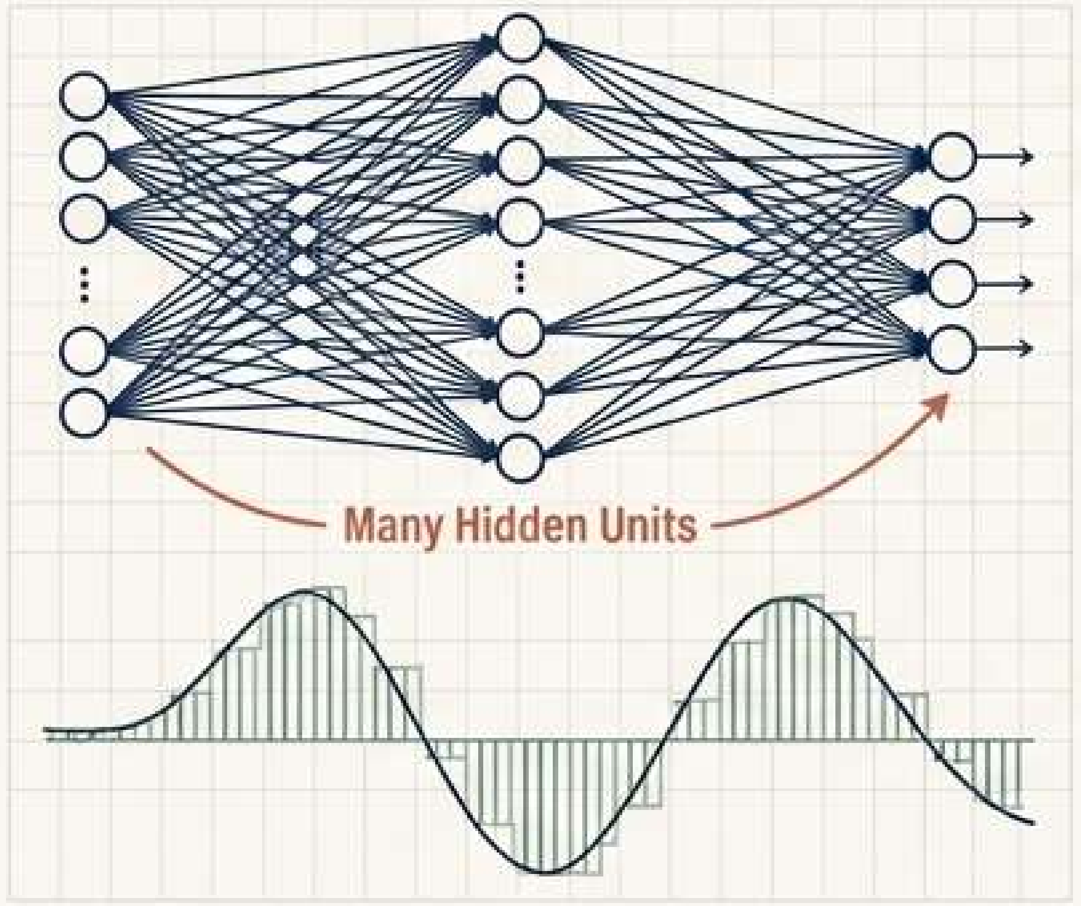
The Paradox of Depth

If shallow networks can do anything, why do we need deep ones?

A single-hidden-layer neural network can, with enough hidden units, approximate any continuous function. This is the Universal Approximation Theorem. So, a fundamental question arises: Why bother with the complexity of adding more layers?

This presentation explores the powerful, and surprisingly intuitive, reason why depth is not just an option, but a fundamental advantage in neural network design.

Shallow Networks (Universal Approximation)

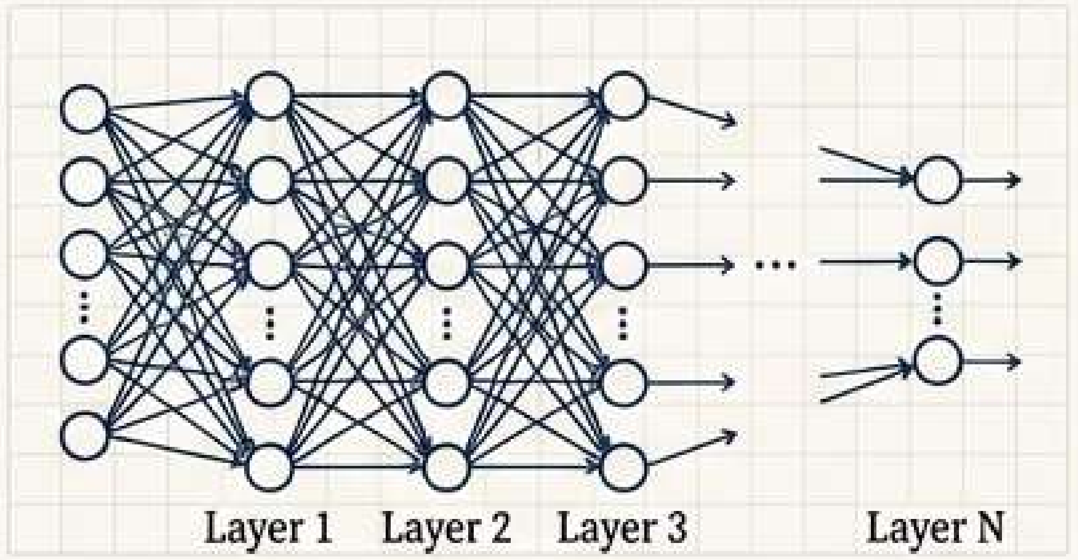


Can approximate any continuous function with enough width.

However, it requires an exponential number of neurons for complex tasks, leading to inefficiency.

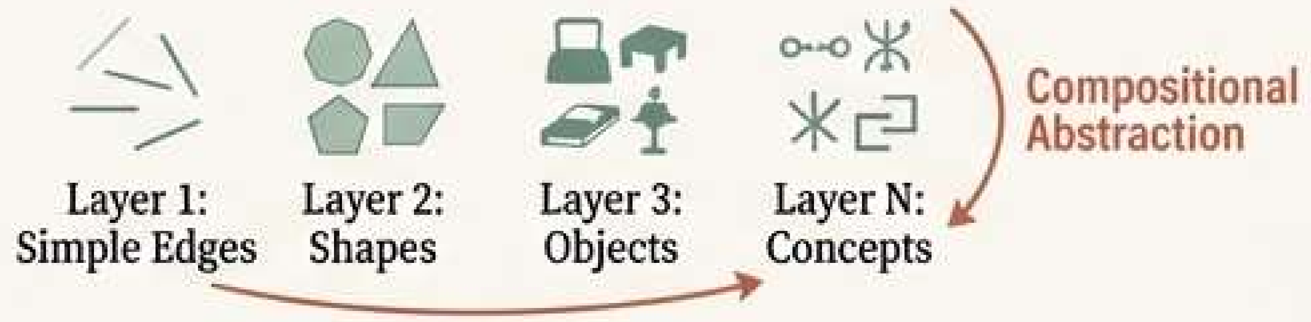
Function Approximation

Deep Networks (Hierarchical Feature Learning)



Build complex functions through compositional hierarchies.

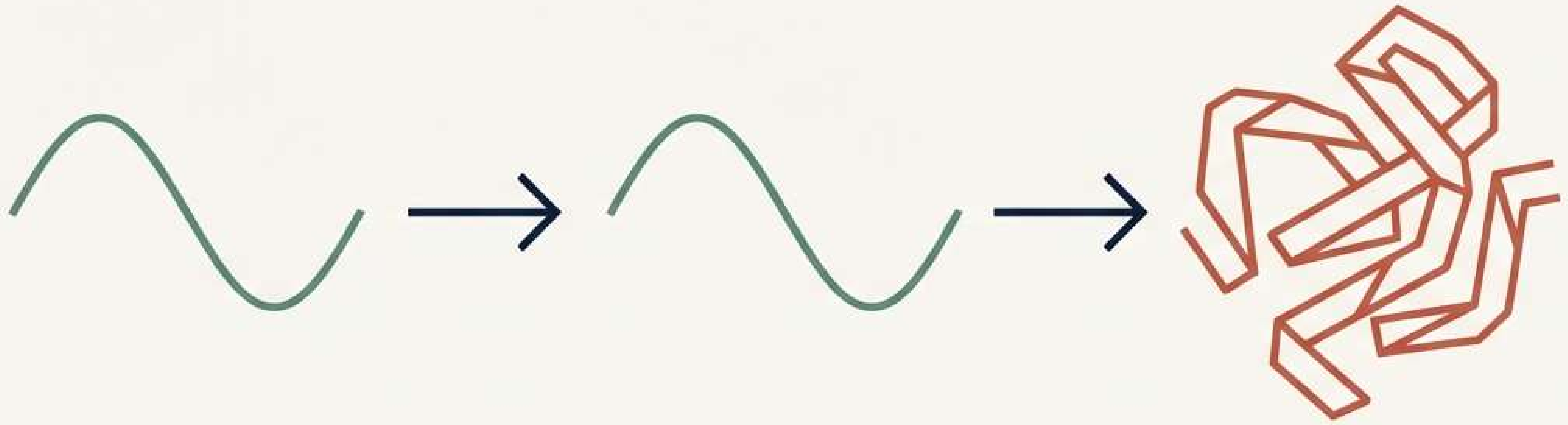
Each layer learns progressively more abstract features, leading to greater parameter efficiency and generalization.



Depth allows for the efficient representation of complex, hierarchical structures inherent in many real-world data, transforming brute force approximation into intelligent, compositional learning.

Key takeaway: Depth enables efficiency and abstraction, not just raw power.

The Insight: Unlocking Complexity Through Composition



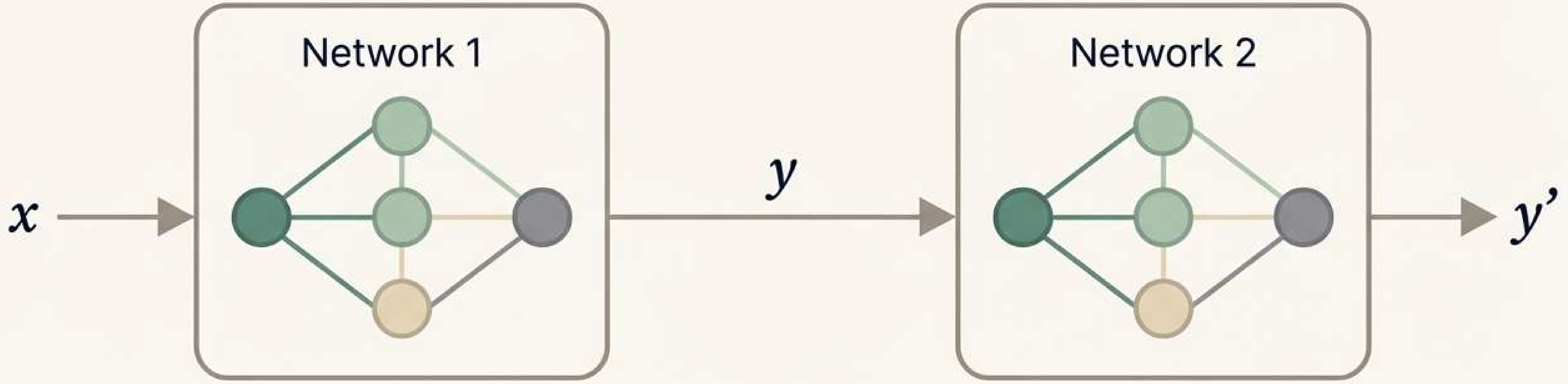
The power of deep networks lies not in a single, complex layer, but in the repeated application of simple transformations. By composing functions—feeding the output of one network into another—we unlock an exponential increase in descriptive power.

We'll explore this through two key intuitions: '**folding**' and '**clipping**'.

A Simple Experiment: What Happens When We Stack Two Networks?

Let's start with two separate, shallow neural networks, each with one input, one output, and three hidden units. With ReLU activations, each network describes a piecewise linear function.

- **Network 1:** Takes an input x and produces an output y.
- **Network 2:** Takes the output y as its input and produces a final output y'.



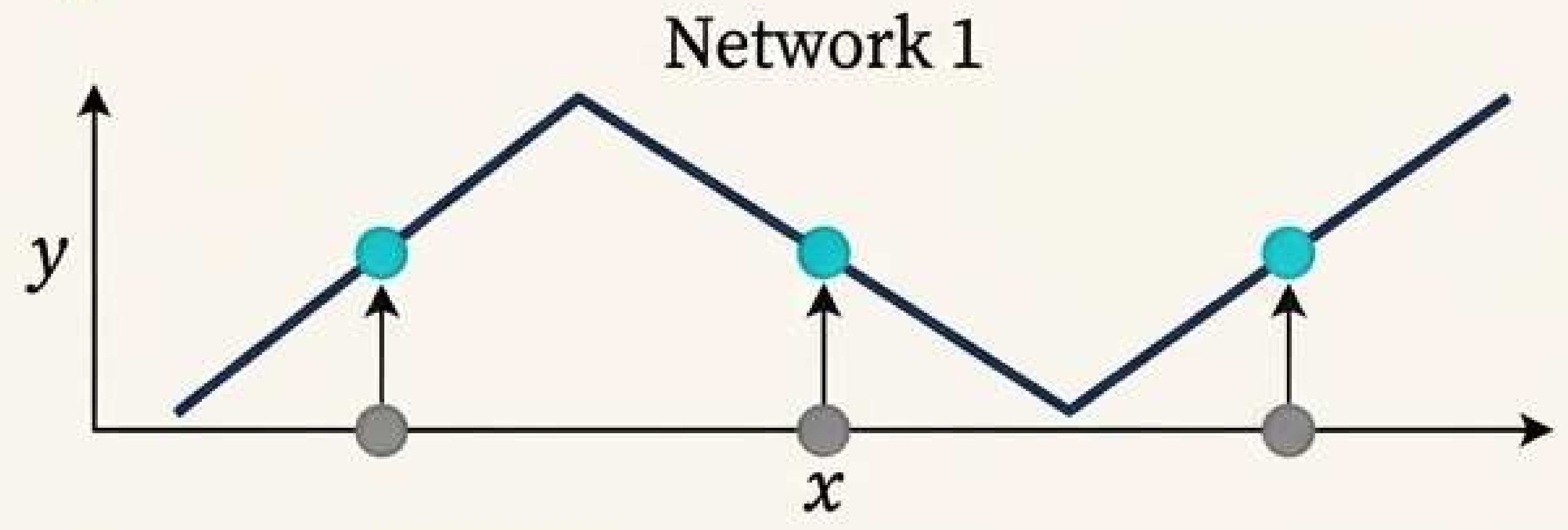
The Multiplicative Effect: From 3 Regions to 9

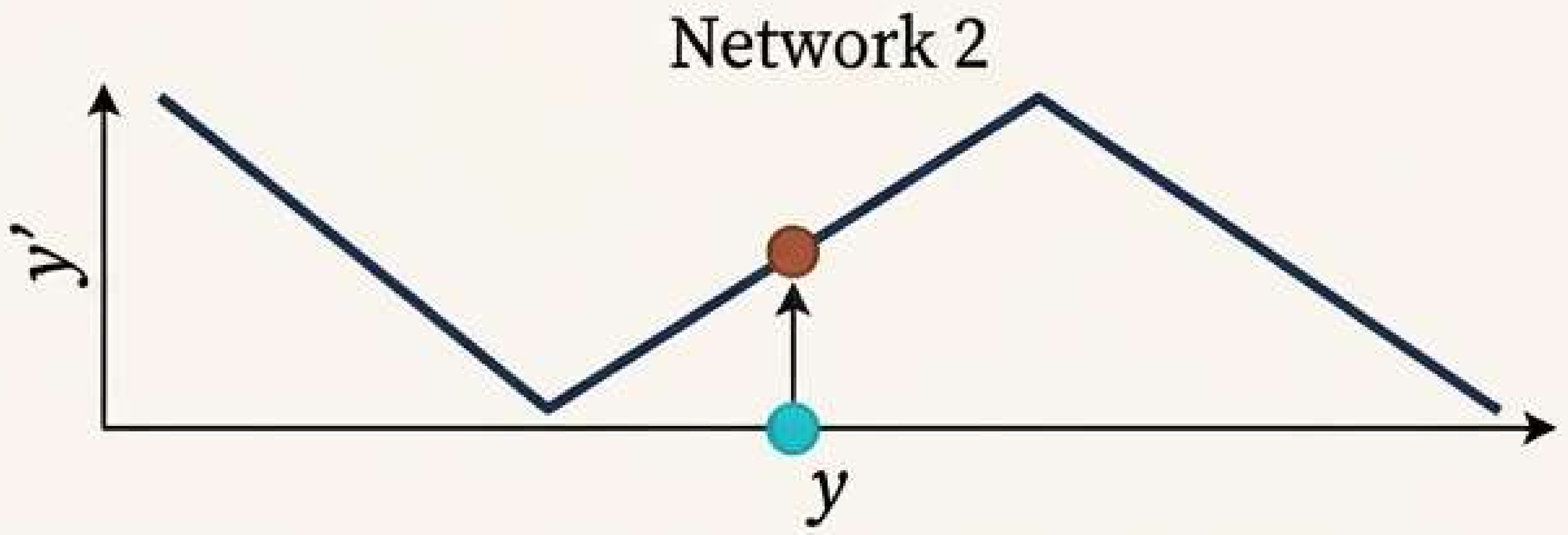
By composing these two simple functions, the number of linear regions multiplies.

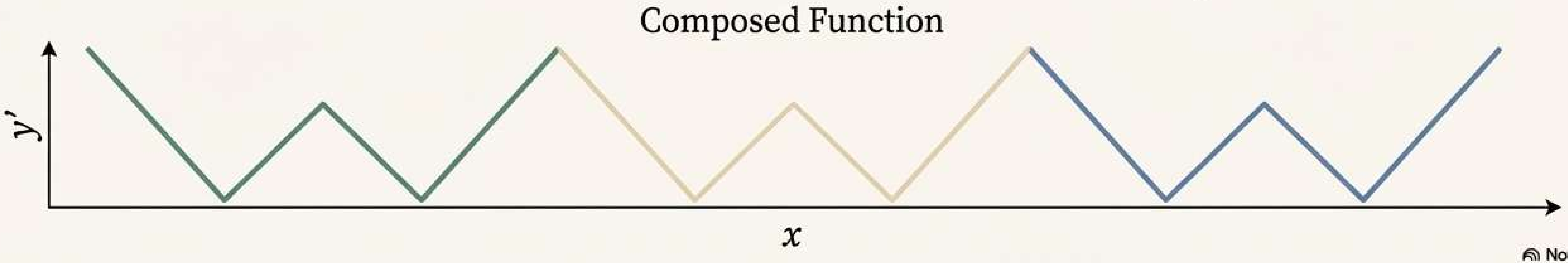
Network 1 (Input *x* to *y*): We configure it to create 3 distinct linear regions. Notice how multiple input ranges of *x* are mapped to the same output range of *y*.

Network 2 (Input *y* to *y'*): This network also defines a function with 3 linear regions.

The Composition (*x* to *y'*): Because Network 1 maps three different input ranges to the same output range, the function from *Network 2* is effectively duplicated three times. The result is **not 3 + 3 = 6 regions, but 3 x 3 = 9 linear regions.**



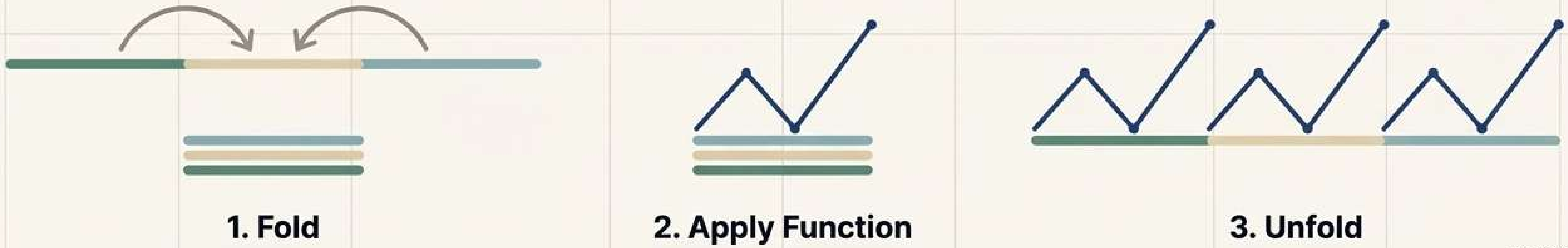




Intuition 1: "Folding" the Input Space

One way to conceptualize the action of the first layer is that it “folds” the input space back onto itself.

1. **Input Space:** Imagine the input x as a simple line.
2. **Fold:** The first network maps multiple parts of this line onto the same segment, like folding a piece of paper.
3. **Apply Function:** The second network then applies its function to this compressed, "folded" space.
4. **Unfold:** When we "unfold" the space to see the final output, the second network’s function has been replicated across all the folded sections.

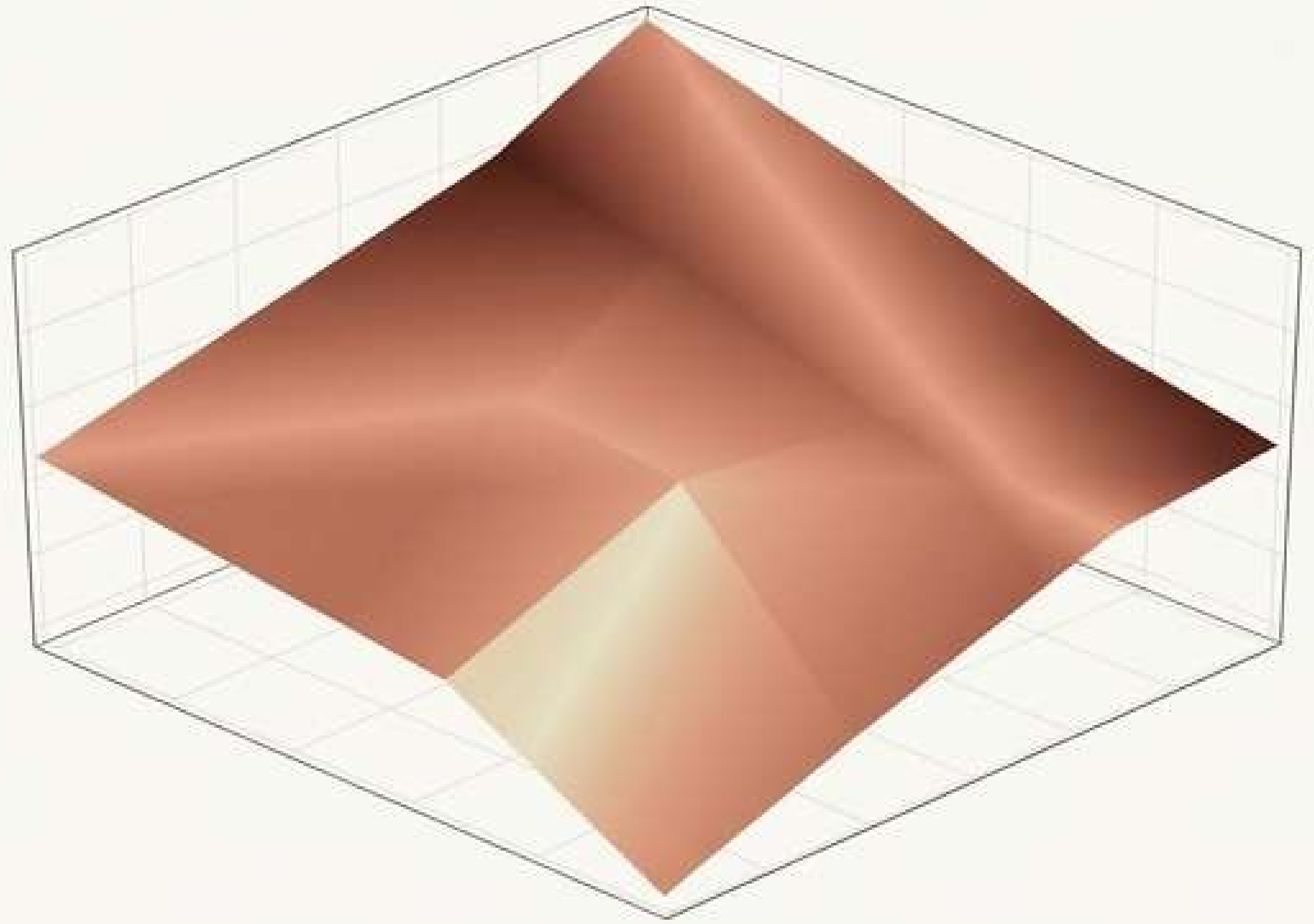


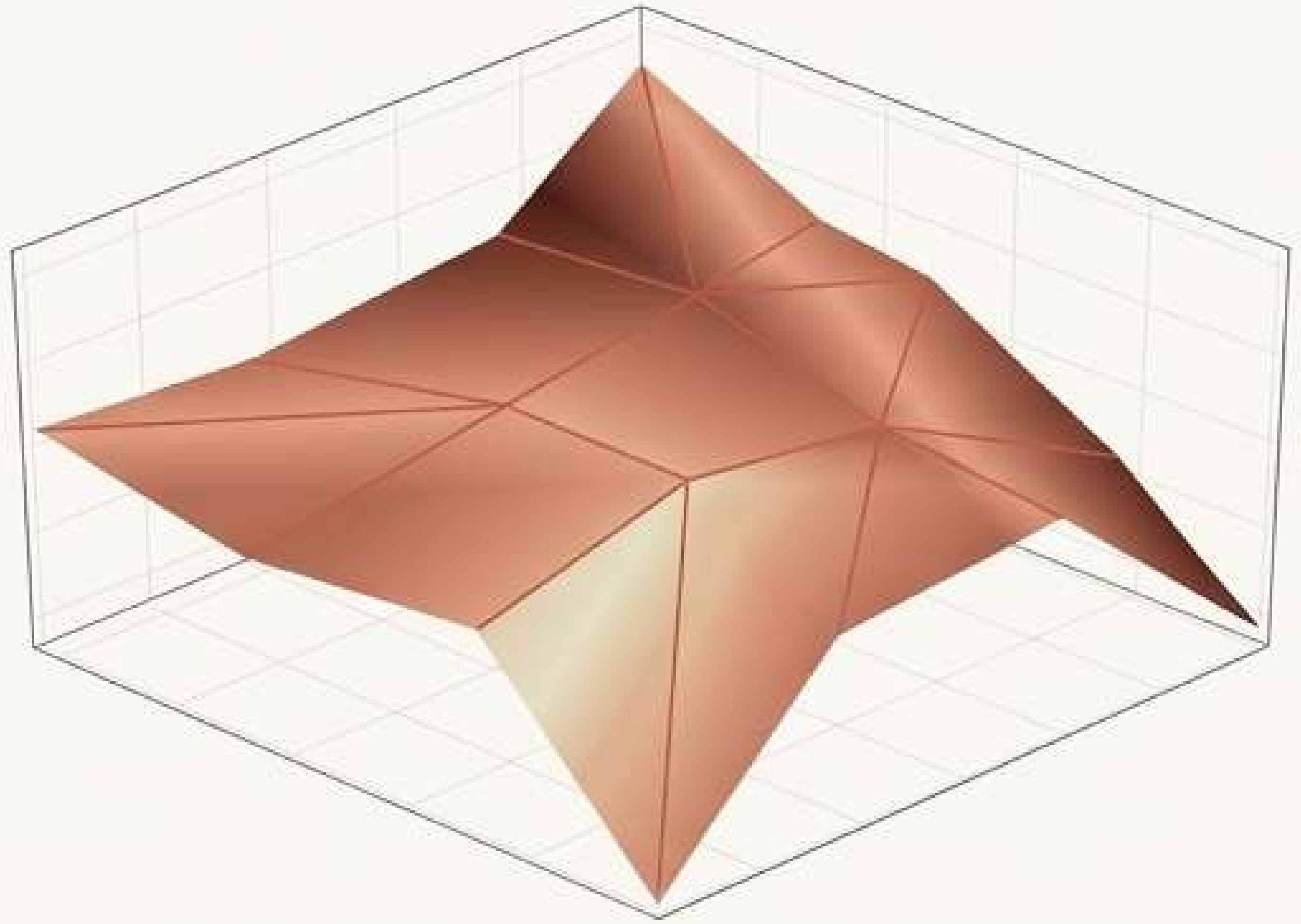
NotebookLM

The Same Principle Applies in Higher Dimensions

This multiplicative effect is not limited to single-dimensional inputs. When the input is 2D (like a plane), the first network can create a function with several flat regions. The second network then divides each of these regions, leading to a dramatic increase in complexity.

- Network 1 (2D input): Produces a function with 7 linear regions.
- Network 2 (1D input from Network 1): Produces a function with 2 linear regions.
- Composition: The result is a function where each of the original 6 non-flat regions is divided, creating a total of 13 linear regions.


Function of Network 1 (7 Regions)

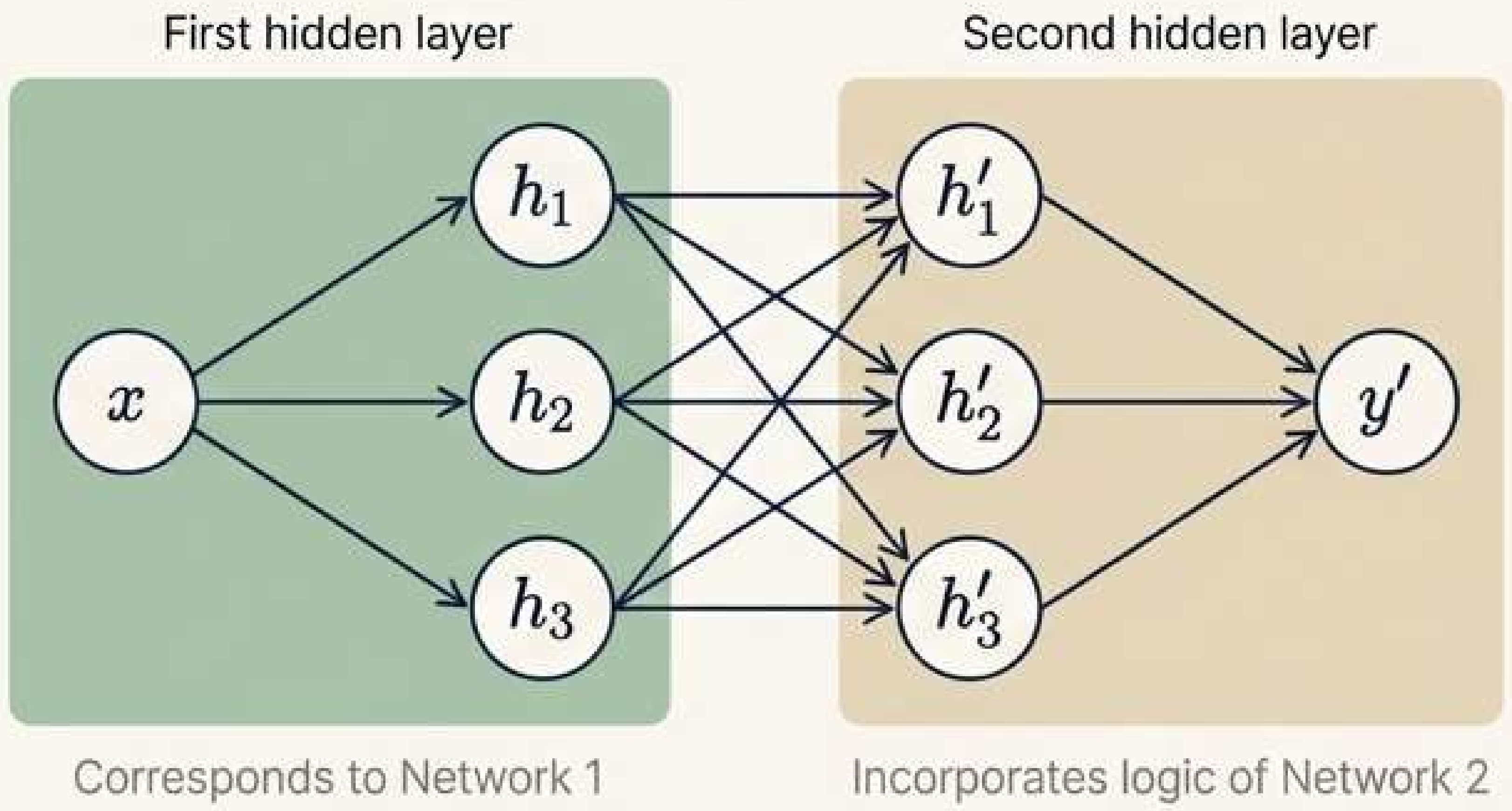

Composed Function (13 Regions)

From Composition to a True Deep Network

The process of composing two shallow networks is mathematically equivalent to a *special case* of a deep network with two hidden layers.

The output of the first network (y) is a linear combination of its hidden units (h).The second network performs a linear operation on y. Applying a linear function to another linear function yields a new linear function.

This means we can rewrite the entire composed system as a single, integrated network with two hidden layers. A general two-layer network is even more powerful, as its parameters are not constrained in the same way as the composed network.



$$h' = a[\theta' y] = a[\theta'(\phi h)]$$

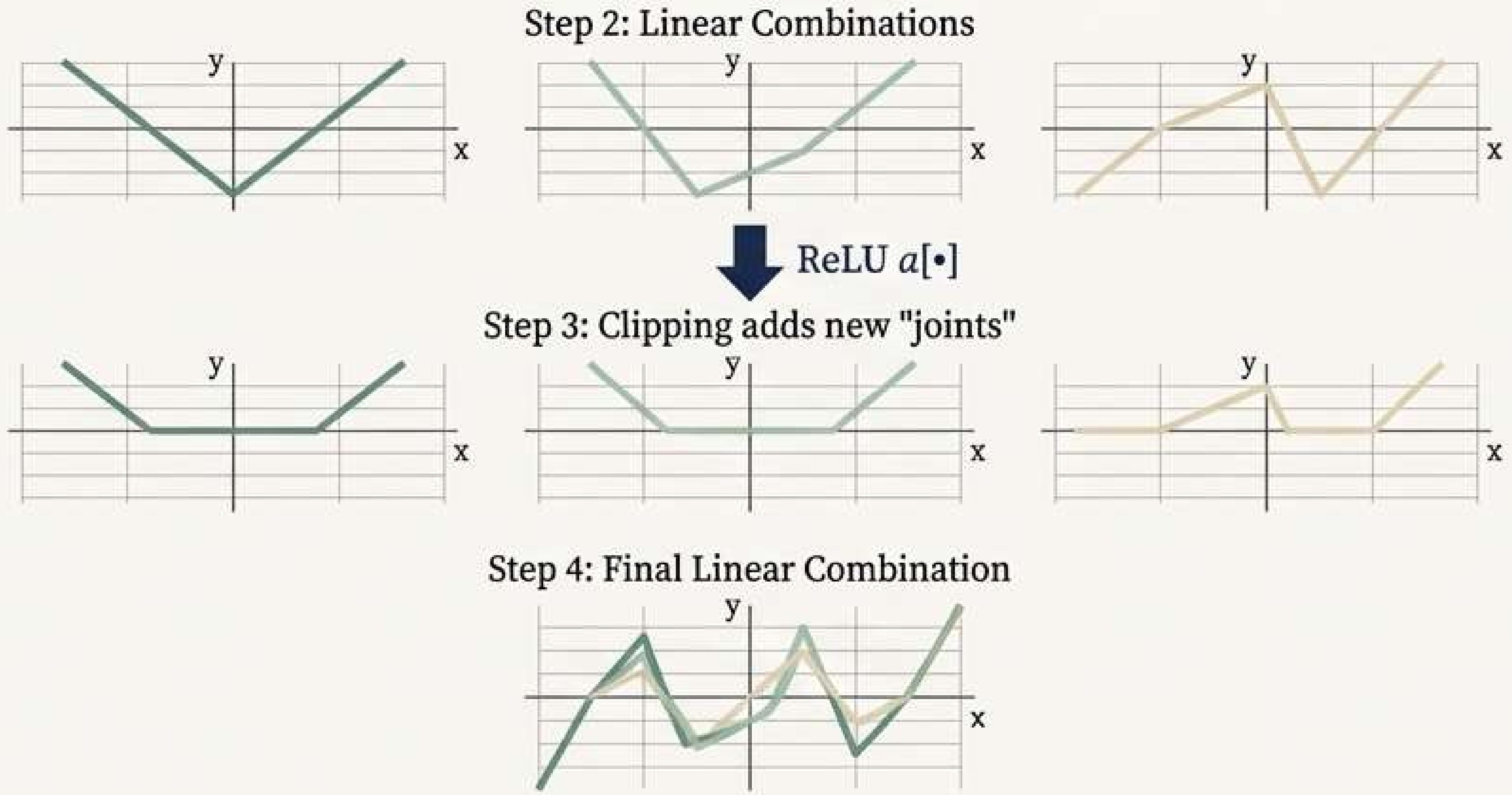
$$\longrightarrow h' = a[\Psi h]$$

NotebookLM

Intuition 2: Building Functions by “Clipping”

Another way to understand deep networks is to see each layer as creating, clipping, and recombining functions.

1. First Layer: Creates a set of standard piecewise linear functions (with "joints" in the same places).
2. Second Layer Pre-Activations: These are new linear combinations of the first layer's outputs, resulting in a new set of piecewise linear functions.
3. Second Layer Activation (Clipping): The ReLU activation function $a[\bullet]$ is applied, "clipping" each function at zero. This crucial step adds new joints and fundamentally alters the function's shape.
4. Final Output: The final output is a linear combination of these newly clipped and jointed functions.


Step 2: Linear Combinations
y
x
ReLU $a[\bullet]$
Step 3: Clipping adds new "joints"
Step 4: Final Linear Combination

The Language of Deep Networks: A Single Equation

Ultimately, a deep network is just a single, though complex, equation relating input x to output y'. All the steps of folding, clipping, and combining can be expressed as one nested function. The equation for our two-layer network is:

$$\begin{aligned} y' = \phi'_0 &+ \phi'_1 a[\psi_{10} + \psi_{11} a[\theta_{10} + \theta_{11} x] + \psi_{12} a[\theta_{20} + \theta_{21} x] + \psi_{13} a[\theta_{30} + \theta_{31} x]] \\ &+ \phi'_2 a[\psi_{20} + \psi_{21} a[\theta_{10} + \theta_{11} x] + \psi_{22} a[\theta_{20} + \theta_{21} x] + \psi_{23} a[\theta_{30} + \theta_{31} x]] \\ &+ \phi'_3 a[\psi_{30} + \psi_{31} a[\theta_{10} + \theta_{11} x] + \psi_{32} a[\theta_{20} + \theta_{21} x] + \psi_{33} a[\theta_{30} + \theta_{31} x]] \end{aligned}$$

While difficult to parse at a glance, its structure reveals the core idea: a composition of linear transformations (θx, $\Psi\mathbf{h}$) and non-linear activations ($a[\bullet]$).

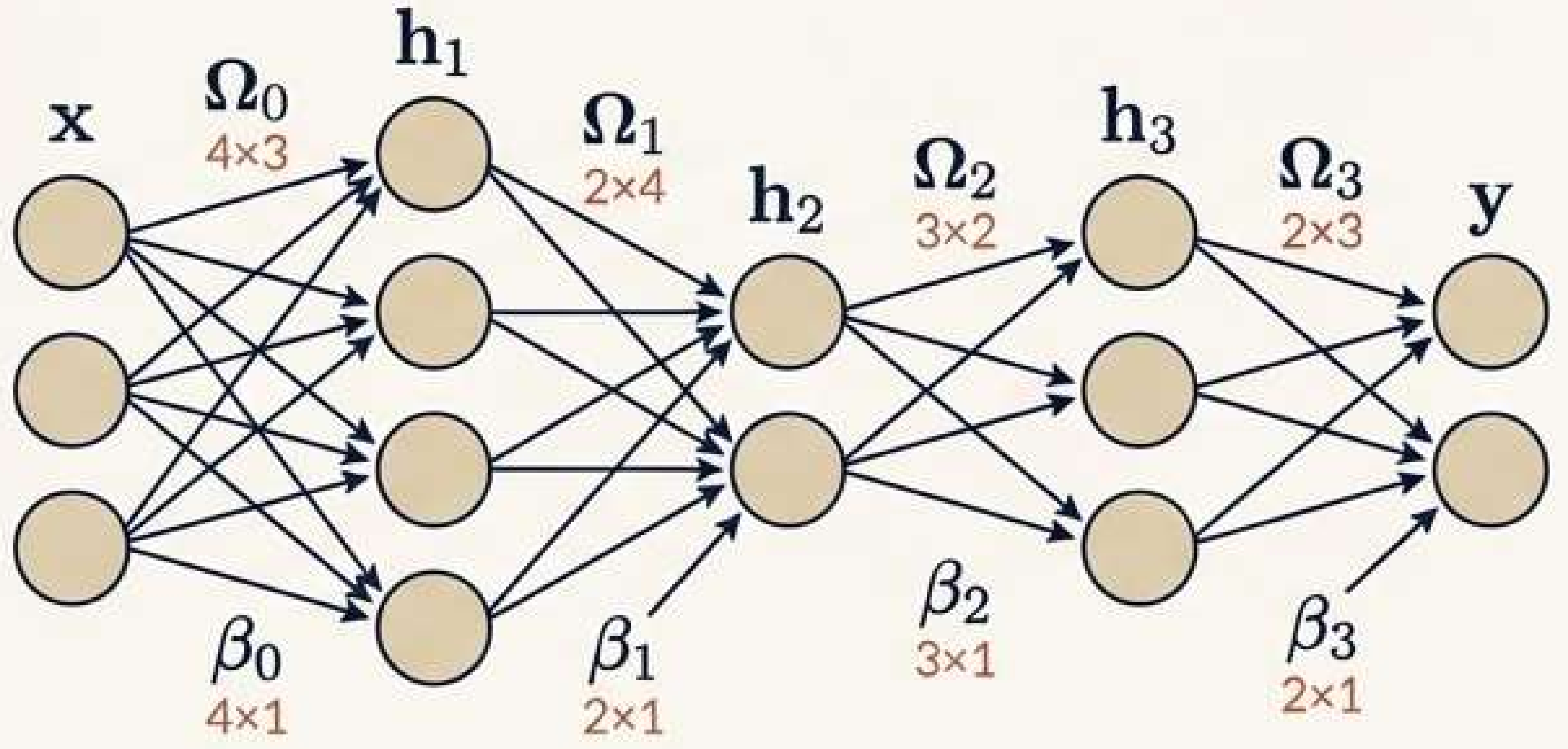
Generalizing to K Layers: The Formal Notation

To describe networks with many layers, we use a more compact matrix notation. A deep network with K layers can be written as a sequence of operations:

$$
\begin{aligned}
\mathbf{h}_1 &= a[\beta_0 + \mathbf{\Omega}_0\mathbf{x}] \\
\mathbf{h}_2 &= a[\beta_1 + \mathbf{\Omega}_1\mathbf{h}_1] \\
&\dots \\
\mathbf{h}_k &= a[\beta_{k-1} + \mathbf{\Omega}_{k-1}\mathbf{h}_{k-1}] \\
\mathbf{y} &= \beta_k + \mathbf{\Omega}_k\mathbf{h}_k
\end{aligned}
$$

Key Terminology:

- **Depth** (K): The number of hidden layers.
- **Width** (D_k): The number of hidden units in layer k.
- **Capacity:** The total number of hidden units.
- **Hyperparameters:** Structural choices like depth and width, set before training begins.



NotebookLM

The Verdict: Why Depth Wins

We began with a paradox: if shallow networks are universal approximators, why do we need deep ones?

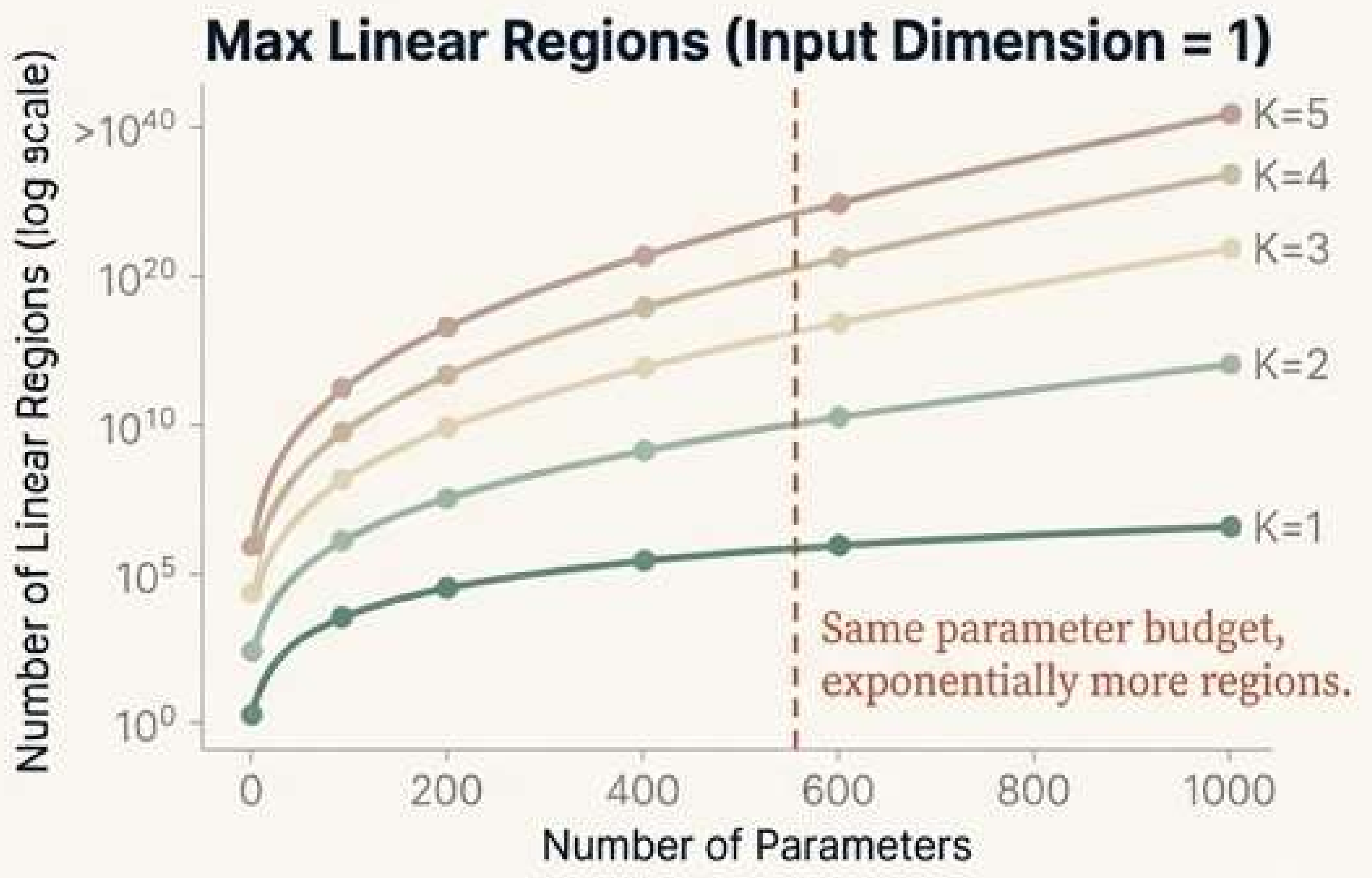
Now, equipped with an understanding of composition, we can provide a comprehensive answer. The advantage of depth is not about what functions can be modeled, but about the *efficiency* and *structure* with which they are modeled.

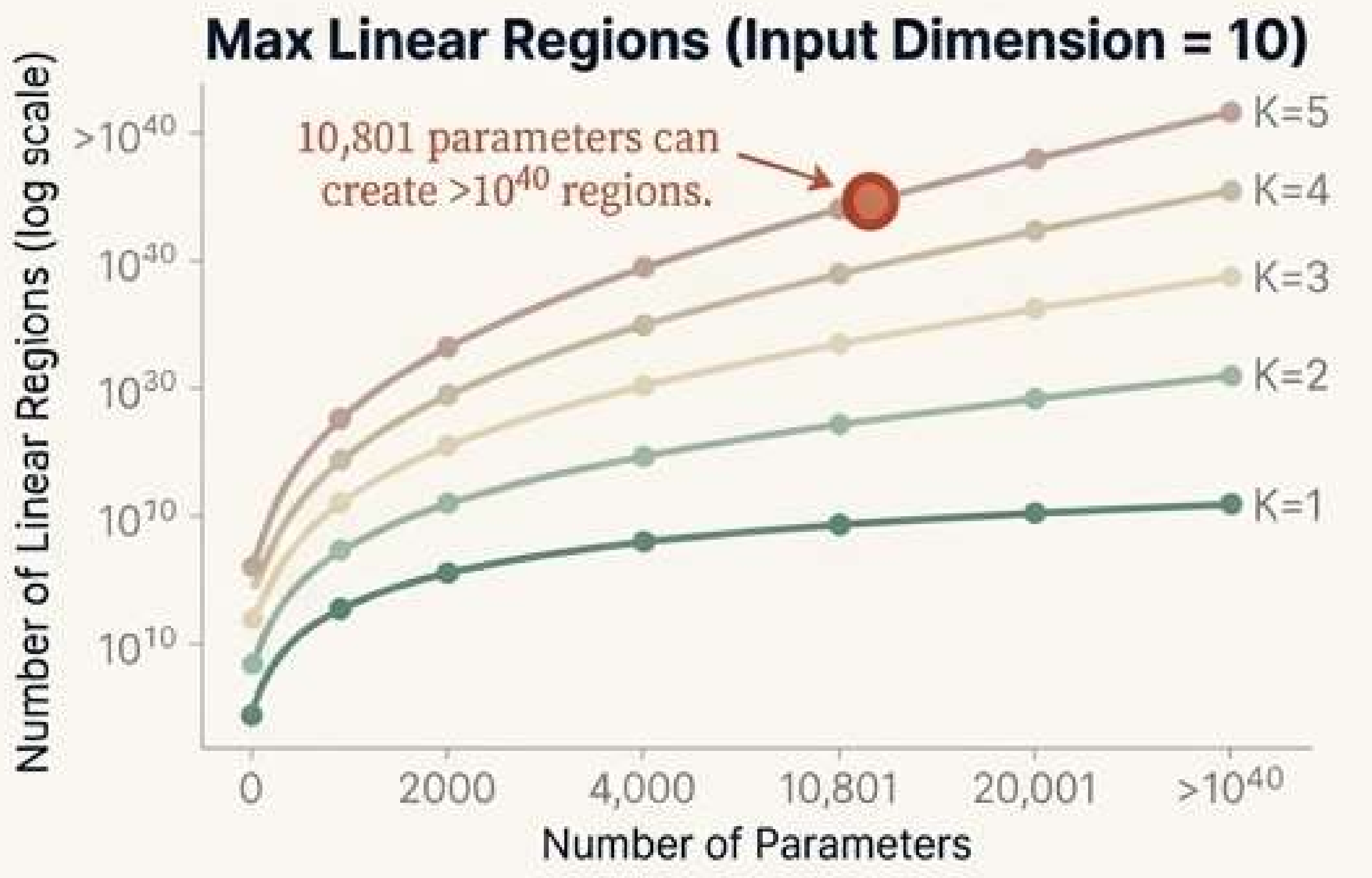
Advantage 1: An Exponential Gain in Expressive Power

For a fixed number of parameters (our "budget"), deep networks can create vastly more linear regions than shallow networks. This isn't a minor improvement; the effect is exponential.

- A shallow network with D hidden units creates at most D+1 linear regions.
- A deep network with K layers of D hidden units can create up to $(D+1)^K$ regions.

Example (D_i=10 inputs): A network with 5 layers and 50 units per layer (10,801 parameters) can create more than 10^{40} linear regions. A shallow network with a similar parameter budget would be orders of magnitude less expressive.

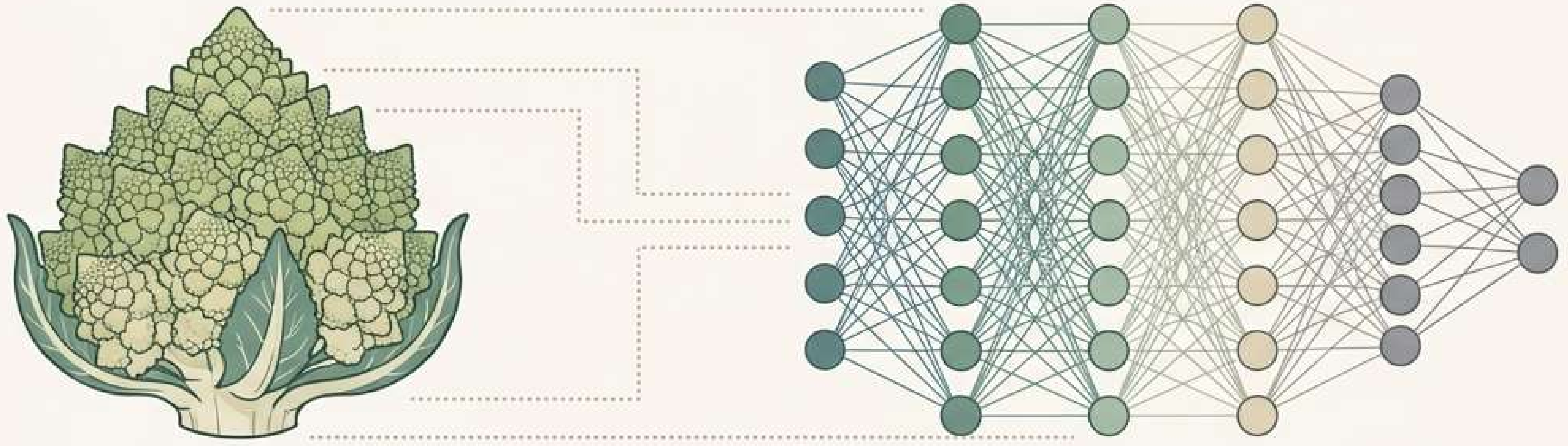




Advantage 2: Finding the Right Symmetries

The immense number of linear regions in a deep network isn't random. They contain complex dependencies and symmetries inherited from the 'folding' process. This is a feature, not a bug.

- **Depth Efficiency**: For some functions, a shallow network would require exponentially more hidden units to achieve the same approximation quality as a deep network.
- **The World is Compositional**: This advantage is most powerful if the real-world functions we want to model are also compositional–that is, they are built from a hierarchy of simpler functions. This is a foundational assumption in many deep learning applications.

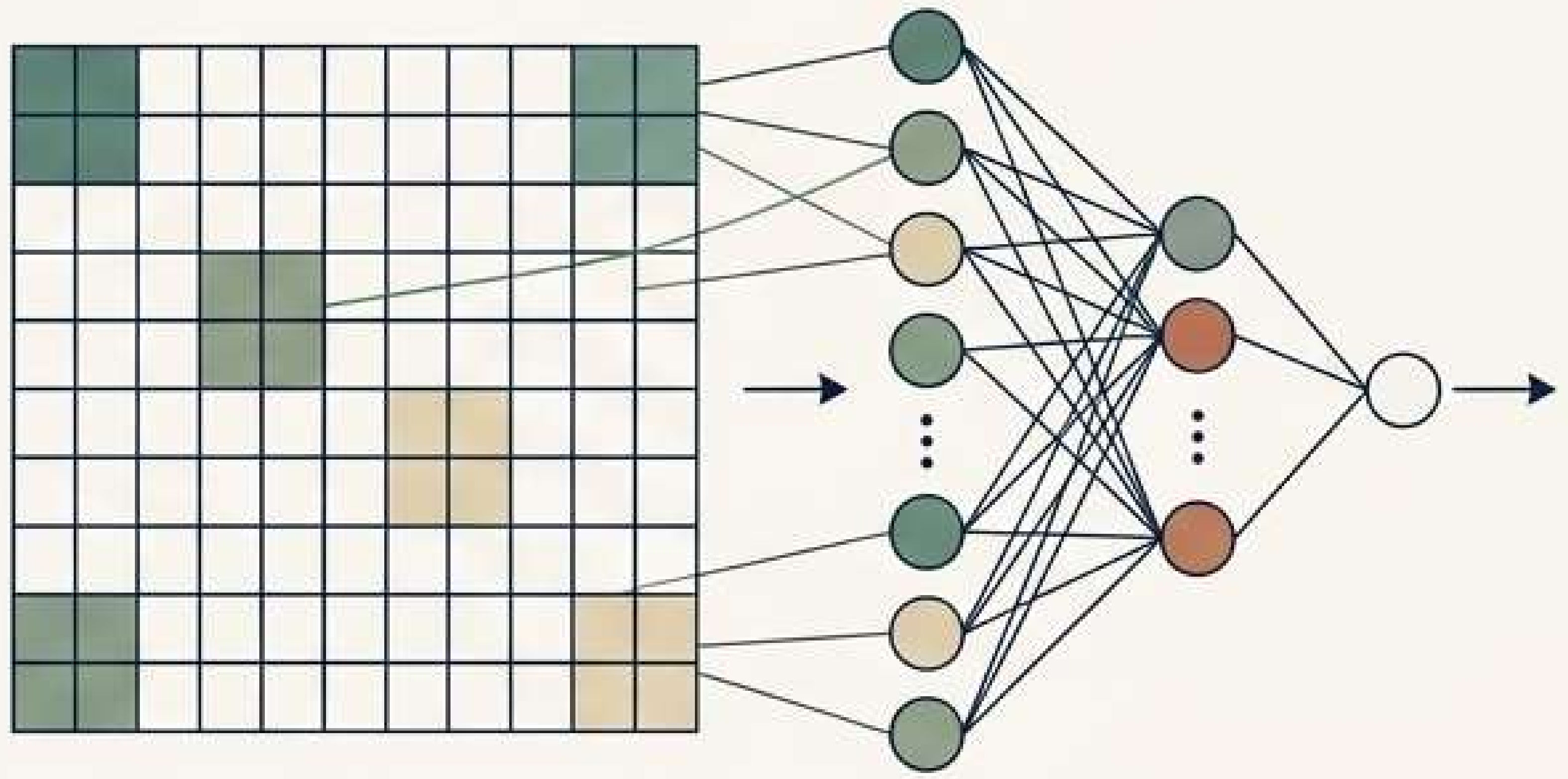

Compositional Structure in Nature
Compositional Structure in Model

NotebookLM

Advantage 3: Practical Benefits for Real-World Problems

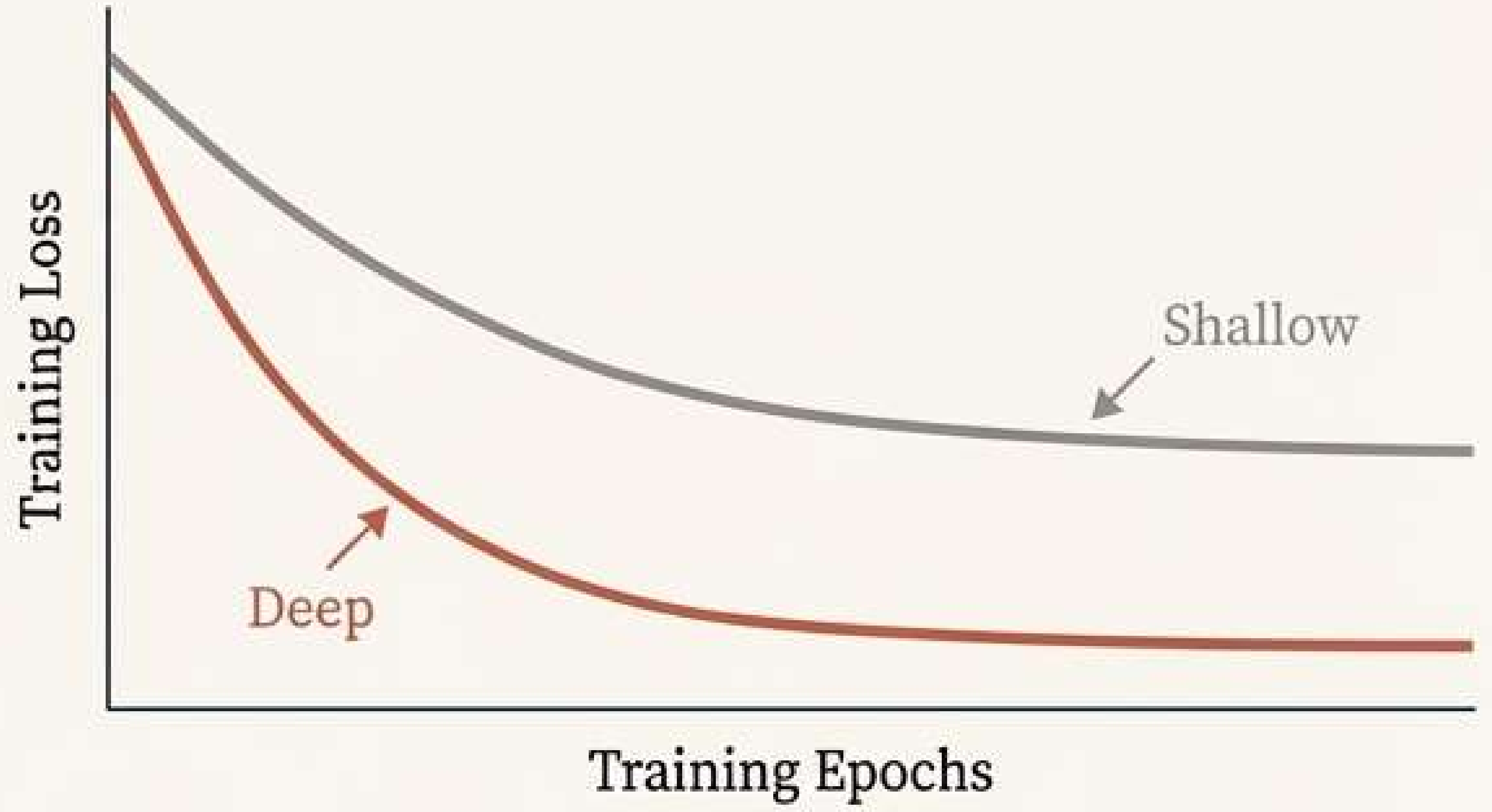
Handling Large, Structured Inputs

For data like images (~10^6 pixels), fully connected shallow networks are computationally prohibitive. Deep, layered architectures allow for local-to-global processing (like in Convolutional Neural Networks), where information from small regions is gradually integrated. This is impossible without multiple layers.



Easier Training and Better Generalization

Empirically, moderately deep networks are often easier to train than very wide shallow ones. They also tend to generalize better to new, unseen data. While not fully understood, these practical benefits are a primary reason for the dominance of deep learning.



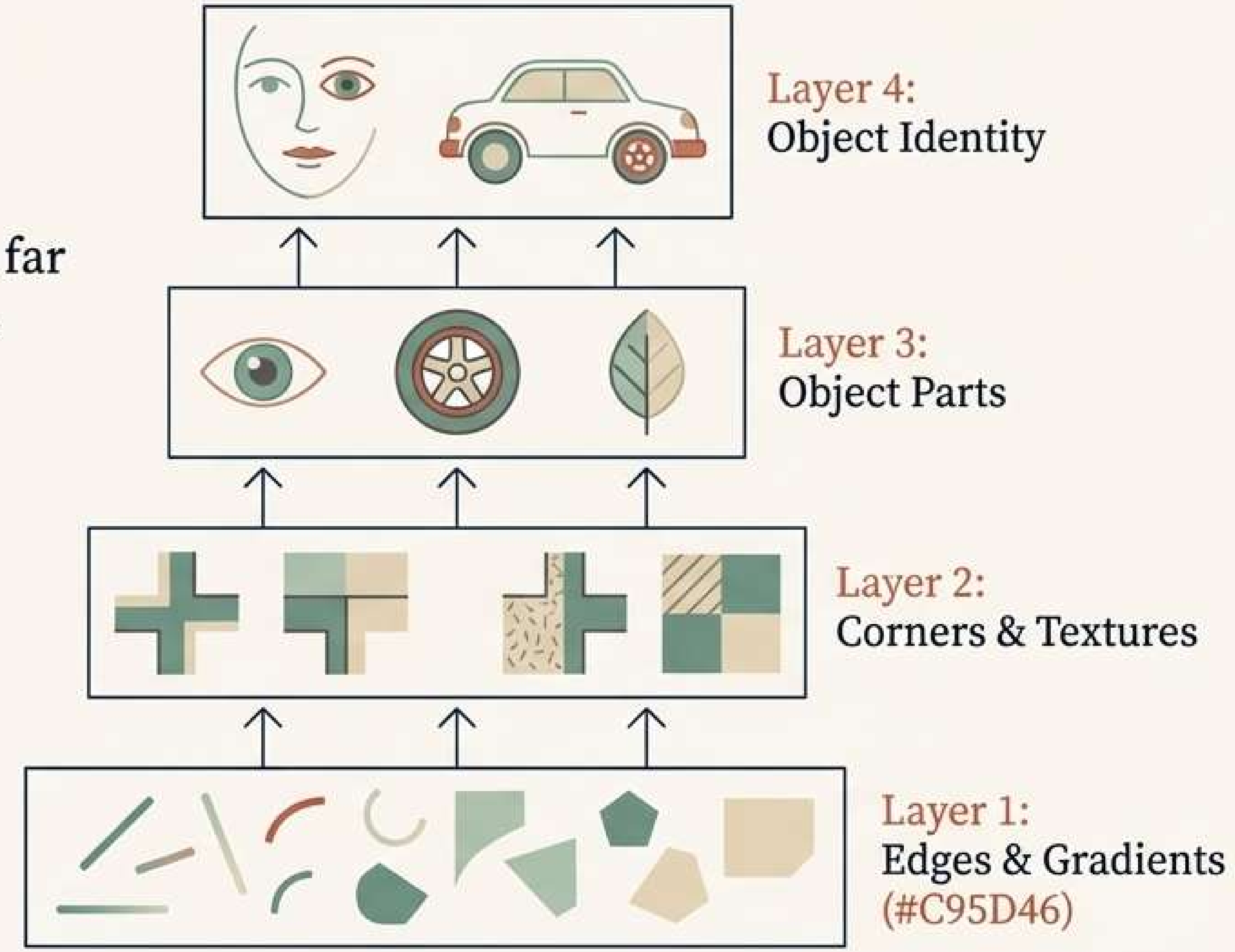
NotebookLM

Depth is How Networks Learn the Compositional Structure of the World

The power of deep networks is not just their ability to approximate any function, but their profound efficiency in doing so. By composing simple functions layer by layer, deep networks can represent complex, hierarchical patterns with far fewer parameters than their shallow counterparts.

They succeed because they embody a fundamental assumption: that the world we seek to model is itself compositional, built from a hierarchy of simpler structures and concepts.

Depth gives our models the right architecture to discover and exploit this structure.



NotebookLM