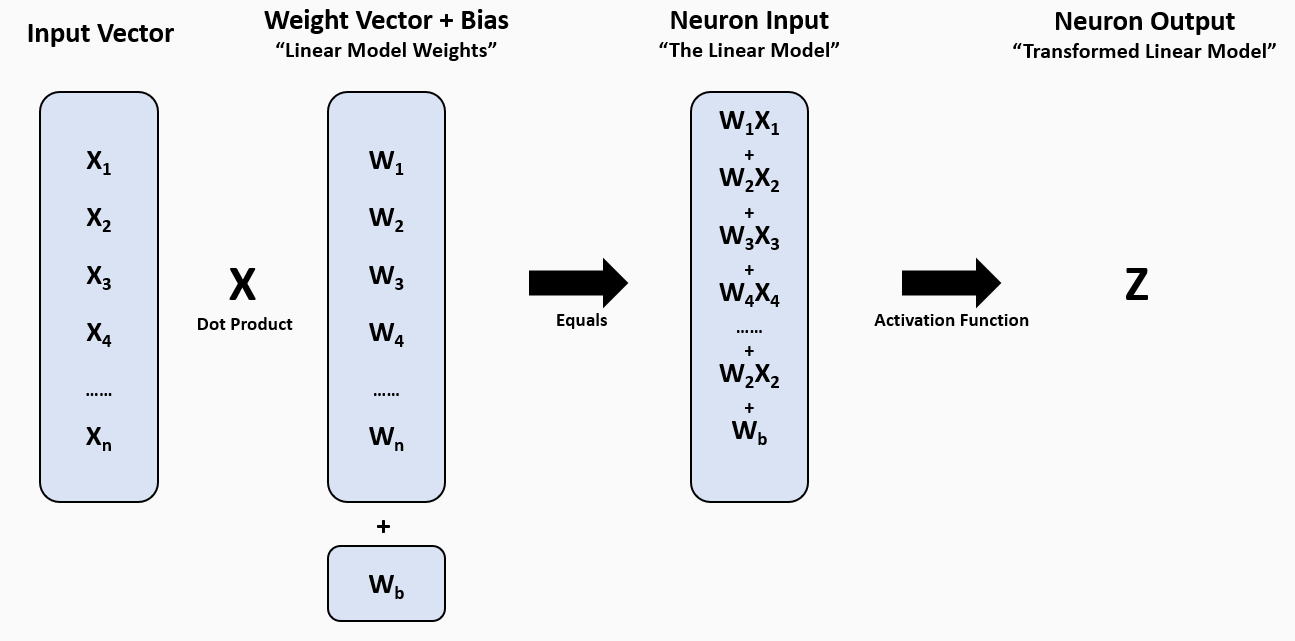
**What is a Neural Network**

This website is concerned with neural networks in the context of regression. In regression we are fundamentally seeking a function that will take an input signal ( of N dimensions) and convert it to a singular outcome. In that context a neural network is just like any other transformative function. It’s sole difference lies in it’s construction – which in turn allows to adapt to complex data.

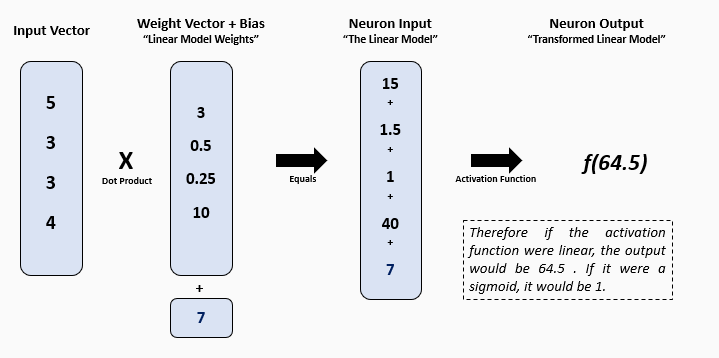
The basic unit of a neural network is a neuron – they are therefore the key to understanding neural networks. A neuron is a self-contained many to one function, consisting of a weight vector and activation function, that can take an input of any dimensionality and return a single value. A neuron processes inputs in the following stages:

* Pre-processing
* Activation

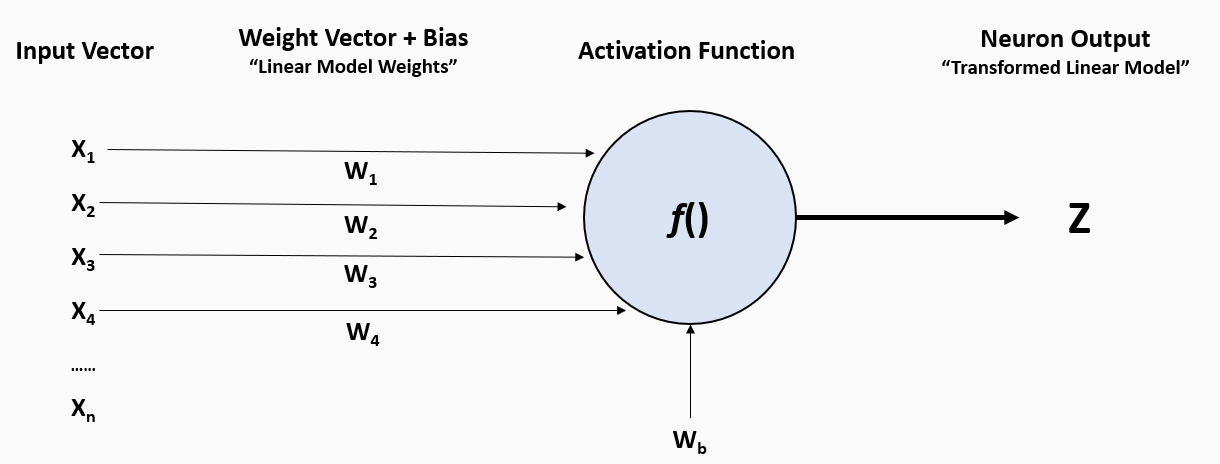
In the pre-processing phase, the neuron converts its inputs into a linear model using its weight vector. This reduces the dimensionality of the input into a single value. This value is then fed into the neuron’s activation function (for example linear, sigmoid or tanh) to generate the final output of the neuron. This is represented on the image below:



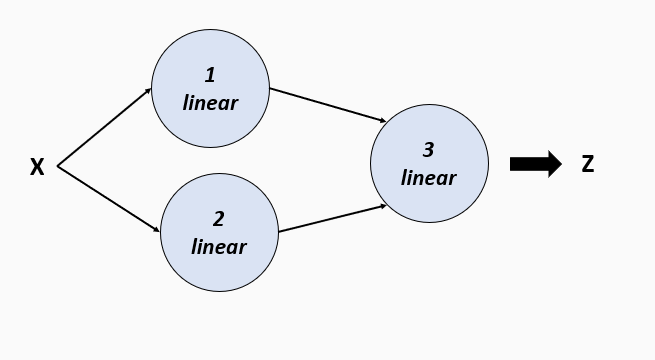
This process is best explained by the example below. In this example a vector of [ 5 , 3 , 3 , 4 ] is used an input into a single neuron with pre-initialized weight vector of [3, 0.5, 0.25, 10 ,7] – returning a value of 64.5 if it possessed a linear activation and 1 if it were sigmoid.



Make sense? This simple function is the building block of a neural network, which is fundamentally just a collection of layered neurons – with the outputs of each layer forming the inputs of the next. Please note that the normal graphical depiction of a neuron is:



Therefore, a simple Neural Network consisting of 3 neurons in a 2x1 formation could be shown as:



With Neurons understood, we can see that a Neural Network is fundamentally:

* A series of distinct layers
* Each layer consists of between 1 and N Neurons
* Each neuron is a basic multidimensional linear model (including an intercept) that sums the product of its inputs and its internal coefficients (weights) – which are generally randomly initialized
* A neurons output is defined as the output of the neurons linear model, transformed via an activation function
* Common activations include linear, sigmoid, tanh and reclu
* This output - plus the outputs of all other neurons on the layer - form the input into each of the next layers neurons

It is therefore intuitive that in the case of regression, the final layer of the network will always contain a single neuron - as we require a single output.

**WHAT IS BACKPROPAGATION**

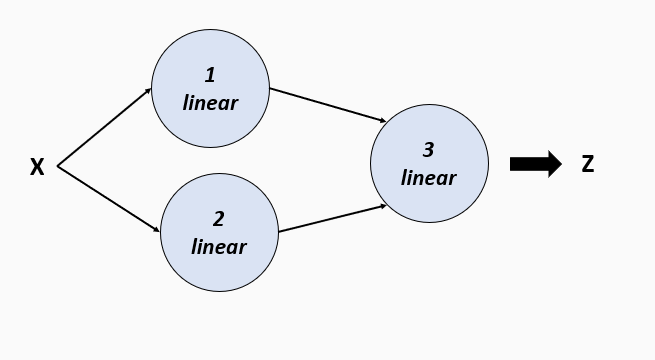
So far on this website we have defined what a neural network is, and how it can calculate an output from randomly initialized weights. But this is useless – a neural network is only powerful if it can learn and adapt to the data that we feed to it.

This is achieved via a method referred to as backpropagation.

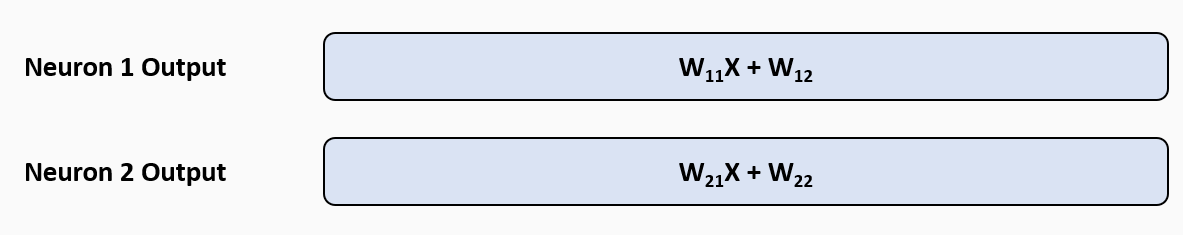
Backpropagation works in the following steps:

* We initialize our neuron with random weights
* We feed our neural network between 1 and N training examples – generally the fewer the better (referred to as a batch)
* We compute the average error between the predictions and the true responses
* We compute the partial derivative of the neural network for each of the weights of each neuron as these are our levers of control over the system
* Once we have determined the derivative, we use a method called gradient make tiny adjustments to each of the systems weights as to reduce the average prediction error
* We repeat this process N times until we meet some stopping criteria or reach some incrementor limit
* Every time we cycle through all training examples it is referred to as an epoch

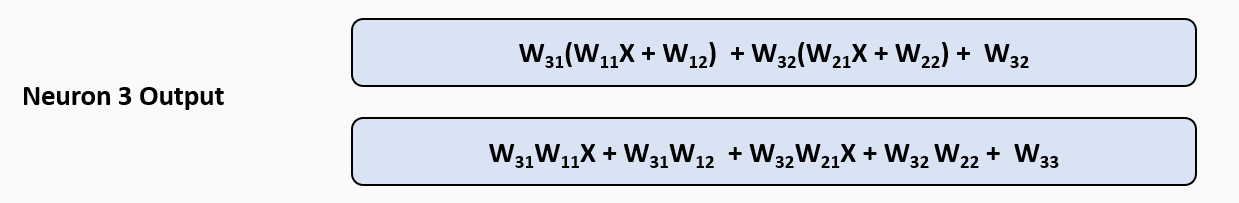
This whole process is best explained again via example – remember our simple 3 linear neuron network from before?



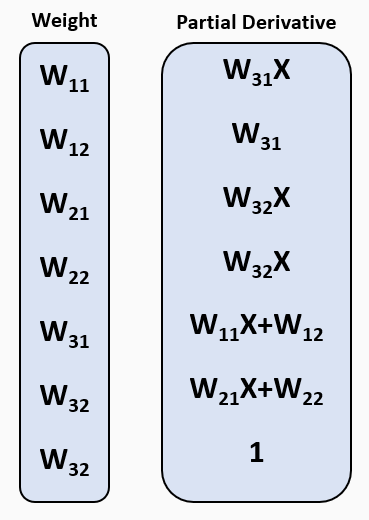
Well, it can be shown that the outputs of the first two neurons are:



And from this, that the output of the entire network can effectively be represented by the following equation. Please note that this holds true – albeit just in a more complicated form – irrespective of the activation function:



From this, using calculus, we can calculate the partial derivative of the output in respect to any weight in the network:



But the question remains – how does this help us? Imagine we were to feed this neural network a training observation – the model would consequently have some error i.e. it would have over or underpredicted the response value. We can then use the partial derivatives of each weight, multiplied by a “learning rate” so as to prevent overfitting, and adjust each of the weight values with one of the following formulae (the choice of which depending of whether we want to increase or decrease the model output):



This methodology is the principle upon which neural networks learn and as such it is immensely powerful – and yet surprisingly simple and elegant.