Activation Functions

@S.Domain

April 2026

Contents

1 Introduction **2**

2 1. The Sigmoid Function **2**

3 2. Tanh (Hyperbolic Tangent) **2**

4 3. ReLU (Rectified Linear Unit) **2**

5 4. Leaky ReLU **3**

6 5. Softmax **3**

7 Quick Summary **3**

Introduction

Activation functions are what make a neural network actually learn. Without them, a network is just a bunch of math equations that only produce a straight line. By adding these functions, we introduce non-linearity, which lets the network figure out complicated stuff like identifying images or processing language.

1. The Sigmoid Function

Sigmoid was the go-to for a long time. It squashes any input into a range between 0 and 1.

Math

Function: $\sigma(x) = \frac{1}{1+e^{-x}}$
Derivative: $\sigma'(x) = \sigma(x)(1 - \sigma(x))$

Why we use it: It works well if you need to predict a probability for a binary choice (like Yes/No). **The problem:** The *Vanishing Gradient*. If your number is really big or really small, the curve gets super flat. Because the derivative is tiny, the signal gets lost when you try to train deep networks, and the model basically stops learning.

2. Tanh (Hyperbolic Tangent)

Think of Tanh as Sigmoid's stronger sibling. It maps inputs to a range between -1 and 1.

Math

Function: $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Derivative: $\tanh'(x) = 1 - \tanh^2(x)$

Why we use it: It's centered around zero. This keeps the data balanced, which usually helps the network train faster than it would with Sigmoid. **The problem:** Like Sigmoid, it still has the vanishing gradient issue if your inputs get too large.

3. ReLU (Rectified Linear Unit)

It's what basically everyone uses in hidden layers now because it's fast and simple.

Math

Function: $f(x) = \max(0, x)$
Derivative: $f'(x) = 1$ if $x > 0$, else 0

Why we use it: It's super fast because it doesn't need exponents. It also doesn't flatten out for positive numbers, so the gradient keeps flowing, making deep networks easier to train. **The problem:** *The Dying ReLU*. If a neuron gets stuck with negative inputs, it outputs 0 and effectively dies, meaning it stops learning completely.

4. Leaky ReLU

This is the fix for the Dying ReLU problem. It lets a little bit of the negative signal through instead of just killing it.

Math

Function: $f(x) = \max(\alpha x, x)$
Derivative: $f'(x) = 1$ if $x > 0$, else α (where $\alpha \approx 0.01$)

Why we use it: By keeping a tiny slope for negative numbers, you prevent neurons from dying. You get all the speed of ReLU without the risk of losing neurons during training.

5. Softmax

You only use this at the very end of your network. It's for when you're classifying something into one of many groups.

Math

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

Why we use it: It turns a bunch of random scores into a list of probabilities that add up to exactly 100%. It's perfect for seeing which class the model is most confident about.

Quick Summary

Function	Range	Use Case
Sigmoid	(0, 1)	Binary Classification
Tanh	(-1, 1)	Hidden layers
ReLU	[0, ∞)	Default for hidden layers
Leaky ReLU	($-\infty$, ∞)	When neurons are dying
Softmax	(0, 1)	Final output layer