Linear Algebra for Data Science Project

Neural Networks and Activation Functions

Derek Regier

December 4, 2025

1. What Are Activation Functions?

What do Activation Functions do?

Activation functions serve as the neural network's decision makers. They introduce nonlinearity, allowing networks to learn complex patterns. Without them, even a 100-layer network would behave like a single linear regression model. Additionally, the shape of an activation function's derivative directly impacts how quickly and effectively a network learns during training.

Derivatives, Backpropagation, and Gradient Flow

During training, neural networks learn through a process called backpropagation, which computes how much each weight should change to reduce the loss. This process relies on the chain rule of calculus, which requires multiplying derivatives as the gradient flows backward through each layer. Specifically, the gradient contribution from a neuron depends on the derivative of its activation function. This is where activation function choice becomes critical: if the derivative is small, the gradient contribution diminishes, and the layer learns slowly or not at all.

The *vanishing gradient problem* occurs when these derivatives are multiplied together across many layers. Consider sigmoid activation, which has a maximum derivative of 0.25. In a deep network, the gradient flowing backward must pass through multiple sigmoid derivatives: $0.25 \times 0.25 \times 0.25 \times \ldots$ For a 10-layer network, this results in $(0.25)^{10} \approx 9.3 \times 10^{-7}$—essentially zero. When gradients become this small, early layers in the network cannot learn effectively, as their weight updates become negligible. This limitation prevented the training of truly deep networks for decades. Activation functions like ReLU, which maintain a constant derivative of 1.0 for positive inputs, avoid this exponential decay: $1.0 \times 1.0 \times 1.0 \times \ldots = 1.0$. This property enabled the training of much deeper networks.

2. Comparative Analysis of Activation Functions

Sigmoid

The sigmoid function is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$, mapping all inputs to the range $(0, 1)$. Its derivative is $\sigma'(x) = \sigma(x)(1 - \sigma(x))$, which reaches a maximum value of 0.25 at $x = 0$.

The sigmoid has always been the standard for neural network development due to its smooth and differential nature. However, it suffers from severe vanishing gradients in deep networks. Since the maximum derivative is only 0.25, repeated multiplication during backpropagation through multiple layers causes gradients to decay exponentially.

Sigmoid is often used for binary classification problems.

Tanh

The hyperbolic tangent function is defined as $\tanh(x)$, mapping inputs to the range $(-1, 1)$. Its derivative is $\tanh'(x) = 1 - \tanh^2(x)$, which reaches a maximum value of 1.0 at $x = 0$.

Tanh is similar to sigmoid but with two important improvements: it is zero-centered (outputs range from -1 to 1 rather than 0 to 1), and its maximum derivative is 1.0 instead of 0.25. The zero-centered property helps with convergence, and the larger maximum derivative slows the vanishing gradient problem. However, tanh still suffers from vanishing gradients at extreme input values, since the derivative approaches 0 as $x \to \pm\infty$, which can be limiting for learning. It can be better than sigmoid at times.

ReLU

The Rectified Linear Unit is defined as $\text{ReLU}(x) = \max(0, x)$, outputting the input directly for positive values and 0 for negative values. Its derivative is simple: $\text{ReLU}'(x) = 1$ if $x > 0$, else 0.

ReLU's most significant advantage is its constant derivative of 1.0 for all positive inputs. This means that during backpropagation, gradients do not decay through layers: $1.0 \times 1.0 \times 1.0 \times \ldots = 1.0$. This property enabled the training of much deeper networks. Additionally, ReLU is computationally efficient as it requires only a comparison operation. Thus, it is often faster than sigmoid and tanh.

However, ReLU has a significant drawback: the *dying ReLU problem*. Neurons that consistently receive negative inputs will output 0, resulting in zero gradients. Once a neuron "dies" in this way, it stops learning entirely and provides no useful information to the network. Despite this limitation, ReLU remains the most widely used activation function for hidden layers in modern deep networks.

Leaky ReLU

Leaky ReLU is a modification of ReLU designed to address the dying neuron problem. It is defined as $\text{Leaky ReLU}(x) = x$ if $x > 0$, else αx (typically $\alpha = 0.01$). The derivative is $\text{Leaky ReLU}'(x) = 1$ if $x > 0$, else α.

By allowing a small, non-zero gradient for negative inputs, Leaky ReLU prevents neurons from dying completely. Even when a neuron receives negative inputs and outputs a small negative value, it can still receive gradient signals ($\alpha = 0.01$) during backpropagation, allowing it to recover and learn. This maintains the advantages of ReLU while eliminating the dying neuron issue.

The trade-off is minimal: the additional hyperparameter α requires tuning, though the default value of 0.01 works well in most cases.

ELU

The Exponential Linear Unit is defined as $\mathrm{ELU}(x) = x$ if $x > 0$, else $\alpha(e^x - 1)$ (typically $\alpha = 1.0$). The derivative is $\mathrm{ELU}'(x) = 1$ if $x > 0$, else αe^x.

ELU combines the advantages of both ReLU-like functions and smooth activation functions like sigmoid or tanh. For positive inputs, it behaves like ReLU with a constant derivative of 1.0, avoiding vanishing gradients. For negative inputs, it uses a smooth exponential curve that approaches $-\alpha$, providing nearly zero-centered outputs similar to tanh. This smooth curve on the negative side means the function is continuously differentiable everywhere (unlike ReLU and Leaky ReLU, which have a kink at $x = 0$), potentially leading to better gradient flow and faster convergence.

However, ELU requires computing the exponential function for negative inputs, which is slower than the simple operations in ReLU and Leaky ReLU.

Comparison of Functions and Derivatives

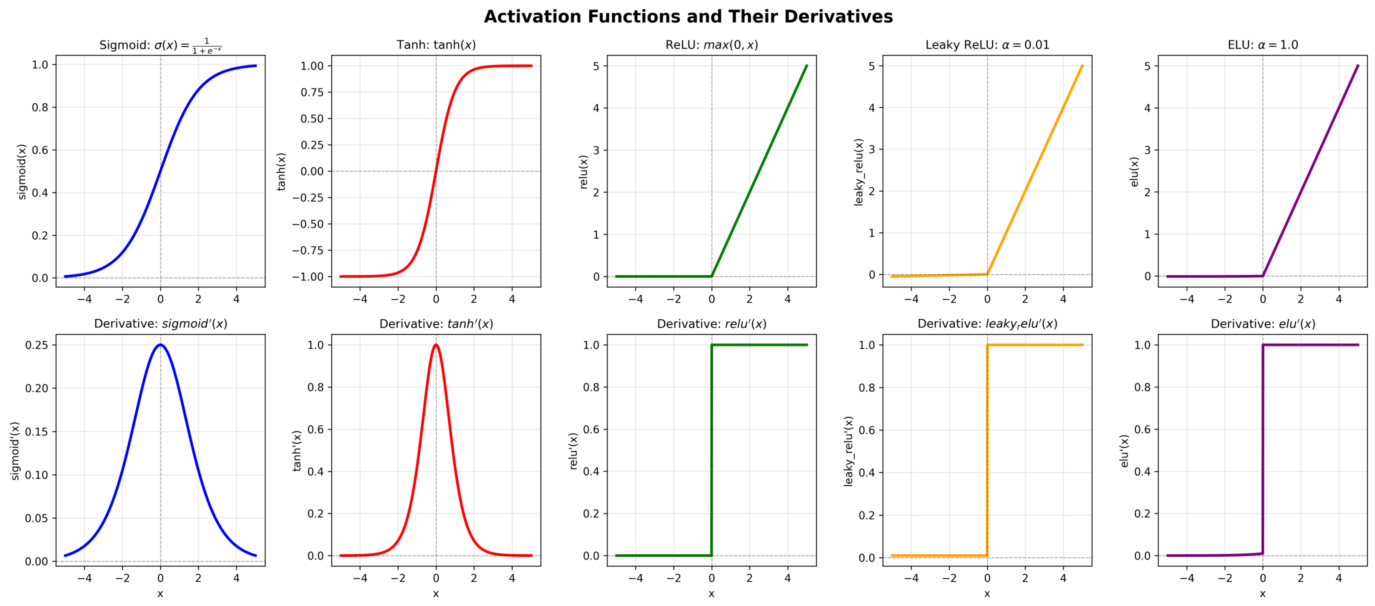

Figure 1: Activation functions and their derivatives. The top row shows the forward pass of each function, bottom row shows the corresponding derivative.

In Figure 1, we have from left to right: Sigmoid, Tanh, ReLU, Leaky ReLU, and ELU. Here we show the full comparison of each activation function and its derivatives.

Notice that the derivative shapes of the ReLU-like functions are similar. The Leaky ReLU has barely noticeable differences due to extremely small values for negative inputs; ELU, on the other hand, has a smooth curve for negative inputs. In general, for small networks, there will be little difference between the ReLU-like functions. However, in deep learning, the choice of activation function is crucial. These derivatives play a huge role in how a model learns, so knowing what they are good at and not good at is vital for building a good neural network for a given task.

3. Performative Comparison in an MLP

Single-Layer Performance

To quickly test the overall training performance, we set up a linearly separable task, which means a straight line can separate the two classes of data. All activation functions get 100% accuracy, which is expected. What is different is the overall training performance when they converge.

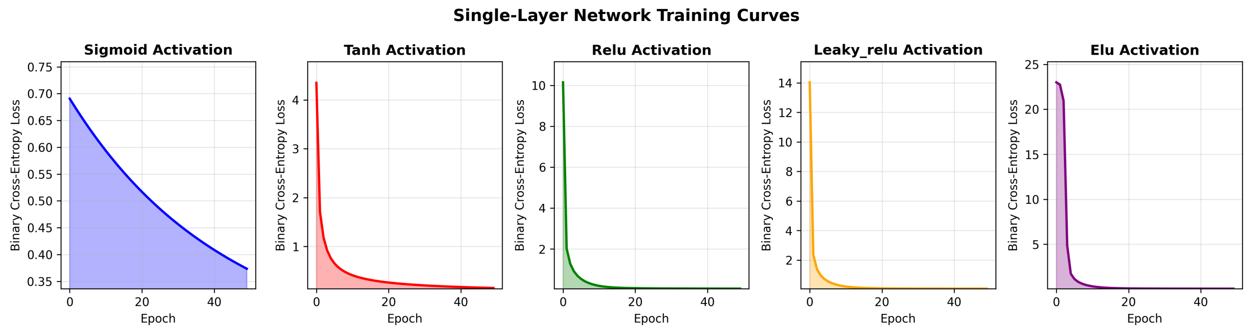

Figure 2: Activation function single layer performance on binary classification for linear separable data

In Figure 2, we can see that the ReLU-like functions train much quicker. Tanh is somewhere in the middle, with sigmoid having the slowest learning. So whilst sigmoid easily accomplishes the task equally for accuracy, it is not good for speed in comparison. There are other tasks where quicker learning is more pronounced.

Multi-Layer Perception Networks

Things can get a bit more complicated with data that is not linearly separable. To solve this, we can set up a Multi-Layer Perception Network or MLP. We set up a simple MLP with a 2-neuron input layer, 10-neuron hidden layer with activation functions of sigmoid, tanh, or ReLU, and a 1-neuron output layer with a sigmoid function. Below is an example of MLP architecture:

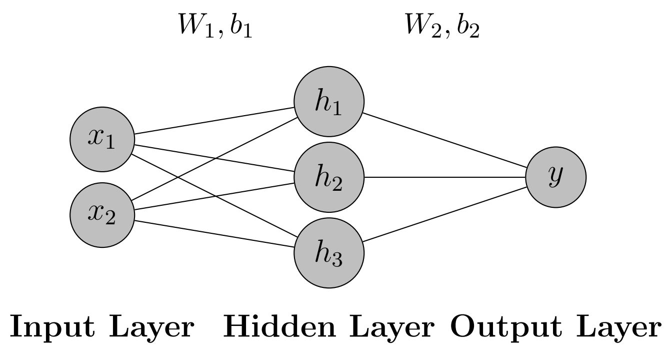

Figure 3: Multi-layer perceptron (MLP) architecture with 2 inputs, 3 hidden units, and 1 output. Data flows left to right through weight matrices W_1 and W_2. Each layer applies an activation function except the input layer.

ELU and Leaky ReLU perform almost the same as ReLU for this task; they only show better performance in deeper learning tasks. Hence, the main experiment uses just ReLU. MLP tests were done over 1000 epochs. An epoch is one complete pass through the entire training dataset. During each epoch, the network sees every training example exactly once, computes predictions, calculates loss, and updates weights through backpropagation.

Hidden Activation	**Final Loss**	**Accuracy**
Sigmoid	0.691948	0.5550
Tanh	0.600268	0.7775
ReLU	0.409428	0.9570

Table 1: Multi-layer network performance with different hidden layer activations on the concentric circles dataset.

Table 1 demonstrates the capabilities for more complicated tasks. ReLU significantly outperforms Sigmoid and Tanh, achieving 95.70% accuracy compared to 55.50% and 77.75%, respectively. This shows where different combinations excel at different tasks. The MLP requires a sigmoid for binary classification, but still needs a better function performance-wise for the hidden layer.

4. Conclusion

Neural networks are still a valuable direction of research today and form the foundations of machine learning, deep learning, and AI like ChatGPT. Activation functions are a key part of that research and are used across a variety of tasks. Whilst sigmoid is typically outdated, it still remains the golden standard that will get the job done and is still used in combination with other layers and functions to perform certain tasks.