Anatomy of a Neural Network

Week 1

Preview

Hello everyone, and thank you for taking the time to read these slides. I hope they are useful.

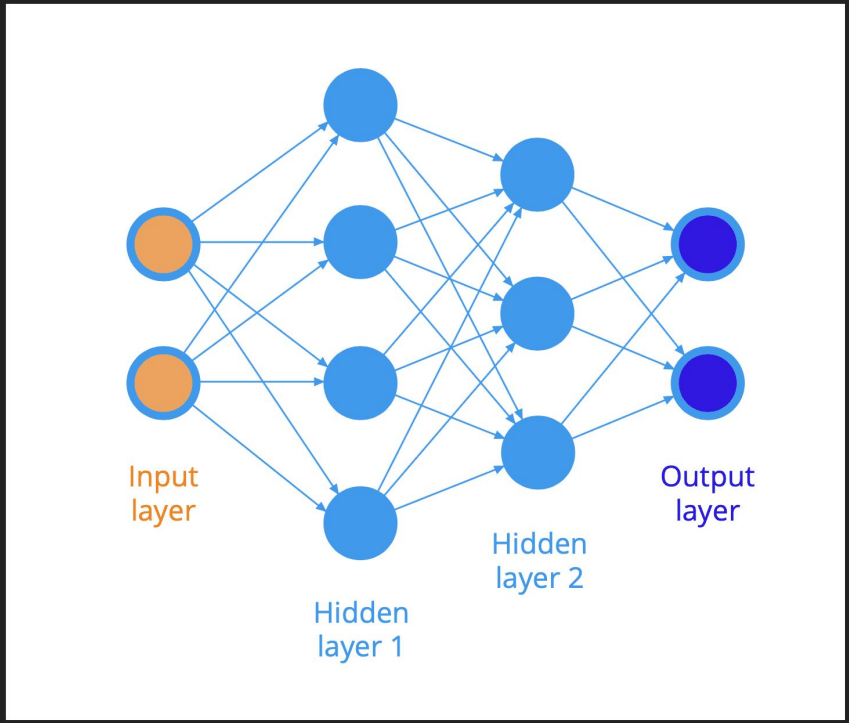
In this slide deck, I hope to cover the foundational concepts on Neural Networks. I'll talk about the following concepts:

- What does a neural network look like
- Its Inputs / outputs
- Activation Values and the Activation function
- Weights / Biases
- Backpropagation (lightly, this topic deserves its own slide deck)

This is a neural network

The visual features in this image heavily correlate with the math behind neural networks. Here's a legend for your use:

- Circles are neurons. These hold values, and modify them
- Lines are weights. They are similar to edges in graphs



Inputs and Outputs

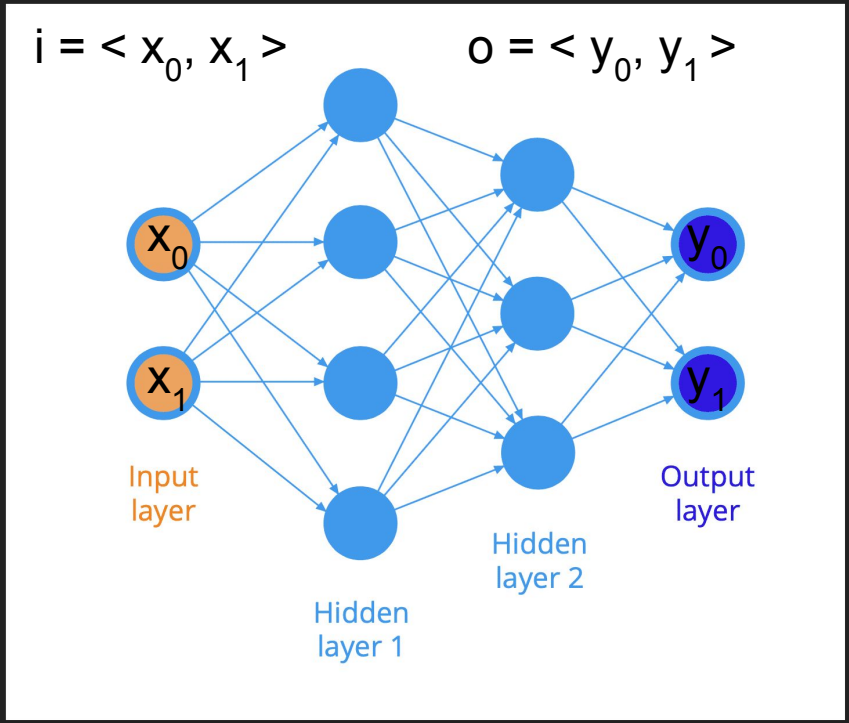
A neural network takes in data as numbers, then outputs a response as numbers. Imagine our model works like a mathematical function.

Let M(i) = o act as our model

Let i be the vector of input values

Let o be the vector of output values.

Each element of i goes into the input layer, and each element of o comes from the output layer.



Recap:

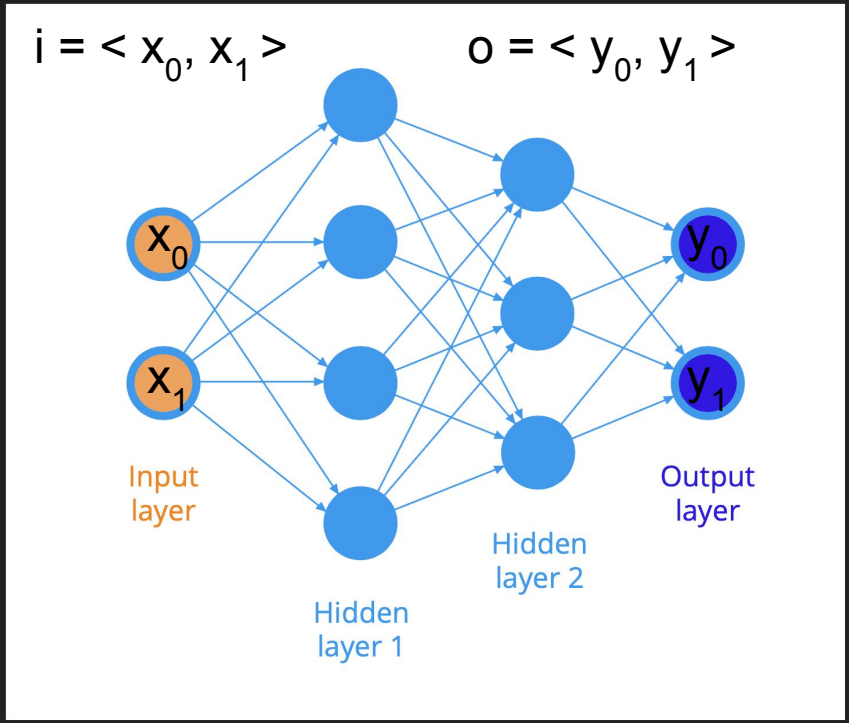
A neural network takes some input data, and outputs numbers.

Its nature is similar to a function

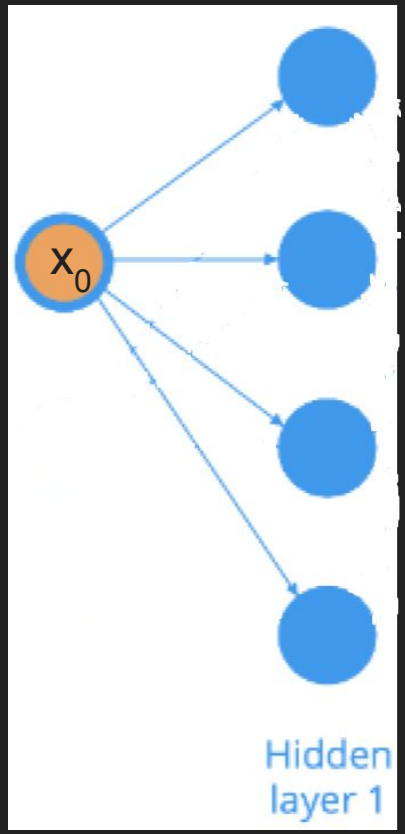
$$M(i) = v$$

Where i and v are vectors with sizes relative to the input and output layers.

The function M() describes the series of operations applied on i to get v.



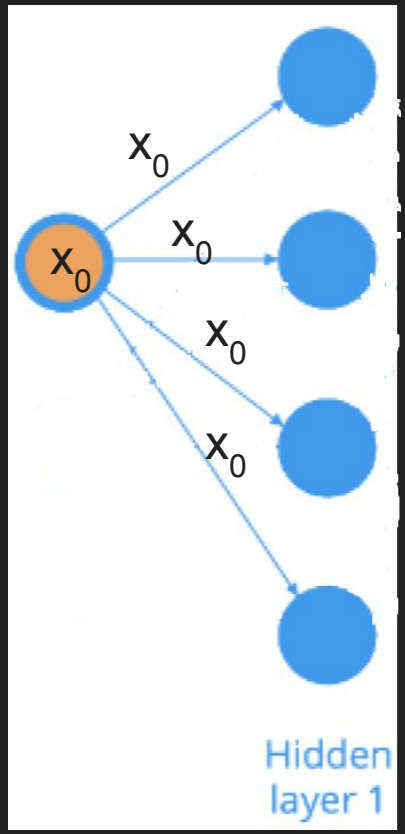
Neurons



Last slide, we defined a neural network as the series of operations on the input vector that yields the output vector. Now we will take a closer look at how this happens.

We're going to close our frame of reference to this (poorly) edited image of our network for now. Let's observe how data moves from the input neuron into the hidden layer.

Neurons

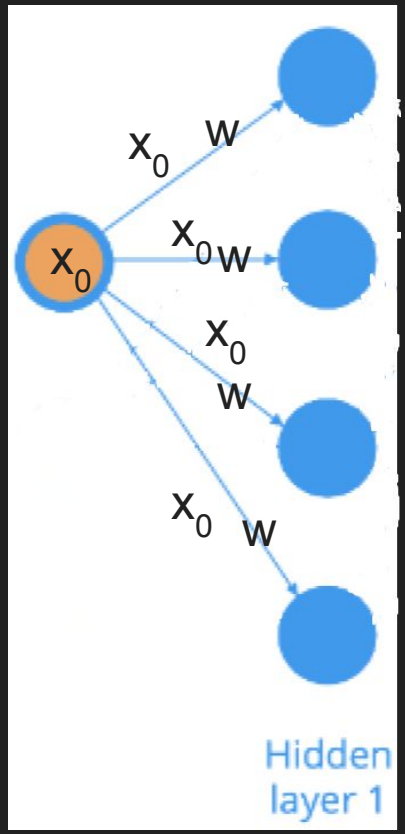


We're going to close our frame of reference to this (poorly) edited image of our network for now. Let's observe how data moves from the input neuron into the hidden layer.

x_0 is our input value, and now its moving to the other neurons along those arrows. The arrows are called weights, and their values will be used in future calculations. Let's call them w

Notice how x_0 gets sent to every neuron in the next layer. When there are multiple input neurons, their input values follow the same path, but are unaffected by previous or future input values

Neurons



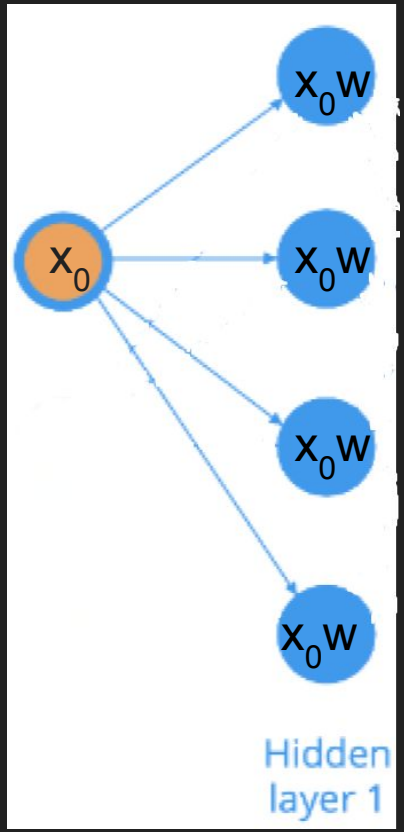
We're going to close our frame of reference to this (poorly) edited image of our network for now. Let's observe how data moves from the input neuron into the hidden layer.

Note: each w is different despite no subscript.

x_0 is our input value, and now its moving to the other neurons along those arrows. The arrows are called weights, and their values will be used in future calculations. Let's call them w.

Notice how x_0 gets sent to every neuron in the next layer. When there are multiple input neurons, their input values follow the same path, but are unaffected by previous or future input values

Neurons

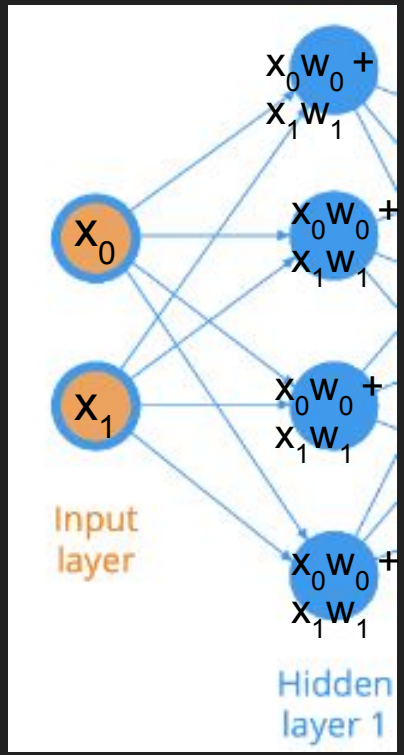


We're going to close our frame of reference to this (poorly) edited image of our network for now. Let's observe how data moves from the input neuron into the hidden layer.

Now, every neuron in the next layer has the input value and weight together. The combination of input values multiplied by their weights is called the unactivated value.

Let's look at an example with multiple neurons in the input layer and see what changes.

Neurons

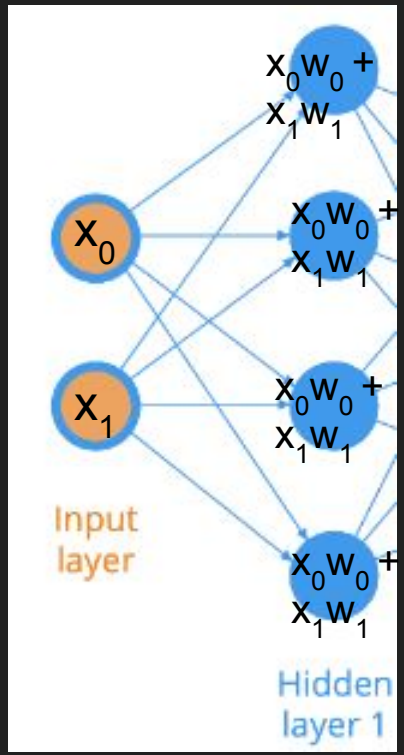


We're going to close our frame of reference to this (poorly) edited image of our network for now. Let's observe how data moves from the input neuron into the hidden layer.

Ok, theres a lot more numbers on the screen. Lets unpack.

Each value of x_0 is still paired with a w, just like the last slide. Similarly, values of x_1 are also paired with a w. However, the w's have subscripts now. Why is that?

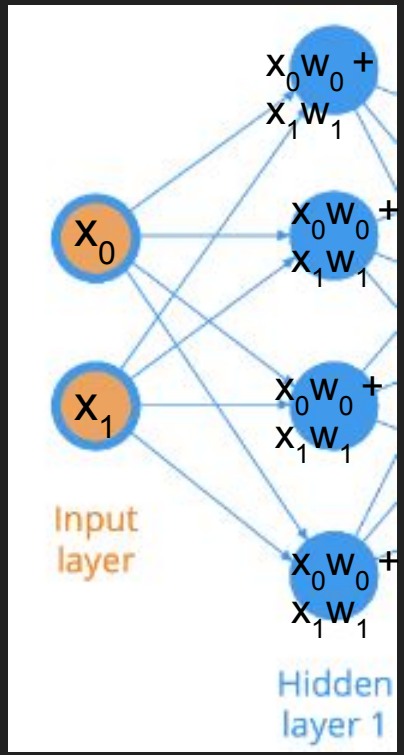
Neurons



We're going to close our frame of reference to this (poorly) edited image of our network for now. Let's observe how data moves from the input neuron into the hidden layer.

It might help to think of things formally. Each neuron has its own vector of weights, w. w_i relates to the i^{th} weight pointing to the neuron (counted top-down). So, in our frame of reference, the values in our hidden layer's neurons are the summation of x_iw_i

Neurons



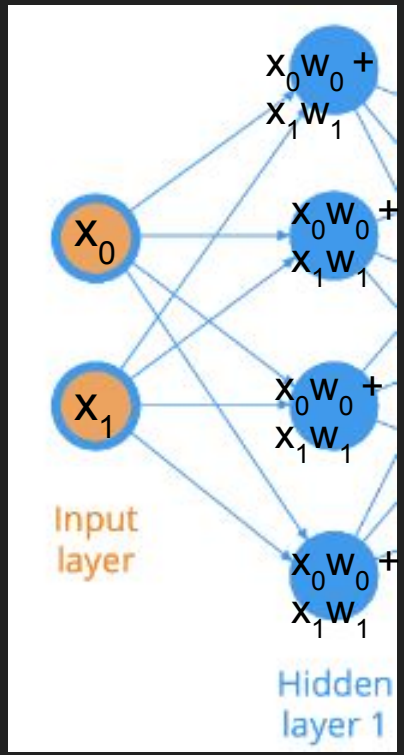
We're going to close our frame of reference to this (poorly) edited image of our network for now. Let's observe how data moves from the input neuron into the hidden layer.

Lets get even more formal. Do you see anything familiar about how the products are summed? Its the dot product!!

let z be the unactivated value of a given neuron, w be the vector of this neuron's weights, and x the vector of inputs.

$$z = w \cdot v$$

Neurons

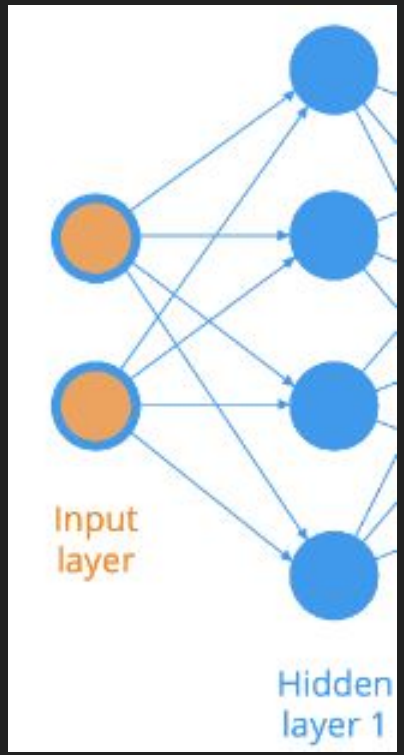


We're going to close our frame of reference to this (poorly) edited image of our network for now. Let's observe how data moves from the input neuron into the hidden layer.

Finally, there's one thing I haven't mentioned yet. Each neuron also has a bias term b. b is a constant, and like w, is defined by the programmer. So the complete equation for the unactivated value of a neuron looks like this:

$$z = w \cdot v + b$$

Neurons:



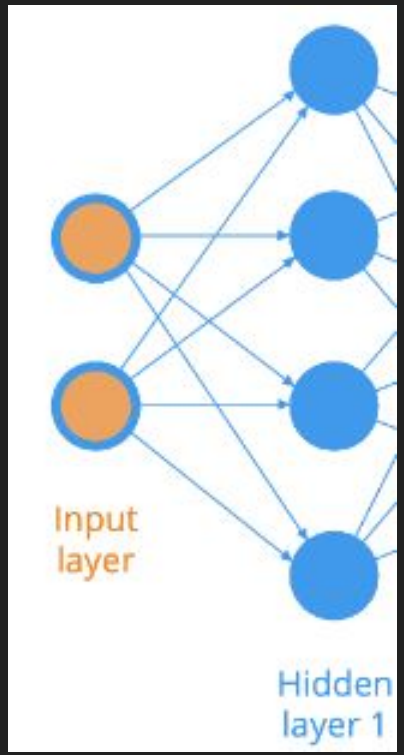
Each neuron has an unactivated value, denoted as z. It also has an activated value a. Lets explore how to find a

To get the final, activated value of a neuron, we apply the activation function, f(z) = a

There are many activation functions, but they all do something similar: introduce non-linearity to our model.

This concept of non-linearity is how models pick up on trends, and differentiates them from algorithms and linear regressions

Neurons:

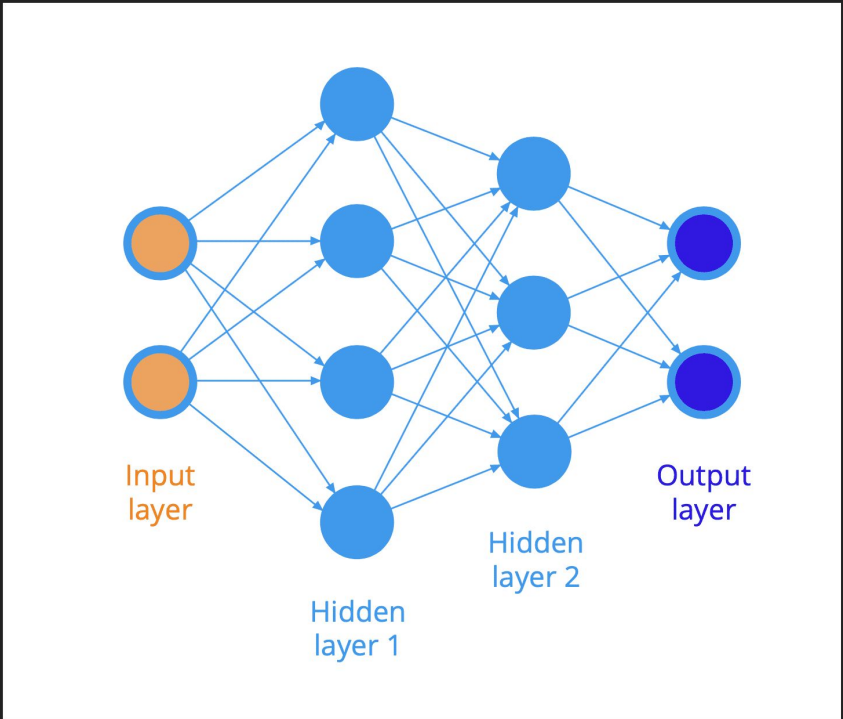


Here are two popular activation functions, ReLU and sigmoid.

$$\text{ReLU}(z) = \begin{cases} z > 0: a = z \\ z \leq 0: a = 0 \end{cases} \qquad \text{Sigmoid}(z) = \frac{e^z}{1 + e^z}$$

The ReLU function outputs 0 if the input is <= 0, or outputs the input. The sigmoid function scales all input values on the numberline to some output value between 0 and 1.

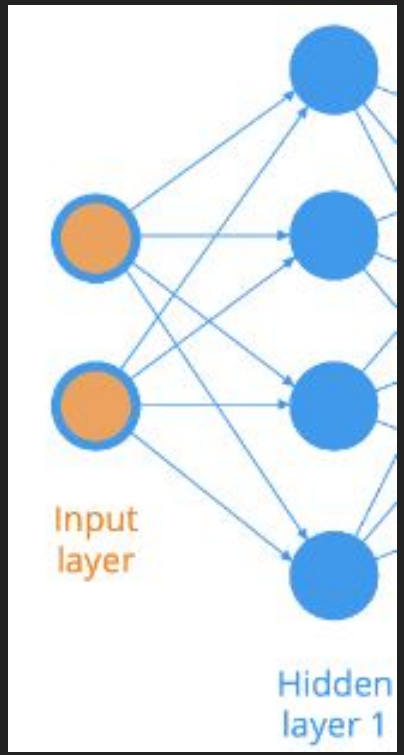
Neurons:



Now that the input layer has populated hidden layer 1, how does hidden layer 2 define its activation values?

The process that a neural network pushes information through itself is recursive. Values in later neurons are defined by activation values in previous layers, all the way back to the input.

Recap on Neurons



We just learned how information travels in a neural network.

Values pass from the input layer along weights, and collect inside neurons.

The sum of the values, weights, and the neuron's bias term is the unactivated value z.

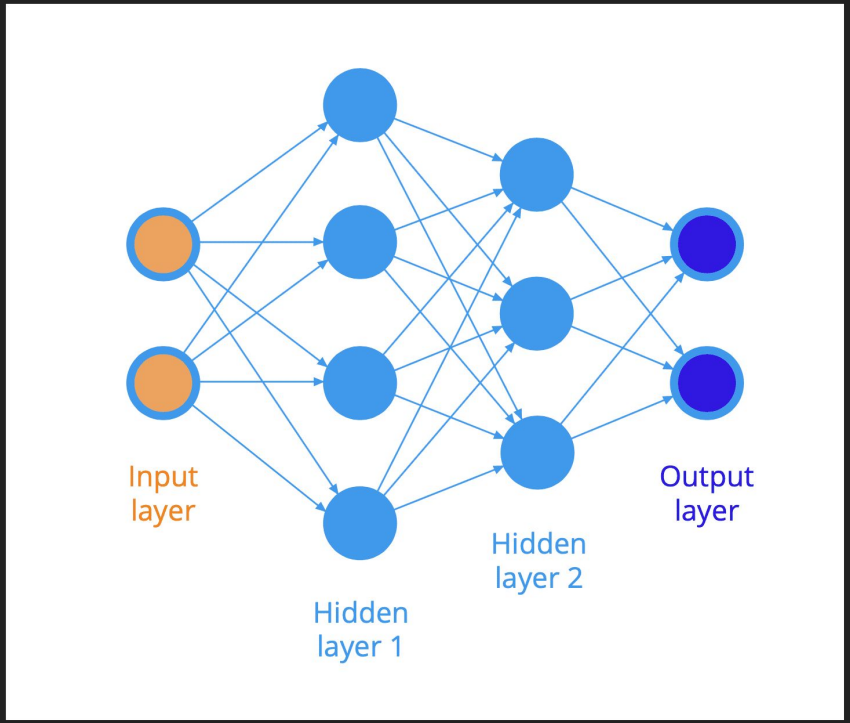
The neuron then applies an activation function that introduces this non-linearity touch to the calculations.

This process continues recursively, with activated values being passed along weights to more layers until its output.

Final Review

Let's quickly what we learned today:

- A neural network takes in data in the form of numbers, and returns other numbers. These numbers can be represented as vectors.
- The unactivated value of a neuron is the sum of previous activated values with their associated weight, plus that neuron's bias term.
- Every neuron has an activation function it applies to its unactivated value to introduce non-linearity



Neurons:

Congratulations on making it this far! You have seen a lot of material, and made a great first step towards learning neural networks. I know there was a lot of information, especially on the values of a neural network. Grab a snack, touch some grass, and try to think of how these topics could apply to your own project.