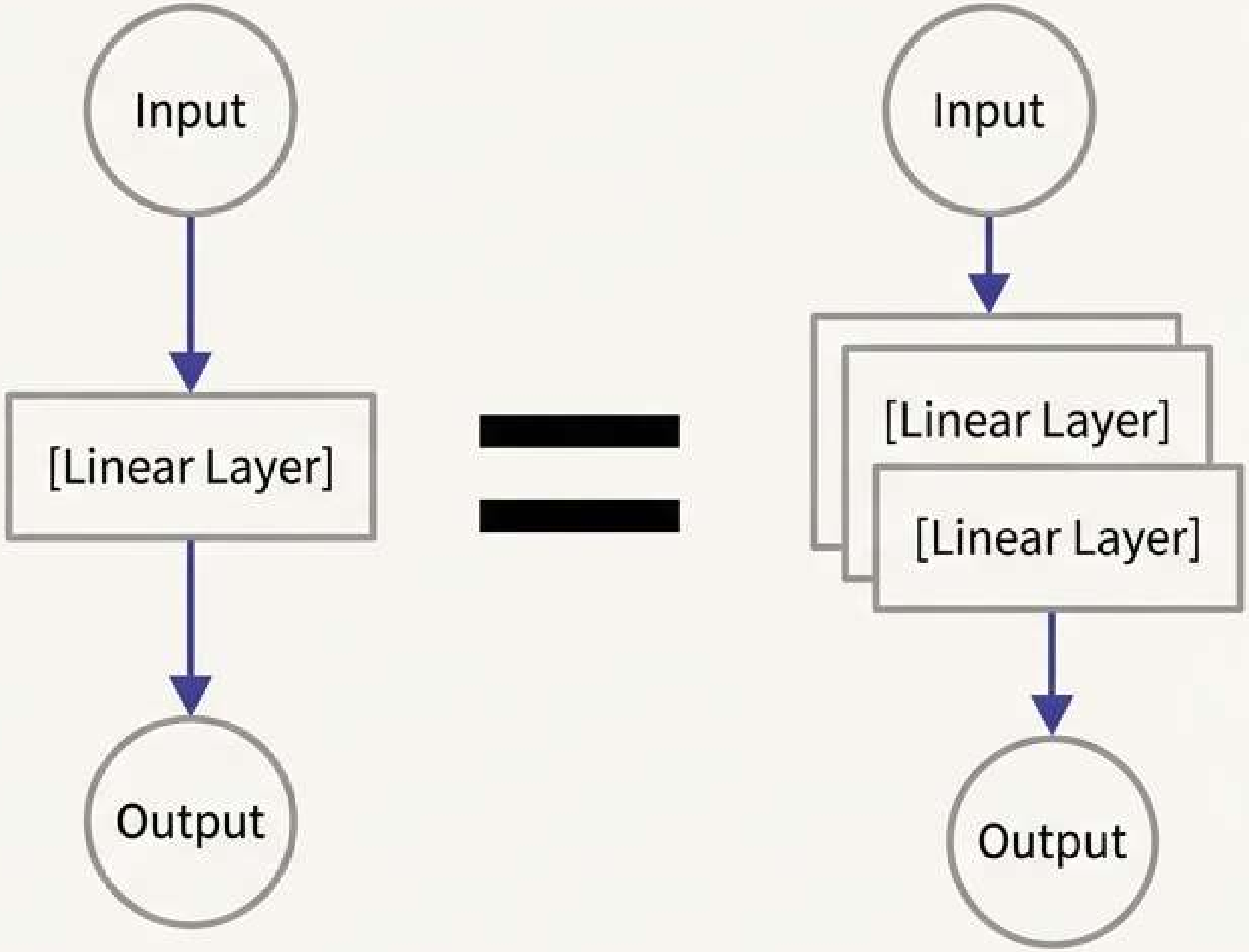
The Unseen Engine of Deep Learning

A Strategic Guide to Activation Functions

Why a 100-Layer Network Can Act Like a Single Layer

Without activation functions, a deep neural network collapses into a simple linear model. Stacking linear operations only results in another linear operation.

Every neuron would only perform a linear transformation on its inputs. It doesn't matter how many hidden layers we attach; all layers will behave in the same way because the composition of two linear functions is a linear function itself. Our model would be no more powerful than a basic linear regression model.

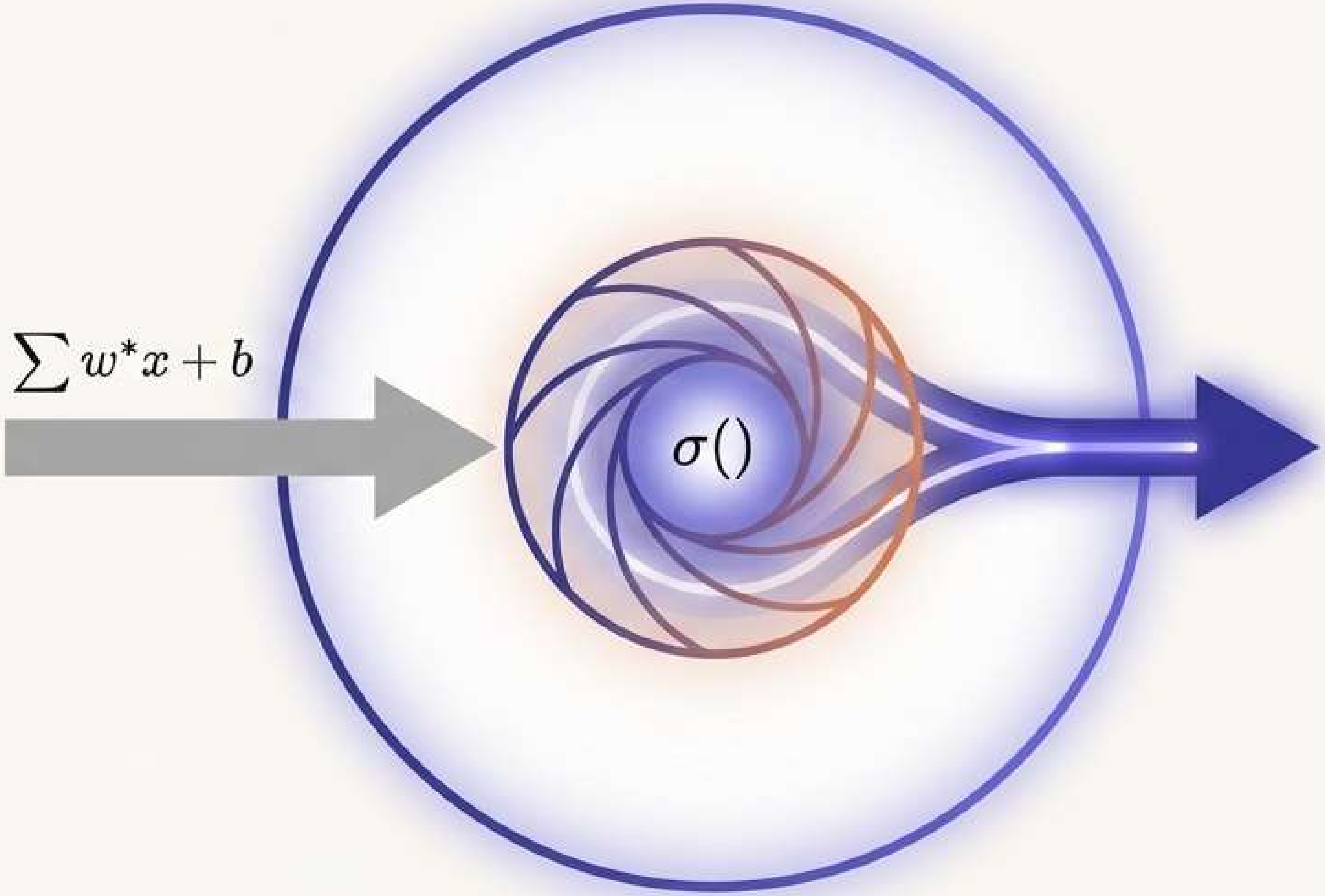


Functionally Equivalent

The Solution: Adding a Non-Linear Gatekeeper

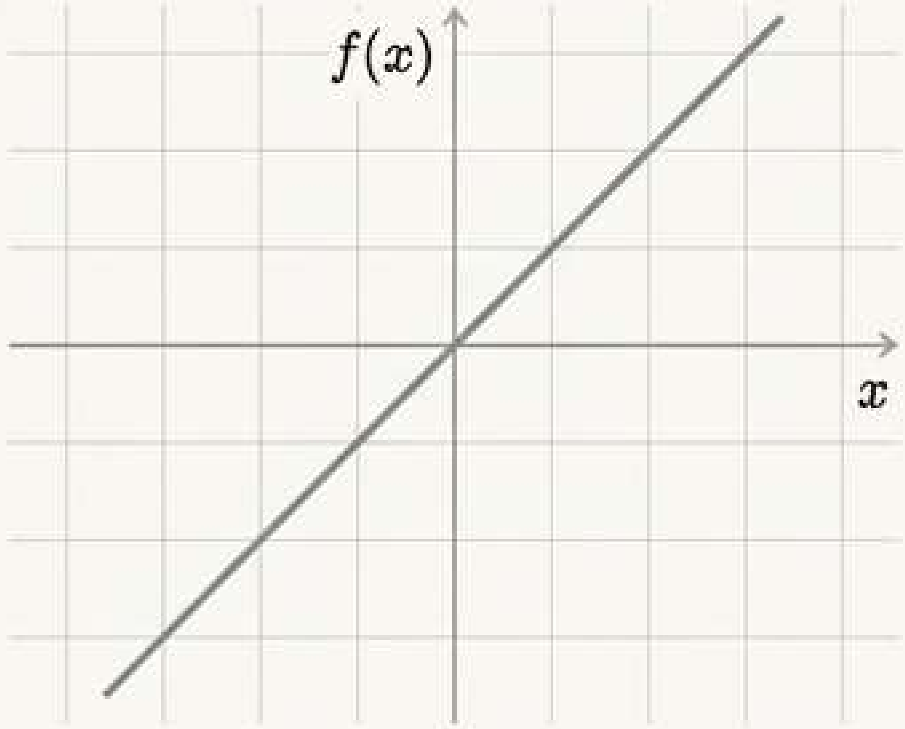
An Activation Function decides whether a neuron should be activated (‘fire’) and with what intensity. Its primary role is to introduce non-linearity into the network.

Think of it as a mathematical ‘gate’ or ‘dimmer switch’ between the input a neuron receives and the output it sends to the next layer. It sends to the next layer. It transforms the summed weighted input into an output value.



Early Attempts and Their Fundamental Flaws

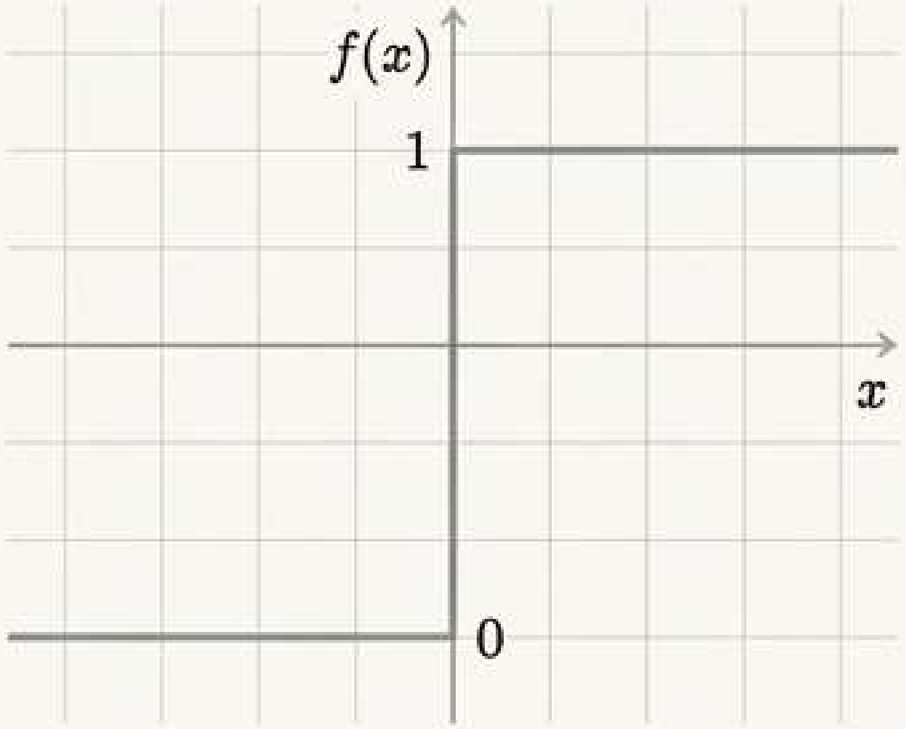
Linear Function



$$f(x) = x$$

No non-linearity. The derivative is a constant, making backpropagation ineffective as it has no relation to the input x. The entire network still collapses into a single layer.

Binary Step Function

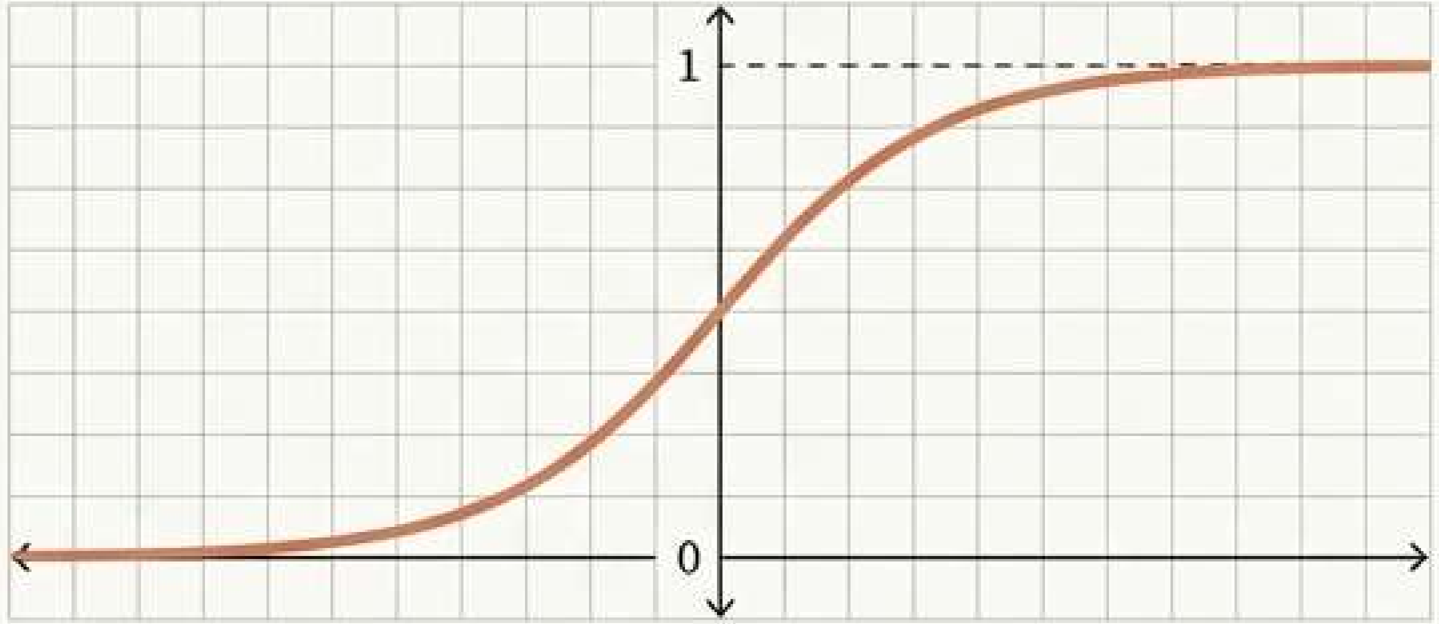


$$f(x) = 1 \text{ if } x > \text{threshold, } 0 \text{ otherwise}$$

Cannot be used for multi-class classification. More critically, the gradient is zero everywhere except at the threshold. This halts the backpropagation process, preventing the network from learning.

The Classic S-Curves: Sigmoid and Tanh

Sigmoid (Logistic)



$$f(x) = \frac{1}{1 + e^{-x}}$$

Role

Squashes real values into a [0, 1] range. Ideal for the output layer in binary classification, where the output can be interpreted as a probability.

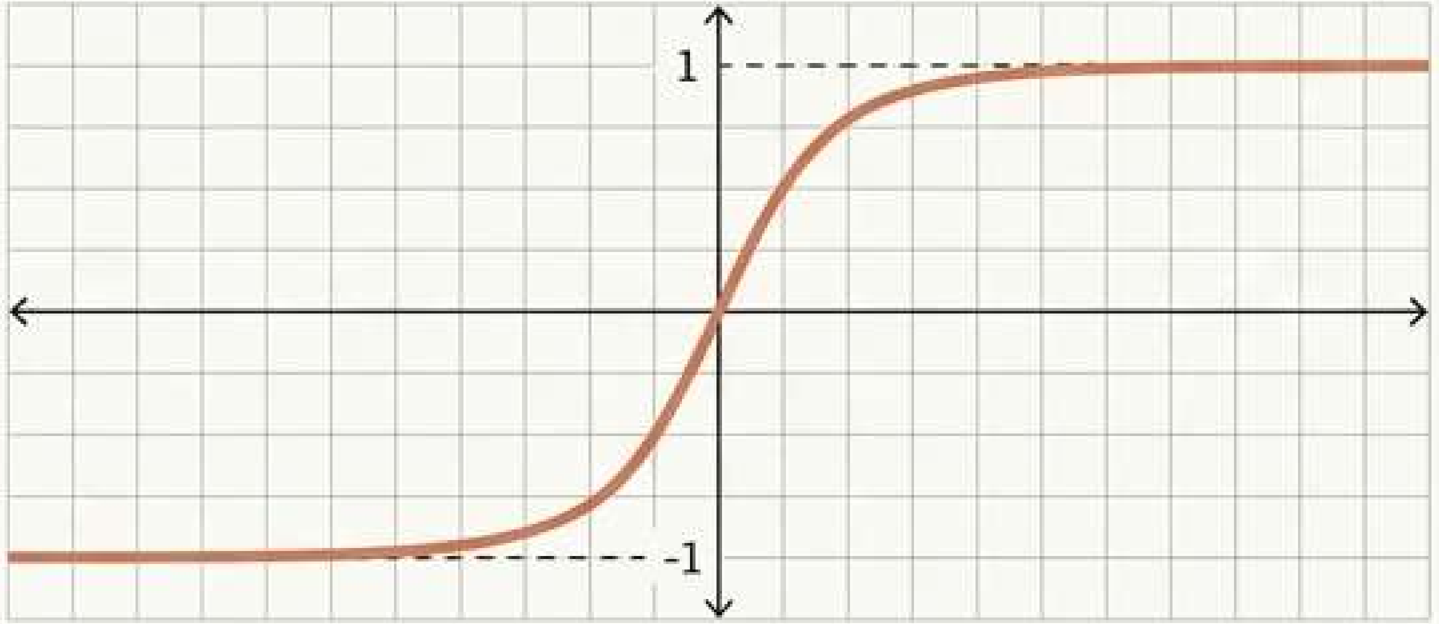
Strengths

- Differentiable
- Smooth gradient

Weakness

- Output is not zero-centered.
- Suffers from the vanishing gradient problem.

Tanh (Hyperbolic Tangent)



$$f(x) = \tanh(x)$$

Role

Often preferred over Sigmoid in hidden layers due to its zero-centered output, which helps the next layer learn more easily.

Strengths

- Zero-centered output.
- Steeper, stronger gradients than Sigmoid.

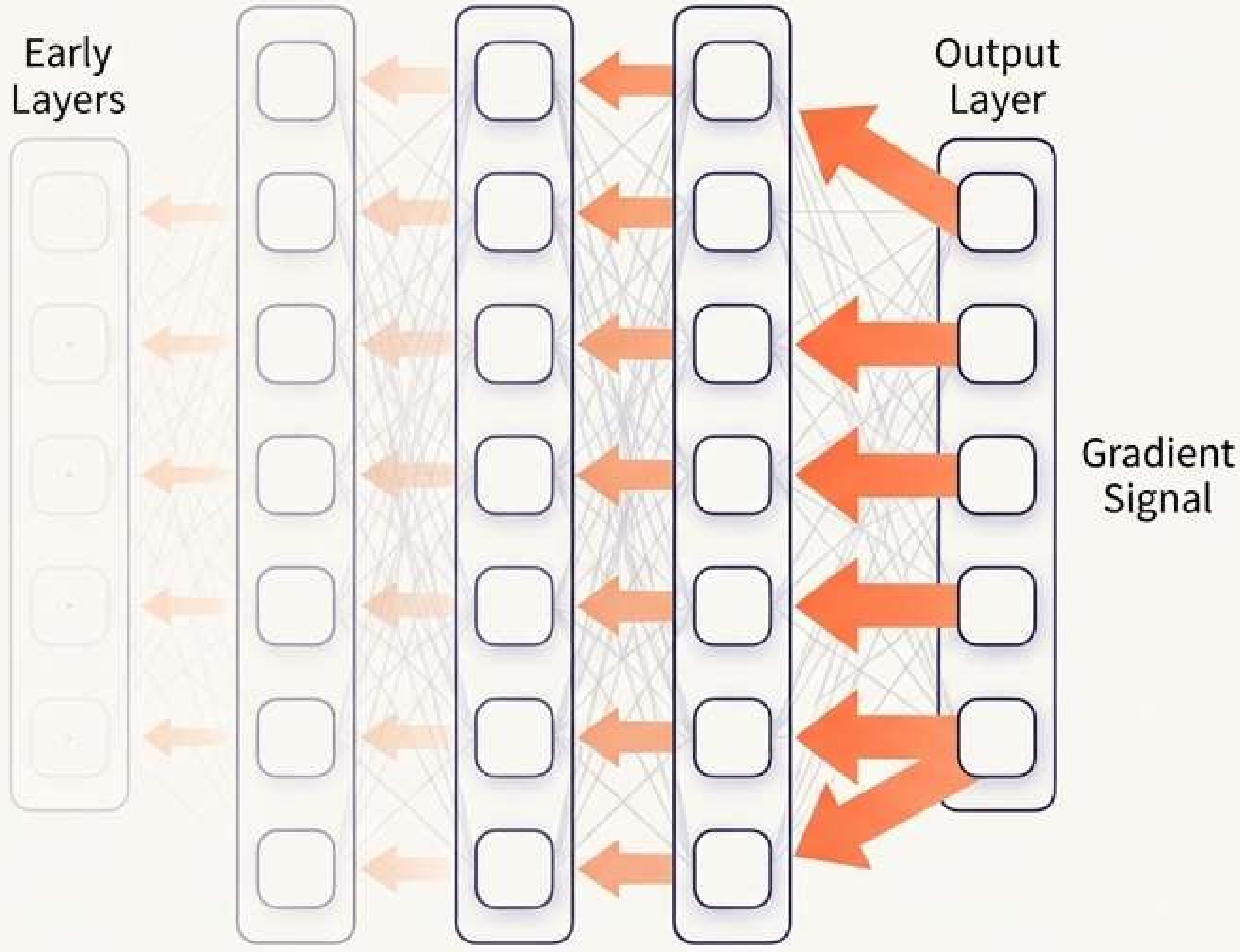
Weakness

- While an improvement, it also suffers from the vanishing gradient problem in its saturated regions.

The Saturation Problem: Vanishing Gradients

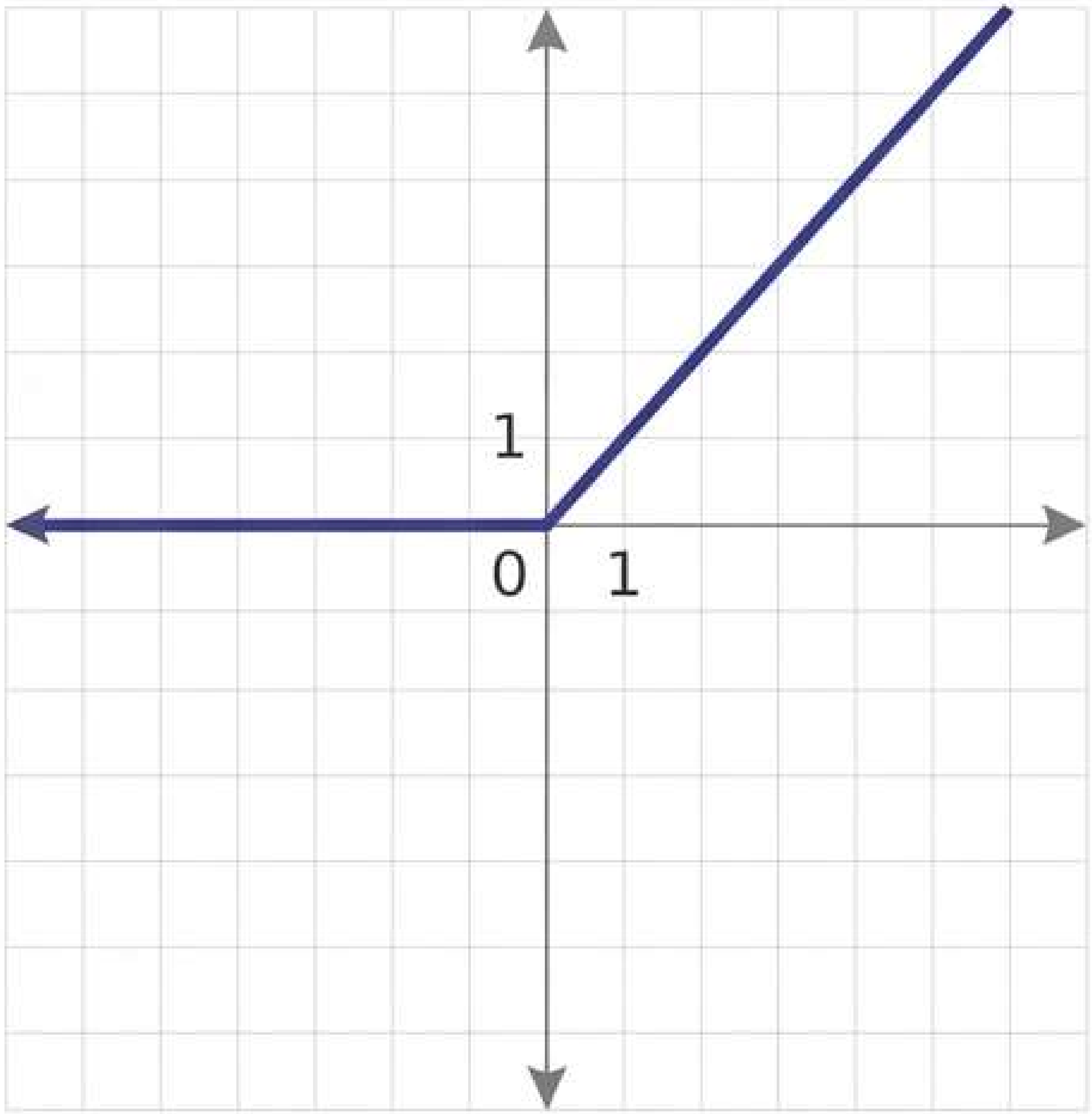
S-shaped functions like Sigmoid and Tanh “squish” a large input space into a small output range. As the input becomes very large or very small, the function saturates, and the slope becomes extremely flat. The gradient approaches zero.

The Impact: During backpropagation, these tiny gradients are multiplied through each layer. In a deep network, the signal can become so small by the time it reaches the early layers that the weights barely update. The network effectively stops learning.



NotebookLM

The Breakthrough: ReLU (Rectified Linear Unit)



$$f(x) = \max(0, x)$$

ReLU is a non-linear function that does not activate all neurons at the same time. It simply thresholds the input at zero.

Why It Works (Pros)

- **Computationally Efficient:** Involves a simple thresholding operation.
- **Solves Vanishing Gradients:** For positive inputs, the gradient is a constant 1, allowing it to pass through the network without shrinking.
- **Accelerates Convergence:** Its linear, non-saturating property helps gradient descent converge much faster.

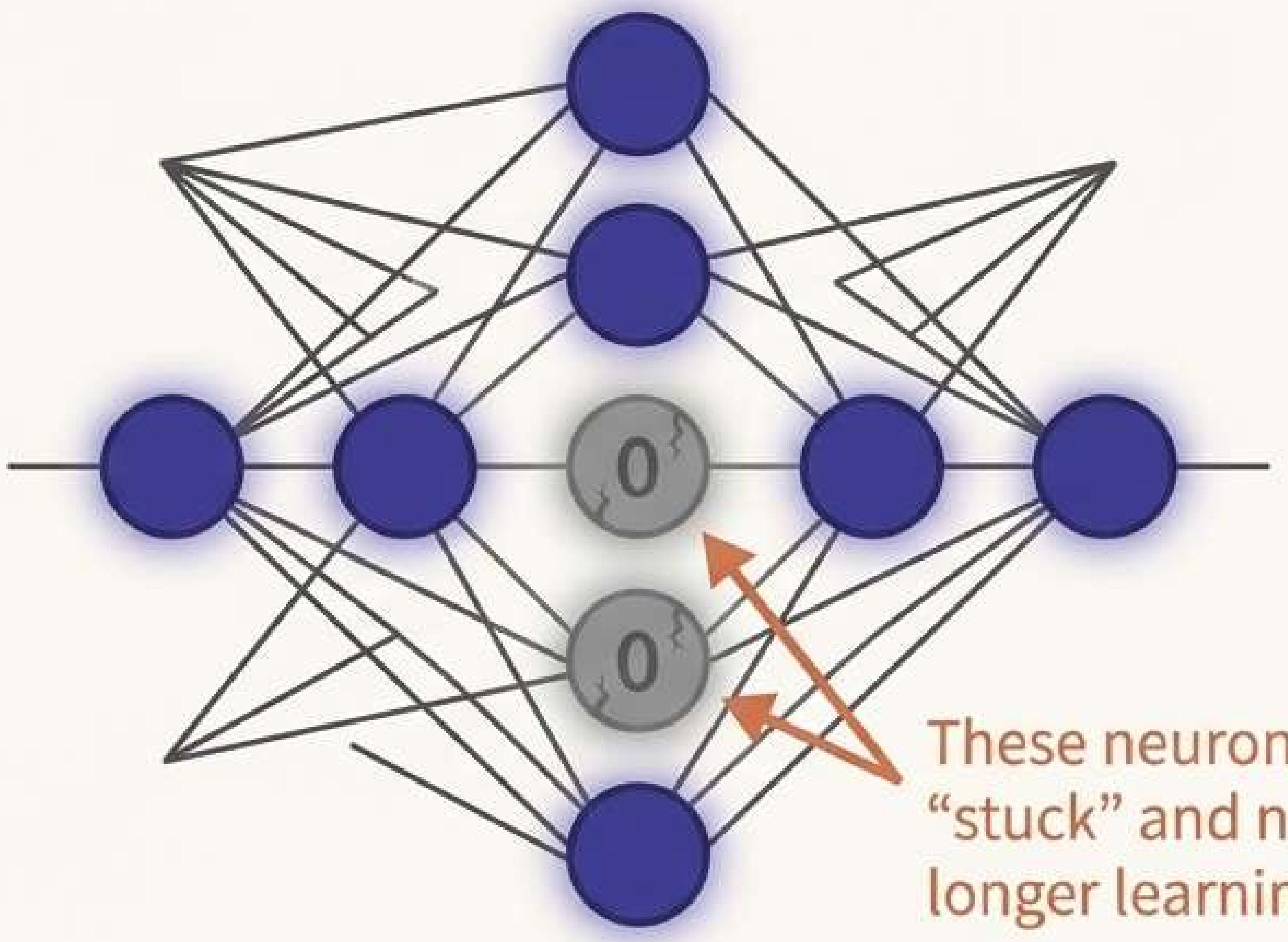
The New Problem (Con)

- Introduces the ‘Dying ReLU’ problem.

The Downside of Simplicity: The Dying ReLU Problem

If a neuron's input is consistently negative, the output will always be zero. Consequently, the gradient flowing through that neuron will also be permanently zero. Because the weights are updated based on the gradient, this neuron's weights will never be updated.

This can create "dead" neurons that never activate, effectively being removed from the network. This decreases the model's capacity to fit and learn from the data properly.

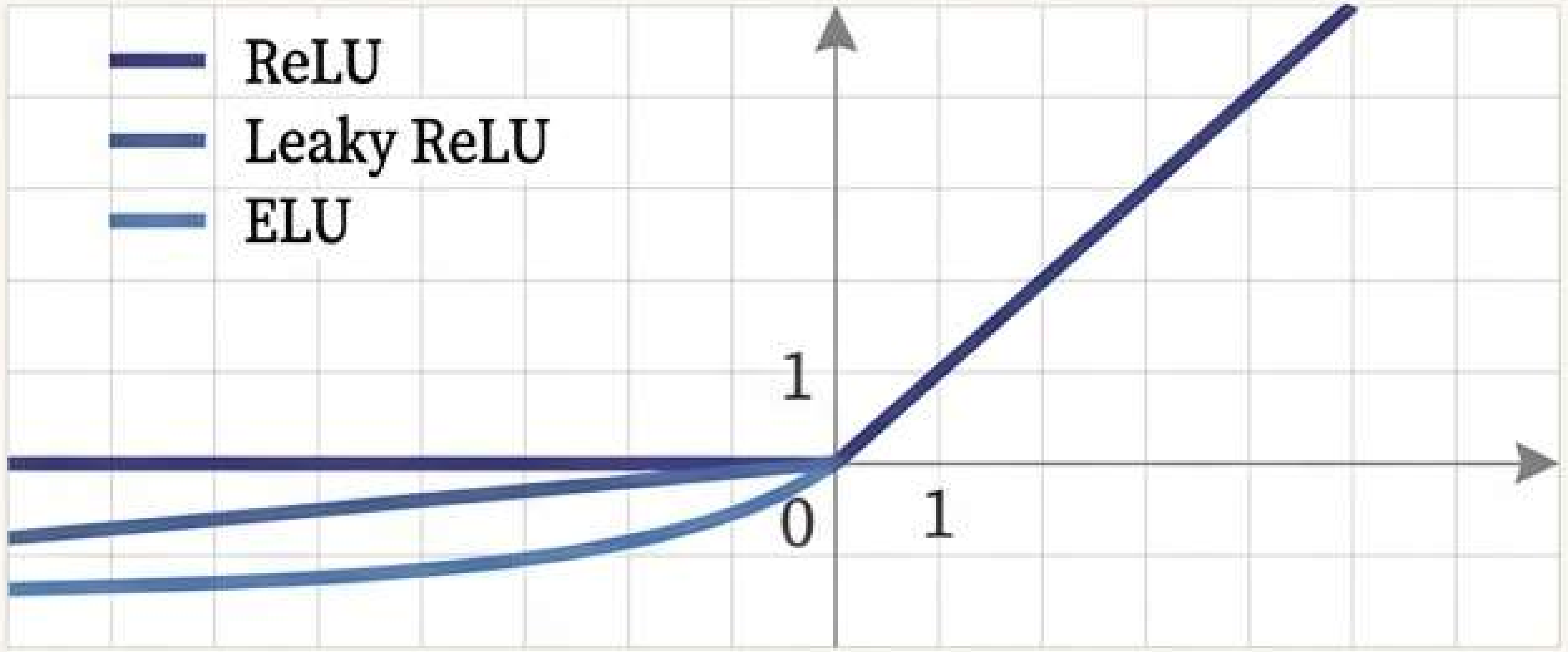


These neurons are "stuck" and no longer learning.

NotebookLM

The ReLU Family: Iterating on a Good Idea

Variants of ReLU aim to solve the Dying ReLU problem by introducing a small, non-zero slope for negative inputs.



Leaky ReLU

Introduces a small, fixed positive slope (e.g., 0.01) for negative values.

$$f(x) = \begin{cases} 0.01x & \text{if } x < 0, \\ x & \text{if } x \geq 0 \end{cases}$$

Parametric ReLU (PReLU)

Makes the slope for negative values a learnable parameter, *a*, which is adjusted during backpropagation.

$$f(x) = \begin{cases} ax & \text{if } x < 0, \\ x & \text{if } x \geq 0 \end{cases}$$

Exponential Linear Unit (ELU)

Uses a smooth, logarithmic curve for negative inputs, which can help push the mean activation closer to zero.

$$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x < 0, \\ x & \text{if } x \geq 0 \end{cases}$$

For the Output Layer: Softmax Turns Scores into Probabilities

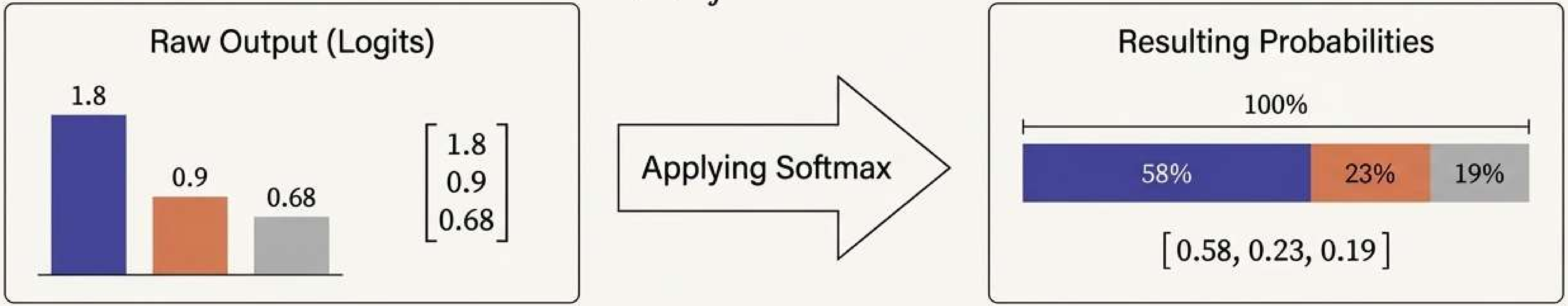
The Problem It Solves

In multi-class classification, the final layer outputs raw scores, or "logits," for each class. How do we convert these into a probability distribution where the probabilities of all classes sum to 1?

The Solution

Softmax, also known as the normalized exponential function, calculates the relative probabilities. It exponentiates each output and then normalizes it by the sum of all exponentiated outputs.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum e_{j}^{z_j}} \text{ for j=1 to K classes}$$



Raw Output (Logits)

$$\begin{bmatrix} 1.8 \\ 0.9 \\ 0.68 \end{bmatrix}$$

Applying Softmax

Resulting Probabilities

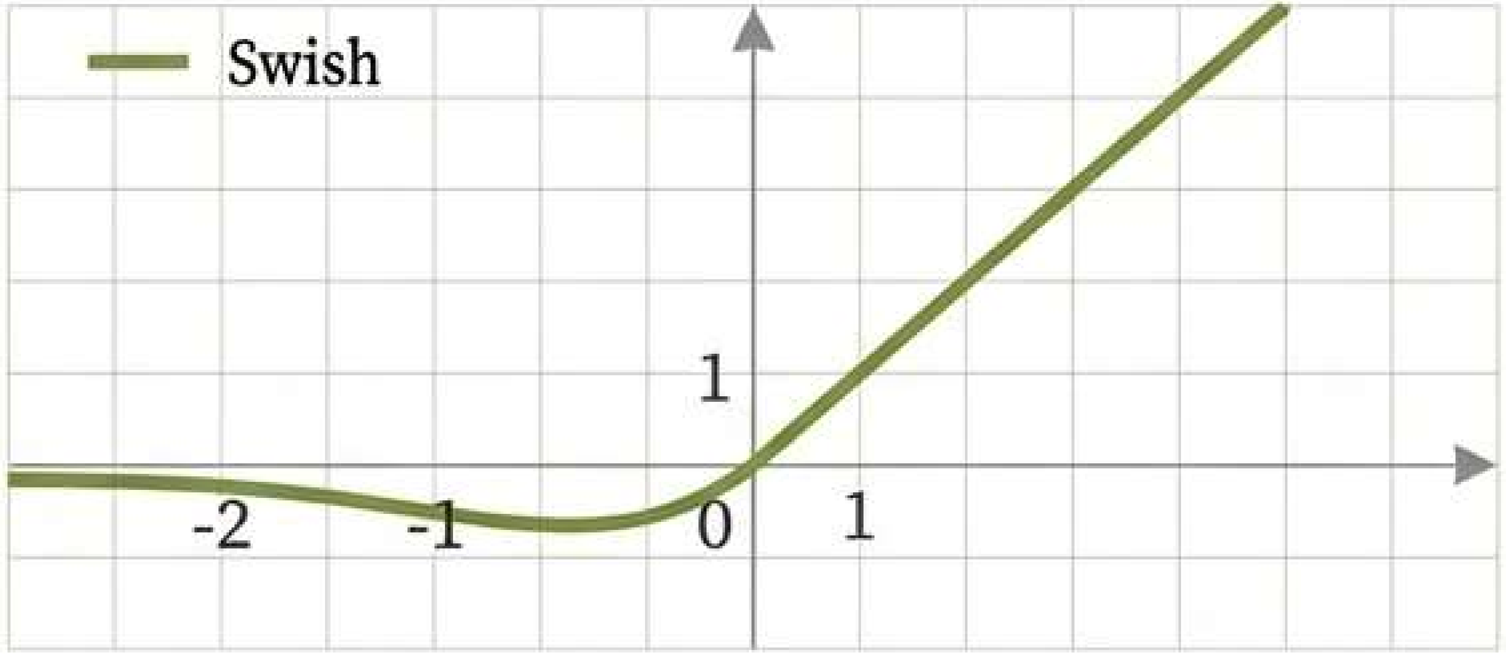
$[0.58, 0.23, 0.19]$

Conclusion: The model predicts the first class with 58% confidence.

The Modern Contenders: Swish and GELU

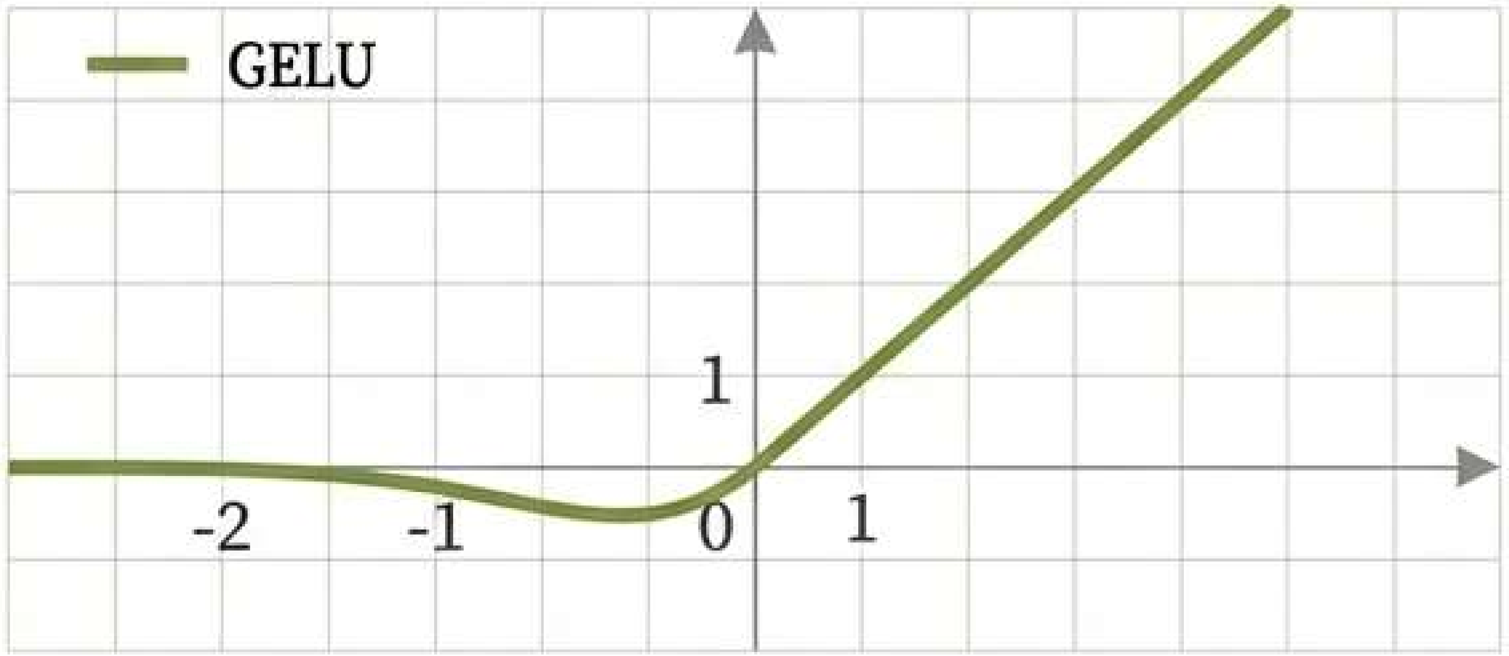
Swish

Origin: A self-gated function developed by Google.
Formula: $f(x) = x * \sigma(x)$
Key Property: A smooth, non-monotonic function that consistently matches or outperforms ReLU on very deep networks (>40 layers). Unlike ReLU, it allows small negative values to be preserved, which can be relevant for capturing underlying patterns.



GELU (Gaussian Error Linear Unit)

Usage: The activation function used in top NLP models like BERT, RoBERTa, and ALBERT.
Motivation: Combines properties from dropout, zoneout, and ReLU. It stochastically multiplies the input by 0 or 1, with the probability determined by the input's value in a cumulative normal distribution.
Key Property: Finds performance improvements across computer vision, NLP, and speech recognition tasks.



An Alternative Path: SELU and Self-Normalization

Core Concept

The Scaled Exponential Linear Unit (SELU) is designed for self-normalizing neural networks. It preserves the mean and variance from the previous layer, enabling a form of internal normalization.

The Advantage

Internal normalization can be faster than external normalization (like Batch Norm), potentially leading to faster network convergence.

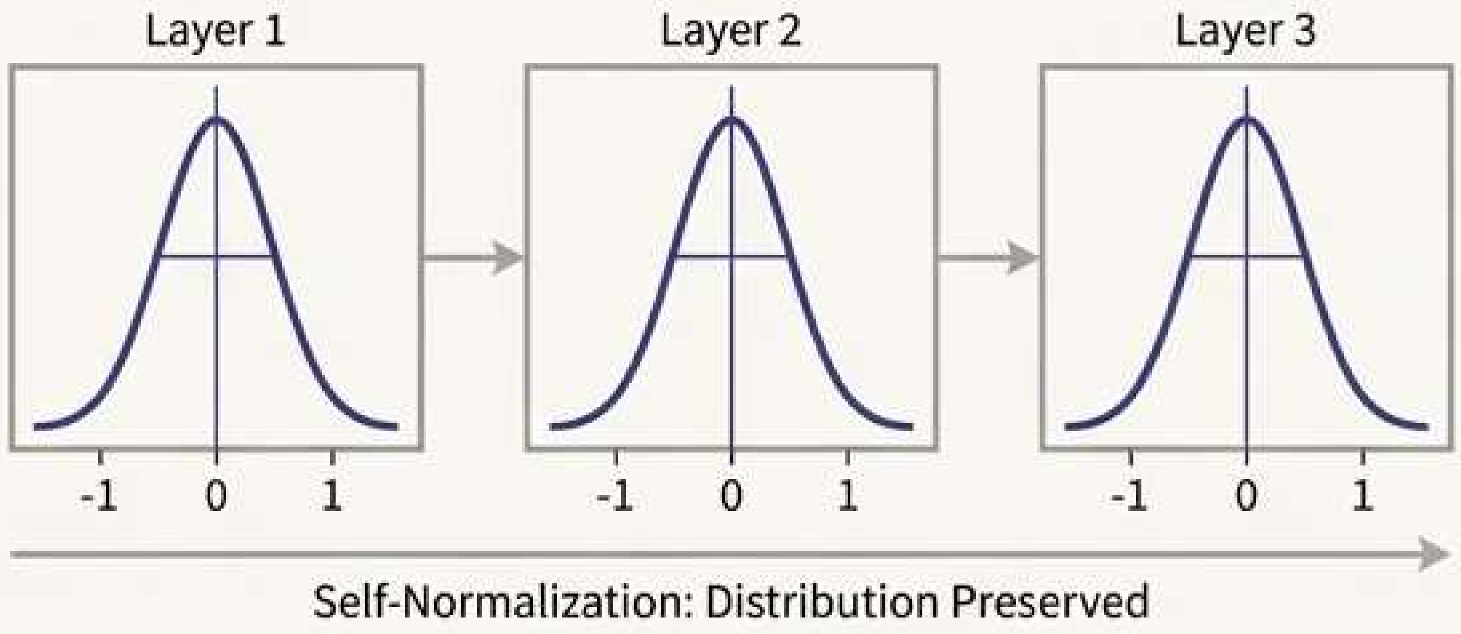
Mechanism

SELU has both positive and negative values to shift the mean, and a gradient larger than one in a specific region to adjust the variance. The `alpha` (α) and `lambda` (λ) parameters are predefined to ensure this property.

$$f(x) = \lambda(\alpha e^{x} - \alpha) \text{ if } x < 0, \lambda x \text{ if } x \geq 0$$

Status

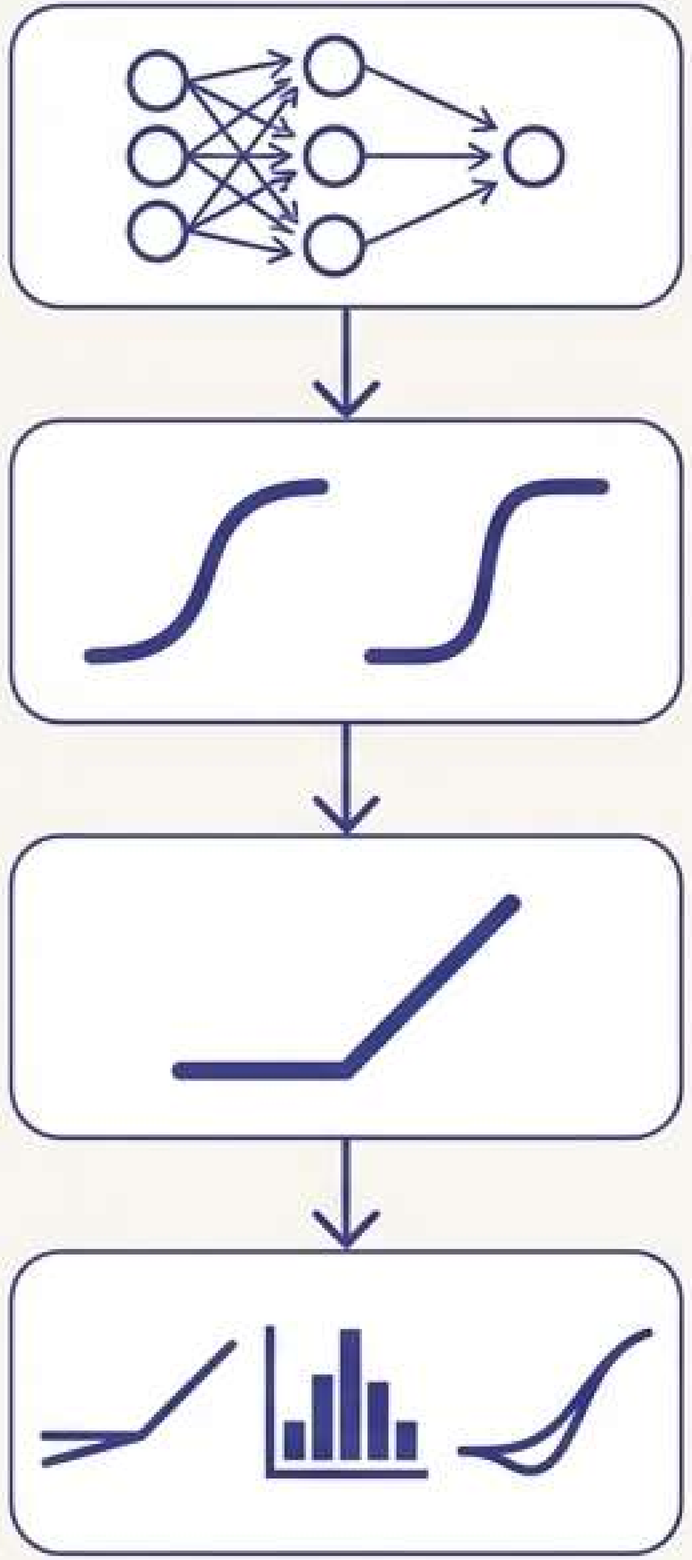
A powerful but more specialized function, primarily explored in the context of self-normalizing architectures.



The Decision Framework: A Cheatsheet for Choosing the Right **Activation Function**

	Choosing for HIDDEN LAYERS		Choosing for the OUTPUT LAYER (This is determined by your problem type)
1	Your Default Starting Point: **ReLU**. It's fast, effective, and the most common choice.	1	Problem: **Regression** (predicting a continuous value) → Function: **Linear / Identity.**
2	If You Experience Dying Neurons: **Leaky ReLU** or **ELU**.	2	Problem: **Binary Classification** (one class or another) → Function: **Sigmoid.**
3	For Very Deep Networks (>40 layers): **Swish** may outperform ReLU.	3	Problem: **Multi-Class Classification** (one of many classes) → Function: **Softmax.**
4	For Recurrent Neural Networks (RNNs): **Tanh** is a strong traditional choice due to its zero-centered range.	4	Problem: **Multi-Label Classification** (can have multiple classes) → Function: **Sigmoid** (applied to each output neuron independently).

The Story of Activation Functions: A Journey of Problem and Solution



Deep networks without non-linearity are just shallow models.

Sigmoid & Tanh introduced curves but created the **Vanishing Gradient** problem.

ReLU solved vanishing gradients with elegant simplicity but introduced the **Dying ReLU** problem.

The **ReLU Family** (Leaky, PReLU, ELU) fixed the dying neuron issue.
Softmax was perfected for multi-class probability outputs.
Swish, GELU, SELU push performance boundaries for state-of-the-art models.

The choice of an activation function is a strategic decision based on a history of trade-offs and innovation.

Sources & Further Exploration

- **Article:** Activation Functions in Neural Networks [12 Types & Use Cases]
  Pragati Baheti, V7 Labs

- **Article:** Introduction to Activation Functions in Neural Networks
  Moez Ali, DataCamp

- **Guide:** Neural networks: Activation functions
  Machine Learning Crash Course, Google for Developers