Multilayer Perceptrons

December 11, 2025

Contents

Chapter 1

The Basics of Multilayer Perceptrons

A **multilayer perceptron** (otherwise known as a *neural network*) is a set of nodes called **neurons**, each containing an activation – a number (usually) between 0 and 1 – which designates how active that specific neuron is.

This is loosely analogous to how our brains work, where each of our neurons can fire to send an electrical signal which, somewhere down the chain, might move our arm or let us perceive the smell of some food.

The image people usually associate with neural networks is the following:

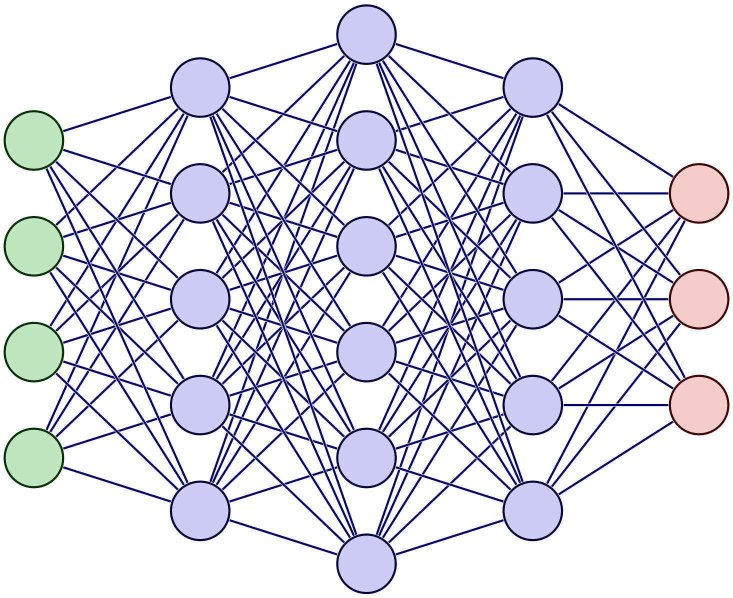

FIGURE 1.1: A simple neural network.

Figure 1.1 shows the structure of a neural network. At its core, a neural network has an **input layer** (colored in green), analogous to some sensory input such as our eyes capturing the light around us, an **output layer** (colored in red), analogous to the output of our vision, which is a colored image of our environment, and then some "hidden layers," which are basically the meat of the network and encode everything about how the inputs and corresponding outputs are related to one another.

The actual math behind these networks is surprisingly simple. We'll focus on a single connection for now:

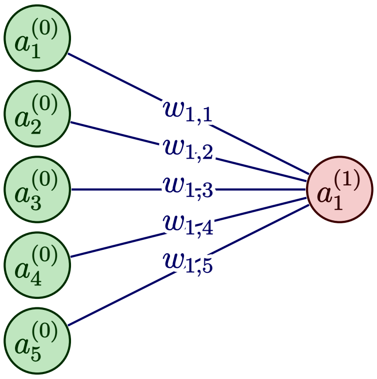

FIGURE 1.2: The activation of two neurons and the weight between them.

Let's decypher these symbols one-by-one:

- $a_n^{(0)}$: The activations of the nodes within the first layer.
- $w_{1,n}$: The **weights** of each of the connections. Essentially, this encodes how strongly connected the neurons are (so how much they influence each-other).
- $a_1^{(1)}$: The output activation.

In order to determine the output activation $a_1^{(1)}$ from the input activations $a_i^{(0)}$, we simply take a weighted sum whose weights are the coefficients $w_{1,i}$; that is,

$$a_1^{(1)} = w_{1,1}a_1^{(0)} + w_{1,2}a_2^{(0)} + \cdots + w_{1,5}a_5^{(0)}$$

Usually we also add in what's called a **bias** term, which is just a value that we add to this sum. We can interpret the bias as the "sensitivity" of the neuron. That means that the whole expression in the general case (so, for n neurons in the input layer) is

$$a_1^{(1)} = b_1^{(0)} + \sum_{i=1}^{n} w_{1,i}a_i^{(0)}$$

There's only one issue: Remember we said that the activations must be between 0 and 1, but with a weighted sum like this, we can essentially get any number. What we need now is some kind of way to squish this value into the range $[0, 1]$. The most common function for this purpose is the **sigmoid** function:

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

The graph of this function is shown below:

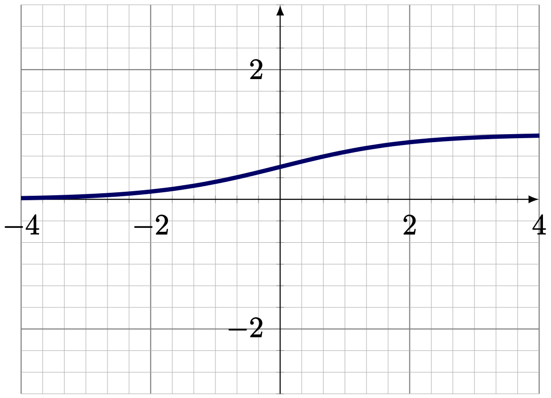

FIGURE 1.3: The graph of $\sigma(x)$.

From Figure 1.3, we can see how the function approaches 0 when x is negative, and approaches 1 when x is positive. This is exactly what we needed! Putting all of this together, we have the following expression for $a_1^{(1)}$:

$$a_1^{(1)} = \sigma\left(b_1^{(0)} + \sum_{i=1}^{n} w_{1,i} a_i^{(0)}\right)$$

Remember, this is just for the activation of a *single* neuron, so for a whole layer, we need to do this many times. There is a way we can simplify this, but we'll need to consider the more general case in order to do so.

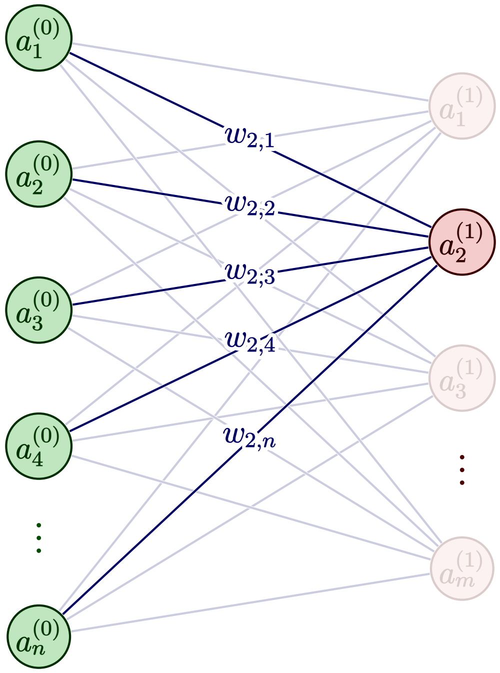

FIGURE 1.4: The general case of the connections between two layers.

The key insight here is that we can simplify this process immensely with the tools of linear algebra.

In particular, we can group all the activations in the first layer (layer 0) into a column vector:

$$\mathbf{a}^{(0)} = \begin{bmatrix} a_1^{(0)} \\ a_2^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix}$$

We can do the same thing for the output activations:

$$\mathbf{a}^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_m^{(1)} \end{bmatrix}$$

Similarly, we can group the biases up into a vector as well:

$$\mathbf{b}^{(0)} = \begin{bmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_m^{(0)} \end{bmatrix}$$

Finally, the weights simply become a matrix which we multiply with $\mathbf{a}^{(0)}$:

$$\mathbf{W}^{(0)} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \cdots & w_{2,n} \\ w_{3,1} & w_{3,2} & w_{3,3} & \cdots & w_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & w_{m,3} & \cdots & w_{m,n} \end{bmatrix}$$

Now, if we imply that the function $\sigma(x)$ applies component-wise to vectors and matrices, we get the following incredibly neat formula for the activations in layer $L+1$ given those in layer L:

$$\mathbf{a}^{(L+1)} = \sigma\Big(\mathbf{b}^{(L)} + \mathbf{W}^{(L)}\mathbf{a}^{(L)}\Big)$$

Other Activation Functions

The sigmoid is not the only function we can use to remap our values. In fact, there are infinitely many functions we can use. Another common choice is the tanh function (which remaps values between -1 and 1). Beyond that, we sometimes don't want to remap between finite values at all. This is why other (arguably more common) activation functions include the ReLU, SiLU, and, more recently, GELU functions.

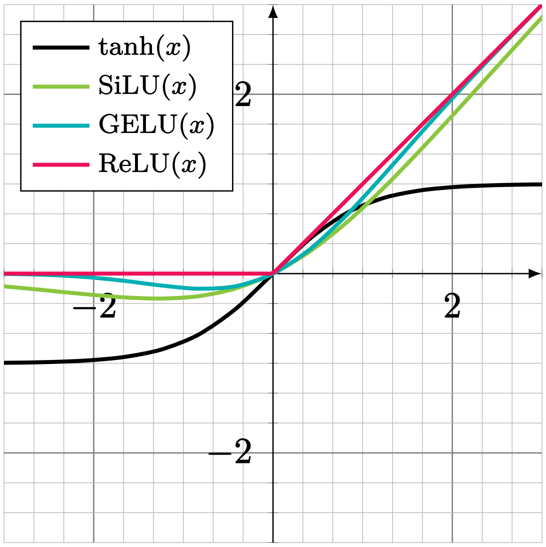

FIGURE 1.5: The graphs of the different activation functions.

ReLU is very common because it's very strict for model behavior; the activation simply gets clamped to zero if it's negative. The expressions for these functions are the following:

$$\begin{aligned}\mathrm{ReLU}(x) &= \max(x, 0)\\ \mathrm{SiLU}(x) &= x\sigma(x)\\ \mathrm{GELU}(x) &\approx 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right) \text{ [1]}\end{aligned}$$

The expression for GELU is simply an approximation because it is actually derived from the cumulative distribution function of the normal distribution, which has no elementary form. In fact, the *true* expression defining GELU(x) is

$$\mathrm{GELU}(x) = x\Phi(x) = x\frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right],$$

where

$$\mathrm{erf}(x) = \int_0^x e^{-s^2} ds$$

is the error function.

Bibliography

[1] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2023. arXiv: 1606.08415 [cs.LG]. URL: https://arxiv.org/abs/1606.08415.