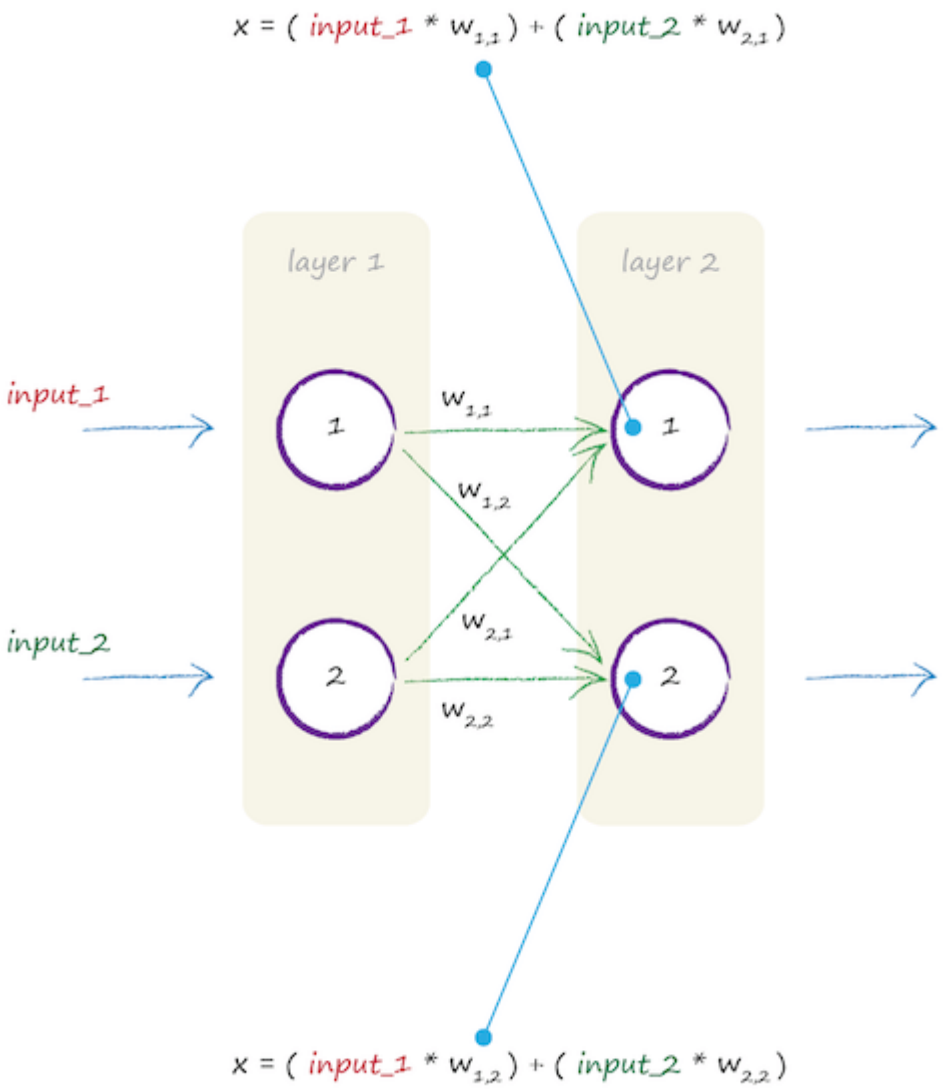
Calculating Inputs for Internal Layers

As we know, the inputs to internal nodes are calculated from the output generated by the previous nodes. In this lesson, we will try to calculate the inputs to an internal layer using previous layer's outputs.

In some guides, you'll see this kind of matrix multiplication called a *dot product* or an *inner product*. There are actually different kinds of multiplication possible for matrices, such as a cross product, but the dot product is the one we want here. Why have we gone down what looks like a rabbit hole of dreaded matrix multiplication and distasteful algebra? There is a very good reason ... hang in there! Look what happens if we replace the letters with words that are more meaningful to our neural networks. The second matrix is a two by one matrix, but the multiplication approach is the same.

Magic! The first matrix contains the weights between nodes of two layers. The second matrix contains the signals of the first input layer. The answer we get by multiplying these two matrices is the combined moderated signal into the nodes of the second layer. Look carefully, and you'll see this. The first node has the first $input_1$ moderated by the weight $w_{1,1}$ added to the second $input_2$ moderated by the weight $w_{2,1}$. These are the values of x before the sigmoid activation function is applied. The following diagram shows this even more clearly.



This is really very useful! Why? Because we can express all the calculations that go into working out the combined moderated signal, x, into each node of the second layer using matrix multiplication. And this can be expressed as concisely as:

$$X = W * I$$

That is, W is the matrix of weights, **I** is the matrix of inputs, and **X** is the resultant matrix of combined moderated signals into layer 2. Matrices are often written in bold to show that they are in fact matrices and don't just represent single numbers. We now don't need to care so much about how many nodes there are in each layer. If we have more nodes, the matrices will just be bigger. But we don't need to write anything longer or larger. We can simply write $W \ . \ I$ even if **I** has 2 elements or 200 elements!

Now, if a computer programming language can understand matrix notation, it

Now, if a computer programming language can understand matrix notation, it can do all the hard work of many calculations to work out the $X = W . I$,

without us having to give it all the individual instructions for each node in each layer. This is fantastic! A little bit of effort to understand matrix multiplication has given us a powerful tool for implementing neural networks without lots of effort from us.

What about the activation function? That's easy and doesn't need matrix multiplication. All we need to do is apply the sigmoid function $y = \frac{1}{1=e^{-x}}$ to each individual element of the matrix **X**. This sounds too simple, but it is correct because we're not combining signals from different nodes here, we've already done that and the answers are in **X**. As we saw earlier, the activation function simply applies a threshold and squishes the response to be more like that seen in biological neurons. So the final output from the second layer is:

$$O = sigmoid(X)$$

That O written in bold is a matrix, which contains all the outputs from the final layer of the neural network.The expression $W\ .\ I$ applies to the calculations between one layer and the next. If we have 3 layers, for example, we simply do the matrix multiplication again, using the outputs of the second layer as inputs to the third layer but of course, combined and moderated using more weights. Enough theory — let's see how it works with a real example, but this time we will use a slightly larger neural network of 3 layers, each with 3 nodes.