From Simple Non-Linear Methods to Deep Learning

Lecture 3

Elodie Chervin Daniel Barbosa

TT 2026

University of Oxford & Nuffield College

Today's plan

The supervised learning frame

Why we need non-linearity

Trees and forests

Neural networks

Training Neural Nets

Recap

The supervised learning frame

Where this lecture sits

Most of the models seen so far lived inside the linear class. Shrinkage trades bias for variance *within* that class — it cannot fix a wrong functional form.

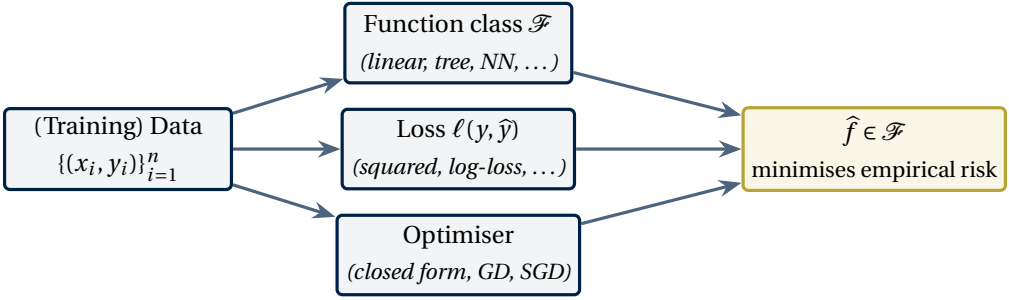
Yet almost nothing in economics is truly linear: Mincer curves, Engel curves, Phillips kinks, the zero lower bound. We need tools that model non-linearities directly.

Two ways to get non-linearity:

1. **Hand-crafted features** — polynomials, interactions, splines. Doesn't scale: you must *know in advance* which non-linearities matter.
2. **Learned non-linearity** — let the method discover the structure. Trees partition; neural networks compose simple non-linear maps from features (X) to responses (Y).

Once the function class learns its own features X, *scale becomes a lever you can turn*: more layers, more units, more parameters. That is the path from a handful of βs to the billions of weights inside today's large models.

The supervised learning template



Every supervised method we will see fits the same skeleton:

$$\widehat{f} = \arg\min_{f \in \mathscr{F}} \frac{1}{n} \sum_{i=1}^{n} \ell\big(y_i, f(x_i)\big).$$

The only thing that changes between methods is which class $\mathscr{F}$.

Optional recap: ML recap

OLS as an instance of the template

Linear regression in the language above:

- **Function Class:** $\mathcal{F} = \{f(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p : \beta \in \mathbb{R}^{p+1}\}$ — the affine functions of x.
- **Loss:** squared error, $\ell(y, \hat{y}) = (y - \hat{y})^2$.
- **Empirical risk:** $\widehat{R}(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i^\top \beta)^2 = \frac{1}{n}\|y - X\beta\|^2$.
- **Optimiser:** closed-form first-order condition. Setting $\nabla_\beta \widehat{R} = 0$ gives the normal equations $X^\top X\,\widehat{\beta} = X^\top y$, hence

$$\widehat{\beta}_{\text{OLS}} = (X^\top X)^{-1} X^\top y.$$

What's special: the class is closed under linear combinations and the loss is quadratic, so the minimiser has a closed-form expression. *Once $X^\top X$ is invertible, we do not need a numerical optimiser to find the solution.*

Logistic regression as an instance of the template

Same skeleton, qualitative response $Y \in \{0, 1\}$.

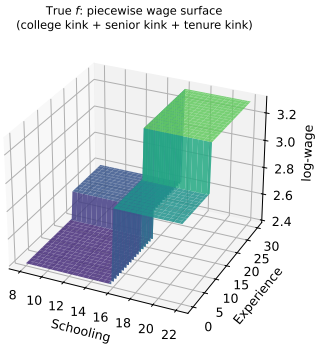
- **Class:** probabilities pushed through the sigmoid, $\mathscr{F} = \left\{ p(x) = \sigma(x^\top \beta) : \beta \in \mathbb{R}^{p+1} \right\}$, with $\sigma(z) = 1/(1 + e^{-z})$.
- **Loss:** negative log-likelihood (cross-entropy), $\ell(y, p) = -\, y \log p \; - \; (1 - y) \log(1 - p)$.
- **Empirical risk:** $\widehat{R}(\beta) = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log \sigma(x_i^\top \beta) + (1 - y_i) \log(1 - \sigma(x_i^\top \beta)) \right]$.
- **Optimiser:** no closed form. $\widehat{R}(\beta)$ is convex in β, solve numerically (Newton–Raphson / IRLS / gradient descent).

Already a hint of what's coming. Same linear index as OLS, but compose it with a non-linearity (σ) and an information-theoretic loss: there is no algebraic formula like $(X^\top X)^{-1} X^\top y$, so we solve by iteration. A neural network keeps composing affine maps and non-linearities, *layer after layer.*

Optional detail: loss functions

Why we need non-linearity

Checkpoint: why non-linearity matters

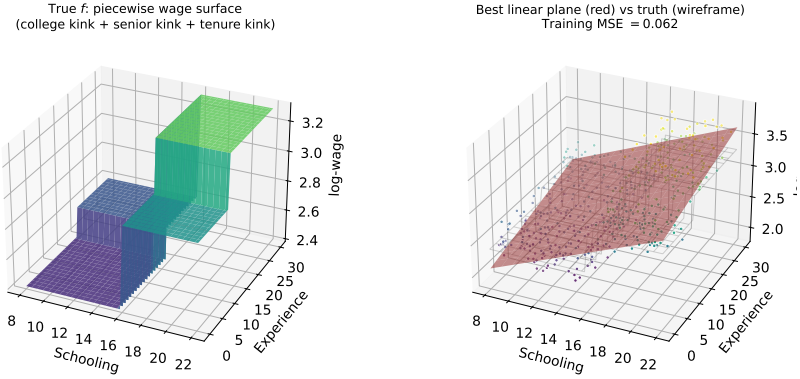


True wage surface. Returns jump at degree and experience thresholds; the slope is not constant everywhere.

Before we fit a tree

1. Why should an economist care about non-linearities at all? Give one example where "one more unit of X" probably does not have the same effect at low and high levels of X.
2. In a wage equation, why might the return to an extra year of schooling differ before and after a degree threshold? What would a purely linear model miss?

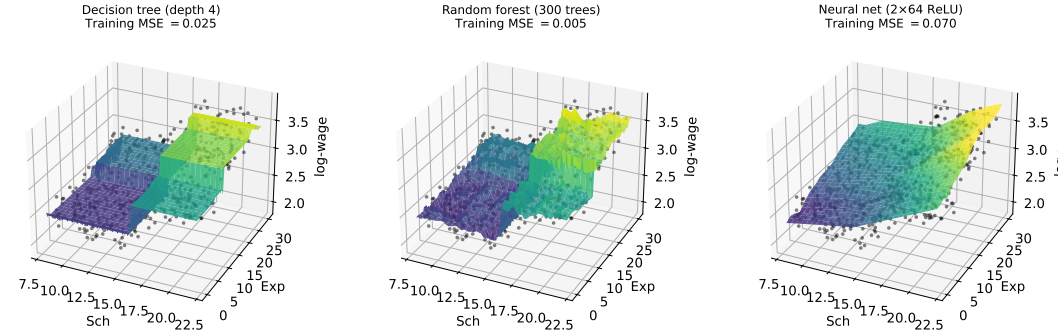
Where linear models break: Mincer wages in 3D



$$f(s,e) = 2.4 + 0.55\,\mathbf{1}\{s > 16\} + 0.35\,\mathbf{1}\{s > 16,\ e > 12\} + 0.20\,\mathbf{1}\{s \le 16,\ e > 18\}.$$

Truth (left): three threshold jumps — college, senior-college, and senior non-college. **Best linear plane** (right, red) tilts upward in both directions and misses every step. Training MSE = 0.062, almost entirely *bias*.

Non-linear methods recover the kinks



Same training data, three different hypothesis classes. The decision tree produces sharp axis-aligned steps that match the truth's piecewise structure. The random forest averages 300 jagged trees into a smoother step pattern; almost-zero training MSE on this DGP. The neural network (two ReLU hidden layers of 64 units) builds smooth ramps that approximate each kink. All three are vastly better than the linear plane on the previous slide.

Two routes to non-linearity

Route 1: hand-crafted features

- Build polynomials, interactions, splines.
- Feed to a linear model.
- This is what economists already do.
- Scales badly: you have to know in advance which non-linearities matter.

Route 2: learned non-linearity

- Pick a class that is itself non-linear.
- Let the optimiser discover the relevant structure from data.
- Trees, forests, neural networks.
- This is what distinguishes modern ML.

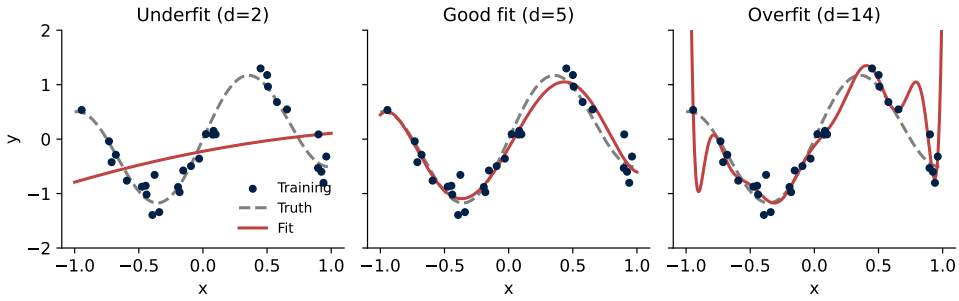
Checkpoint: hand-code or learn?

Specification judgement

When is it better to hand-code a non-linearity yourself, such as adding a square or an interaction term, and when might it be better to let a machine-learning method search for non-linear structure?

Bias-variance, in one picture — setup

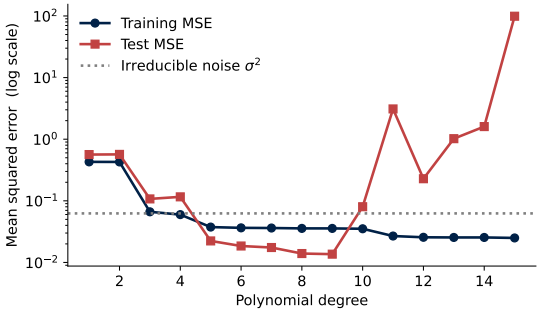
Synthetic DGP. $Y = f(X) + \varepsilon$ with $f(x) = \sin(1.5\pi x) + 0.5\,x$, $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, $\sigma = 0.25$, $X \sim \text{Unif}[-1, 1]$, training $n = 30$. We fit OLS in the polynomial basis $\{1, x, x^2, \ldots, x^d\}$ and sweep the degree d.



Same data, three different degrees. The right panel has *lower* training error than the middle one — and *worse* test error. That is the trade-off, made visible.

Bias-variance, in one picture — the U-curve

$$\mathrm{MSE}(\hat{f}) = \underbrace{\mathrm{Bias}(\hat{f})^2}_{\text{class too simple}} + \underbrace{\mathrm{Var}(\hat{f})}_{\text{fit too sensitive}} + \underbrace{\sigma^2}_{\text{irreducible noise}} .$$



Sweep d from 1 to 15, log-MSE on a 1000-point test grid.

- Training MSE falls monotonically: a richer class always fits the train set better.
- Test MSE is U-shaped: it bottoms out near $d \approx 5$, then climbs as variance dominates.
- Dotted line: the irreducible noise floor $\sigma^2 = 0.0625$. *No model can dip below it.*

Reading. Left of the minimum we are *bias-limited*; right of it, *variance-limited*. Cross-validation is the empirical instrument we use to find the bottom.

Optional detail: bias-variance derivation · double descent

Cross-validation: choosing complexity without using the test set

The U-curve is a warning: training error keeps falling, but test error eventually rises.
Cross-validation is how we estimate where the bottom is *before* touching the final test set.

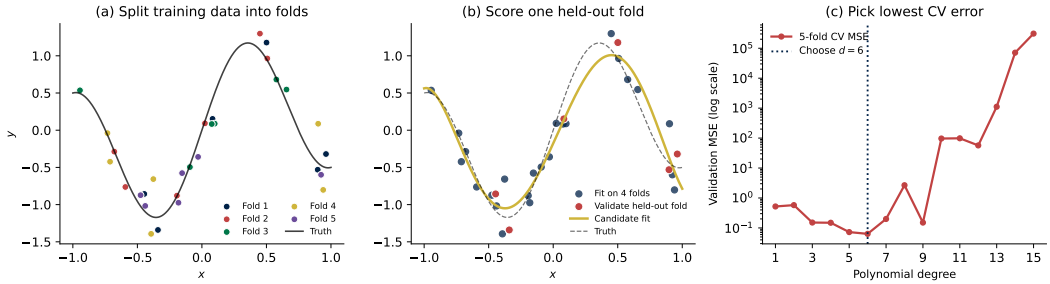
1. Split the training data into K folds.
2. For each candidate complexity λ (degree, tree depth, number of trees, NN size), repeat:

$$\widehat{R}_{\mathrm{CV}}(\lambda) = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{|I_k|}\sum_{i\in I_k}\ell\left(y_i, \widehat{f}_{\lambda}^{(-k)}(x_i)\right).$$

3. Choose the λ with the lowest validation loss, then refit on all training data.
4. Use the test set once, at the end, for the final reported number.

Key discipline. Validation is for model choice; test is for honest evaluation.

Cross-validation on the polynomial example



Same data as the bias-variance example. Split the training set into folds, fit each candidate polynomial degree on four folds, score it on the held-out fold, and average across folds. The chosen degree is the one with lowest validation error, not lowest training error.

Trees and forests

Decision trees: the model (ISLP §8.1)

Partition feature space into J disjoint rectangles $R_1, \dots, R_J$, predict a constant in each.

$$\widehat{f}(x) = \sum_{j=1}^{J} \widehat{c}_j \, \mathbf{1}\{x \in R_j\}, \qquad \widehat{c}_j = \frac{1}{|R_j|} \sum_{i : x_i \in R_j} y_i.$$

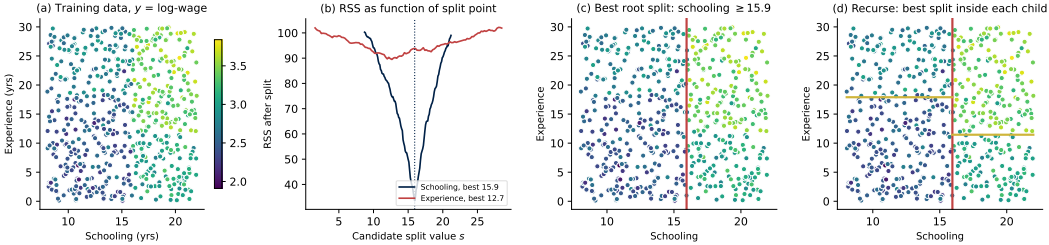
How are the rectangles found? **Recursive binary splitting**: at each node, find the variable j and cutpoint s that minimise the within-node RSS,

$$(j^\star, s^\star) = \arg\min_{j,s} \sum_{i : x_{ij} < s} (y_i - \bar{y}_L)^2 + \sum_{i : x_{ij} \geq s} (y_i - \bar{y}_R)^2,$$

where $\bar{y}_L, \bar{y}_R$ are the means of y in the resulting left and right children. Recurse on each child until a stopping rule kicks in (max depth, min leaf size).

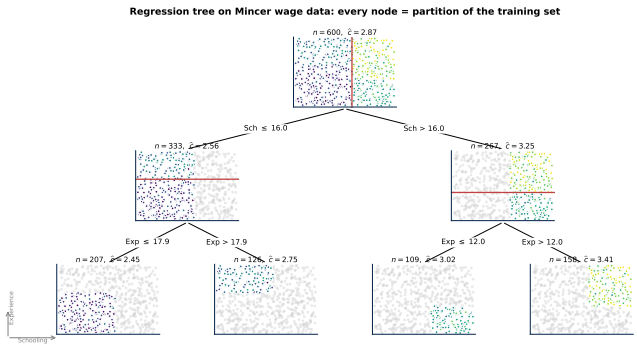
Classification trees replace RSS with a node impurity: *Gini* $\sum_k \hat{p}_k(1 - \hat{p}_k)$ or *cross-entropy* $-\sum_k \hat{p}_k \log \hat{p}_k$.

How the first split is chosen — a Mincer wage example



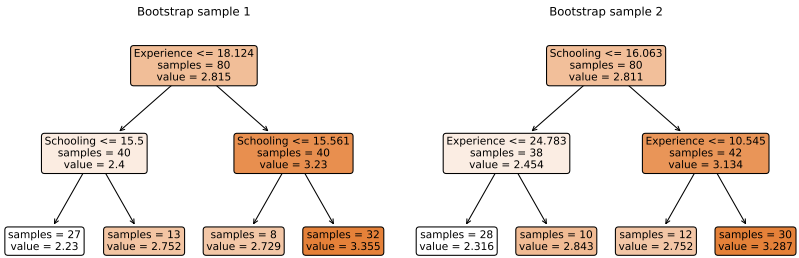
- Data: synthetic Mincer wages, $n = 600$, with schooling, experience, and log-wage.
- At each candidate split, compute RSS in the two child nodes and pick the smallest RSS.
- First split: schooling at $s^\star \approx 16$, recovering the college threshold.
- Then recurse: each child gets its own best split, here both on experience.

The fitted tree on the same data



Each node = the slice of (schooling, experience)-space it owns. Leaves: *junior non-college* ($\hat{c} \approx 2.45$), *senior non-college* (2.75), *junior college* (3.02), *senior college* (3.41) — the tree has rediscovered the Mincer kinks.

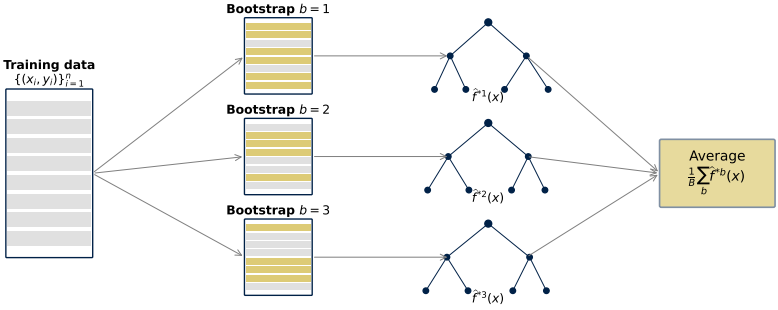
Trees are unstable: same DGP, two bootstrap samples



The bootstrap: sample n observations *with replacement* from $\{(x_i, y_i)\}_{i=1}^{n}$. Some points appear multiple times, others drop out — a cheap proxy for redrawing from the population.

Each panel: a depth-2 tree fit to one resample. Different cutpoints, sometimes different variables. Greedy splitting is discontinuous; observations near a cutpoint flip the downstream tree. *Fix: average many trees.*

Bagging (ISLP §8.2.1): average B bootstrap trees

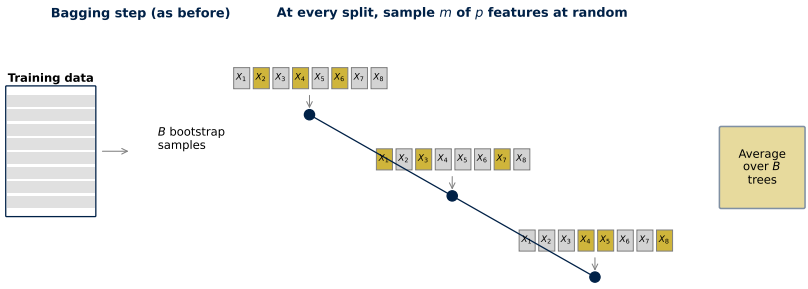


Each bootstrap reuses some rows and drops others ⇒ each tree sees a slightly different sample.

- Fit many trees on bootstrap resamples, then average their predictions: $\widehat{f}_{\text{bag}}(x) = B^{-1}\sum_{b=1}^{B}\widehat{f}^{*b}(x)$.
- Averaging reduces variance: unstable trees become a more stable predictor.
- Issue: bagged trees often choose the same strong root split, so their errors stay correlated.

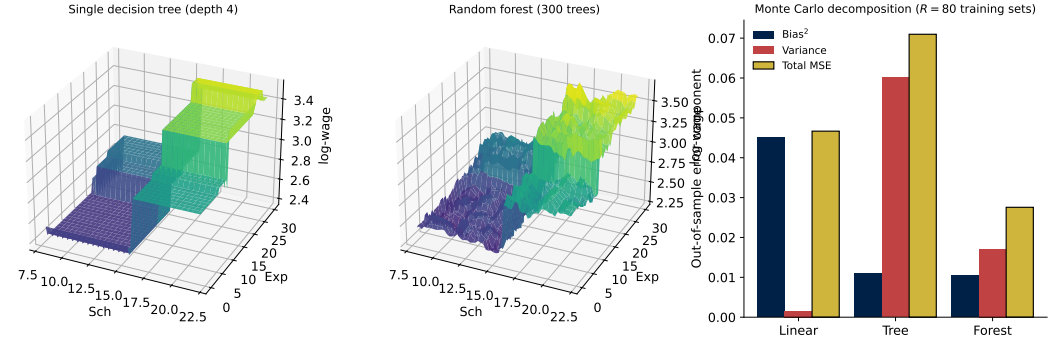
Optional detail: bootstrap, OOB, CV

Random forests (ISLP §8.2.2): bagging plus decorrelation



At every split, sample m of p features at random ($m = \sqrt{p}$ classification, $m = p/3$ regression). For B correlated estimators with pairwise correlation ρ, $\mathrm{Var}(\frac{1}{B}\sum Z_b) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$. Lower $\rho \Rightarrow$ better averaging.

Tree vs. random forest in 3D + bias / variance



- **Surfaces:** the single tree is coarse-stepped; the forest averages 300 deep trees and smooths the steps.
- **Bias:** linear model misses the kinks; tree and forest have low bias.
- **Variance:** the fully-grown tree is unstable, so its MSE is worst.
- **Forest:** keeps the tree's low bias but cuts variance by about 4×.

Boosting (ISLP §8.2.3): the algorithm

Bagging averaged B trees *independently*. Boosting fits them *sequentially*, each one targeted at what the previous ensemble got wrong.

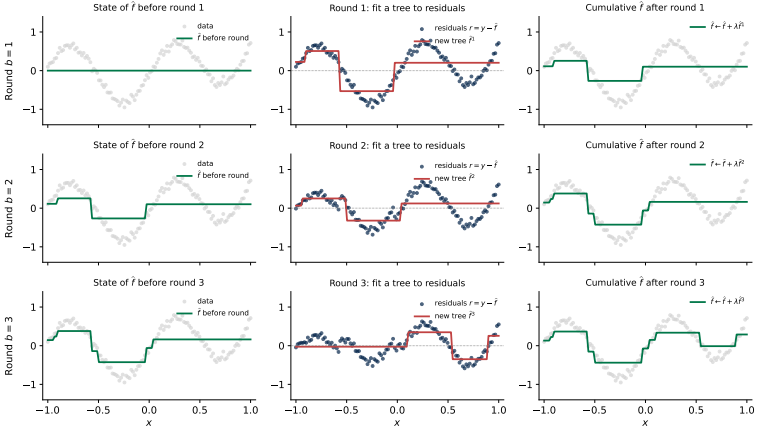
Algorithm (regression boosting).

1. Initialise $\widehat{f}(x) = 0$ and residuals $r_i = y_i$.
2. For $b = 1, \dots, B$: fit a small tree $\widehat{f}^b$ (depth $d \in [2, 6]$) to $\{(x_i, r_i)\}$; shrink-and-add $\widehat{f} \leftarrow \widehat{f} + \lambda \widehat{f}^b$; update $r_i \leftarrow r_i - \lambda \widehat{f}^b(x_i)$.
3. Return $\widehat{f}(x) = \sum_{b=1}^{B} \lambda \widehat{f}^b(x)$.

Three tuning knobs:

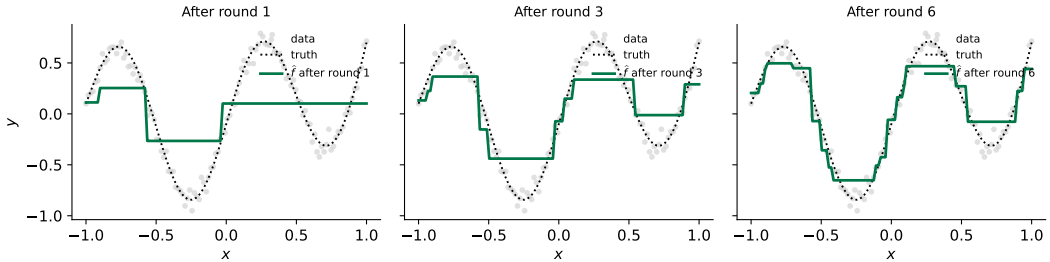
- B — number of rounds. Larger B overfits eventually.
- λ — learning rate ($\in (0, 1]$). Smaller λ needs more rounds, generalises better.
- d — per-tree depth. Controls the order of interaction each tree captures.

Boosting: rounds 1–3 in detail



Each row = one round. **Left:** $\widehat{f}$ before. **Middle:** residuals + new tree. **Right:** cumulative $\widehat{f}$. Each tree explains only the leftover.

Boosting: convergence after a few rounds



Same DGP. After round 1, $\widehat{f}$ is a single coarse tree — bias-dominated. After round 3, it has resolved the main wave. After round 6, it is close to the truth (dotted black). With $\lambda < 1$ each tree contributes a small correction, so B has to be reasonably large; that is why B and λ trade off: halving λ roughly doubles the B you need.

Boosting: why it dominates tabular ML

- **Bagging vs. boosting — different goals.** Bagging reduces *variance* by averaging high-variance trees. Boosting reduces *bias* by sequentially correcting a low-variance starting point. The two are complementary, but boosting typically wins on structured tabular data where bias is the binding constraint.
- **Practical recipe (XGBoost / LightGBM / CatBoost).** Set $d \in \{4, 6, 8\}$, $\lambda \in \{0.01, 0.05, 0.1\}$, then choose B by early stopping on a validation set. Modern implementations add: column-subsampling per tree, ℓ_2 regularisation on leaf values, histogram-based split-finding for speed.
- **Gradient boosting view.** Replacing residuals $r_i = y_i - \widehat{f}(x_i)$ with the negative gradient of *any* differentiable loss, $-\partial_{\widehat{f}}\ell(y_i, \widehat{f}(x_i))$, generalises the algorithm to logistic / Poisson / quantile regression. The residual update is the squared-error special case.
- **In economics.** Kleinberg, Lakkaraju, Leskovec, Ludwig, Mullainathan (2018, *QJE*) train gradient boosting on bail-decision data and show that ranking defendants by predicted risk Pareto-dominates judges. Boosting, not OLS, is what made the predictive performance possible.

What economists actually use trees for

Three roles, one example each. (Worked HTE / DML examples in Appendix A.0 and B.0.)

1. **Prediction-policy problems** — some decisions only need $\widehat{y}$, not $\widehat{\beta}$.
   - *Kleinberg et al. (2018, QJE)*. Gradient boosting trained on NYC bail data: ranking defendants by predicted re-arrest risk Pareto-dominates the judges. Same crime rate with ~ 25% fewer jailings, or 25% lower crime with the same number of jailings.
2. **Heterogeneous treatment effects** — which subgroups are most affected by a treatment?
3. **Nuisance estimation in double/debiased ML** — absorb high-dimensional controls flexibly, then run a clean low-dimensional causal regression on top.

Optional detail: variable importance | importance plot | causal forests | DML

Neural networks

Why neural networks, given that trees already work?

Trees / forests / GBMs are excellent on *tabular* data. So why pile on another model class?

- **Trees ingest a tidy table.** Every method we have seen so far requires a finite-dim row $x \in \mathbb{R}^p$ chosen in advance by the analyst. They do not eat raw text, raw pixels, raw audio, or sequences.
- **Trees cannot extrapolate.** A regression tree predicts a constant inside each leaf. Outside the training range of Y, every prediction is bounded by what was seen — harmful for forecasting.
- **Trees cannot share structure across inputs.** Each split looks at one variable. There is no mechanism to learn, say, that "edges" are a useful summary of pixels regardless of where they appear in an image. Convolution, recurrence, and attention encode such priors directly into the architecture.
- **Trees do not learn features.** A neural network learns its own basis $h_k(X)$ end-to-end with the prediction objective. Once you have a learned representation you can hand the embedding back to a forest, a GBM, or a double-ML pipeline. *This is what next lecture is about.*

Rule of thumb. Tabular and modest n: GBM. Raw unstructured data: deep learning earns its keep.

Single-layer neural network (ISLP §10.1)

Build K hidden-unit features from the inputs:

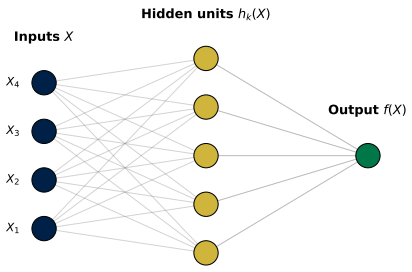
$$h_k(X) \equiv A_k = g\Big(w_{k0} + \sum_{j=1}^{p} w_{kj}\, X_j\Big),$$

then a linear output layer in the activations:

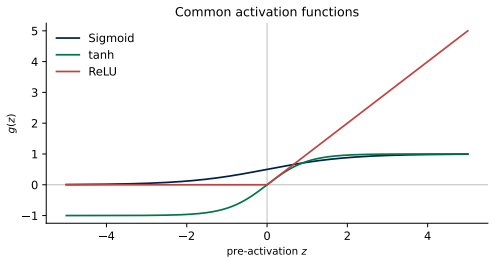
$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k\, h_k(X).$$

- $g(\cdot)$: fixed non-linear activation.
- Parameters: weights w_{kj}, biases w_{k0}, β_k.
- $h_k(X)$ are learned basis functions: each unit learns one non-linear transformation of X.

Optional detail: OLS/logit as NNs



Activation functions $g(z)$



- **Sigmoid:** $g(z) = 1/(1+e^{-z})$. Smooth probability-shaped curve; saturates at both tails.
- **ReLU:** $g(z) = \max(0, z)$. Zero for negative inputs, linear for positive inputs. *Modern default.*
- **tanh:** S-shaped like sigmoid, but centred at zero.

Why the non-linearity is essential. Without $g(\cdot)$, every A_k is linear in X, and $f = \beta_0 + \sum_k \beta_k A_k$ collapses back to a single linear model in X. The non-linearity is what buys us anything new.

Why this class can capture interactions

Take $p = 2$, $K = 2$, and the toy activation $g(z) = z^2$ with weights chosen so that

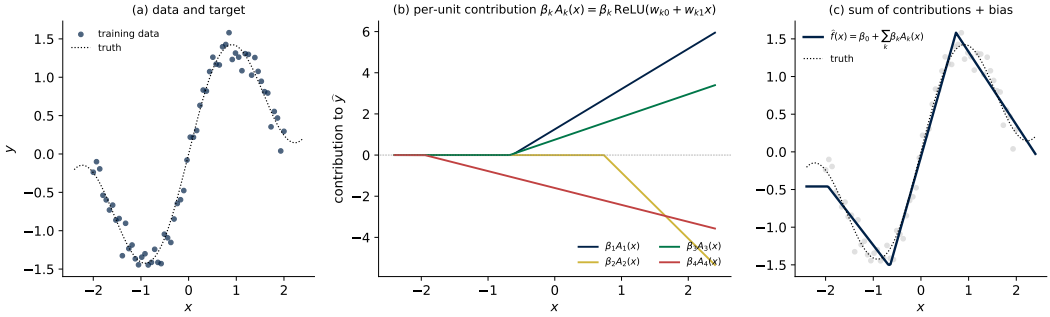
$$A_1 = (X_1 + X_2)^2, \qquad A_2 = (X_1 - X_2)^2.$$

Then for output coefficients $\beta_1 = \frac{1}{4}$, $\beta_2 = -\frac{1}{4}$,

$$f(X) = \beta_1 A_1 + \beta_2 A_2 = X_1 X_2.$$

- A sum of *linear functions of X pushed through a non-linearity* produces a genuine *interaction* in X.
- No one wrote $X_1 X_2$ down. The network found it inside its hypothesis class.
- ReLU / sigmoid do this generically, not just for one polynomial. (ISLP §10.1, the small worked example.)

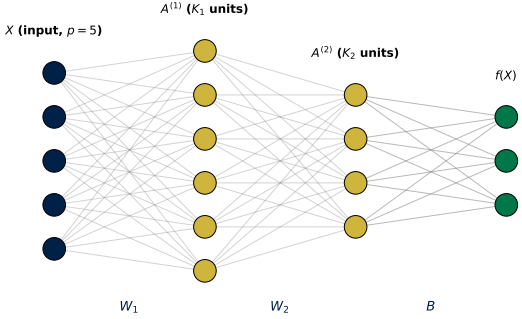
Worked example I (regression): how a tiny NN maps x to $\widehat{y}$



- Single hidden layer: $K = 4$ ReLU units, scalar input x, scalar output $\widehat{y}$.
- Each hidden unit is a ramp: $\beta_k A_k(x) = \beta_k \mathrm{ReLU}(w_{k0} + w_{k1}x)$.
- Ramps can switch on at different kinks and point in different directions.
- The prediction is their sum plus β_0: non-linearity is built from simple pieces.

Optional detail: numeric forward pass | classification example

Multilayer networks (ISLP §10.2)



Stack the same construction: $A_k^{(1)} = g(w_{k0}^{(1)} + \sum_j w_{kj}^{(1)} X_j)$, then $A_\ell^{(2)} = g(w_{\ell 0}^{(2)} + \sum_k w_{\ell k}^{(2)} A_k^{(1)})$, then $f(X) = \beta_0 + \sum_\ell \beta_\ell A_\ell^{(2)}$. Weights gather into W_1, W_2, B. ISLP's MNIST example: 235,146 parameters on 60,000 training images.

Optional detail: why depth · softmax outputs · universal approximation

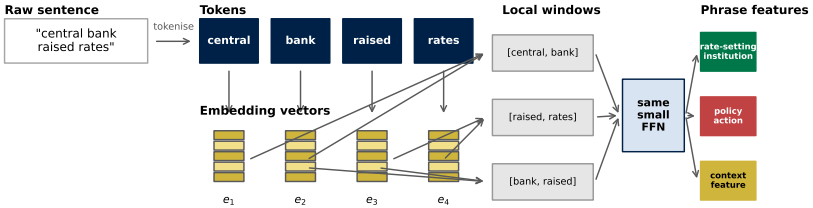
Why depth matters

A deeper network learns features in stages, rather than trying to jump from raw input to prediction in one step.

- **Tabular data:** early layers can build interactions; later layers combine those interactions into predictions.
- **Images:** pixels → edges → shapes → objects.
- **Text:** tokens → phrases → syntax and meaning → next-token probabilities.
- **Economic data:** the same idea applies when raw inputs are documents, news, earnings calls, job ads, satellite images, or transaction sequences.

Intuition. Depth is useful when the signal is compositional: useful high-level features are built from lower-level features.

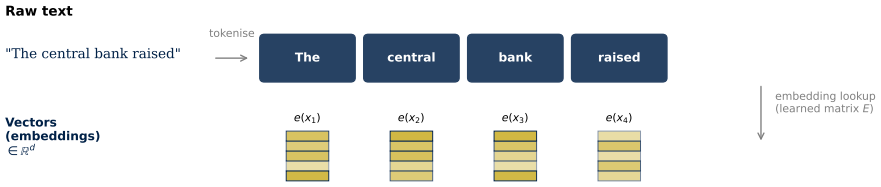
A tiny text example: tokens → phrases



Point: after tokenisation, the network no longer sees words as strings. It sees vectors, and feed-forward layers combine nearby vectors into useful phrase-level features.

- Tokenisation turns the sentence into a short sequence: `central`, `bank`, `raised`, `rates`.
- Each token becomes a vector. A feed-forward layer can combine nearby vectors to create phrase-level features.
- Here, `central bank` and `raised rates` become useful local patterns. Syntax and richer meaning require stronger sequence models, which we return to later.

From a sequence of words to a vector NNs can read



A neural network takes vectors. Text is a sequence of symbols. Bridge:

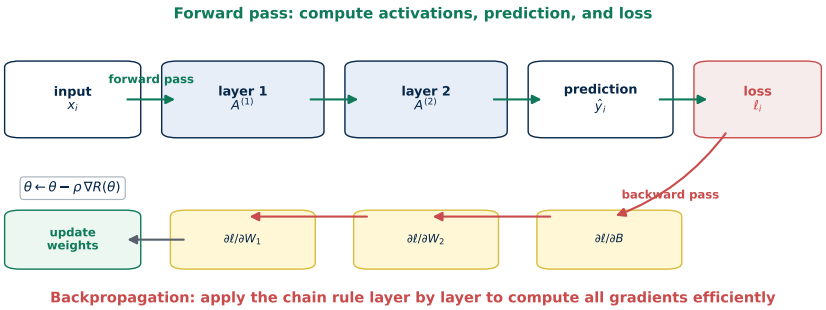
- **Tokenise.** Split the string into a finite vocabulary V of subwords.
- **Embed.** Each token id j is mapped to a vector $e_j \in \mathbb{R}^d$ via a learned lookup matrix $E \in \mathbb{R}^{|V| \times d}$. Tokens that play similar roles end up near each other in $\mathbb{R}^d$.
- **Stack.** A sentence becomes an ordered list of vectors $(e_{x_1}, e_{x_2}, \ldots, e_{x_T})$ that an NN can ingest.

E is learned jointly with the rest of the network — nothing has to be specified by hand.

Optional detail: width vs. depth · architectures · CNNs · scaling laws

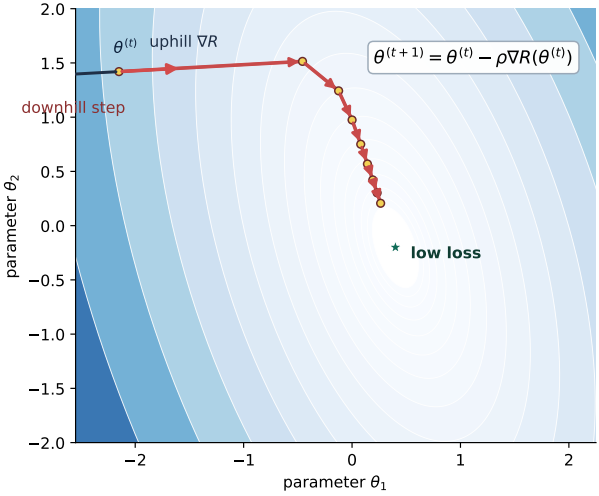
Training Neural Nets

Forward pass, backward pass, backpropagation



- **Forward pass:** push one minibatch through the network to compute activations, predictions $\hat{y}$, and loss $R(\theta)$.
- **Backward pass:** propagate the loss sensitivity backwards through the layers.
- **Backpropagation:** chain rule applied efficiently to compute all gradients $\partial R / \partial \theta_j$.

Gradient descent: the basic idea



We have an objective $R(\theta)$: average loss over the training data.

- The gradient $\nabla R(\theta)$ points in the direction where loss rises fastest.
- So we step the other way:

$$\theta^{(t+1)} = \theta^{(t)} - \rho_t \nabla R(\theta^{(t)}).$$

- In a neural net, θ is every weight and bias. Backpropagation computes this gradient efficiently.
- The practical question is how to make these steps cheap, stable, and generalisable.

Gradient methods for numerical optimisation

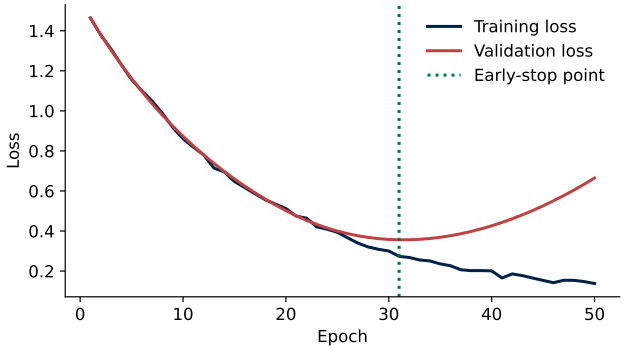
Once the model is non-linear, $R(\theta)$ has no closed form. Gradient methods iterate $\theta^{(t+1)} = \theta^{(t)} - \rho_t \mathscr{G}^{(t)}$, $\mathscr{G}^{(t)} \approx \nabla_\theta R(\theta^{(t)})$. Methods differ only in how $\mathscr{G}^{(t)}$ is formed:

- **Full-batch GD.** $\mathscr{G}^{(t)} = \nabla R(\theta^{(t)})$ exactly. Deterministic; one pass per step.
- **SGD.** Random minibatch $\mathscr{B}$: $\mathscr{G}^{(t)} = \frac{1}{|\mathscr{B}|} \sum_{i \in \mathscr{B}} \nabla \ell_i$. Cheap, unbiased, noisy.
- **Momentum.** Smooth past gradients: $\mathscr{G}^{(t)} = \mu \mathscr{G}^{(t-1)} + (1-\mu)\nabla R$. Helps narrow valleys.
- **Adaptive (Adam, AdamW).** Per-coordinate rescaling by running RMS of past gradients.
- **Newton.** $[\nabla^2 R]^{-1} \nabla R$: quadratic convergence near minima, but $O(P^3)$ per step.

Why first-order dominates ML. The gradient costs one forward+backward pass; higher-order info is too expensive at $P \sim 10^8$. Theory in Appendix C.

Optional detail: parameter count · matrix GD · GD vs SGD · learning rate · SGD code · theory

Training and validation loss across epochs



Training loss falls (almost) forever; validation loss starts to creep up.

Early stopping: halt at the validation minimum. The simplest and most-used NN regulariser.

Optional detail: double descent

Three regularisers for neural networks

- **Ridge (weight decay):** add $\lambda \sum \theta_j^2$ to $R(\theta)$. Same idea as ridge regression in Week 1; usually applied to W_1, W_2 and not to the output biases.
- **Early stopping:** halt SGD when held-out loss stops improving. Implicitly limits how far θ travels from its random initialisation.
- **Dropout:** on each training step, randomly zero out a fraction ϕ of the activations $A_k^{(\ell)}$ and rescale the survivors by $1/(1-\phi)$. Prevents co-adaptation between units; equivalent in spirit to averaging an ensemble of sub-networks.

All three exist for the same reason as ridge/lasso in Week 1: very rich classes need to be *constrained*.

Checkpoint: what do we lose?

Prediction is not the only criterion

What do we lose when we move from a simple linear model to a flexible non-linear model? Think about interpretation, overfitting, and explaining the result to a policymaker.

Recap

Practical recipe for any new prediction problem

1. **Baseline:** linear regression (or logistic for classification).
2. **Regularise:** ridge / lasso / elastic net with cross-validated λ.
3. **Capture interactions cheaply:** gradient boosting.
4. **Only if 1–3 are visibly under-performing on a known-non-linear problem, or if you're working with text / images:** reach for a neural network.

Occam, restated: when several methods give roughly equivalent performance, pick the simplest.

Optional detail: Hitters example

Checkpoint: final model choice

Prediction ties are not decision ties

Suppose a random forest, a boosted tree model, and a neural network all predict equally well. Why might the simplest or most interpretable one still be the best choice for an empirical economics project?

Recap

- Supervised learning template: hypothesis class + loss + optimiser.
- Bias-variance trade-off: every modelling decision sits on this curve.
- Two routes to non-linearity: hand-crafted features vs. learned non-linearity.
- Trees → bagging → random forests → gradient boosting: the workhorse for tabular data.
- Neural networks: composed affine maps + non-linearities; trained by SGD + backprop.
- Deep learning shines on rich, structured data with lots of it. *Often overkill for tabular work.*

Coming up: chapter 4

Next week: **Text data, pre-processing, and a first look at language models.**

- Why text is unlike anything we've seen so far.
- Bag-of-words, tf-idf, topic models.
- Word embeddings: turning semantics into geometry.
- Cliffhanger into chapter 5: contextual embeddings → transformers.

Tutorial on Friday: 4 exercises on bias-variance, trees, UK House Price Index, tiny PyTorch MLP.
Notebook is in `lectures/week_03/tutorial_03.ipynb`.

Appendix 0.1 — Loss functions: what changes?

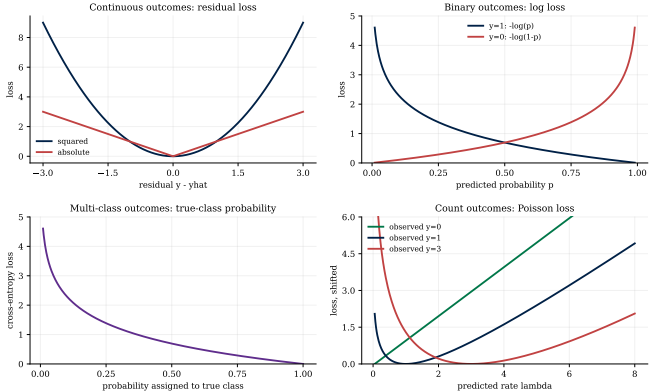
The loss function says what kind of mistake is costly. The function class decides what shapes $\widehat{f}$ can take; the loss decides how we score each candidate shape.

Outcome type	Typical prediction	Common losses
Continuous $Y \in \mathbb{R}$	Mean / point forecast	Squared error, absolute error, Huber
Binary $Y \in \{0, 1\}$	Probability $p(x)$	Log loss / cross-entropy, hinge loss
Multi-class $Y \in \{1, \ldots, C\}$	Class probabilities	Multinomial cross-entropy
Counts $Y \in \mathbb{N}$	Rate $\lambda(x) > 0$	Poisson negative log-likelihood

Rule. Pick a loss whose geometry matches the data object you are predicting: real value, probability, class label, or count.

Back

Appendix 0.2 — Loss functions: what the curves do



Geometry. Squared/absolute losses score residual size; cross-entropy punishes confident wrong probabilities; Poisson loss scores predicted count rates.

Back

Appendix 0.3 — Bootstrap, OOB error, and CV

- **Bootstrap sample.** Draw n rows with replacement from the training data. Some observations appear more than once; about one third are left out.
- **Bagging.** Fit one tree per bootstrap sample and average predictions. This reduces variance because each tree sees a slightly different dataset.
- **Out-of-bag (OOB) error.** For observation i, predict using only trees whose bootstrap sample omitted i. This gives a validation-style error estimate without a separate validation split.
- **Connection to CV.** Both OOB and cross-validation estimate out-of-sample error using only the training set; OOB is the version that comes almost for free with bagging.

Caveat. OOB estimates performance, but it does not decorrelate the trees. Random forests add feature subsampling to do that.

Back

[RECAP] Three things ML cares about that classical econometrics often does not

1. **Out-of-sample prediction is the gold standard.**
   Always split into train/test; report test error.
2. **The hypothesis class can be enormous.**
   From a handful of βs to millions of weights. Regularisation becomes first-class.
3. **Parameters often don't have a clean interpretation.**
   The *prediction* $\hat{f}(x)$ has meaning; individual weights inside an RF or NN often don't.

Mullainathan & Spiess (2017): ML lives in the $\hat{y}$ part of the toolbox, not the $\hat{\beta}$ part.

◂ Back

Discussion: questions to keep in mind

1. Why is a linear model with hand-crafted features (squares, interactions, splines) often *not enough* when p is moderately large? What goes wrong?
2. A neural network is described as "a linear model in learned features." What exactly is being learned, and why does that matter for an economist?
3. Suppose two methods — a tuned random forest and a deep neural network — produce the same test error on your dataset. Which would you ship, and why?

We come back to all three by the end of the lecture.

Back

The bias-variance trade-off

Take a fresh test pair $(X, Y) \sim \mathbb{P}$ with $Y = f(X) + \varepsilon$, $\mathbb{E}[\varepsilon \mid X] = 0$, $\mathrm{Var}(\varepsilon \mid X) = \sigma^2$. Expected squared error of $\widehat{f}$, integrated over both X *and* the training sample:

$$\mathbb{E}\Big[(Y - \widehat{f}(X))^2\Big] = \underbrace{\mathbb{E}_X\big[(\mathbb{E}\,\widehat{f}(X) - f(X))^2\big]}_{\text{Bias}^2} + \underbrace{\mathbb{E}_X\big[\mathrm{Var}\,\widehat{f}(X)\big]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Irreducible}}.$$

- **Bias** is the systematic miss of the class (averaged over the input distribution): even with $n \to \infty$, can $\mathscr{F}$ *represent* f?
- **Variance** is the sensitivity of $\widehat{f}$ to which training set we drew (averaged over X): how much would $\widehat{f}$ wobble if we resampled the data?
- **Irreducible** σ^2 is a floor: the noise in Y that no $\widehat{f}$ can absorb.

- Linear model: low variance, possibly high bias.
- Deep neural net: potentially low bias, potentially huge variance.
- **The art:** pick a class *just rich enough* to capture the structure without chasing the noise.

Back

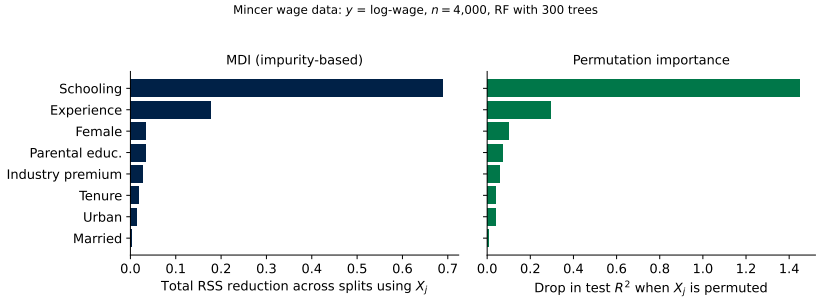
Variable importance: two flavours, one question

Individual weights inside a forest are uninterpretable. We instead ask: how much does each input X_j contribute to $\widehat{f}$?

- **MDI (Mean Decrease in Impurity).** Sum the RSS reduction across every split that used X_j, averaged over the B trees: $\mathrm{MDI}_j = \frac{1}{B}\sum_b \sum_{t:\, v(t)=j} \Delta\mathrm{RSS}(t)$. Free during training. Caveat: biased toward high-cardinality features and over-credits one winner inside a correlated cluster.
- **Permutation importance.** Fit the model. On a held-out set, shuffle column X_j (breaks the link between X_j and Y). Recompute test R^2. The drop is what the model loses when X_j is destroyed: $\mathrm{PI}_j = R^2(\widehat{f}) - \mathbb{E}\, R^2(\widehat{f}$ on X_j-shuffled test). More expensive, but measures *predictive value* and is robust to correlation.

Back

Reading the Mincer importance plot



RF (300 trees), $n = 4{,}000$, y = log-wage, eight inputs. Both panels agree: schooling is far and away top (it captures the college kink); experience is second (it gates the senior and tenure premia). Tenure ranks lower than experience despite its DGP coefficient because it is correlated with experience and the forest substitutes one for the other.

Back

OLS and logistic regression *are* single-layer NNs

Both classical workhorses are degenerate cases of the previous slide. Set $K = 1$ and pick $g(\cdot)$:

- **OLS.** Take $g(z) = z$ (identity), so $A_1 = w_{10} + \sum_j w_{1j} X_j$ and $f(X) = \beta_0 + \beta_1 A_1$ is affine in X. With squared-error loss, $\widehat{\beta}$ minimises $\sum_i (y_i - f(x_i))^2$ — OLS exactly (modulo a redundant rescaling of w_1. vs. β_1).
- **Logistic regression.** Take $g(z) = \sigma(z) = 1/(1 + e^{-z})$ and freeze $\beta_0 = 0, \beta_1 = 1$:

$$f(X) = \sigma\Big(w_{10} + \sum_{j=1}^{p} w_{1j} X_j\Big) = \Pr(Y = 1 \mid X).$$

  Trained with cross-entropy, this is logistic regression exactly.

Take-away. A neural network is not a different family — it is the same template, with K enlarged and g non-linear. Adding hidden units gets us past the linear class. Adding layers is next.

Back

Worked example I: the same forward pass, in numbers ($x^\star = 0.7$)

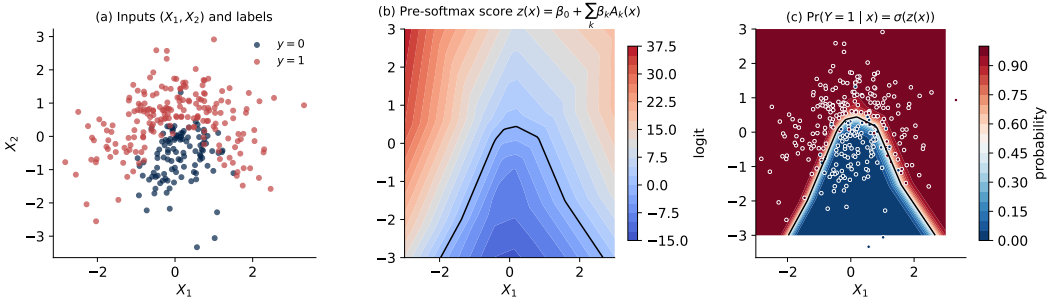
Activations at $x^\star = 0.7$; $\hat{y} = \beta_0 + \sum_k \beta_k A_k = -0.46 + (+1.97) = 1.51$

k	w_{k0}	w_{k1}	$z_k = w_{k0} + w_{k1} x^\star$	$A_k = \mathrm{ReLU}(z_k)$	β_k	$\beta_k A_k$
1	0.48	0.75	1.01	1.01	2.60	2.62
2	-1.54	2.09	-0.07	0.00	-1.53	-0.00
3	0.47	0.70	0.97	0.97	1.57	1.52
4	1.70	0.87	2.31	2.31	-0.94	-2.17

Read each row left-to-right: pre-activation $z_k = w_{k0} + w_{k1} x^\star$, ReLU output $A_k = \max(0, z_k)$, output weight β_k, contribution $\beta_k A_k$ to the prediction. Unit 2 is silent ($z_2 = -0.07 < 0$ so $A_2 = 0$): its weight $\beta_2 = -1.53$ contributes nothing at this x. The remaining three units sum to $+1.97$; add the output bias $\beta_0 = -0.46$ to get $\hat{y}(x^\star) = 1.51$. This is exactly what the network does on every forward pass.

Back

Worked example II (binary classification): from inputs to probabilities



$p = 2$ inputs, $K = 8$ ReLU units, sigmoid output. *(a)* Training labels with a curved true boundary. *(b)* Last linear layer's score $z(x) = \beta_0 + \sum_k \beta_k A_k(x)$; black curve = level set $z = 0$. *(c)* $\Pr(Y = 1 \mid x) = \sigma(z(x))$, bounded in $[0, 1]$ with 0.5 contour at $z = 0$. Rule: qualitative response = score (from learned features) + sigmoid / softmax.

Back

Why we want *multiple* hidden layers

Universal approximation says one wide hidden layer can already represent any continuous function. So why ever go deeper?

- **Compositional data.** Real signals are hierarchical: pixels → edges → shapes → objects; characters → words → sentences. Each hidden layer builds features *from the previous layer's features* — exactly the abstraction ladder the data has.
- **Parameter efficiency.** A function representable by a depth-L network with $\mathcal{O}(L \cdot K)$ parameters can require width $\sim K^L$ in a single-hidden-layer network for the same accuracy (Telgarsky 2016; Eldan & Shamir 2016). Depth is exponentially cheaper for compositional targets.
- **Reusable sub-features.** A single hidden layer must rebuild every interaction from raw X. With multiple layers, a useful intermediate feature is *shared* across all higher-level concepts that depend on it.
- **Empirical regularity.** On vision, language, and speech, depth reliably helps while pure width saturates. Best-in-class architectures are deep ($L \in [10, 100+]$) and only moderately wide.

Template view. Hidden units extend the hypothesis class *horizontally*; hidden layers extend it *vertically*, each new layer learning a basis built on the previous one's.

Back

Multiple outputs and the softmax

For a C-class classification (digits 0–9, $C = 10$), output C linear scores

$$Z_m = \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} A_\ell^{(2)}, \qquad m = 1, \ldots, C,$$

then turn them into probabilities with the **softmax**:

$$f_m(X) \;=\; \Pr(Y = m \mid X) \;=\; \frac{e^{Z_m}}{\sum_{m'=1}^{C} e^{Z_{m'}}}.$$

Train by minimising the **cross-entropy** (negative multinomial log-likelihood):

$$R(\theta) \;=\; -\sum_{i=1}^{n} \sum_{m=1}^{C} y_{im} \log f_m(x_i).$$

Continuity with week 2: this is exactly the multinomial-logit loss — now with *learned* features $A_\ell^{(2)}$ in place of raw x_{ij}.

Back

Universal approximation

Theorem (informal). A single-hidden-layer network with sufficient width K can approximate any continuous function on a compact set to arbitrary precision, under modest conditions on g.

- Reassures us that the hypothesis class $\{h_k\}$ is *rich enough*.
- Says nothing about how to *find* a good function inside it from finite data.
- In practice, *depth* (more layers) turns out to be more useful than width (more units in one layer) for hard problems — a hierarchical basis is a much more efficient way to spend parameters.

Back

Fitting a neural network: collecting the parameters

Every weight and bias is a scalar that must be optimised. We *flatten* all of them into one long vector $\theta \in \mathbb{R}^P$. For a network with two hidden layers,

$$\theta = \big[\, \mathrm{vec}(W_1), \mathrm{vec}(W_2), \mathrm{vec}(B) \,\big] \in \mathbb{R}^P, \qquad P = (p+1)K_1 + (K_1+1)K_2 + (K_2+1)C,$$

where $\mathrm{vec}(\cdot)$ stacks a matrix's columns into a vector and the "+1" counts the bias row in each layer.

The objective averages a per-example loss ℓ over the training set:

$$R(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell\big(y_i, f_\theta(x_i)\big), \qquad \ell = \tfrac{1}{2}(y-\hat{y})^2 \text{ (regression)}, \quad \ell = -\sum_m y_m \log \hat{y}_m \text{ (classification)}.$$

- $R(\theta)$ is *non-convex* in θ — multiple local minima, no closed form.
- In ISLP's MNIST example, θ has $P = 235{,}146$ entries on $n = 60{,}000$ training images.
- Strategies: slow iterative descent + regularisation ($\|\theta\|_2^2$ weight decay, dropout, early stopping).

Back

Stochastic gradient descent: cheap, noisy, fast

Replace the full-batch gradient with a minibatch estimate: sample $|\mathcal{B}| \in [32, 512]$ rows and use $\widehat{\nabla}_\theta R = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_\theta \ell(y_i, f_\theta(x_i))$, with $\mathbb{E}[\widehat{\nabla}_\theta R] = \nabla_\theta R(\theta)$. Update rule unchanged: $\theta \leftarrow \theta - \rho \, \widehat{\nabla}_\theta R$.

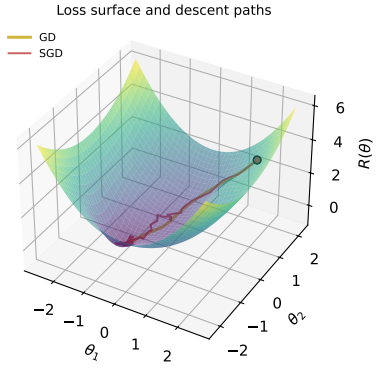
Pseudocode (one epoch):

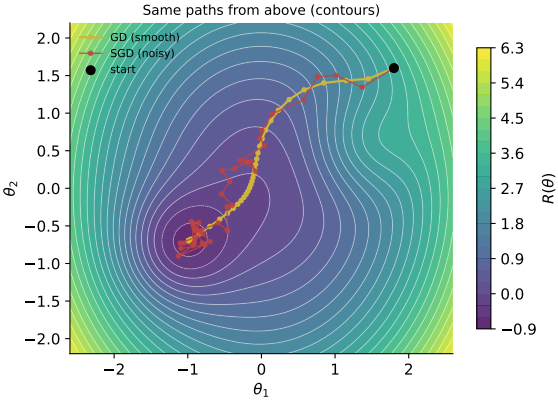
```
shuffle indices 1..n
for each minibatch B of size |B|:
    y_hat <- forward(X[B], theta)
    grad  <- backprop(X[B], y[B], y_hat, theta)   # vector in R^P
    theta <- theta - rho * grad
```

- One epoch = one full pass through the data ($\lceil n/|\mathcal{B}| \rceil$ steps); orders of magnitude faster wall-clock.
- Minibatch noise acts as mild regularisation. `PyTorch`: `loss.backward()` populates `.grad`; `optimizer.step()` applies the update. Variants (Adam, AdamW) rescale per-coordinate.

Back

GD vs SGD on a 2-D loss surface

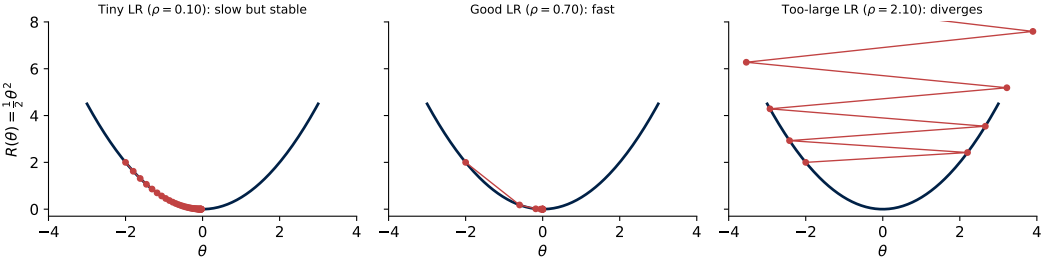




Same start, same ρ. GD (gold) follows the negative gradient smoothly into the basin. SGD (red) uses a noisy minibatch estimate, zigzags, but on average heads the same way.

Back

The learning rate matters



One-dimensional bowl $R(\theta) = \frac{1}{2}\theta^2$, gradient $= \theta$, update $\theta \leftarrow (1-\rho)\theta$. Stable when $|1-\rho| < 1$, i.e. $\rho \in (0, 2)$. Tiny ρ: converges, but every step is small. Good ρ: fast geometric decay. $\rho > 2$: the update overshoots and the iterate diverges. In practice we tune ρ on validation loss; modern optimisers (Adam, AdamW) adapt ρ per coordinate.

Back

Stochastic gradient descent: cheap, noisy, fast

Replace the full-batch gradient with a minibatch estimate: sample $|\mathcal{B}| \in [32, 512]$ rows and use $\widehat{\nabla}_\theta R = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_\theta \ell(y_i, f_\theta(x_i))$, with $\mathbb{E}[\widehat{\nabla}_\theta R] = \nabla_\theta R(\theta)$. Update rule unchanged: $\theta \leftarrow \theta - \rho \, \widehat{\nabla}_\theta R$.

Pseudocode (one epoch):

```
shuffle indices 1..n
for each minibatch B of size |B|:
    y_hat <- forward(X[B], theta)
    grad  <- backprop(X[B], y[B], y_hat, theta)   # vector in R^P
    theta <- theta - rho * grad
```

- One epoch = one full pass through the data ($\lceil n/|\mathcal{B}| \rceil$ steps); orders of magnitude faster wall-clock.
- Minibatch noise acts as mild regularisation. `PyTorch`: `loss.backward()` populates `.grad`; `optimizer.step()` applies the update. Variants (Adam, AdamW) rescale per-coordinate.

Back

Double descent (1/2): why the classical story breaks

Week 2's U-shaped curve: too simple → underfit; too rich → overfit. The classical advice was to stop adding parameters before you can interpolate the training set.

The puzzle. Modern networks have $P \gg n$. They interpolate ($\widehat{R} = 0$) and still generalise. The classical curve says this should not work.

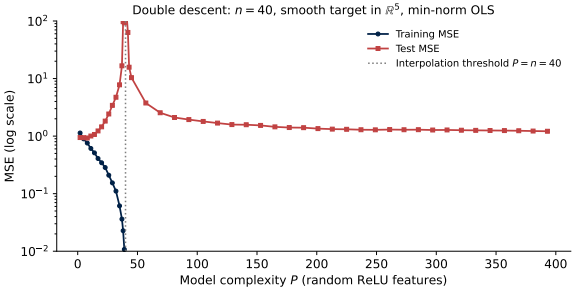
What changes near the interpolation threshold.

- $P < n$: OLS has a unique minimiser; $\mathrm{Var}(\widehat{\beta}) \propto \sigma^2 \,\mathrm{tr}((X^\top X)^{-1})$.
- $P \to n$: smallest eigenvalue of $X^\top X$ tends to 0; $\|\widehat{\beta}\| \to \infty$; test error spikes.
- $P > n$: $X\beta = y$ has *infinitely many* solutions. Pick the minimum-norm one $\widehat{\beta} = X^+ y$ (Moore-Penrose). With $P \gg n$, this $\widehat{\beta}$ has small norm and the predictor stabilises.

Implicit regularisation. Min-norm OLS, SGD from small init, and dropout all bias estimation toward small-norm interpolators. The classical U is just the first half of the curve.

Back

Double descent (2/2): a numerical simulation



Setup. $n = 40$, $x_i \in \mathbb{R}^5$. Min-norm OLS in random ReLU features, sweep P from 2 to 400 (avg. 30 draws).

- $P < n$: classical U.
- $P = n$: peak. Training error 0; norm explodes.
- $P > n$: min-norm interpolator has small weights $\Rightarrow$ second descent, eventually below the pre-peak best.

Belkin et al. 2019, *PNAS*; Nakkiran et al. 2020. Why huge networks can generalise.

Back

Width vs. depth

Two networks with the same parameter count:

- One hidden layer, 1,000 units.
- Ten hidden layers, 100 units each.

Universal approximation says either can in principle represent any function.

Empirically, depth wins on hard problems. The standard story: *depth allows hierarchical feature learning* that matches the compositional structure of real-world data.

Back

Architectural specialisations

- **CNNs** (convolutional): grid data — images, satellite imagery, spatial econometrics. *Weight sharing across spatial positions.*
- **RNNs / LSTMs** (recurrent): sequence data — text, time series. *Largely displaced by transformers.*
- **Transformers**: sequence and set data. The architecture behind every frontier language model. *Lecture 5.*

Pattern: pick an inductive bias appropriate to the data, encode it in the architecture.

Back

The Hitters lesson

ISLP §10.6: fit linear regression, lasso, and a one-hidden-layer NN to 263 baseball players.

Method	# Parameters	Test MAE
OLS	20	254.7
Lasso	12	252.3
Neural network	1,345	257.4

The neural network has 5x more parameters than training observations and *loses to OLS*.

Back

Appendix A.0 — HTE: a worked example

Setting. *One Summer Plus* (Chicago, 2012–13): ~ 1,600 disadvantaged high-school students were randomly assigned to a 25-hour-per-week summer job. Davis & Heller (2020, *ReStat*) ask whether the programme reduces violent-crime arrests — and *for whom*.

What the trees do. Each tree splits on baseline covariates X (age, prior arrests, school records, neighbourhood). Splits are scored not by RSS in y but by how different the within-leaf treatment effect $\hat{\tau}$ becomes between siblings (Athey & Imbens 2016). Honest splitting (Wager & Athey 2018) uses one half of the sample to choose splits and the other half to estimate $\hat{\tau}$ in each leaf, restoring asymptotic normality. The forest averages B such trees and outputs $\hat{\tau}(x)$ for any new x, with pointwise CIs.

Finding. The pooled ATE on violent-crime arrests is small and noisy. The forest shows the effect is *concentrated* on youth with extensive prior justice contact: large negative $\hat{\tau}(x)$ for the top quartile of baseline risk, indistinguishable from zero for the rest. Policy implication: target the programme rather than scale it.

`grf` (R), `econml` (Python). Theory in A.1–A.5 below.

◂ Back

Appendix A.1 — Set-up and target

Each unit has potential outcomes $\big(Y_i(0), Y_i(1)\big)$, treatment $D_i \in \{0,1\}$, and pre-treatment covariates $X_i \in \mathbb{R}^p$. Throughout we maintain

- Unconfoundedness. $\big(Y_i(0), Y_i(1)\big) \perp D_i \mid X_i$.
- Overlap. $\eta < e(x) := \Pr(D_i = 1 \mid X_i = x) < 1 - \eta$ for some $\eta > 0$.
- Consistency. $Y_i = D_i Y_i(1) + (1 - D_i) Y_i(0)$.

The estimand of interest is the conditional average treatment effect (CATE)

$$\tau(x) \;=\; \mathbb{E}\big[\, Y_i(1) - Y_i(0) \,\big|\, X_i = x \,\big].$$

Pooled ATE $= \mathbb{E}[\tau(X)]$ averages this away. Causal forests aim for the function $\tau(\cdot)$ itself.

Appendix A.2 — Causal trees: the splitting criterion

A standard regression tree picks splits to minimise $\sum_{\ell} \sum_{i \in \ell} (Y_i - \bar{Y}_{\ell})^2$. A *causal* tree (Athey & Imbens 2016, *PNAS*) instead targets variation in $\tau(x)$.

Inside each candidate leaf ℓ, define a leaf-specific treatment effect estimate

$$\hat{\tau}_{\ell} = \overline{Y}_{\ell, D=1} - \overline{Y}_{\ell, D=0},$$

the simple difference in means. The criterion (Athey–Imbens, expected MSE for treatment effects) approximately reduces to

$$\arg\max_{(j,s)} \sum_{\ell \in \{L,R\}} n_{\ell} \hat{\tau}_{\ell}^2 - \text{penalty}\left(\frac{\sigma_1^2}{n_{\ell,1}} + \frac{\sigma_0^2}{n_{\ell,0}} \right).$$

The first term rewards splits whose children disagree about $\hat{\tau}$; the second penalises noise from splitting either treatment arm too thinly. Reading. A tree built this way carves up $\mathcal{X}$ into regions of *treatment-effect homogeneity*.

Appendix A.3 — Honest splitting: why it matters

The problem. If the same observations both choose the splits and estimate $\hat{\tau}_\ell$, leaf-level estimates are biased: splits intentionally separate residuals into "high-$\hat{\tau}$" and "low-$\hat{\tau}$" bins, so $\hat{\tau}_\ell$ overstates within-leaf differences. Inference is invalid.

The fix. Split the training sample $\mathcal{S}$ in half:

- **Splitting half** $\mathcal{S}_{\text{tr}}$: used to choose the splits $(j_t^\star, s_t^\star)$ at every node.
- **Estimation half** $\mathcal{S}_{\text{est}}$: used *only* to compute $\hat{\tau}_\ell = \overline{Y}_{\ell, D=1} - \overline{Y}_{\ell, D=0}$ once the partition is fixed.

Independence between split selection and leaf estimation kills the bias. Wager & Athey (2018, *JASA*) show that for *honest* regression trees on i.i.d. data, with subsampling rate s_n satisfying $s_n / n \to 0$, $s_n \log(n)^p / n \to 0$,

$$\frac{\hat{\tau}(x) - \tau(x)}{\sqrt{V_n(x)}} \xrightarrow{d} \mathcal{N}(0, 1).$$

$V_n(x)$ is computed via the *infinitesimal jackknife* on the bootstrap replicates — analogous to bagging's OOB variance.

Appendix A.4 — The causal forest as a kernel-weighted estimator

Equivalent re-statement (Athey, Tibshirani, Wager 2019, *Annals Stat.*). A random forest defines a *forest-induced weight function*

$$\alpha_i(x) \;=\; \frac{1}{B}\sum_{b=1}^{B} \frac{\mathbf{1}\{X_i \in L_b(x)\}}{|L_b(x)|},$$

where $L_b(x)$ is the leaf containing x in tree b. The forest's prediction at x is the $\alpha_i(x)$-weighted average of the responses — a data-adaptive nearest-neighbour estimator.

For the causal target, replace "regression at x" with "treatment-effect estimate at x":

$$\widehat{\tau}(x) \;=\; \arg\min_{\tau} \sum_{i=1}^{n} \alpha_i(x)\left(Y_i - \widehat{\mu}(X_i) - \tau\cdot(D_i - \widehat{e}(X_i))\right)^2,$$

the *generalised random forest* (GRF) estimator. The forest only supplies the weights $\alpha_i(x)$; the moment condition that defines τ is the standard partial-linear one.

Appendix A.5 — Practical pipeline (`grf`)

1. Estimate nuisance functions: $\widehat{m}(x) = \widehat{E}[Y \mid X = x]$ and $\widehat{e}(x) = \widehat{\Pr}(D = 1 \mid X = x)$, by plain regression / classification forests with cross-fitting.
2. Form *partialled-out* responses and treatments: $\widetilde{Y}_i = Y_i - \widehat{m}(X_i)$, $\widetilde{D}_i = D_i - \widehat{e}(X_i)$.
3. Grow B *honest* causal trees on $(\widetilde{Y}, \widetilde{D}, X)$, scoring splits on heterogeneity in the residualised effect. Average to get $\widehat{\tau}(x)$.
4. Variance via the infinitesimal jackknife. Pointwise z-tests of $\widehat{\tau}(x) = 0$ at any x; ANOVA-style tests of constant treatment effect.

Caveats. Honest splitting halves the effective sample; CIs are pointwise, not uniform; overlap can fail in tails of X and inflate variance.

Appendix B.0 — DML: a worked example

Setting. *Effect of institutions on long-run growth* (Acemoglu, Johnson, Robinson, 2001). The structural equation is $Y = \theta D + g_0(X) + \varepsilon$ with Y = log GDP/capita, D = a measure of institutional quality, and X = *hundreds of historical, geographic, and colonial controls*. Picking X by hand is the empirical bottleneck.

What the trees do. Belloni, Chernozhukov & Hansen (2014, *ReStud*) put trees / forests in the *first stage*: separately estimate $\hat{m}(x) = \hat{E}[Y \mid X]$ and $\hat{g}(x) = \hat{E}[D \mid X]$ from the same data, with K-fold cross-fitting. Then form residuals $\tilde{Y}_i = Y_i - \hat{m}(X_i)$, $\tilde{D}_i = D_i - \hat{g}(X_i)$ and regress $\tilde{Y}$ on $\tilde{D}$:

$$\hat{\theta}_{\text{DML}} = \left(\tilde{D}^\top \tilde{D}\right)^{-1} \tilde{D}^\top \tilde{Y}, \qquad \sqrt{n}\left(\hat{\theta}_{\text{DML}} - \theta\right) \xrightarrow{d} \mathcal{N}(0, V).$$

What forests buy. They absorb the high-dimensional non-linear dependence of Y and D on X *without* forcing the analyst to specify a parametric control function, while the partial-linear coefficient $\hat{\theta}$ retains $\sqrt{n}$ inference.

Finding. The DML coefficient on institutions is positive, significant, and similar in magnitude to the AJR IV estimate, but no longer relies on the analyst choosing controls.

`DoubleML` (R, Python). Theory in B.1–B.5 below.

Back

Appendix B.1 — The partially-linear model

Target. The structural equation is

$$Y_i \;=\; \theta_0\, D_i \;+\; g_0(X_i) \;+\; U_i, \qquad \mathbb{E}[U_i \mid D_i, X_i] = 0,$$

with treatment / regressor of interest D_i, high-dimensional controls X_i, and unknown nuisance function g_0. We want $\widehat{\theta}$ for θ_0 that is $\sqrt{n}$-consistent and asymptotically normal, even if g_0 is non-parametric.

The problem with naive plug-in. Define $m_0(x) := \mathbb{E}[Y \mid X = x]$ and $e_0(x) := \mathbb{E}[D \mid X = x]$, so that $Y_i - m_0(X_i) = \theta_0(D_i - e_0(X_i)) + U_i$. If we estimate $\widehat{m}, \widehat{e}$ with ML and run OLS on residuals, the regularisation bias of $\widehat{m}, \widehat{e}$ *leaks* into $\widehat{\theta}$ at rate n^{-r} with $r < 1/2$. The estimator is no longer $\sqrt{n}$-consistent.

Appendix B.2 — Neyman orthogonality

Define the *partial-linear orthogonal score*

$$\psi(W;\theta,\eta) \;=\; \big(Y - m(X) - \theta\,(D - e(X))\big)\big(D - e(X)\big), \qquad \eta = (m,e), \quad W = (Y,D,X).$$

θ_0 is the solution to $\mathbb{E}[\psi(W;\theta_0,\eta_0)] = 0$. The score ψ is Neyman-orthogonal:

$$\frac{\partial}{\partial t}\,\mathbb{E}\big[\psi(W;\theta_0,\eta_0 + t\cdot h)\big]\Big|_{t=0} \;=\; 0,$$

for every direction h in the nuisance space. Read it. A small perturbation of η around the truth has *zero first-order effect* on the moment that defines θ_0. Estimation error in $\widehat{\eta}$ enters $\widehat{\theta}$ only through second-order terms.

Appendix B.3 — The DML estimator and its rate condition

Cross-fitting. Split the data into K folds. For each fold k, train $\widehat{\eta}^{(-k)} = (\widehat{m}^{(-k)}, \widehat{e}^{(-k)})$ on the *other* $K-1$ folds (e.g. random forests, GBM, NN). Plug the held-out predictions into the score:

$$\widehat{\theta}_{\mathrm{DML}} \;=\; \frac{\sum_i (D_i - \widehat{e}^{(-k(i))}(X_i))\,(Y_i - \widehat{m}^{(-k(i))}(X_i))}{\sum_i (D_i - \widehat{e}^{(-k(i))}(X_i))^2}.$$

This is exactly OLS of $\widetilde{Y}$ on $\widetilde{D}$ on residualised data.

Theorem (Chernozhukov et al. 2018). Under regularity conditions plus the *rate condition*

$$\|\widehat{m} - m_0\|_{L^2} \cdot \|\widehat{e} - e_0\|_{L^2} \;=\; o_p\!\left(n^{-1/2}\right),$$

e.g. both nuisances converge at $n^{-1/4}$, we have

$$\sqrt{n}\left(\widehat{\theta}_{\mathrm{DML}} - \theta_0\right) \xrightarrow{d} \mathcal{N}(0,\, \Sigma), \qquad \Sigma = \mathbb{E}\Big[(D - e_0(X))^2\Big]^{-1} \mathbb{E}\Big[\psi^2\Big] \mathbb{E}\Big[(D - e_0(X))^2\Big]^{-1}.$$

Appendix B.4 — Why $n^{-1/4}$? Where the bias terms cancel

Expand the empirical score around θ_0 and the truth η_0:

$$\widehat{\theta} - \theta_0 = \underbrace{-\mathbb{E}_n[\psi(W;\theta_0,\eta_0)]/\mathbb{E}_n[\partial_\theta \psi]}_{\text{leading term, } O_p(n^{-1/2})} + \underbrace{\text{first-order in } (\widehat{\eta}-\eta_0)}_{=0 \text{ by Neyman orthogonality}} + \underbrace{\text{second-order in } (\widehat{\eta}-\eta_0)}_{O_p(\|\widehat{m}-m_0\|\cdot\|\widehat{e}-e_0\|)}.$$

For the second-order remainder to be $o_p(n^{-1/2})$, the *product* of the two nuisance errors must vanish faster than $n^{-1/2}$. Each individual error converging at $n^{-1/4}$ is enough — which is what trees, forests, GBM, and modest NNs achieve under standard smoothness or sparsity assumptions.

Practical reading. You can use almost any flexible learner for $\widehat{m}$ and $\widehat{e}$, as long as it is consistent at a polynomial rate. The orthogonal score does the algebraic heavy lifting that makes $\widehat{\theta}$ behave like classical OLS.

Appendix B.5 — Beyond partially-linear: the recipe

The same recipe applies to many causal estimands. Pick a moment that is Neyman-orthogonal at the truth, plug in cross-fit ML nuisances, average:

- **ATE under unconfoundedness** (binary D): $\psi = \left(\frac{D(Y-m_1(X))}{e(X)} - \frac{(1-D)(Y-m_0(X))}{1-e(X)}\right) + m_1(X) - m_0(X) - \theta$. The doubly-robust score.
- **LATE / IV with high-dim controls**: orthogonal score in (Y, D, Z, X), nuisances $m(x), \pi(x) = \Pr(Z = 1 \mid X)$.
- **Heterogeneous treatment effects**: combine DML residualisation with a causal forest second stage (Athey, Tibshirani, Wager 2019).

Implementations: `DoubleML` (R, Python), `econml`.

Take-away. DML is a *two-step* story. First step: any flexible learner that can absorb g_0 at $n^{-1/4}$. Second step: a small, low-dimensional regression on residualised data with classical inference. Trees and forests dominate the first step in practice because they are robust, fast, and require almost no tuning.

C.1 — Convexity and smoothness, the two assumptions

Throughout, $R : \mathbb{R}^P \to \mathbb{R}$ is the empirical risk and we want to minimise it.

- R is convex if $R(\alpha\theta_1 + (1-\alpha)\theta_2) \le \alpha R(\theta_1) + (1-\alpha)R(\theta_2)$ for all θ_1, θ_2, $\alpha \in [0,1]$. Equivalently $\nabla^2 R \succeq 0$. *Implication:* every local min is global.
- R is L-smooth if its gradient is L-Lipschitz: $\|\nabla R(\theta) - \nabla R(\theta')\| \le L\|\theta - \theta'\|$. Equivalently $\nabla^2 R \preceq L\, I$. *Implication:* a quadratic upper bound,

$$R(\theta') \;\le\; R(\theta) + \nabla R(\theta)^\top (\theta' - \theta) + \frac{L}{2}\|\theta' - \theta\|^2.$$

- R is μ-strongly convex if $\nabla^2 R \succeq \mu\, I > 0$. Equivalently a quadratic *lower* bound.

Linear regression with ℓ_2 loss is μ-strongly convex with L-smooth gradient: classical OLS lives in this regime. Neural networks are smooth (with very large L) but *not* convex.

Back

C.2 — The descent lemma (the workhorse identity)

Take an L-smooth R. Apply the quadratic upper bound at $\theta' = \theta - \rho \nabla R(\theta)$:

$$R(\theta - \rho \nabla R(\theta)) \;\le\; R(\theta) - \rho \|\nabla R(\theta)\|^2 + \frac{L\rho^2}{2} \|\nabla R(\theta)\|^2.$$

Choosing $\rho = 1/L$ gives the descent lemma:

$$R(\theta^{(t+1)}) \;\le\; R(\theta^{(t)}) - \frac{1}{2L} \|\nabla R(\theta^{(t)})\|^2.$$

Each GD step *strictly decreases* R unless the gradient is zero. Summing and using $R^* = \inf_\theta R$:

$$\sum_{t=0}^{T-1} \|\nabla R(\theta^{(t)})\|^2 \;\le\; 2L\big(R(\theta^{(0)}) - R^*\big),$$

hence $\min_{t<T} \|\nabla R(\theta^{(t)})\|^2 = O(1/T)$. Reading. Even *without convexity*, GD with $\rho \le 1/L$ drives the gradient to zero at rate $1/T$ — it finds stationary points.

C.3 — Convergence rates of gradient descent

Add convexity and the rates sharpen.

- **Convex + L-smooth.** With $\rho \le 1/L$, $R(\theta^{(T)}) - R^* \le \frac{L\|\theta^{(0)} - \theta^*\|^2}{2T}$. Sublinear $O(1/T)$ rate. (Nesterov's accelerated GD improves this to $O(1/T^2)$ with the same per-step cost.)
- **μ-strongly convex + L-smooth.** With $\rho = 1/L$, $\|\theta^{(t+1)} - \theta^*\|^2 \le \left(1 - \mu/L\right)\|\theta^{(t)} - \theta^*\|^2$. Linear (geometric) rate; condition number $\kappa = L/\mu$ controls speed.
- **Non-convex + L-smooth.** Only $\min_{t<T} \|\nabla R(\theta^{(t)})\| = O(1/\sqrt{T})$. We are guaranteed to find a stationary point but not a global minimum.

Reading for NNs. Neural-network risk is the third regime: smoothness gives finite-step descent, but no global guarantee. The empirical observation that SGD finds usable minima despite non-convexity is the main mystery of deep learning theory.

C.4 — SGD: convergence in expectation

Replace ∇R with an unbiased estimate $\widehat{g}$ satisfying $\mathbb{E}[\widehat{g}] = \nabla R$ and $\mathbb{E}\|\widehat{g}\|^2 \le G^2$. Take a non-summable but square-summable step-size schedule ρ_t (e.g. $\rho_t = \rho_0/\sqrt{t}$).

- **Convex + L-smooth.** Robbins–Monro: with $\sum \rho_t = \infty$, $\sum \rho_t^2 < \infty$,

$$\mathbb{E}\Big[R(\bar{\theta}^{(T)})\Big] - R^* \;=\; O\big(1/\sqrt{T}\big),$$

  where $\bar{\theta}^{(T)}$ is the running average of iterates.
- **Strongly convex + L-smooth.** With $\rho_t = c/(t+t_0)$, $\mathbb{E}\,\|\theta^{(t)} - \theta^*\|^2 = O(1/t)$.
- **Constant step size.** $\theta^{(t)}$ converges to a *neighbourhood* of θ^* whose radius scales with $\rho \cdot G^2$. Smaller $\rho \Rightarrow$ tighter neighbourhood, slower convergence.

Reading. SGD trades a smaller computational cost per step for a slower theoretical rate. In practice, the trade-off is overwhelmingly worth it for n in the millions.

C.5 — Momentum and adaptive methods

- **Polyak momentum (heavy ball).**

$$v^{(t+1)} = \mu\, v^{(t)} - \rho\, \nabla R(\theta^{(t)}), \qquad \theta^{(t+1)} = \theta^{(t)} + v^{(t+1)}.$$

  Acts like a low-pass filter on the gradient; provably accelerates GD on quadratics.

- **Nesterov.** Evaluate the gradient at the *look-ahead* point $\theta^{(t)} + \mu v^{(t)}$. $O(1/T^2)$ rate for convex smooth problems — optimal among first-order methods.

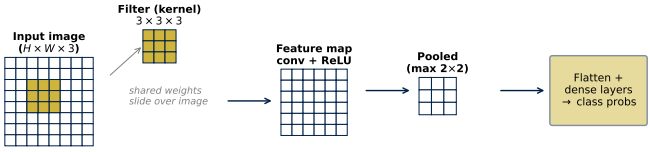
- **AdaGrad.** Per-coordinate step size: $\rho_{t,j} = \rho / \sqrt{\sum_{s \le t} (\partial_j R^{(s)})^2}$. Coordinates with small historical gradients get larger steps. Useful for sparse features.

- **Adam / AdamW.** Combine momentum (first moment) and AdaGrad (second moment), with bias correction and (in AdamW) decoupled weight decay:

$$m^{(t)} = \beta_1 m^{(t-1)} + (1-\beta_1)\widehat{g}^{(t)}, \quad v^{(t)} = \beta_2 v^{(t-1)} + (1-\beta_2)\widehat{g}^{(t)2}, \quad \theta^{(t+1)} = \theta^{(t)} - \rho\, \widehat{m}/(\sqrt{\widehat{v}} + \varepsilon).$$

  Default optimiser for transformers. Robust to hyperparameter choices, bad on some convex tasks but excellent in deep learning practice.

D.1 — The convolution operation



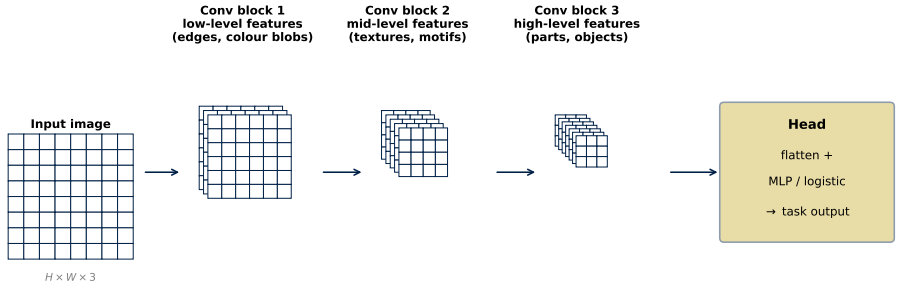
Two key inductive biases: (i) locality -- each filter sees only a small patch; (ii) weight sharing -- the same filter is applied at every spatial position.

Stack many such conv→pool blocks: edges → textures → parts → objects.

Conv layer: $X \in \mathbb{R}^{H\times W\times C_{\text{in}}} \to Y \in \mathbb{R}^{H'\times W'\times C_{\text{out}}}$. Each output channel c has a filter $K_c \in \mathbb{R}^{k\times k\times C_{\text{in}}}$: $Y_{i,j,c} = \sum_{u,v,c'} K_c[u,v,c'] \cdot X[i+u, j+v, c'] + b_c$, followed by ReLU. Locality (filter sees $k \times k$ only), weight sharing (same K_c everywhere). Conv has $k\,k\,C_{\text{in}}\,C_{\text{out}}$ parameters — independent of H, W.

Back

D.2 — Building a feature hierarchy



Same convolutional backbone learns transferable features; swap the head for the task (classification, regression, segmentation, ...).

Stack many conv-and-pool blocks. Spatial resolution shrinks; channel dimension grows. Block 1 sees pixels and learns edges / colour blobs; block 2 combines those into corners and textures; block 3 into object parts (eyes, wheels); deeper blocks into whole objects. Confirmed empirically by filter visualisations (Zeiler & Fergus 2014): layer 1 ≈ Gabor filters, layer 5 ≈ object templates.

D.3 — The features are reusable; the head is task-specific

The same convolutional backbone produces a set of features $\phi(X) \in \mathbb{R}^{H'' \times W'' \times C_{\text{out}}}$. To solve a task, attach a *head* on top:

- **Image classification.** Flatten $\phi(X)$, pass through a small MLP, softmax to K classes. Train with cross-entropy.
- **Regression.** Flatten $\phi(X)$, pass through an MLP with a scalar output. Train with squared error.
- **Object detection.** On each spatial location of $\phi(X)$, predict a bounding box and class. Train with a multi-task loss.
- **Segmentation.** Replace flatten with an upsampling decoder; predict a class *per pixel*.

Transfer learning. Train the backbone once on a huge source task (e.g. ImageNet, 1.2M labelled images). Freeze its weights. On a new task with much less data, replace only the head and fine-tune. Standard recipe in applied econ work that uses satellite imagery, scanned documents, or product photos: never train a CNN from scratch on a small dataset — start from a pretrained backbone and swap the head.

E.1 — Empirical scaling laws

Train a family of models of varying parameter count N on varying training-token counts D. For each (N, D), plot the converged test loss L. Across many architectures (Kaplan et al. 2020, OpenAI; Hoffmann et al. 2022, Chinchilla), the same functional form fits:

$$L(N, D) \approx L_\infty + \left(\frac{A}{N}\right)^\alpha + \left(\frac{B}{D}\right)^\beta,$$

with exponents $\alpha, \beta \approx 0.3$. Bigger model + more data $\Rightarrow$ predictably lower loss across many orders of magnitude.

Compute-optimal scaling (Chinchilla). For a fixed compute budget $C \propto N \cdot D$, minimise $L(N, D)$ subject to that constraint. The optimum gives $N^\star \propto C^{1/2}, \quad D^\star \propto C^{1/2}$. Implication: *train models that are about one token per parameter*, not the much-larger N/much-smaller D that earlier (GPT-3-era) practice favoured. Chinchilla showed that GPT-3 was substantially under-trained.

Back

E.2 — What scaling laws are not

- **Not derived from theory.** The form $L(N, D) = L_\infty + (A/N)^\alpha + (B/D)^\beta$ is fit to data. There is no first-principles derivation of $\alpha, \beta \approx 0.3$. The robustness of the fit across architectures is an *empirical regularity*, not a theorem.
- **Loss is not capability.** The scaling law tracks cross-entropy on held-out tokens. Downstream task performance (accuracy on benchmarks, helpfulness, reasoning) is a discontinuous function of L — many tasks "emerge" once L falls below a threshold.
- **Holds in distribution.** The fit is for the same train and test distributions. Out-of-distribution performance does not obey the same curve.
- **Compute is the binding constraint.** The curve says "loss falls predictably with N and D". In practice both are bounded by hardware. A 10× larger model needs 10× more compute and roughly 10× more data — it is the compute axis that ultimately limits frontier model size.