in the thousands!).
- Easy to extend to the case where y is a vector (e.g., a probability distribution), but the notation becomes cumbersome.

The key idea: Approximate a function in the "right" representation of the data

- Traditional function approximations:

$$f(X) \approx h(X)$$

  use a flat functional form $h(\cdot)$ (i.e., a linear combination of Chebyshev polynomials, a piecewise linear function).

- Deep neural networks (and machine learning more in general) use nested mappings:

$$f(X) \approx g(\phi(X))$$

  where $\phi : \mathcal{X} \to \mathcal{Z}$, $g : \mathcal{Z} \to \mathbb{R}$.

  1. $\mathcal{Z}$ is the "representation space."
  2. The function $\phi(\cdot)$ finds a representation of the data X in the "right" geometrical space.
  3. The function $g(\cdot)$ approximates $f(\cdot)$ in the representation of the data.

An example: The non-stationary stochastic neoclassical growth model

- A social planner's problem:

$$\max \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \log(c_t)$$

$$c_t + k_{t+1} = \left(e^{z_t}\right)^{(1-\alpha)} k_t^{\alpha} + (1-\delta)\, k_t$$

$$\lim_{T\to\infty} \mathbb{E}\beta^T k_{T+1} c_T^{-1} = 0$$

- Technology is a random walk with a drift:

$$z_t = \lambda + z_{t-1} + \sigma\nu_t, \quad \nu_t \sim \mathcal{N}(0,\sigma)$$

- State variables: $X_t = (k_t, z_t)$.
- Challenge: this dynamic programming problem is non-stationary.

Finding the state is an art

- Standard solution: "engineer" a transformation of the variables ("feature engineering") using our economic understanding of the problem (our "domain knowledge").
- For example $\widetilde{c}_t = \frac{c_t}{e^{z_{t-1}}}$ and $\widetilde{k}_t = \frac{k_t}{e^{z_{t-1}}}$.
- New equivalent but stationary social planner's problem:

$$\max \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \log(\widetilde{c}_t)$$

$$\widetilde{c}_t + e^{\lambda+\sigma\nu_t} \widetilde{k}_{t+1} = (e^{\lambda+\sigma\nu_t})^{(1-\alpha)} \widetilde{k}_t^{\alpha} + (1-\delta)\, \widetilde{k}_t$$

$$\lim_{T\to\infty} \mathbb{E}\beta^T \widetilde{k}_{T+1} \widetilde{c}_T^{-1} = 0$$

- New state variables: $\widetilde{X}_t = (\widetilde{k}_t, \nu_t, z_{t-1})$.

Why did the transformation help?

- The new problem of the social planner can be solved using a high-order perturbation nearly to machine precision using `Dynare` in a fraction of a second.
  - This problem only depends on $\widetilde{k}_t$ and ν_t.
- With the solution above, we use the third state variable, z_{t-1}, to simulate the economy by "untransforming" the endogenous variables into levels.
- In our notation before:
  1. The function $\phi(\cdot)$: $(k_t, z_t) \rightarrow (\widetilde{k}_t, \nu_t, z_{t-1})$.
  2. The function $g(\cdot)$: high-order perturbation on $(\widetilde{k}_t, \nu_t)$ plus simulation "untransforming" endogenous variables with z_{t-1}.

A few common misconceptions

1. This is *not* about dimensionality reduction. We have gone from two to three state variables! A more general case: Cover theorem (1965).

Cover theorem

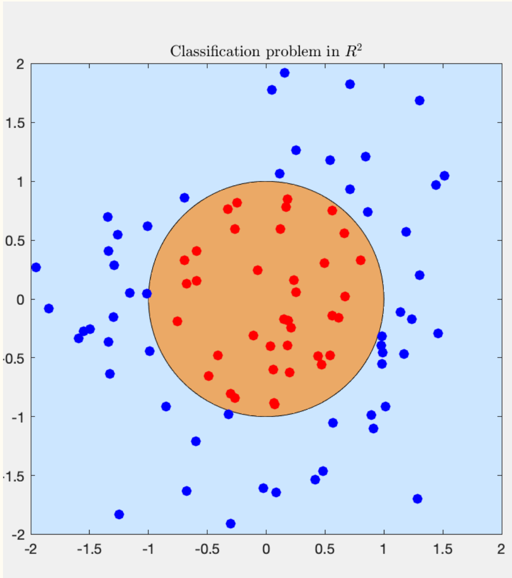
A sorting problem is more likely to be linearly separable in a high-dimensional space than in a low-dimensional space, provided that the space is not densely populated.

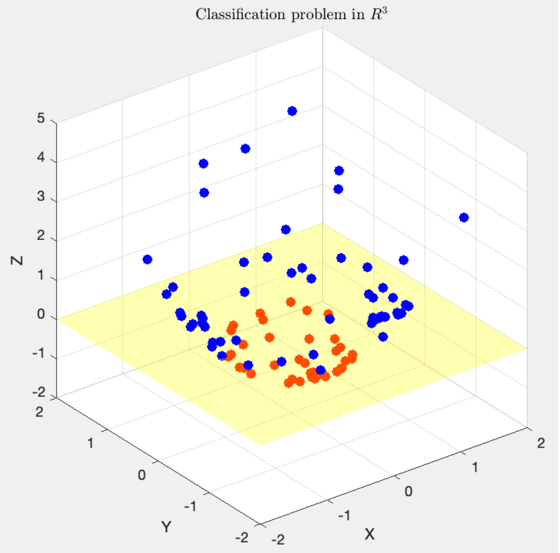
   Sometimes, good representations are in lower dimensions; sometimes, they are in higher dimensions.

2. This is *not* about big data. Data has played no role.

3. This is *not* even about deep neural networks per se! We have "engineered" the transformation by hand.

The kernel trick for sorting: Transform $(x, y) \to (x, y, x^2 + y^2 - 1)$





Deep neural networks

So, what is the role of deep neural networks in all this?

- Recall: the whole point is to find the right representation of the data.
    - Everything is about the geometry of the problem!
- The challenge is that our domain knowledge is limited.
    - What is the best representation of the states of a macro climate change model? (My work with Kenneth Gillingham and Simon Scheidegger).
- Can we design a function $\phi(\cdot)$ that has enough expressiveness to discover the most informative representation without much input from the researcher?
    - Often (but not always): yes, we can!
    - Sometimes we can find $g(\cdot)$ and $\phi(\cdot)$ simultaneously, sometimes we find $g(\cdot)$ by hand or a different algorithm.

A basic example: the feedforward fully-connected deep neural network

- We start by taking affine transformations of the features of the data:

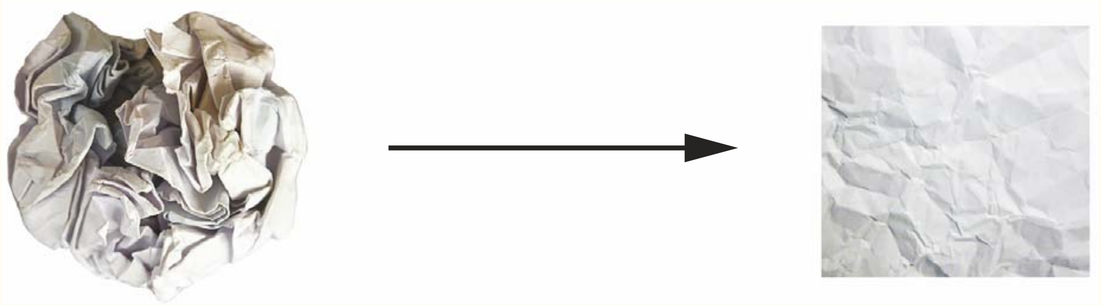
$$z_m^0 = \theta_{0,m}^0 + \sum_{n=1}^{N} \theta_{n,m}^0 x_n$$

- We take $M^{(1)}$ of these affine transformations, we apply a continuous deformation (such as stretching or bending) with the activation function $\phi(\cdot)$, and take a second affine transformation:

$$g_{\theta^1}(X) = z_m^1 = \theta_{0,m}^1 + \sum_{m=1}^{M^{(1)}} \theta_m^1 \phi^1\left(z_m^0\right)$$

- Iterate the previous steps J times:

$$\begin{aligned} y \approx g_\theta(X) &= \theta_0^J + \sum_{m=1}^{M^{(J)}} \theta_m^J \phi^J\left(z_m^{J-1}\right) \\ &= g_{\theta^J}\left(g_{\theta^{J-1}}\left(\ldots g_{\theta^1}(X)\right)\right) \end{aligned}$$



Returning to our example

- Recall our social planner's problem:

$$\max \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \log(c_t)$$
$$c_t + k_{t+1} = (e^{z_t})^{(1-\alpha)} k_t^{\alpha} + (1-\delta) k_t$$
$$z_t = \lambda + z_{t-1} + \sigma \nu_t, \quad \nu_t \sim \mathcal{N}(0,\sigma)$$
$$\lim_{T\to\infty} \mathbb{E}\beta^T k_{T+1} c_T^{-1} = 0$$

  with state variables: $X_t = (k_t, z_t)$.

- There is a recursive problem associated with the previous sequential problem:

$$V(X) = \max_{k'} \{u(c) + \beta \mathbb{E} V(X')\}$$
$$c = (e^z)^{(1-\alpha)} k^{\alpha} + (1-\delta) k - k'$$
$$z' = \lambda + z + \sigma \nu', \quad \nu \sim \mathcal{N}(0,\sigma)$$

Our approximation

- We approximate:
$$V(X) \approx g(\phi(X))$$
by minimizing a loss function $\mathscr{L}(\theta^*; X, y)$.
- The optimizer forces $\phi(X)$ to reconfigure X in informationally efficient representations.
- Why? Because a first-order optimizer picks weights to push the representation toward a simpler, low-Kolmogorov complexity manifold.

The geometric unification

- I have presented a basic feedforward deep neural network.
- There are many alternative architectures (e.g., GANs, transformers) designed to capture specific problems.
- Often, researchers are puzzled by the multiplicity of architectures.
- However, there are significant theoretical connections among different architectures.
- "Geometric unification" effort in the spirit of the *Erlangen Program* formulated by Felix Klein: a common framework to think about different architectures.

Why do we care?

Overparameterization

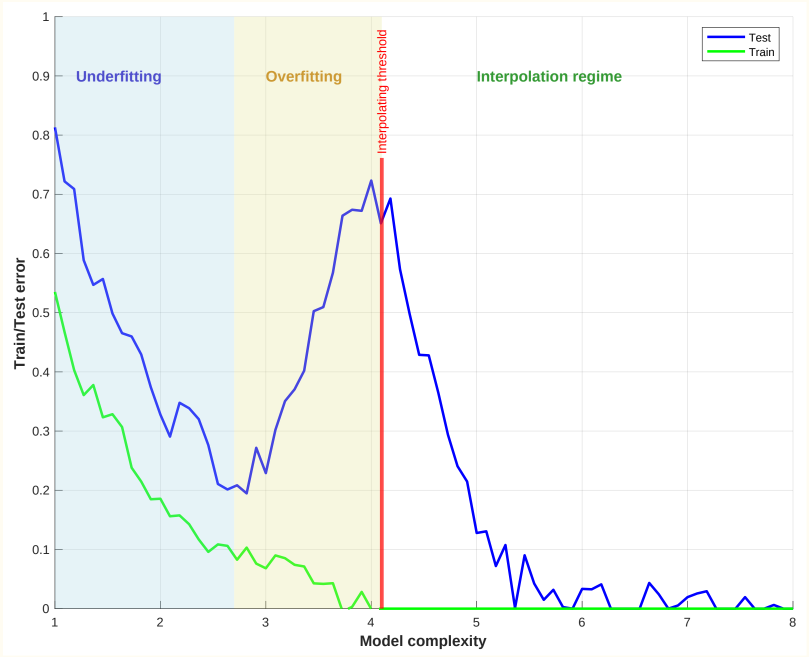
- Deep neural networks are massively overparameterized ($\dim(\theta) \gg$ num. data).
- We want to have enough expressiveness for our representations.
- But, how can this not be a problem?

The classical view: Enrico Fermi, 1953

I remember my friend Johnny von Neumann used to say, with four parameters I can fit an elephant, and with five I can make him wiggle his trunk.

Background

- A classical result in statistics: bias-variance tradeoff.
  - U-shaped test-error curve: Adding regressors to a statistical model reduces bias but increases variance.
  - This tradeoff has long guided model choice in econometrics.
- Machine learning research (Belkin et al., 2019), however, has uncovered a "second descent" in test error when model complexity grows well beyond the sample size:
  - First document case: Vallet et al. (1989).
  - Present in the solution of DSGE models as an inductive bias: Ebrahami Kahou et al. (2025).


Underfitting
Overfitting
Interpolating threshold
Interpolation regime
Test
Train
Train/Test error
Model complexity
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1
2
3
4
5
6
7
8

Why do we have a double descent?

- Double descent in a nutshell:

  1. The first descent is because the approximation is learning the patterns of the data (loosely speaking, $g(\cdot)$).
  2. The first ascent is because the approximation is "memorizing" the data.
  3. The second descent is because the approximation is learning the representations of the data that generalize well (loosely speaking, $\phi(\cdot)$).

- Can we get more precise?

Implicit regularization

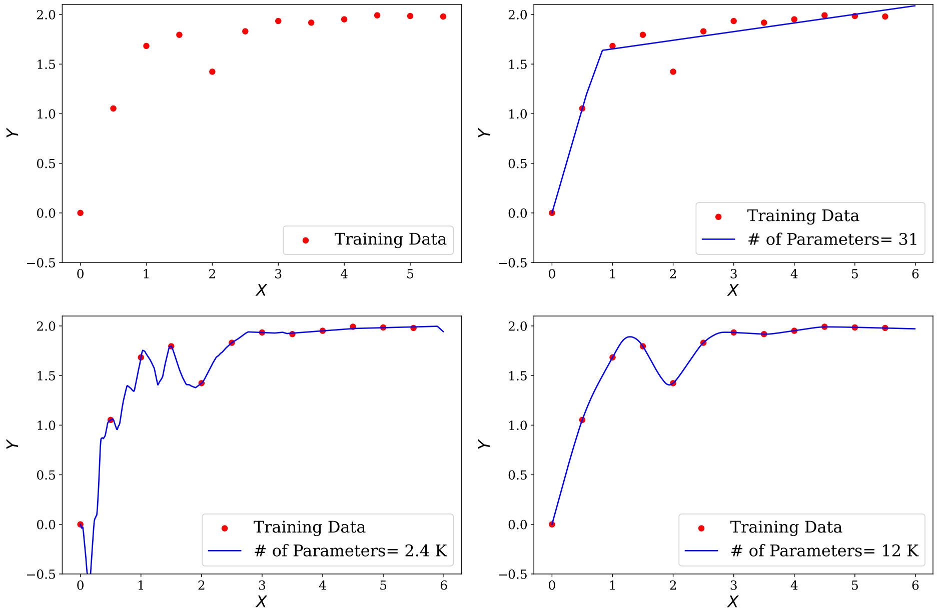
- High-dimensional gradient-based optimizers select min-norm solutions:

$$g_\theta^* \equiv \min_{g_\theta \in \mathcal{H}(\theta)} \|g_\theta\|_\psi$$

$$\text{s.t. } \mathscr{L}(\theta^*; X, y) = 0, \text{ for all } X \in \mathbf{X}$$

where recall that $\mathscr{L}(\theta^*; X, y)$ is a loss function.

1. Theoretical proofs for some cases (Belkin, 2021) and plenty of "real-life" cases where this holds (many of my papers).
2. This is a form of implicit regularization: the "soft inductive bias" of machine learning.
3. Identification of the individual θ's is irrelevant, since all min-norm solutions are equivalent.
4. Nonetheless, this should remind you of the ridge regression.


Training Data
# of Parameters= 31
# of Parameters= 2.4 K
# of Parameters= 12 K
X
Y

Going beyond implicit regularization

- But, why do we have implicit regularization?
- The theoretical proofs offer little analytic insight.
- Hypotheses such as the winning lottery tickets fail in many circumstances.
- And it turns out that we do not even need first-order optimizers to have a soft inductive bias: large models have a wide region of low loss.
- Let me walk you through a much simpler linear environment.

A univariate DGP

- Consider a univariate time series Y_t generated by some unspecified stationary data-generating process (DGP).
- $\{Y_t\}_{t=1}^{T}$ is the training data.
- Also, some extra test data $\{Y_t\}_{t=T+1}^{T+J}$ that I do not use for estimation.

An AR(n) model

- An AR(n) model for this training data is:

$$Y_t = \phi_0 + \sum_{j=1}^{n} \phi_j Y_{t-j} + \varepsilon_t$$

  with $\mathbb{E}[\varepsilon_t] = 0$, and $\text{Var}(\varepsilon_t) = \sigma^2$.

- $\beta = (\phi_0, \phi_1, \ldots, \phi_n)^\top \in \mathbb{R}^N$ with $N = n + 1$.
- I am not assuming the DGP behind the training data is an AR(n) process; I am simply fitting this model to the observations.

Design matrix and response vector

- The design matrix $X \in \mathbb{R}^{T_{eff} \times N}$ with:

$$X = \begin{bmatrix} 1 & Y_n & Y_{n-1} & \cdots & Y_1 \\ 1 & Y_{n+1} & Y_n & \cdots & Y_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & Y_{T-1} & Y_{T-2} & \cdots & Y_{T-n} \end{bmatrix}$$

  where $T_{eff} = T - n$.

  - For simplicity, I am ignoring the rank condition $\text{rank}(X) = N$.

- The response vector:

$$y = \begin{bmatrix} Y_{n+1} \\ \vdots \\ Y_T \end{bmatrix}$$

- If I estimate β using OLS, I will have three cases.

Case I: The underparameterized regime

- When $T_{eff} > N$, the OLS estimator is:

$$\hat{\beta}_{\text{under}} = (X^\top X)^{-1} X^\top y$$

- For a test regressor x_{test}, the prediction of the fitted model is:

$$\hat{y}_{\text{test,under}} = x_{\text{test}}^\top (X^\top X)^{-1} X^\top y$$

- The test error follows a U-shape: as N increases, the bias decreases, and the variance grows roughly in proportion to N/T_{eff}.

Case II: The interpolation threshold

- When $T_{eff} = N$, I still have $\hat{\beta}_{\text{under}} = (X^\top X)^{-1} X^\top y$, but the AR($n$) fits the training data perfectly.
- Hence, the prediction is still:
$$\hat{y}_{\text{test,interp}} = x_{\text{test}}^\top (X^\top X)^{-1} X^\top y$$
- Spike in the test error.

Case III: The overparameterized regime

- When $T_{eff} < N$, X is rank-deficient, and $X^\top X$ is singular.
- That is, there exist infinitely many interpolating solutions that fit the data perfectly.
- If I use the Moore–Penrose pseudoinverse:

$$\hat{\beta}_{\text{over}} = X^+ y = X^\top (XX^\top)^{-1} y$$

with prediction:

$$\hat{y}_{\text{test,over}} = x_{\text{test}}^\top X^\top (XX^\top)^{-1} y$$

- This approach yields the minimum-norm interpolator $\arg\min \|\beta\|_2$, a form of implicit regularization.
- In our model, implicit regularization is achieved through the pseudoinverse, but the same regularization can also be obtained using gradient-based optimizers (e.g., stochastic gradient descent).

Singular value decomposition (SVD)

- Test errors can be analyzed in terms of the singular values of X.
- Recall that the SVD of X is:

$$X = U\Sigma V^\top$$

  where $\sigma_1 \geq \cdots \geq \sigma_R > 0$ and $R = \text{rank}(X)$.

- Nonzero singular values $\{\sigma_r\}$ capture directions of variation in the data.

Decomposition around an “ideal” model

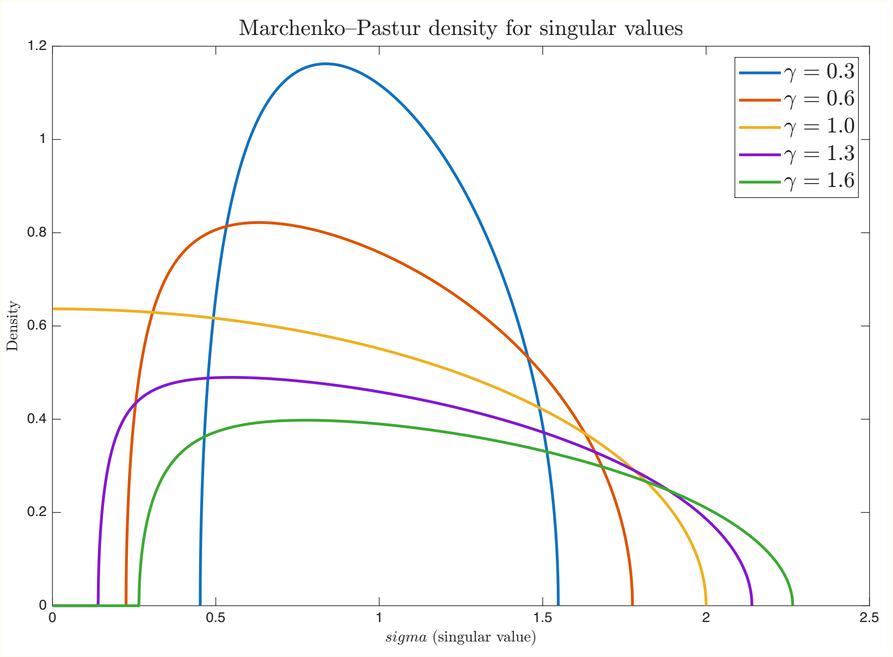
- Write $y = X\beta^\star + E$, where:
  - $\beta^\star$: best linear predictor (minimizes test risk).
  - E: residuals (noise and/or misspecification).
- Test error (both in the under- and overparameterized regimes):

$$\sum_{r=1}^{R} \frac{1}{\sigma_r} (x_{\text{test}}^\top v_r)(u_r^\top E)$$

- Test error diverges when:
  1. Small singular values ($1/\sigma_r$ large).
  2. Test features align with vulnerable directions.
  3. Residuals amplify these directions.

The appearance and disappearance of small singular values

- Let us use random matrix theory.
  - You can get the same results using differential geometry or algebraic topology.
- The Marchenko-Pastur law characterizes the density of singular values.
  - Think about this as a CLT for singular values of random matrices.
- Define $\gamma = \frac{N}{T_{eff}}$.


Marchenko–Pastur density for singular values
Density
$sigma$ (singular value)
$\gamma = 0.3$
$\gamma = 0.6$
$\gamma = 1.0$
$\gamma = 1.3$
$\gamma = 1.6$
0
0.2
0.4
0.6
0.8
1
1.2
0
0.5
1
1.5
2
2.5

Why do we move away from zero in high dimensions?

- A fundamental feature of high-dimensional geometry is that the angles of random vectors are almost right and norms are almost constant.
    - Intuition: Throw many very high-dimensional arrows with random entries in a very high-dimensional room: they will be nearly perpendicular to each other and have roughly the same size.
- In other words, random vectors are almost orthogonal and spread variance evenly: every projection axis sees roughly the same amount of stretching.
- But the stretching is just the singular value.
- High dimensionality is not a curse; it is a blessing!

A univariate application

- The DGP for the simulation:

$$y_t = \varphi y_{t-1} + \varepsilon_t + \theta \varepsilon_{t-1}, \qquad \varepsilon_t \sim \mathcal{N}(0, \sigma^2)$$

- $|\varphi| < 1$, $|\theta| < 1$ (stationarity and invertibility).
- ARMA(1,1) has AR(∞) representation with slow decay $\Rightarrow$ truncation bias at small n.
- To obtain long AR(∞) tail: $\varphi \approx -\theta$.
- Simulation parameters:
  - $\varphi = 0.95$, $\theta = -0.90$, $\sigma = 1$.
  - $T = 300$ chosen to match typical quarterly macro data.

Train/test protocol

- Train/test split: $T_{\text{train}} = \lfloor 0.8T \rfloor$.
- No global standardization (to avoid leakage).
- For each n:
  1. Build $(X_{\text{train}}, \mathbf{y}_{\text{train}})$.
  2. Compute $\hat{\phi} = X^{+}\mathbf{y}$.
  3. Predict 1-step ahead using true lagged values.
  4. Test error:
$$\mathrm{MAE} = \frac{1}{N} \sum_{t \in \text{test}} |\hat{y}_t - y_t|$$

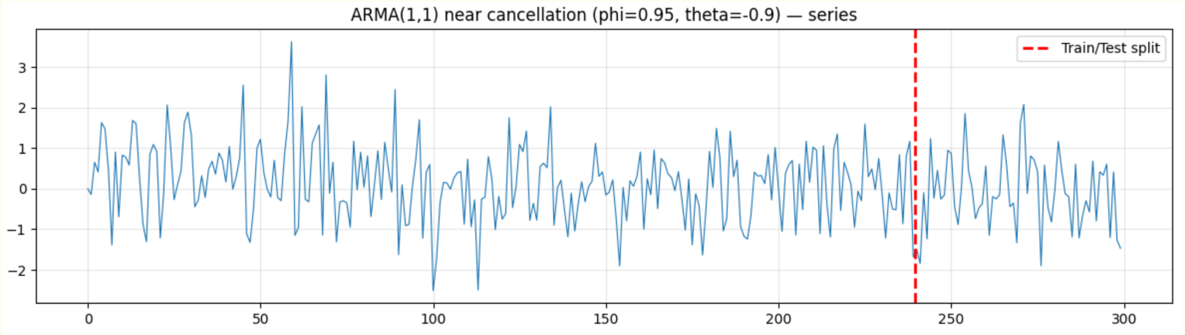


Figure 1: ARMA(1,1) data.

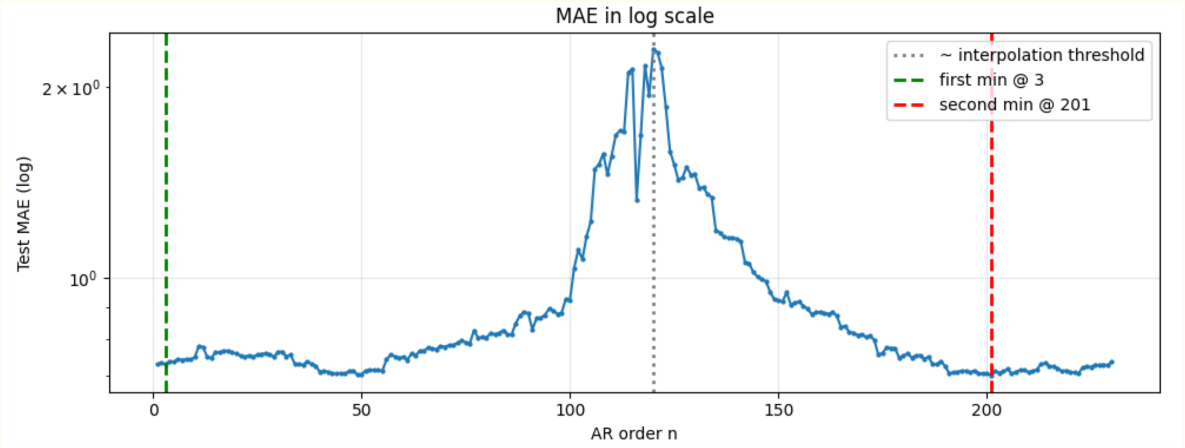


Figure 2: Test MAE of AR(n) vs. AR order n for ARMA(1,1).

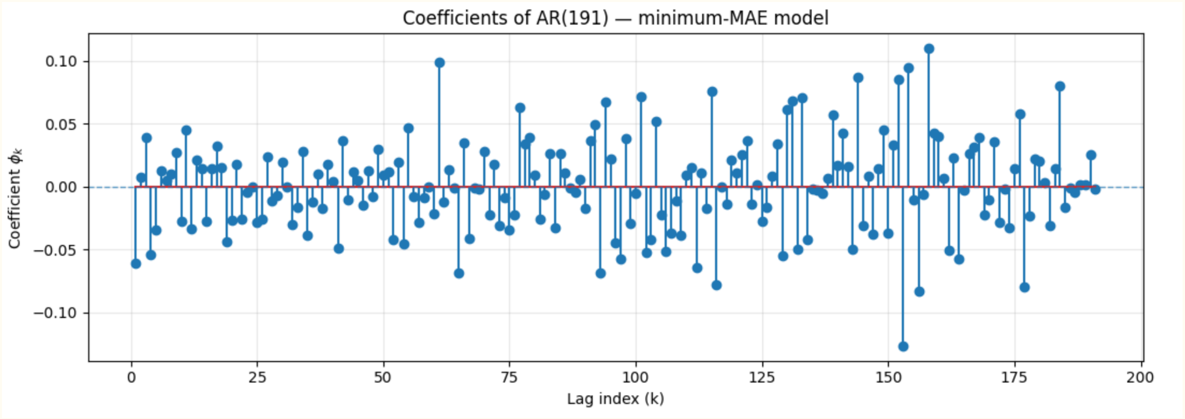


Figure 3: Coefficient Estimates of Best AR ($n = 191$).

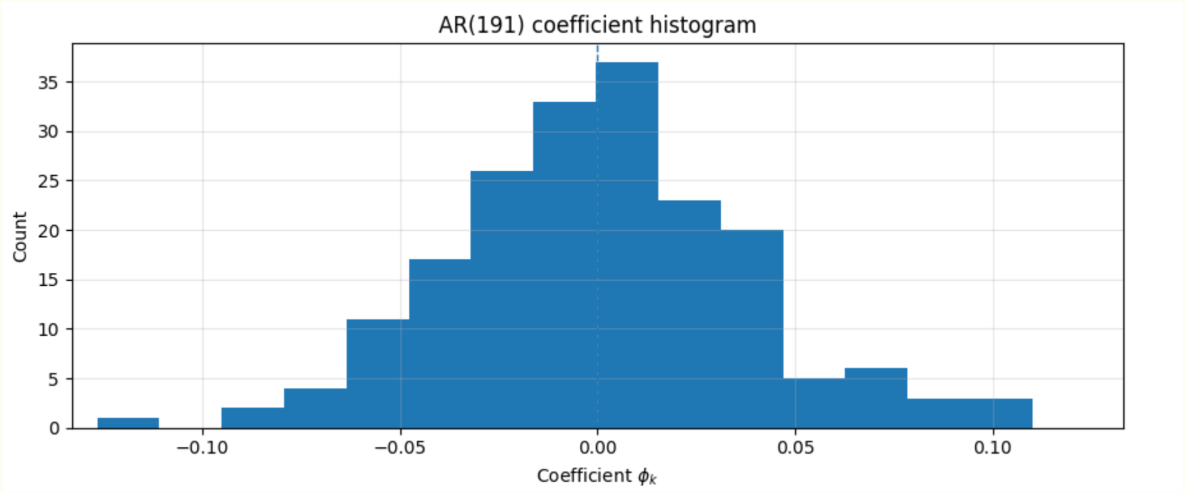


Figure 4: Histogram of Coefficient Estimates of Best AR ($n = 191$).

Taking stock

- Deep learning works because it searches for a better geometric representation of the function approximation problem at hand.
- Understanding deep learning geometrically helps economists see that model expressiveness, rather than parameter count, drives generalization and the double descent phenomenon.