

Activation Functions

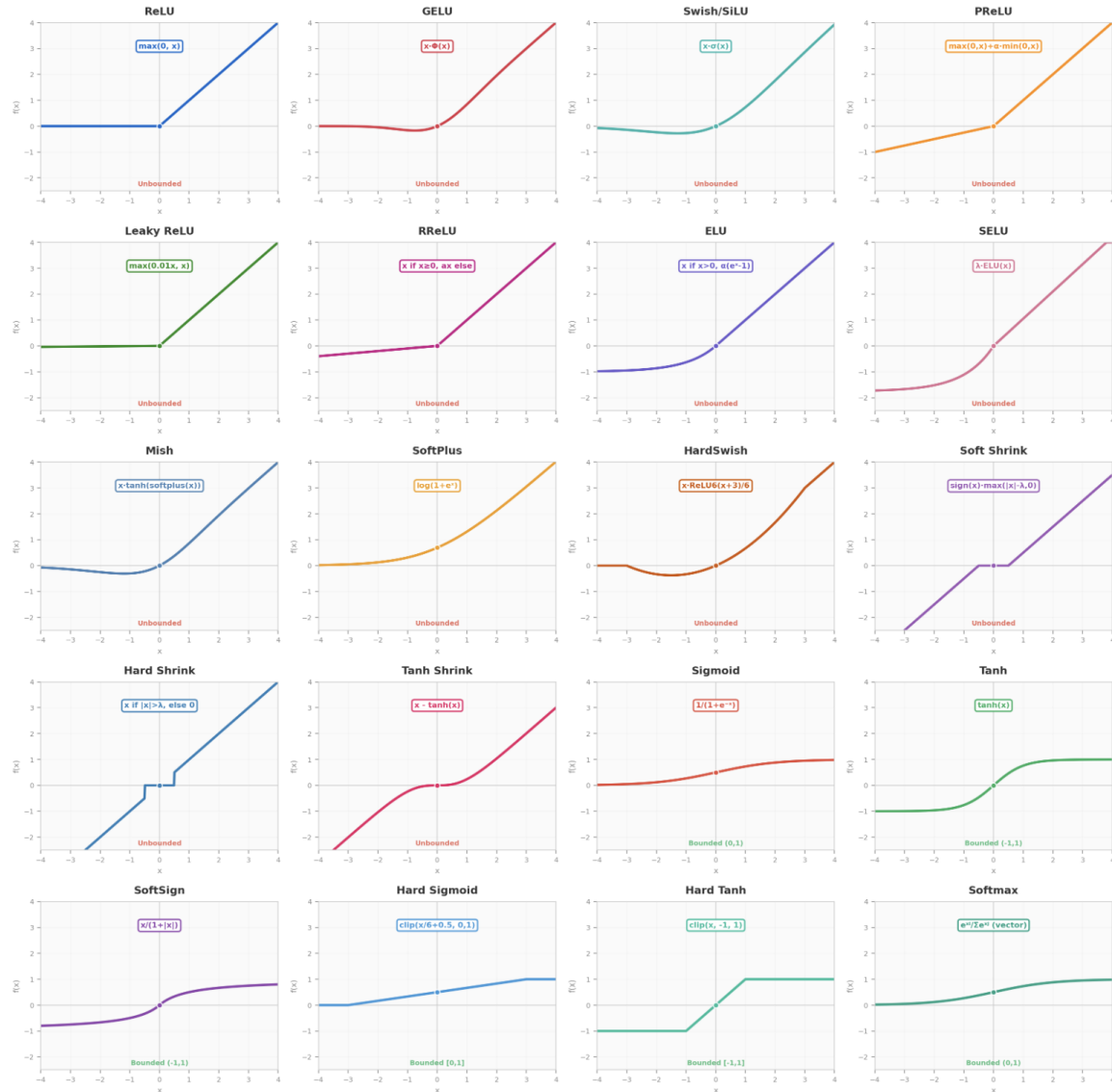


Table of Contents

The Problem Activation Functions solve	3
1. ReLU (Rectified Linear Unit)	4
2. GELU (Gaussian Error Linear Unit)	5
3. Swish / SiLU (Sigmoid Linear Unit)	6
4. SwiGLU (Swish-Gated Linear Unit)	7
5. PReLU (Parametric ReLU)	7
6. Leaky ReLU / RReLU	8
7. ELU (Exponential Linear Unit)	9
8. SELU (Scaled ELU)	10
9. Mish	11
10. Sigmoid	12
11. Tanh	13
12. SoftPlus	14
13. SoftSign	15
14. Hard Sigmoid / Hard Tanh / HardSwish	16
15. Shrink Functions (Soft/Hard/Tanh Shrink)	17
16. Softmax	18
 Ranking: Best for each Use Case	19
 Evolution Timeline	20
 Key Takeaways	20

The Problem Activation Functions solve

The Linearity Problem:

A neural network layer computes:

$$\text{output} = W \cdot x + b$$

If you stack layers without activation:

$$\text{Layer 1: } h_1 = W_1 \cdot x + b_1$$

$$\text{Layer 2: } h_2 = W_2 \cdot h_1 + b_2 = W_2 \cdot (W_1 \cdot x + b_1) + b_2$$

$$= (W_2 \cdot W_1) \cdot x + (W_2 \cdot b_1 + b_2)$$

$$= W' \cdot x + b' \text{ (Still linear)}$$

No matter how many layers it always collapses to a single linear transformation.

The Solution is non-linearity:

$$\text{Layer 1: } h_1 = \text{activation}(W_1 \cdot x + b_1)$$

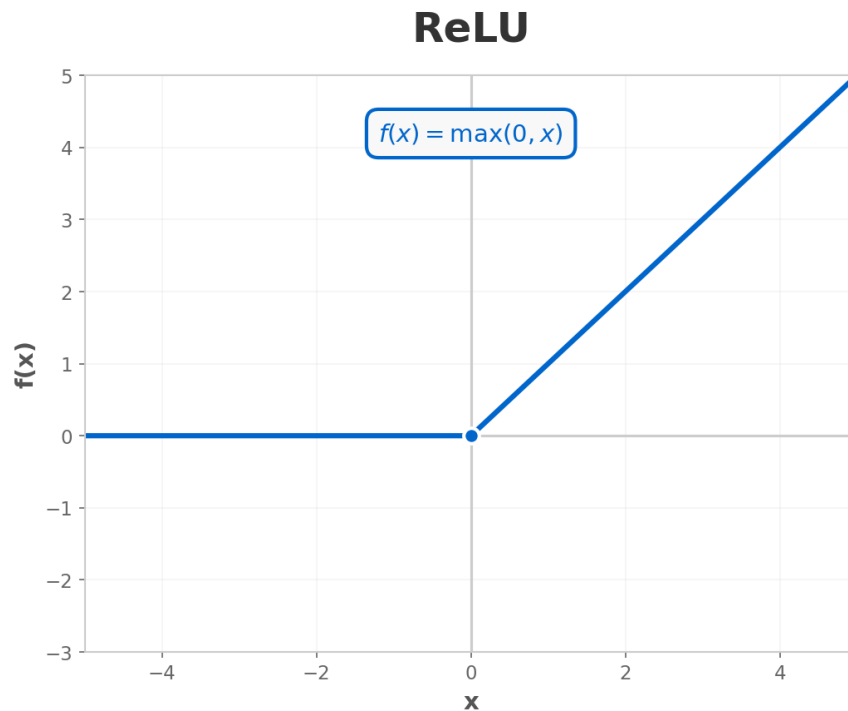
$$\text{Layer 2: } h_2 = \text{activation}(W_2 \cdot h_1 + b_2)$$

Now each layer creates a new non-linear transformation that can't be simplified. This is what enables:

- Image recognition
- Language understanding
- Voice synthesis
- Game playing AI etc

Activation functions add **curves to the math, enabling Deep Learning.**

1. ReLU (Rectified Linear Unit)



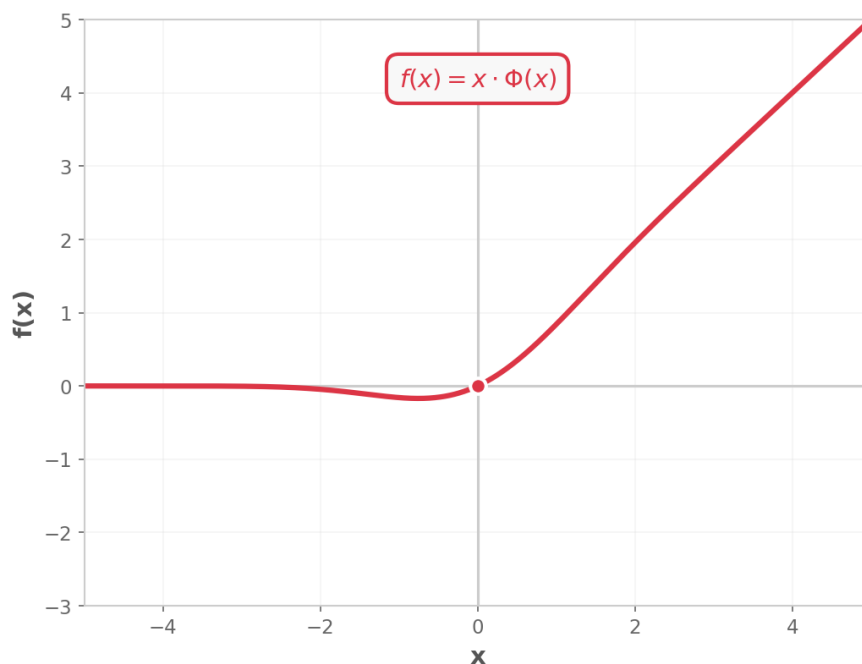
Pros	Cons
Super fast (just max operation)	Dying ReLU - Neurons can permanently die
No vanishing gradient for positive values	Not zero-centered
Sparse activation (efficient)	Unbounded (can explode)

Why it's popular: Simple, fast, works well in most CNNs.

Used in: AlexNet, VGG, ResNet, most CNN architectures

2. GELU (Gaussian Error Linear Unit)

GELU

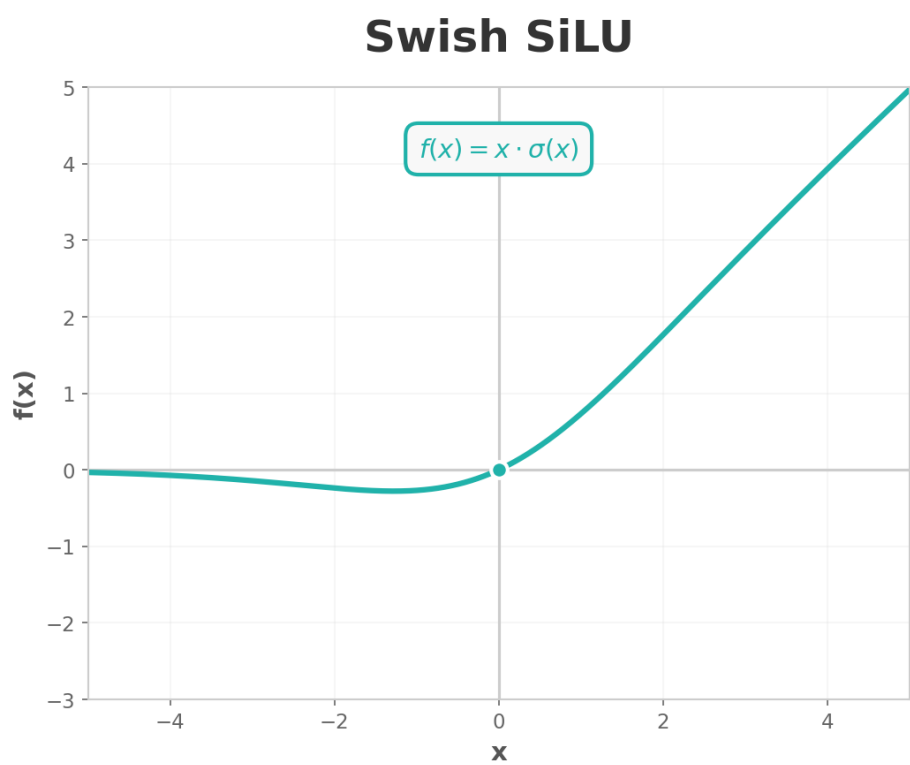


Pros	Cons
Smooth, differentiable everywhere	Computationally expensive
Probabilistic interpretation	Slower than ReLU
No dying Neuron problem	
Better gradient flow	

Why it's better than ReLU: Smoothness helps transformers learn better. Small negative values aren't completely killed.

Used in: GPT-2, GPT-3, GPT-4, BERT, Vision Transformer (ViT)

3. Swish / SiLU (Sigmoid Linear Unit)



Pros	Cons
Smooth & non-monotonic	More compute than ReLU
Self-gating property	Unbounded above
Better gradient flow	
Outperforms ReLU in deep networks	

Why it's better than ReLU: The smooth curve and self-gating allow negative values to contribute slightly, preventing dead neurons.

Used in: EfficientNet, LLaMA 1-4, Mistral, Qwen, MobileNetV3 etc

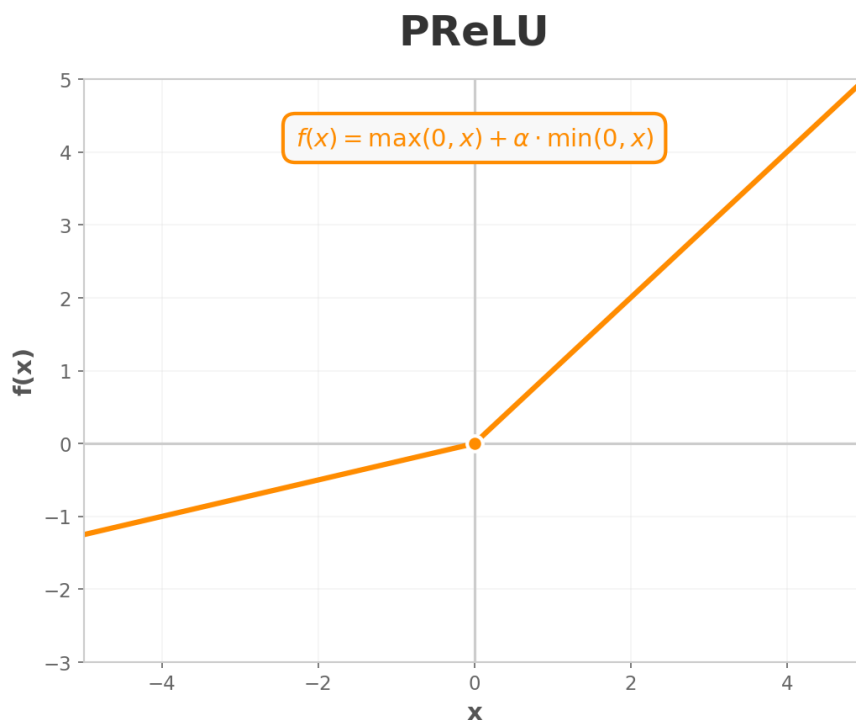
4. SwiGLU (Swish-Gated Linear Unit)

Pros	Cons
State-of-the-art for LLMs	50% more parameters
Gating mechanism adds expressiveness	More memory usage
Better gradient flow	Can't visualize as 2D curve
Outperforms GELU in transformers	

Why it's the 2024-25 king: Combines the smoothness of Swish with gating mechanism. Empirically dominates benchmarks (researchers say it **divine benevolence**).

Used in: LLaMA 2-4, PaLM, Mistral, Mixtral, Qwen 3, DeepSeek-V3

5. PReLU (Parametric ReLU)

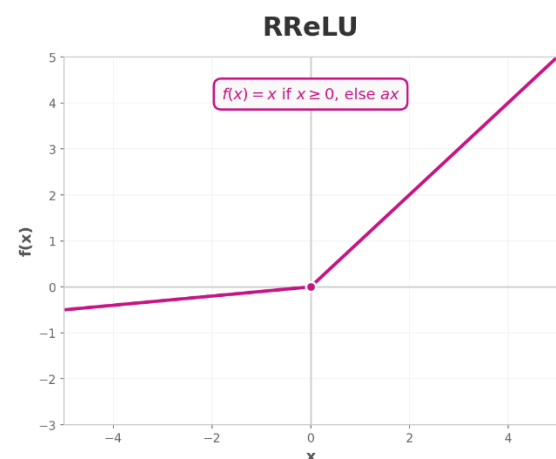
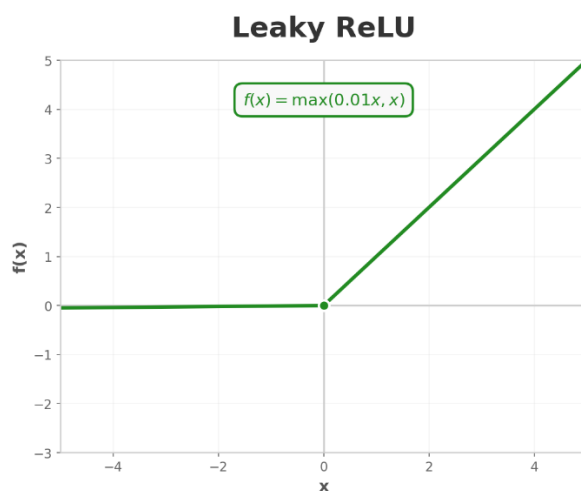


Pros	Cons
Fixes dying ReLU	Extra parameter to learn
Learnable slope	Can overfit on small datasets
Network decides best α	

Why it's better than ReLU: The network learns the optimal slope for negatives instead of using fixed 0.

Used in: ResNets, face recognition models

6. Leaky ReLU / RReLU

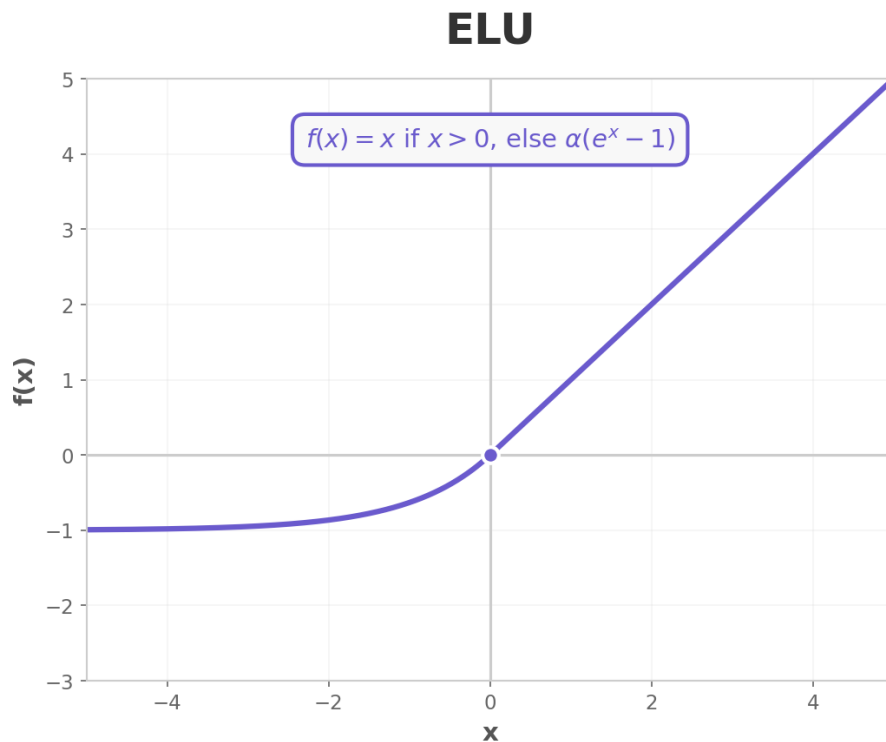


Pros	Cons
Prevents dying ReLU	Fixed α may not be optimal
Simple fix to ReLU	Still not smooth
RReLU adds regularization	

Why it's better than ReLU: Small gradient for negatives keeps neurons alive.

Used in: YOLO variants, GANs

7. ELU (Exponential Linear Unit)



Pros

Smooth for negatives

Outputs closer to zero-mean

Faster convergence than ReLU

Cons

Exponential is slow

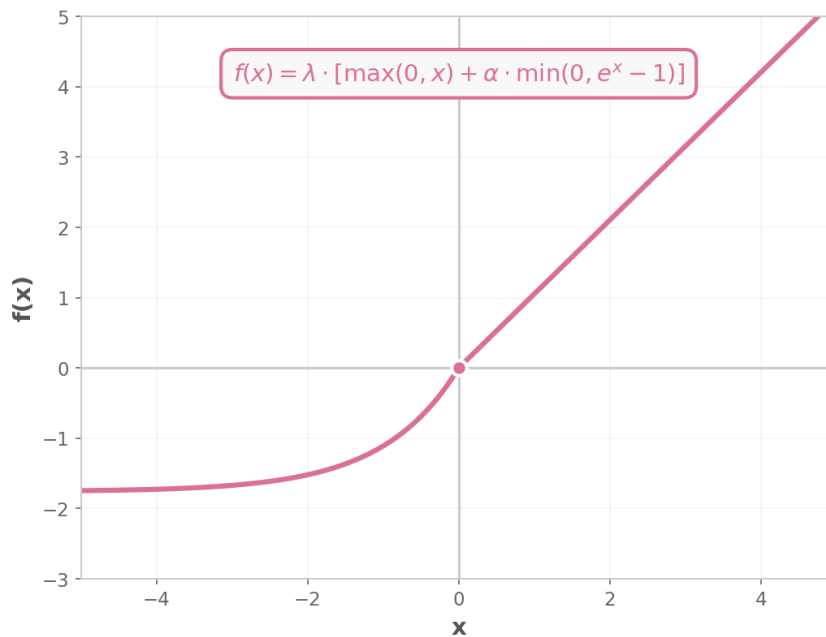
Saturates for very negative values

Why it's better than ReLU: The smooth negative part helps with gradient flow and produces zero-centered outputs.

Used in: Deep networks where batch normalization isn't used

8. SELU (Scaled ELU)

SELU



Pros	Cons
Self-normalizing	Only works with specific initialization
No need for batch norm	Requires LeCun normal init
Great for deep MLPs	Not ideal for CNNs/RNNs

Why it's special: Mathematically designed to maintain mean=0, variance=1 through layers. Built-in normalization.

Used in: Very deep fully-connected networks

9. Mish

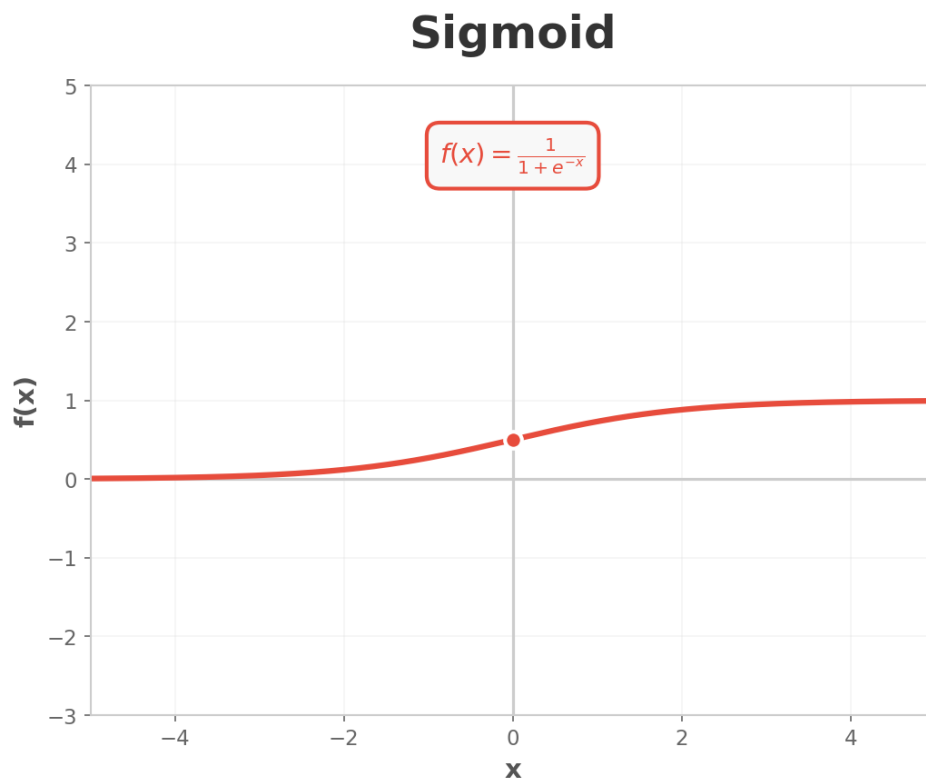


Pros	Cons
Smoother than Swish	Computationally expensive
Unbounded above, bounded below	Complex formula
Self-regularizing	
No dying neurons	

Why it's better than ReLU/Swish: Extra smoothness from $\tanh(\text{softplus})$ provides even better gradient flow in deep networks.

Used in: YOLOv4, YOLOv5, object detection models

10. Sigmoid



Pros

Output bounded (0, 1)

Probabilistic interpretation

Good for output layers

Cons

Vanishing gradient problem

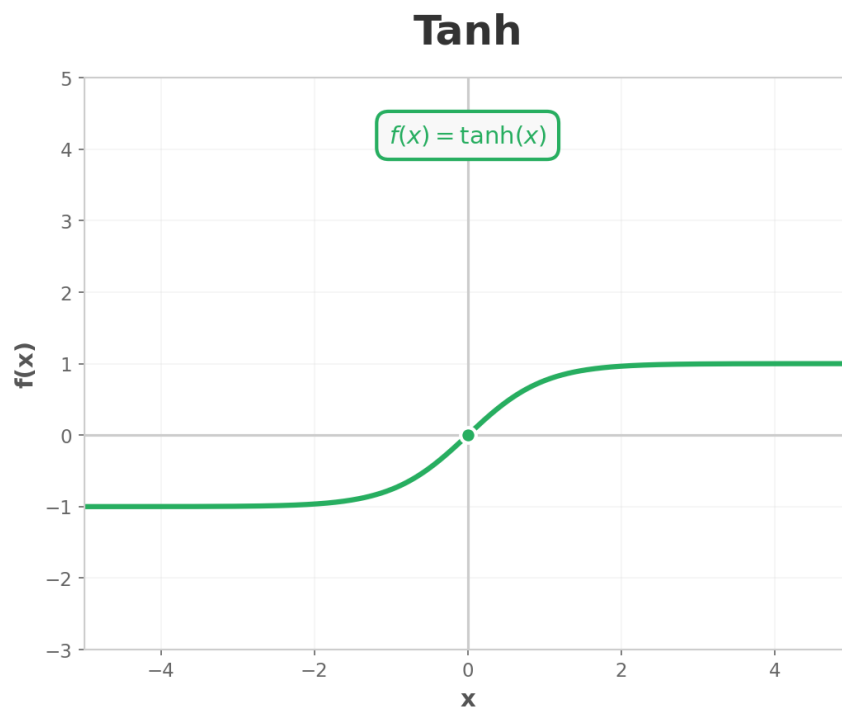
Not zero-centered

Saturates at extremes

Why it's NOT used in hidden layers anymore: Gradients vanish when x is very positive or negative → deep networks can't learn.

Still used for: Binary classification output, gates in LSTMs

11. Tanh

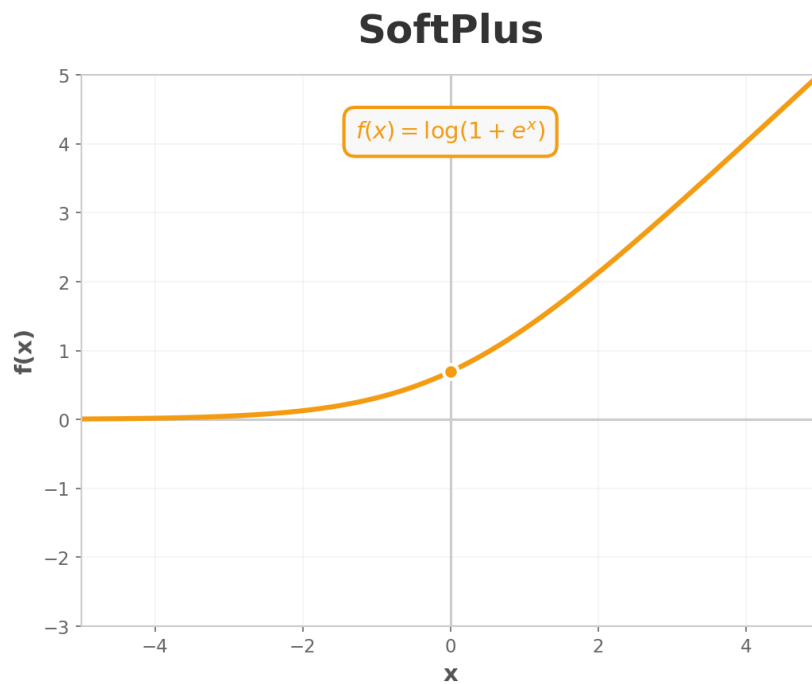


Pros	Cons
Zero-centered (-1 to 1)	Vanishing gradient
Stronger gradients than sigmoid	Saturates at extremes
Good for normalized data	

Why it's better than Sigmoid: Zero-centered outputs help with convergence. But still has vanishing gradient.

Used in: RNNs, LSTMs, GRUs (for gates and states)

12. SoftPlus

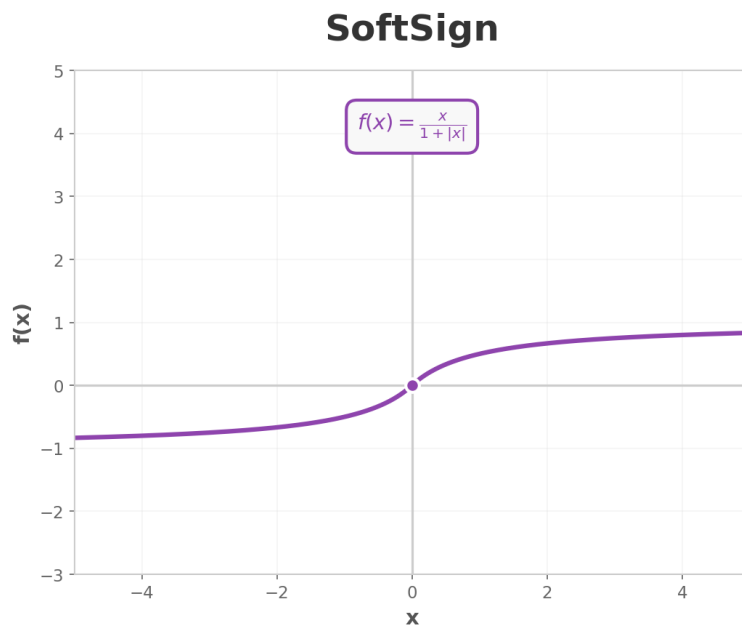


Pros	Cons
Smooth version of ReLU	Never exactly zero
Always positive output	Slower than ReLU
Differentiable everywhere	

Why it exists: Smooth approximation to ReLU when you need continuous derivatives.

Used in: Probabilistic models, variational autoencoders

13. SoftSign



Pros

Smoother than Tanh

Doesn't saturate as fast

Zero-centered

Cons

Slower convergence

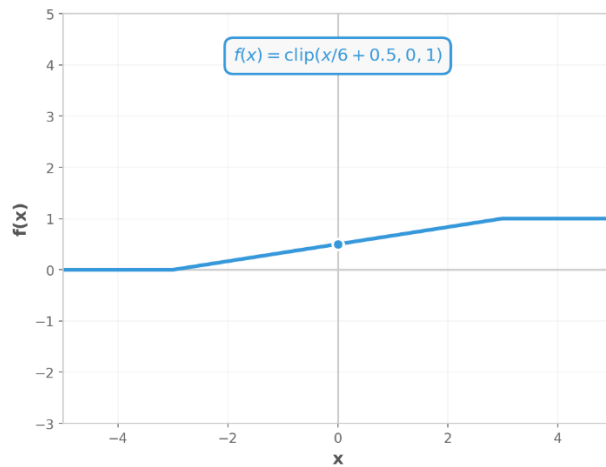
Less commonly used

Why it's better than Tanh: Approaches ± 1 more slowly $\rightarrow$ less gradient vanishing.

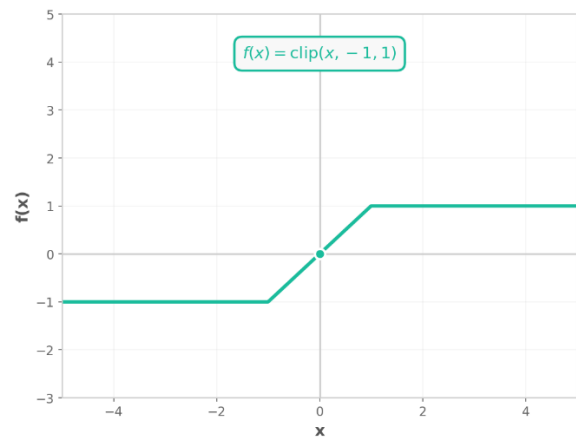
Used in: Some NLP applications, when Tanh saturates too fast

14. Hard Sigmoid / Hard Tanh / HardSwish

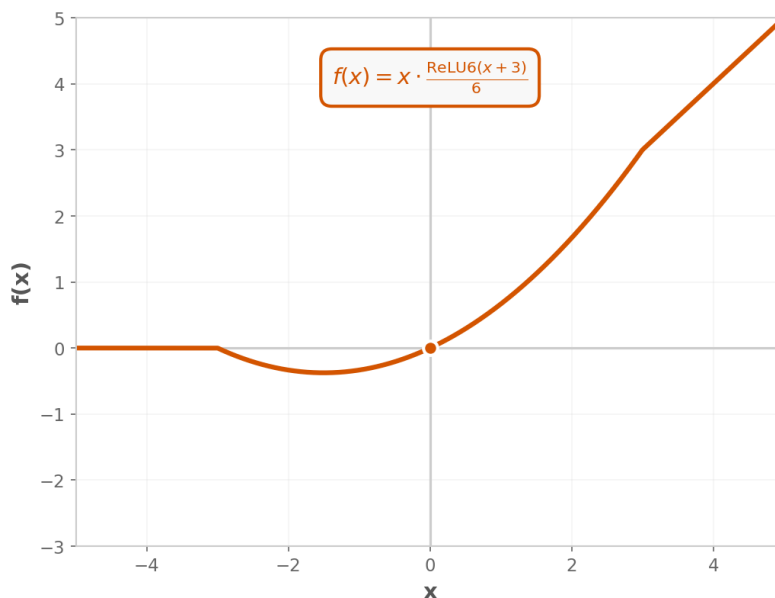
Hard Sigmoid



Hard Tanh



HardSwish

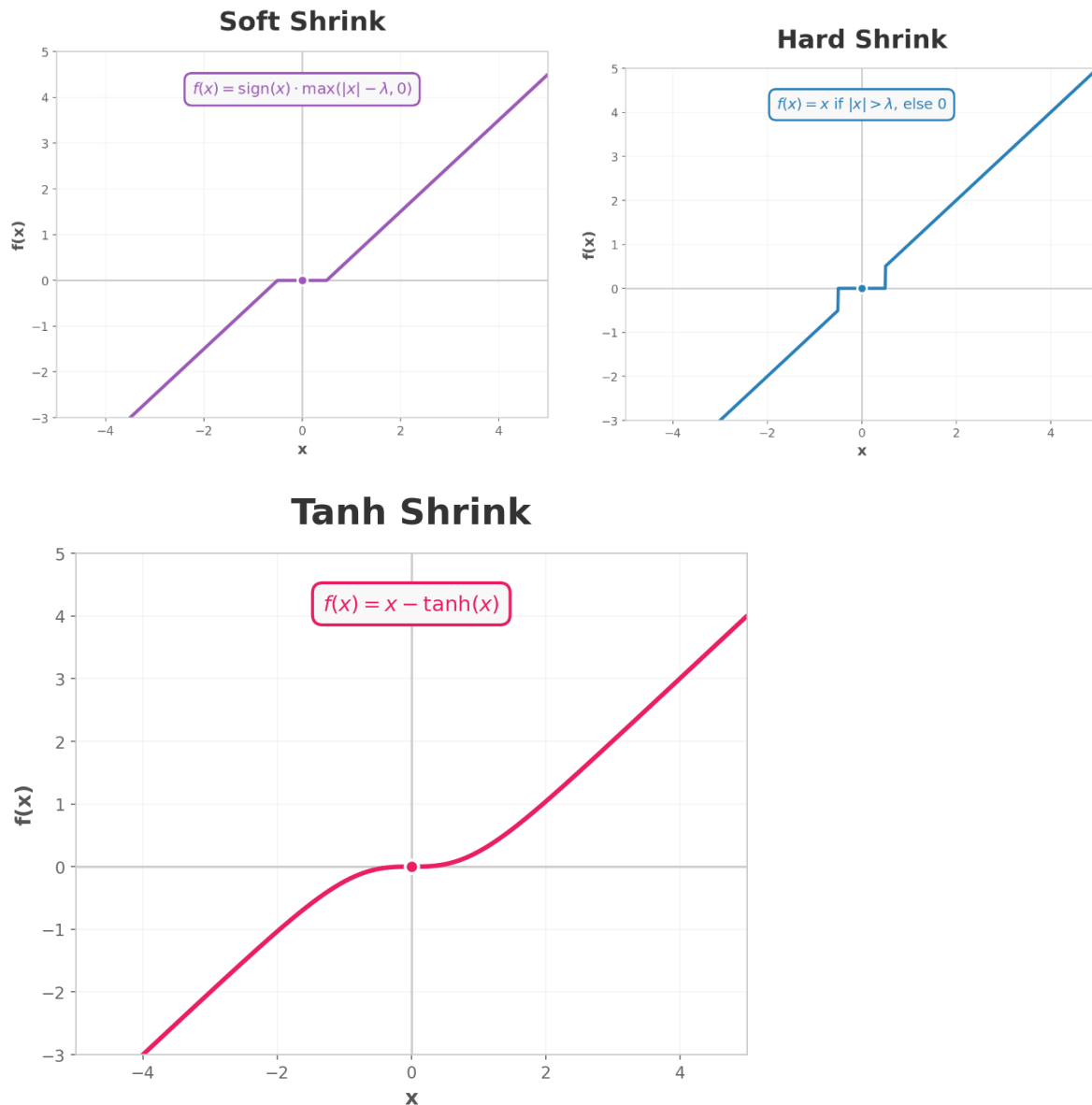


Pros	Cons
Computationally cheap	Not smooth (has kinks)
No exponentials	Less accurate approximation
Great for mobile/edge	

Why they exist: Fast approximations for resource-constrained devices. Trade smoothness for speed.

Used in: MobileNetV3, edge AI, quantized models

15. Shrink Functions (Soft/Hard/Tanh Shrink)



Pros

Sparsity-inducing

Good for denoising

Thresholding behaviour

Cons

Niche applications

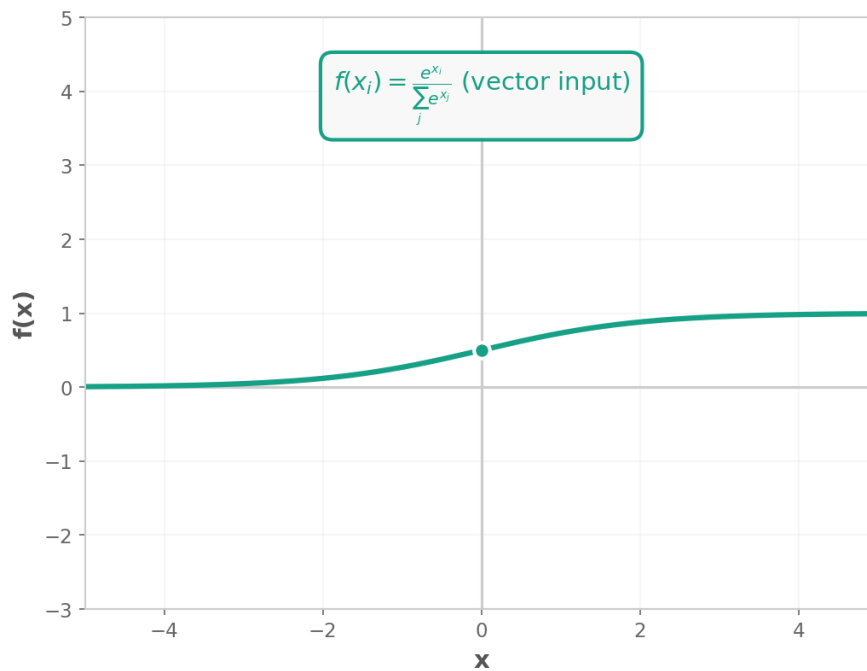
Not mainstream

Why they exist: Push small values to zero → sparse representations, useful in autoencoders and signal processing.

Used in: Wavelet transforms, sparse coding, denoising autoencoders

16. Softmax

Softmax Note



Pros	Cons
Outputs sum to 1	Only for output layer
Probabilistic interpretation	Computationally expensive
Works on vectors	Can't use in hidden layers

Why it's different: Operates on entire vector, not single values. Turns scores into probabilities.

Used in: Classification output layer, LLM next-token prediction (GPT-4, LLaMA, Claude etc)

Ranking: Best for each Use Case

Use Case	Best Choice	Why
CNN hidden layers	ReLU, Mish	Fast, works well
Transformer/LLM FFN	SwiGLU, GELU	Smooth gradients, better performance
RNN/LSTM gates	Sigmoid, Tanh	Bounded outputs needed
Very deep MLPs	SELU	Self-normalizing
Mobile/Edge AI	HardSwish, ReLU	Computationally efficient
Binary classification output	Sigmoid	Outputs probability
Multi-class output	Softmax	Probabilities sum to 1
Regression output	Linear (None)	Unbounded output needed
Object detection	Mish, Leaky ReLU	Better gradient flow
GANs	Leaky ReLU	Prevents dying neurons

Evolution Timeline

1990s: Sigmoid, Tanh (vanishing gradient problems)



2010: ReLU (solved vanishing gradient, but dying neurons)



2015: Leaky ReLU, PReLU, ELU (fixed dying neurons)



2016: SELU (self-normalizing)



2017: Swish/SiLU (smooth, self-gating)



2018: GELU (transformers love it)



2019: Mish (even smoother)



2020: SwiGLU, GeGLU (gated FFN layers)



2024-25: SwiGLU dominates LLMs

Key takeaways

1. **ReLU** is still the default starting point, simple and fast
2. **GELU/SwiGLU** for transformers and LLMs, smooth gradients matter
3. **Sigmoid/Tanh** only for output layers or RNN gates now
4. **Hard variants** for mobile/edge deployment
5. **No single best** - depends on architecture, task and compute budget