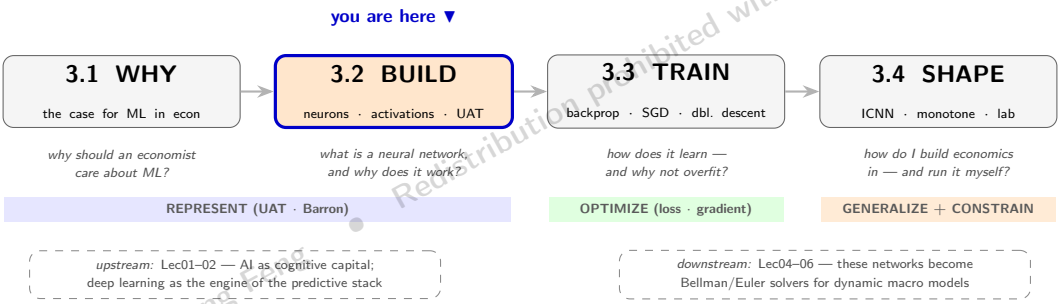
AI for Economic Research: Dynamic Models, Language, and Agents
Lecture 3.2: Neural Networks — Architecture and Activations

Zhigang Feng

2026-07-07 10:05:15

Lecture 3 in One Map

One lecture, four decks, one goal: make neural networks a working tool for economists — why they matter, what they are, how they learn, and how to make them respect economics.



Act 3.2 (here) *opens the box: neurons, layers, and activations assemble into f_θ, and the approximation theorems (Cybenko–Hornik UAT; Barron's dimension-free rate) explain why this class fits the functions macro needs.*

Neural Network Architecture

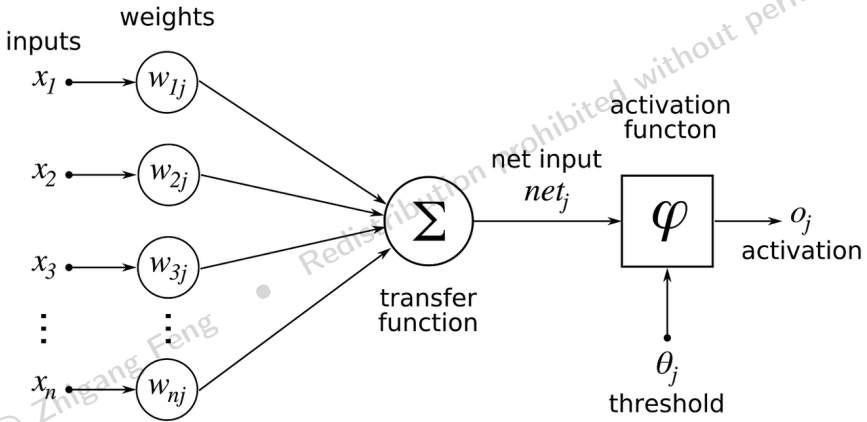
Neurons, layers, depth — the anatomy of f_θ

Neural Networks: The Basic Building Block

- Artificial neural networks are:
  - Inspired by biological neural networks
  - Composed of interconnected neurons organized in layers
  - Trained by adjusting weights based on data

- **Key components:**
  - **Neuron:** Receives inputs, applies an activation function, produces output
  - **Activation function:** Nonlinear map applied by each neuron (sigmoid, tanh, ReLU)
  - **Loss function:** Measures how well predictions match true values — the objective to minimize

Single Neuron: Visual



Single Hidden Layer Network

- Input $x \in \mathbb{R}^M$, output $y \in \mathbb{R}^N$.
- The j-th hidden unit:

$$h_j^{(1)} = f_j^{(1)}(x) = \sigma\left(b_j^{(1)} + \sum_{m=1}^{M} w_{j,m}^{(1)} x_m\right), \quad j = 1, \ldots, U_1 \tag{1}$$

- The i-th output:

$$y_i = \sum_{j=1}^{U_1} \tilde{w}_{i,j}\, h_j^{(1)} + \tilde{b}_i, \quad i = 1, \ldots, N \tag{2}$$

- σ is a nonlinear activation function.
- Common choices: $\sigma(x) = \tanh(x)$, $\sigma(x) = \max(0, x)$ (ReLU), $\sigma(x) = \frac{1}{1+e^{-x}}$ (sigmoid).

Two Hidden Layer Network

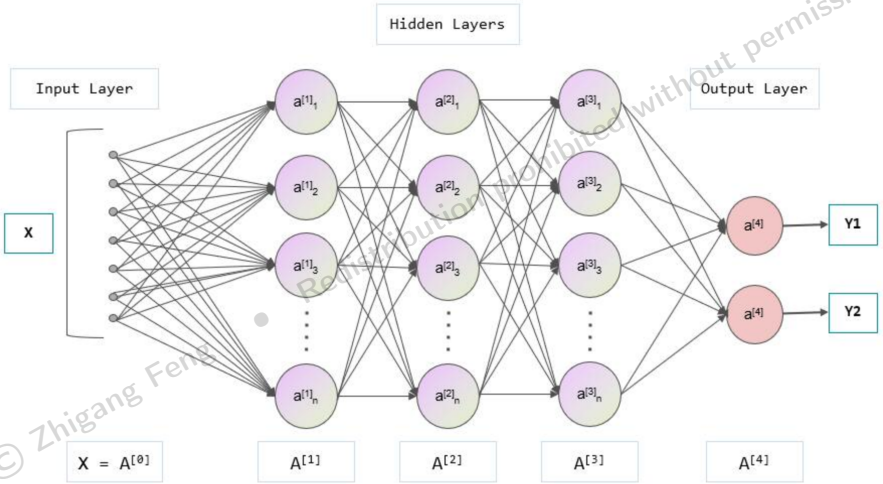
- Take the linear combination of $h^{(1)}$ and pass through another activation:

$$h_j^{(2)} = \sigma\left(b_j^{(2)} + \sum_{k=1}^{U_1} w_{j,k}^{(2)} f_k^{(1)}(x)\right), \quad j = 1, \ldots, U_2 \tag{3}$$

$$y_i = \sum_{j=1}^{U_2} \tilde{w}_{i,j} h_j^{(2)} + \tilde{b}_i, \quad i = 1, \ldots, N \tag{4}$$

- **Pattern:** Each layer performs a linear transformation followed by a nonlinear activation. Depth adds representational power.
- **Macro analogy:** Successive layers are like successive time steps in a dynamic program — each transforms the "state" into a richer representation.

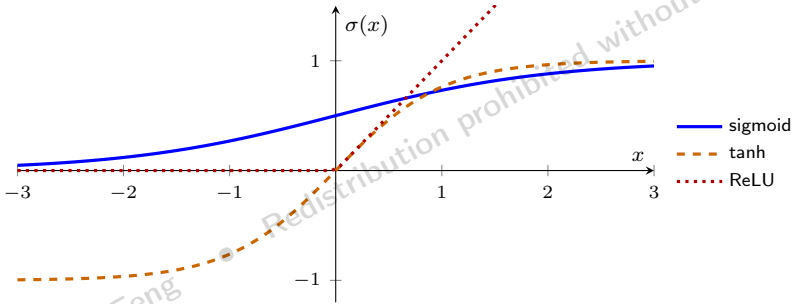
Hidden Layers: Visual



Activation Functions and Universal Approximation

Nonlinearity is the point: sigmoid, tanh, ReLU — and why two layers approximate anything

Activation Functions: Visual



Sigmoid & tanh saturate (⇒ vanishing gradients); ReLU is piecewise linear (⇒ sparse activation, fast gradient flow).

Activation Functions: What They Do

- **Sigmoid:** $\sigma(x) = \frac{1}{1+e^{-x}}$ — outputs in $(0, 1)$, useful for probabilities.
- **Tanh:** $\tanh(x)$ — outputs in $(-1, 1)$, zero-centered, smoother than ReLU.
- **ReLU:** $\max(0, x)$ — simple, sparse activation, dominant in modern networks.
- **Why nonlinearity matters:** without activation functions, stacked layers collapse into one linear map — depth gains us nothing without nonlinearity.
- **For economists:** the activation is the network's analogue of choosing a flexible functional form (translog, CES, semi-parametric) — but the network composes it across layers.

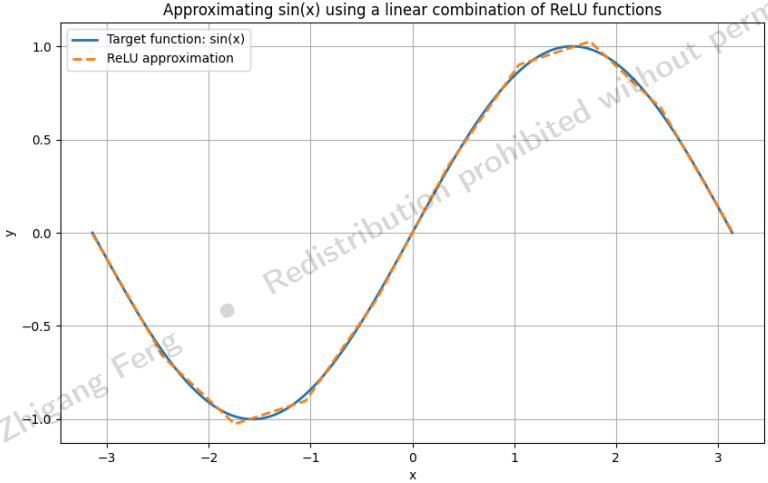
Function Approximation with ReLU

- **Goal:** Approximate arbitrary nonlinear functions using simple building blocks.
- **Key Idea:** Linear combination of shifted ReLU functions:

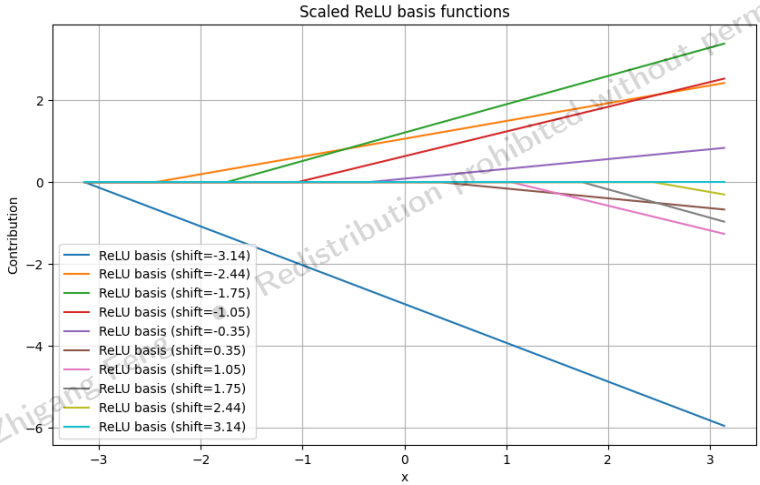
$$\hat{y}(x) = a_0 + \sum_{i=1}^{N} a_i \,\mathrm{ReLU}(x - b_i)$$

- **Shifts b_i:** Define the "kink" points where marginal slope changes.
- **Coefficients a_i:** Scale each ReLU's contribution to match local slope.
- **Example:** Approximating $\sin(x)$ on $[-\pi, \pi]$:
  - Choose shifts b_i evenly over the domain
  - Form design matrix with columns $\mathrm{ReLU}(x - b_i)$ and a bias
  - Solve a least-squares problem for coefficients a_i
- **Economist's take:** ReLU kinks are piecewise-linear splines — the network *learns* the knot locations, unlike a fixed Chebyshev grid.

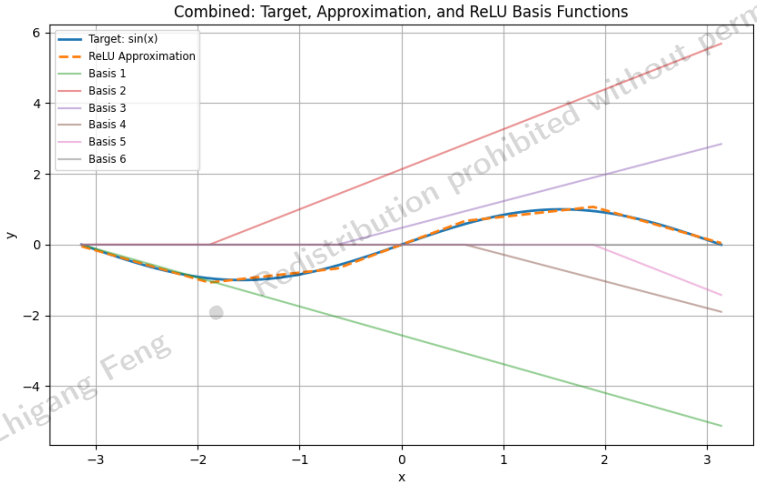
ReLU Approximation: Increasing Complexity



ReLU Approximation: More Kink Points



ReLU Approximation: High Fidelity



Why Neural Networks Work: Universal Approximation

Universal Approximation Theorem (Cybenko 1989; Hornik 1991)

Any continuous function on $[0,1]^d$ can be approximated *uniformly* by a two-layer network — for sigmoidal σ (Cybenko), for any bounded nonconstant σ (Hornik), indeed for *any* non-polynomial σ (Leshno, Lin, Pinkus & Schocken 1993).

Intuition — you just watched the proof. Those ReLU plots *were* the construction: a weighted sum of shifted kinks $\sum_i a_i \,\mathrm{ReLU}(x - b_i)$ bends to fit any curve, and a sigmoid network stacks soft "bumps" the same way. **One hidden unit = one building block; add units ⇒ shrink the error.** Nonlinearity is the glue — drop it and the whole sum collapses back to a single straight line.

Existence, not a recipe. UAT promises a good network *exists* — not that training will *find* it, nor that "enough" units is a *small* number. Approximation ≠ optimization ≠ generalization; T3 takes on the finding. *(The same theorem later is why transformer attention approximates — Lec. 7.)*

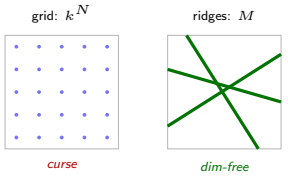
Why Neural Networks Work: Beating the Curse of Dimensionality

Barron (1993): a dimension-free rate

A one-hidden-layer network reaches squared error of order $O(1/M)$ in M units — *independent of the input dimension* N. Fixed series bases (polynomials, splines, Chebyshev) manage only $O(1/M^{2/N})$: to hold accuracy as N grows, M must blow up exponentially.

Why the gap?

- **Grid basis is fixed:** resolution k per axis needs k^N pieces — most parked where f barely moves. That is the curse.
- **A neuron is a ridge** $\sigma(w^\top x + b)$**:** a soft step aimed along a *learned* direction w. The net spends its M units where f varies — so error tracks f's *intrinsic* complexity, not the dimension N.



For economists: this is why deep learning cracks high-dimensional dynamic models — heterogeneous-agent macro, many assets and shocks — exactly where Chebyshev and tensor grids break down (Lec. 4–6; the DeepRamsey case).

DNN = Linear Transformation + Nonlinear Activation

- **Linear Transformation:**
  - Each neuron performs a weighted sum of inputs: projects data onto a new axis
  - The set of transformations across a layer creates a *new representation*
  - Enables: flexible feature representation, information compression, interaction capture
- **Nonlinear Activation:**
  - Allows the network to capture nonlinearity and complex patterns
  - Creates curved decision boundaries in the feature space
  - **Essential** for the universal approximation property
- **Together:** Each layer applies an affine map $Wx + b$ then a pointwise nonlinearity $\sigma(\cdot)$. Stacking layers composes these maps:

$$f(x) = \sigma_L \circ A_L \circ \cdots \circ \sigma_1 \circ A_1(x)$$

where $A_\ell(x) = W_\ell x + b_\ell$.

Bridge: A Good Network Exists — Now We Must Find It

Act 3.2 built the object. A network is a composition of affine maps and pointwise nonlinearities; universal approximation says a good one exists, and Barron says width buys accuracy at a rate the input dimension cannot destroy.

- You have seen *what*: neurons, layers, and activations — and why nonlinearity is essential (drop it and the whole stack collapses to one straight line).
- **Next (3.3):** existence is not a recipe. The finding machinery — forward pass, backpropagation, stochastic gradient descent — plus the regularization discipline that keeps the fit honest.

Arc: 3.1 WHY (the case for ML) $\rightarrow$ **3.2 BUILD** *(architecture + UAT)* $\rightarrow$ *3.3 TRAIN (backprop + SGD)* $\rightarrow$ *3.4 SHAPE (economics-aware nets + lab).*