Section 4 of 11 · 4 lessons

Activation Functions

The common activations and their derivatives, the vanishing-gradient and dying-ReLU problems, and how to choose activations for hidden and output layers.

4.1 Sigmoid, tanh, ReLU and Friends

4.2 Vanishing Gradients

4.3 Dying ReLU, Leaky ReLU and ELU

4.4 Choosing Activation Functions



Free course with quizzes, progress tracking and Scoopy, an AI tutor:
airesearchlabs.si/learn/dl

Deep Learning Foundations · Edition November 2026 · © 2026 AI Research Labs. Free for personal study.

Where this section fits

Every lesson is a step in building a model. These are the steps this section teaches; the Big picture PDF and the online map show how they connect to the rest.

4

BUILDING AND TRAINING A NETWORK · STEP 4

Choose the activations

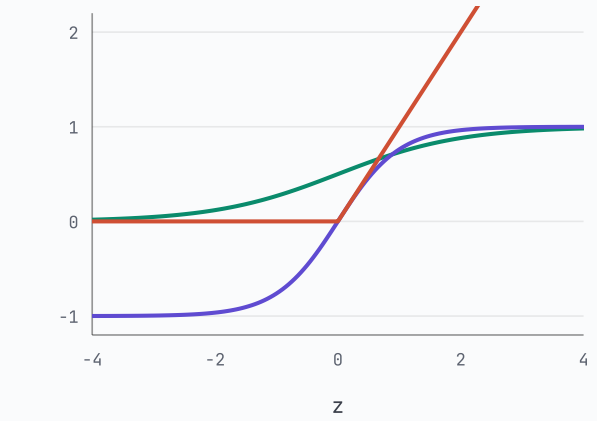
Which non-linearity in hidden layers, and which at the output?

Lessons here: 4.1, 4.2, 4.3, 4.4 · feeds into Compute the gradients

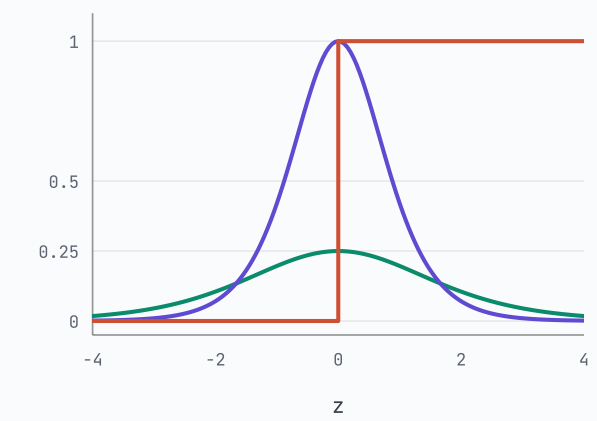
Sigmoid, tanh, ReLU and Friends

The common activation functions side by side: their formulas, ranges and derivatives, where each is used, and why the sign function cannot be trained by gradient descent.

The functions



Sigmoid squashes to (0, 1), tanh to (−1, 1); ReLU is zero for negative inputs and the identity for positive ones.



Derivatives. Sigmoid and tanh fade to zero for large |z| (saturation). ReLU's derivative is exactly 1 on the positive side.

ACTIVATION	FORMULA	RANGE	DERIVATIVE	TYPICAL USE AND LIMITATION
Sigmoid	$\sigma(z) = \frac{1}{1+e^{-z}}$	(0, 1)	$\sigma(1-\sigma)$, at most 0.25	Binary output probability; saturates
tanh	$\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	(−1, 1)	$1 - \tanh^2 z$, at most 1	Zero-centred hidden units; still saturates
ReLU	$\max(0, z)$	[0, ∞)	0 for $z < 0$, 1 for $z > 0$	Default for hidden layers; can "die"
Sign	$+1$ if $z > 0$, -1 if $z < 0$	{−1, +1}	0 almost everywhere	Threshold constructions; untrainable by gradients
Linear	z	(−∞, ∞)	1	Regression output
Softmax	$o_j = \frac{e^{s_j}}{\sum_k e^{s_k}}$	(0, 1), sums to 1	Section 6	Multi-class output probabilities

Two derivatives to know by heart

Sigmoid. $\sigma'(z) = \sigma(z)\,\big(1 - \sigma(z)\big)$. It peaks at $z = 0$, where $\sigma = 0.5$ and $\sigma' = 0.25$. At $z = 2$: $\sigma = 0.881$, $\sigma' = 0.105$. At $z = 5$: $\sigma = 0.993$, $\sigma' = 0.0066$. Large inputs **saturate** the unit and kill its gradient.

tanh. $\tanh'(z) = 1 - \tanh^2(z)$, which peaks at 1. tanh is a rescaled sigmoid, $\tanh z = 2\sigma(2z) - 1$, and being zero-centred it often trains a little better in hidden layers.

Why the sign function can't be trained

Away from $z = 0$, a small change in the weights doesn't change a sign unit's output at all, so its derivative is **zero almost everywhere** (and undefined at 0). Gradient descent gets no signal about which way to move. That's why trainable networks use smooth or piecewise-linear activations instead.

Common doubts

FIRST PASS **How do I remember the sigmoid derivative?**

Write $\sigma = (1 + e^{-z})^{-1}$ and differentiate: $\sigma' = e^{-z}(1 + e^{-z})^{-2}$. Since $e^{-z}/(1 + e^{-z}) = 1 - \sigma$, this is $\sigma(1 - \sigma)$. In backpropagation you already have σ from the forward pass, so the derivative costs one multiplication: $a(1 - a)$.

GOING DEEPER **How do I derive σ′ = σ(1 − σ)? Why isn't the derivative of e^(−z) something like −e^(−2z)?**

Write $\sigma(z) = (1 + e^{-z})^{-1}$ and use the chain rule, outside then inside:

1. **Outside, power rule:** $\frac{d}{du}u^{-1} = -u^{-2}$, so you get $-(1 + e^{-z})^{-2}$.
2. **Inside, exponential rule:** $\frac{d}{dz}(1 + e^{-z}) = -e^{-z}$. The 1 is a constant, so it vanishes. The exponent stays $-z$; you only multiply by its derivative, -1.
3. **Multiply:** $-(1 + e^{-z})^{-2} \cdot (-e^{-z}) = \dfrac{e^{-z}}{(1 + e^{-z})^2} = \dfrac{1}{1 + e^{-z}} \cdot \dfrac{e^{-z}}{1 + e^{-z}} = \sigma\,(1 - \sigma)$.

Thumb rule: "The power rule lowers the power. The exponential rule keeps the exponent and multiplies by its derivative." So $\frac{d}{dz}e^{-z} = -e^{-z}$, never $-e^{-2z}$.

Cheatsheet

Sigmoid

$\sigma(z) = 1/(1 + e^{-z}) \in (0, 1)$; $\sigma' = \sigma(1 - \sigma) \leq 0.25$ (at $z = 0$). Saturates for large $|z|$.

tanh and ReLU

$\tanh \in (-1, 1)$, $\tanh' = 1 - \tanh^2 \leq 1$. ReLU $= \max(0, z)$, derivative 0 or 1.

Sign is untrainable

Its derivative is 0 almost everywhere, so gradient descent gets no signal. Use sigmoid, tanh or ReLU.

Online version with the interactive quiz: airesearchlabs.si/learn/dl/activation-functions/sigmoid-tanh-relu-and-friends

Vanishing Gradients

Why early layers of a deep sigmoid network barely learn: backpropagation multiplies one activation derivative per layer, and sigmoid derivatives are at most 0.25. The numbers, the mechanism and the fixes.

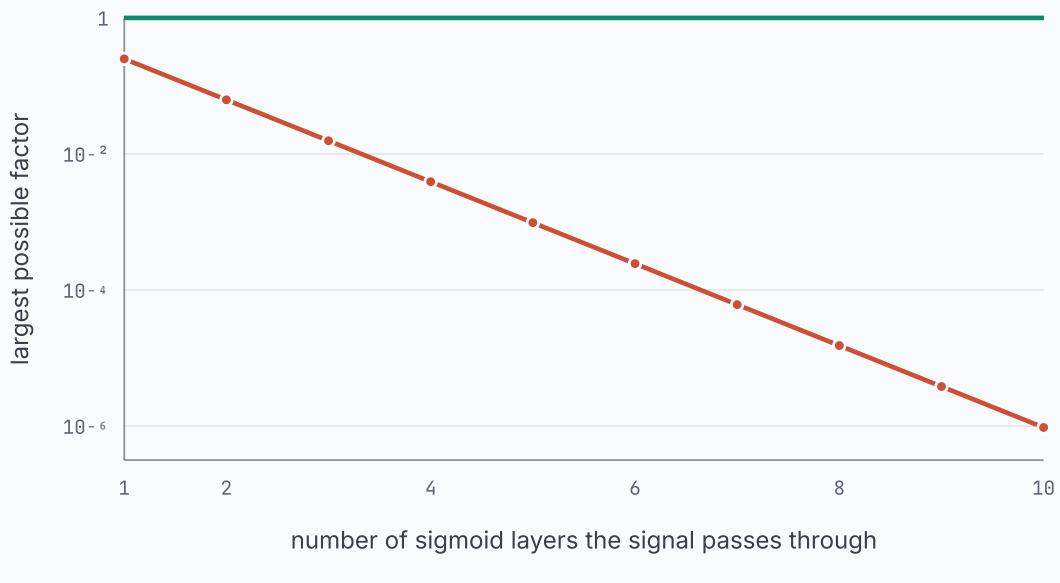
The mechanism

Backpropagation (Section 7) carries the error signal from the output back to earlier layers. At each layer it is multiplied by a transposed weight matrix and by the **activation derivative**. Written compactly, with $D_k = \operatorname{diag}\big(f'(z^{(k)})\big)$:

$$\delta^{(k)} = D_k\, W_{k+1}^{\top} D_{k+1} W_{k+2}^{\top} \cdots \delta^{(L)}$$

For sigmoid units, every factor f' is **at most 0.25**. Ten such factors give at most

$$0.25^{10} = 4^{-10} \approx 9.5 \times 10^{-7}$$



The activation-derivative part of the gradient shrinks geometrically with depth for sigmoid units (log scale). Actual gradients also depend on the weights.

So the output layers keep learning while the first layers receive almost no update. **Saturation** makes it worse: a unit with a large $|z|$ has σ' far below 0.25.

Remedies

- **ReLU** in hidden layers: its derivative is exactly 1 for active units, so the activation doesn't shrink the signal.

- **Careful initialisation** (for example He initialisation for ReLU) and **normalised inputs**, so units start out of saturation.
- Architectural fixes for very deep networks, such as batch normalisation and residual connections (beyond this course).

EXAM PHRASING

"The sigmoid derivative is at most 0.25, so backpropagating through many sigmoid layers multiplies many small factors and the gradients reaching early layers shrink towards zero. ReLU's derivative is 1 for positive inputs, so it doesn't shrink the signal."

Common doubts

GOING DEEPER **Can gradients also become too large?**

Yes. If the weights are large, the product of weight matrices can grow with depth instead of shrinking: **exploding gradients**, which make training unstable. Sensible initialisation keeps the products near 1 in both directions, and gradient clipping is a common safety net.

FIRST PASS **If the earliest layers get tiny gradients, what about the later layers?**

The later layers are fine. A layer's gradient is a product of one slope per layer **between it and the output**, so layers near the output multiply only a few small factors, and layers near the input multiply many.

With sigmoid slopes of at most 0.25: one layer back, the gradient is scaled by 0.25; five layers back, by $0.25^5 \approx 0.001$. So the last layers keep learning while the first layers barely move from their random starting weights, and the network behaves as if it were much shallower.

ReLU helps because its slope is exactly 1 for positive inputs, so the product doesn't shrink.

Cheatsheet

Vanishing gradients

Backprop multiplies one f' per layer. Sigmoid $f' \leq 0.25$, so 10 layers give $\leq 0.25^{10} \approx 10^{-6}$. ReLU ($f' = 1$ when active) avoids the shrinkage.

Online version with the interactive quiz: airesearchlabs.si/learn/dl/activation-functions/vanishing-gradients

Dying ReLU, Leaky ReLU and ELU

ReLU solves vanishing gradients for active units but creates a new failure: a unit stuck on the negative side gets zero gradient forever. How it happens and the remedies.

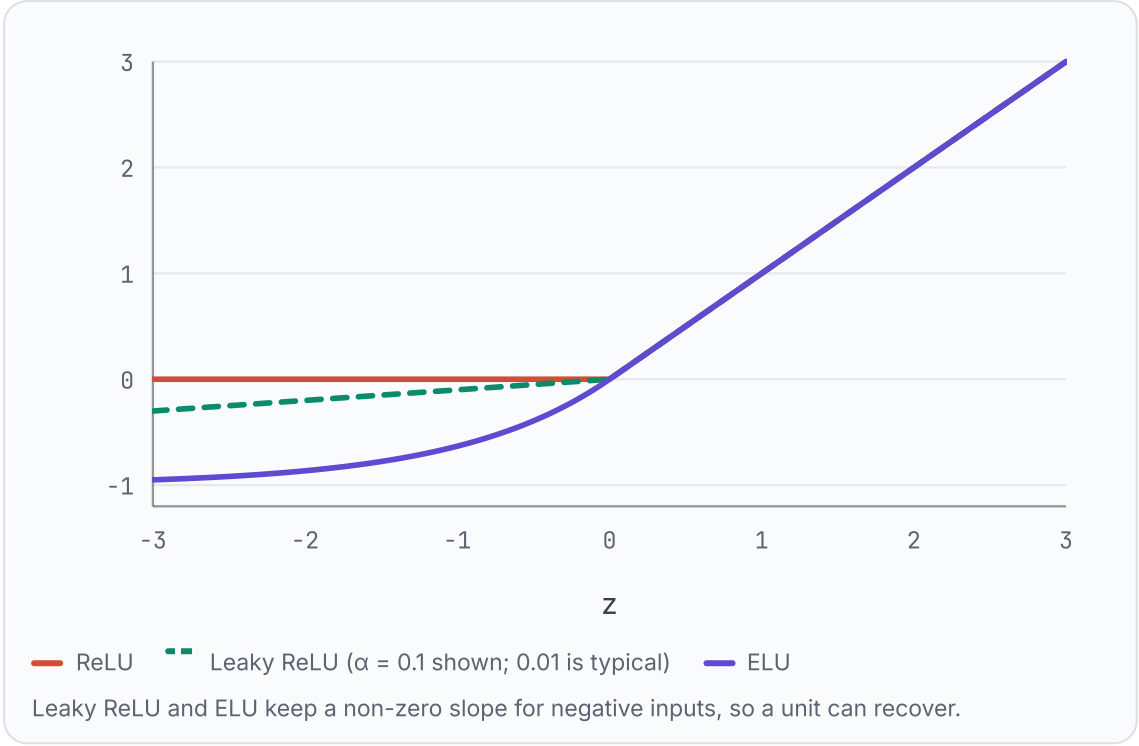
Why ReLU is the default

ReLU, $\max(0, z)$, has derivative 1 on its positive side, so active units pass the error signal back undiminished. It is also very cheap to compute and gives **sparse** activations (many units output exactly 0). It's the recommended default for hidden layers of deep networks.

Dying ReLU

If a unit's pre-activation is negative for **every** training example, its output is always 0 and so is its gradient. Gradient descent never changes its incoming weights, so it stays dead. A large learning-rate step that pushes the bias far negative is a typical cause.

Remedies



Leaky ReLU and ELU keep a non-zero slope for negative inputs, so a unit can recover.

- **Leaky ReLU:** $f(z) = z$ for $z \geq 0$ and αz for $z < 0$, with $\alpha \approx 0.01$. The small negative slope keeps a gradient flowing.
- **ELU:** a smooth curve $e^z - 1$ on the negative side, also with a non-zero gradient.
- A **smaller learning rate** and **He initialisation** make dying less likely in the first place.

DON'T CONFUSE THE TWO PROBLEMS

Vanishing gradients: many small derivative factors multiplied across layers (typical of sigmoid). **Dying ReLU**: one unit stuck at zero output and zero gradient. Different mechanisms, different fixes.

Common doubts

FIRST PASS **Is a dead ReLU unit harmful or just useless?**

Mostly useless: it always outputs 0, so it contributes nothing and wastes capacity. If many units die, the network effectively becomes much smaller than designed and may underfit.

FIRST PASS **What's the difference between "dead neurons" and "not zero-centred"?**

They are two different problems of two different activations.

	DEAD NEURONS	NOT ZERO-CENTRED
Which activation	**ReLU**	**Sigmoid** (outputs 0 to 1)
What happens	$z < 0$ for every input, so the output is 0 and the slope is 0. The neuron never updates again.	Outputs are always positive, so all weight gradients into the next layer share one sign. Updates zig-zag and training is slower.
How bad	The neuron is useless for good	Training is slower, but it still works
Fix	Leaky ReLU or ELU (a small slope for $z < 0$)	tanh (outputs −1 to 1)

Thumb rule: "ReLU can die. Sigmoid is lopsided, and it also saturates, which causes vanishing gradients."

Cheatsheet

Leaky ReLU

$f(z) = z$ for $z \geq 0$, αz for $z < 0$ ($\alpha \approx 0.01$). Keeps a gradient on the negative side; fixes dying ReLU.

Vanishing vs dying

Vanishing: small f' factors multiply across layers (sigmoid). Dying ReLU: a unit stuck at $z < 0$, output and gradient 0.

Online version with the interactive quiz: airesearchlabs.si/learn/dl/activation-functions/dying-relu-leaky-relu-and-elu

Choosing Activation Functions

Which activation to use where: ReLU for deep hidden layers, sigmoid for shallow networks and binary outputs, linear for regression outputs, softmax for multi-class outputs.

Hidden layers

- **Deep networks:** ReLU (or Leaky ReLU / ELU if units die).
- **Shallow networks** (one or two hidden layers): sigmoid or tanh work fine; vanishing gradients only bite with depth.
- Never purely linear: the layers would collapse into one (Section 3).

Output layer: decided by the task

TASK	OUTPUT ACTIVATION	WHY
Regression (a number)	Linear	Predictions can be any real value; a sigmoid would trap them in (0, 1)
Binary classification	One sigmoid unit	Outputs $P(y = 1 \mid x)$
Multi-class classification	Softmax over the classes	Positive outputs that sum to 1, one probability per class

In PyTorch, the sigmoid or softmax is usually folded into the loss function instead of the model (Section 6.4).

> RULE OF THUMB
>
> Hidden layers: ReLU. Output: whatever makes the outputs mean what the task needs: a real number, a probability, or a probability per class.

Common doubts

FIRST PASS **Can I use a sigmoid output for regression if my targets are between 0 and 1?**

You can, if the targets genuinely live in (0, 1), such as proportions. But for general regression targets (prices, temperatures) a sigmoid caps every prediction between 0 and 1, which is wrong, and it also saturates. Use a linear output and standardise the targets instead.

FIRST PASS **Why might the output neuron have no activation function?**

For regression the target can be any real number, like a price or a temperature. ReLU would forbid negative outputs, and sigmoid or tanh would squash the output into a fixed range. The identity lets the output take any value, and the hidden layers still supply the non-linearity.

FIRST PASS **Do all neurons in a layer use the same activation function?**

In practice, yes: a layer uses one activation for all its neurons, because that is simpler and works well. Nothing in the maths forbids mixing. Different **layers** often differ, though: ReLU in the hidden layers and identity, sigmoid or softmax at the output.

Cheatsheet

Output activation by task

Regression: linear. Binary: one sigmoid. Multi-class: softmax. Hidden layers: ReLU (deep) or sigmoid/tanh (shallow).

Online version with the interactive quiz: airesearchlabs.si/learn/dl/activation-functions/choosing-activation-functions