Introduction to neural networks

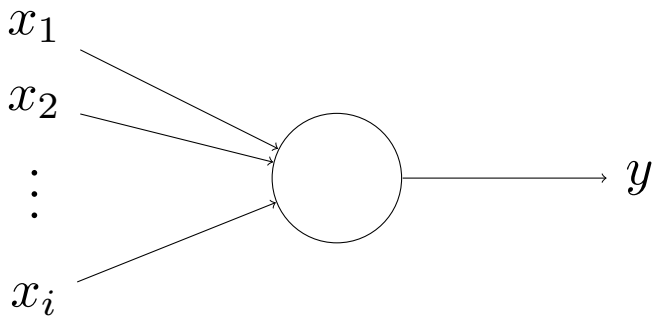
Peter Rowlett

Last week we saw how to generate text using a probability transition model. With 'generative AI', text is typically generated using a neural network. There are lots of tasks that neural networks can be used for, and various 'off the shelf'/'black box' tools that can be used to create these.

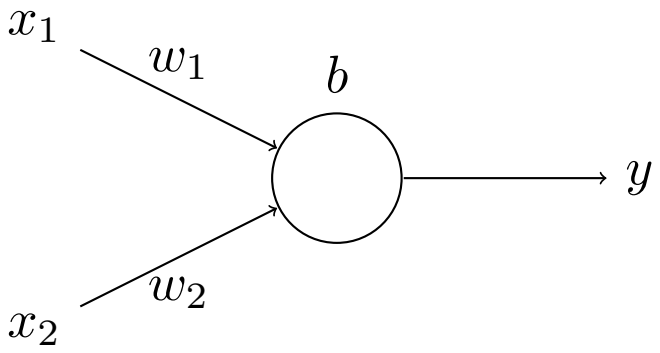
Here we aim to understand some of the basics of how these models work, so instead of using an established tool we will work from first principles through building an artificial neuron and training it to complete a simple classification task.

1 Neurons

Your brain is made of neurons which basically take in electrical impulses, process them and send them on again. Artifical neural networks are modelled on this. An artificial neuron takes some inputs, processes them and outputs the result. Here is an example of a simple neuron with i inputs, $x_1, x_2, \ldots, x_i$, and one output, y.



Say each *input* x_k is a number. As it is passed to the neuron, it is associated with a *weight*, w_j. The neuron also has associated with it a *bias*, b. This gives us the following simple model of an artifical neuron.



The neuron processes the inputs, the weights and the bias. One way we can combine these for

the network above with two inputs, two weights and one bias is

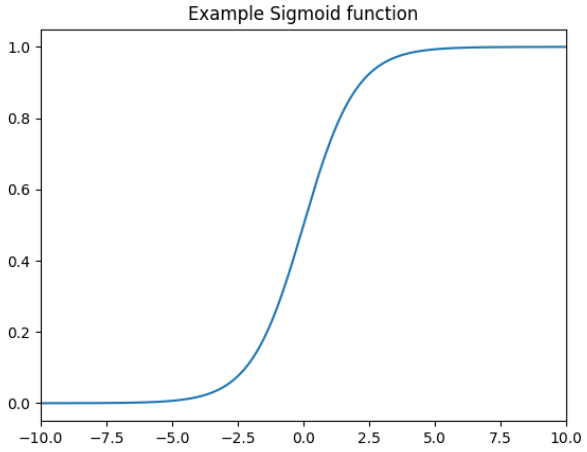
$$Y = \begin{bmatrix} x_1 & x_2 & 1 \\ x_3 & x_4 & 1 \\ \vdots & \vdots & \vdots \\ x_{n-1} & x_n & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix}.$$

A biological neuron either activates or doesn't. For an artificial neuron, we could simply say that if the value of Y meets some threshold then we output 1, otherwise we output 0. However, this does not give the best results. It is better to use an *activation function* that maps our output onto the range 0-1.

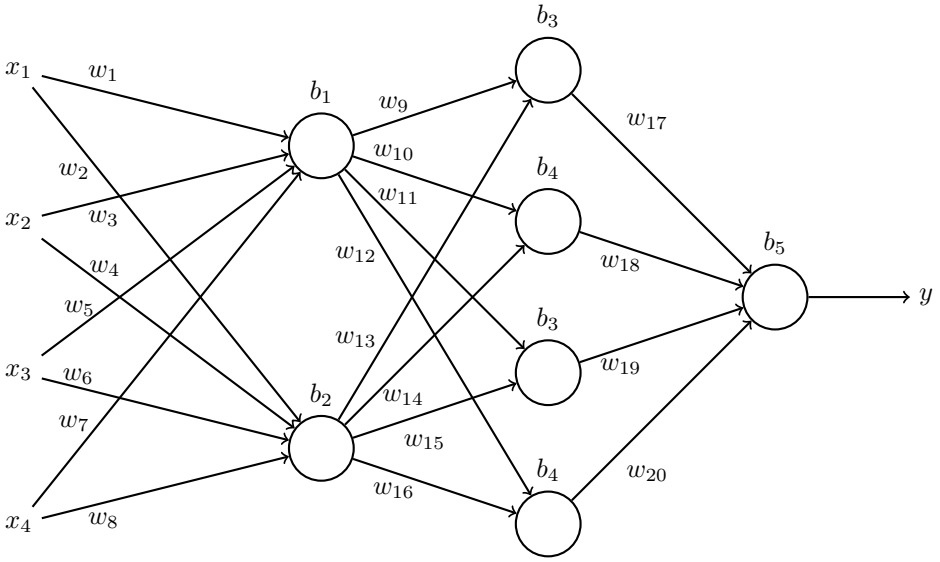
It is common to use the logistic function, called sigmoid, as the activation function.

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

$\sigma(\alpha)$ maps α onto the range 0-1 in a smoother way that just stepping between 0 and 1 and is effective in the learning step we will discuss later.



Neural networks can become much more complicated. For example, here is a network with seven neurons, each of which would process its weighted inputs with the bias in the same way.



Note that this is still quite simple for a neural network.

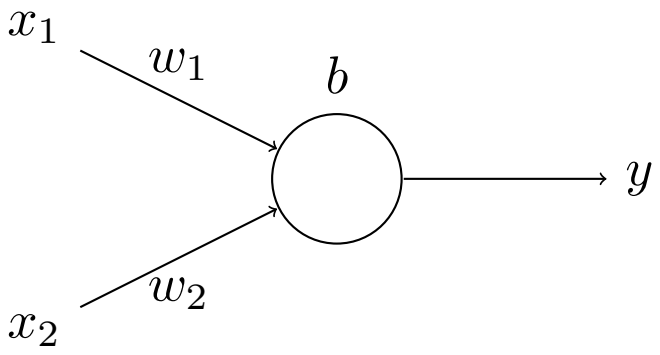
The neurons that accept inputs are called the *input layer*. The ones that deliver the output are the *output layer*. Layers of neurons in between are called *hidden* layers, because they don't interact with the world outside the network.

2 How learning happens

The neural network learns by adjusting the values of the weights and biases. This means the output of each neuron depends to a greater or lesser extent on its inputs, and for more complicated networks means that the final output represents a complex interaction of the inputs, being combined in various different ways.

We 'train' the network by providing some training data for which we know the expected output. Once the network is trained, it is ready to try handling real inputs for which we do not know the output in advance.

For example, consider this simple network of one neuron which we saw above.



To train the network, first we set the weights and bias to random values. Say we set $w_1 = 0.3$, $w_2 = 0.25$, $b = 0.1$.

Then we offer the network a set of training data.

$$X = \begin{bmatrix} 2 & 3 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = 1.45.$$

Now for our matrix of outputs we can calculate the activation function values.

$$\sigma(1.45) = \frac{1}{1 + \mathrm{e}^{-1.45}} = 0.810.$$

Once the network has calculated an output it can compare with a known output, we can update the network using the *backpropogation* algorithm.

First we calculate the error. Because it will be convenient later, we define this as half of the square of the difference between what we are expecting the output to be (the *target*) and the calculated output from our network.

$$E = \frac{1}{2} \left(\text{target} - \text{output} \right)^2 .$$

We use the square of the difference to avoid issues with positive/negative signs.

We can pick values of w_1, w_2 and b to minimise E for these inputs $x_1 = 2$ and $x_2 = 3$ with expected output $t = 1$. However, we are looking to optimise the network across a whole set of training data, and the values we come up with may not work for the next item in our training data. Instead, we seek to nudge the weights and bias each time they meet a piece of training data,

so that after processing a large set of training data they will give a good approximation for all values they might encounter.

We can think of the error E as a function acting on the weights and bias, so $E(w_1, w_2, b)$. We are looking to minimise this error across a set of training data.

To minimise E, we use a method called gradient descent, an iterative algorithm for locating a local minimum. To do this, we take repeated steps in the opposite direction of the gradient of a function at the current point. The size of the steps taken is important because if we move too quickly we might jump past a local minimum without identifying it, but if we move too slowly we might not get very far from our current position.

In order to minimise E, we must consider its gradient with respect to each of the weights and bias. So for example, if we consider w_i, we would be interested in adjusting these along the direction of the partial derivative of E with respect to w_i. To apply gradient descent, we update each w_i by changing its value to

$$w_i - \eta \frac{\partial E}{\partial w_i} x_i$$

where η controls the size of the steps and is called the *learning rate*.

Here is where the advantage of using the sigmoid function comes into play: its derivative is easily expressed in terms of its output. First, consider $1 - \sigma(x)$:

$$\begin{aligned} 1 - \sigma(x) &= 1 - \frac{1}{1 + e^{-x}} \\ &= \frac{1 + e^{-x}}{1 + e^{-x}} - \frac{1}{1 + e^{-x}} \\ &= \frac{1 + e^{-x} - 1}{1 + e^{-x}} \\ &= \frac{e^{-x}}{1 + e^{-x}}. \end{aligned}$$

Now consider the derivative

$$\frac{\mathrm{d}\sigma(x)}{\mathrm{d}x} = \frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{1}{1 + e^{-x}}\right).$$

First, let $u = 1 + e^{-x}$ so $\sigma(u) = u^{-1}$. Then

$$\frac{\mathrm{d}u}{\mathrm{d}x} = -e^{-x}$$

and

$$\frac{\mathrm{d}\sigma}{\mathrm{d}u} = -u^{-2}.$$

Now we use the chain rule,

$$\begin{aligned} \frac{\mathrm{d}\sigma}{\mathrm{d}x} &= \frac{\mathrm{d}\sigma}{\mathrm{d}u}\frac{\mathrm{d}u}{\mathrm{d}x} \\ &= -u^{-2} \cdot -e^{-x} \\ &= e^{-x}\left(1 + e^{-x}\right)^{-2} \\ &= \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} \\ &= \sigma\left(1 - \sigma\right). \end{aligned}$$

So we can write $\frac{\mathrm{d}\sigma}{\mathrm{d}x} = \sigma(1-\sigma)$.
To find the derivative of the the error with respect to one of its weights w_i, we find

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial E}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial w_i} \\ &= \frac{\partial E}{\partial \sigma} \cdot \sigma\left(1-\sigma\right).\end{aligned}$$

Since

$$E = \frac{1}{2}\left(t-\sigma\right)^2$$

we have

$$\begin{aligned}\frac{\partial E}{\partial \sigma} &= \frac{1}{2}\frac{\partial}{\partial \sigma}\left(t-\sigma\right)^2 \\ &= \frac{1}{2} \cdot -2\left(t-\sigma\right) \\ &= -\left(t-\sigma\right).\end{aligned}$$

So

$$\frac{\partial E}{\partial w_i} = -\left(t-\sigma\right)\sigma\left(1-\sigma\right).$$

So we adjust the weights using

$$w_i = w_i + \eta \times (\text{target} - \text{output}) \times \text{output} \times (1 - \text{output}) \times x_i.$$

and adjust the bias using

$$b = b + \eta \times (\text{target} - \text{output}) \times \text{output} \times (1 - \text{output}).$$

η is called the *learning rate*, and might for example be set to 0.1.
For example, we had $x_1 = 2$, $x_2 = 3$, $\begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.25 \\ 0.1 \end{bmatrix}$, $\sigma = 0.810$ and $t = 1$. So

$$\begin{aligned} w_1 &= 0.3 + 0.1 \times (1 - 0.810) \times 0.810 \times (1 - 0.810) \times 2 \\ &= 0.3058; \\ w_2 &= 0.25 + 0.1 \times (1 - 0.810) \times 0.810 \times (1 - 0.810) \times 3 \\ &= 0.2588; \\ b &= 0.1 + 0.1 \times (1 - 0.810) \times 0.810 \times (1 - 0.810) \\ &= 0.1029. \end{aligned}$$

Now we would move onto the next item of training data. The same set of training data may be used multiple times, called epochs.