

Neural Networks for NLP

ANN Basics

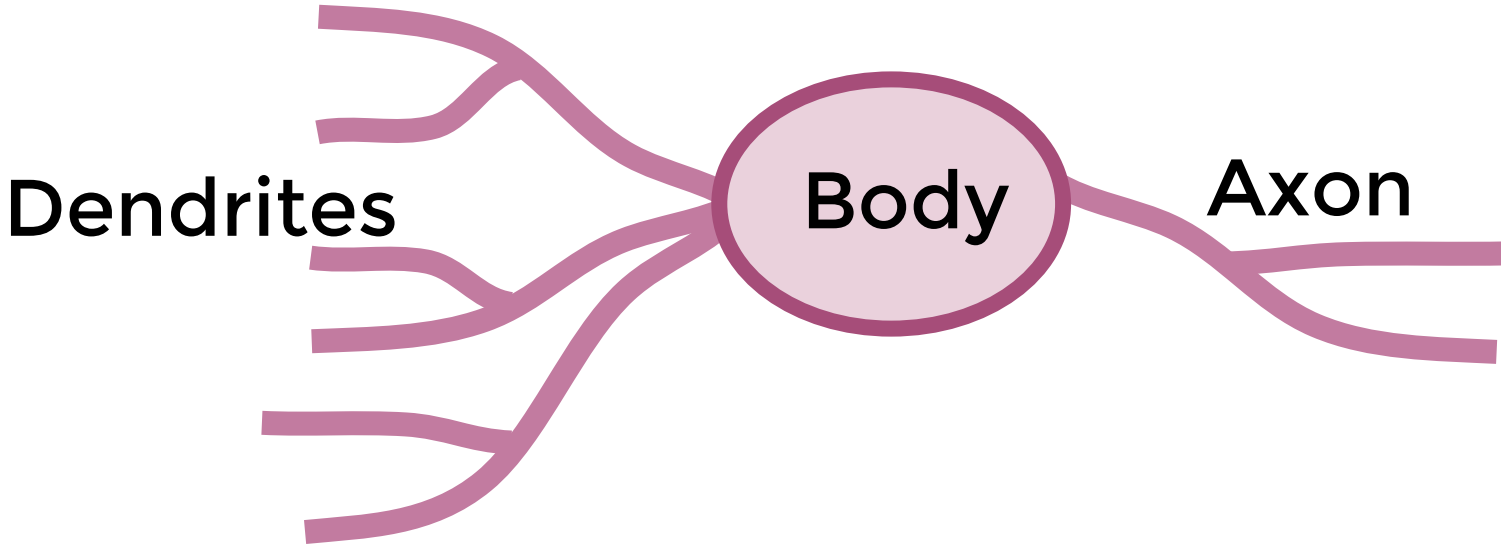
Deep Learning

- Before we launch straight into neural networks, we need to understand the individual components first, such as a single “neuron”.

ANN

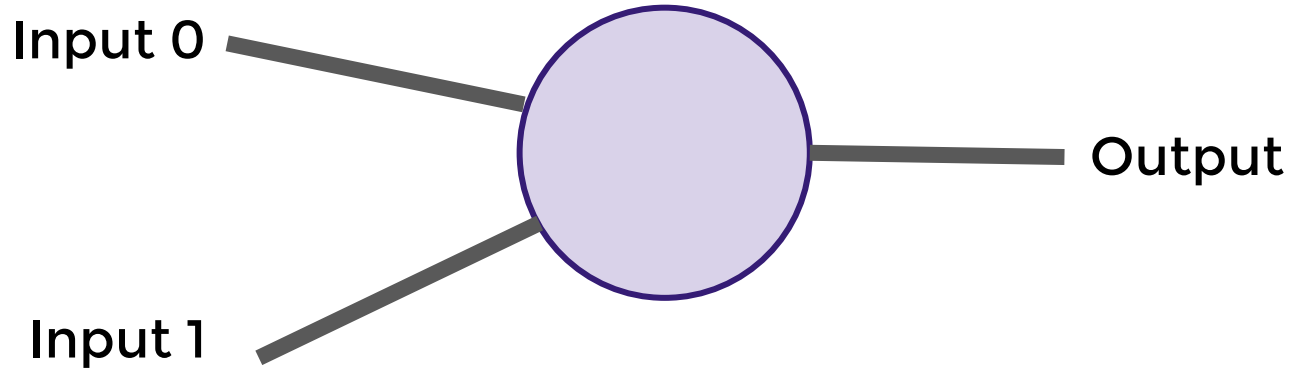
- Artificial Neural Networks (ANN) actually have a basis in biology!
- Let's see how we can attempt to mimic biological neurons with an artificial neuron, known as a perceptron!

The biological neuron



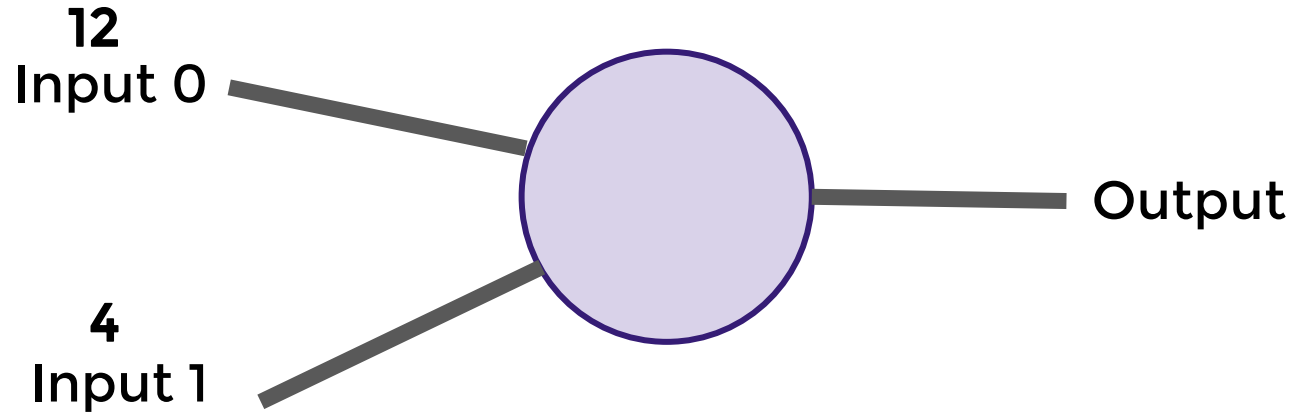
Artificial Neuron aka Perceptron

The artificial neuron also has inputs and outputs!



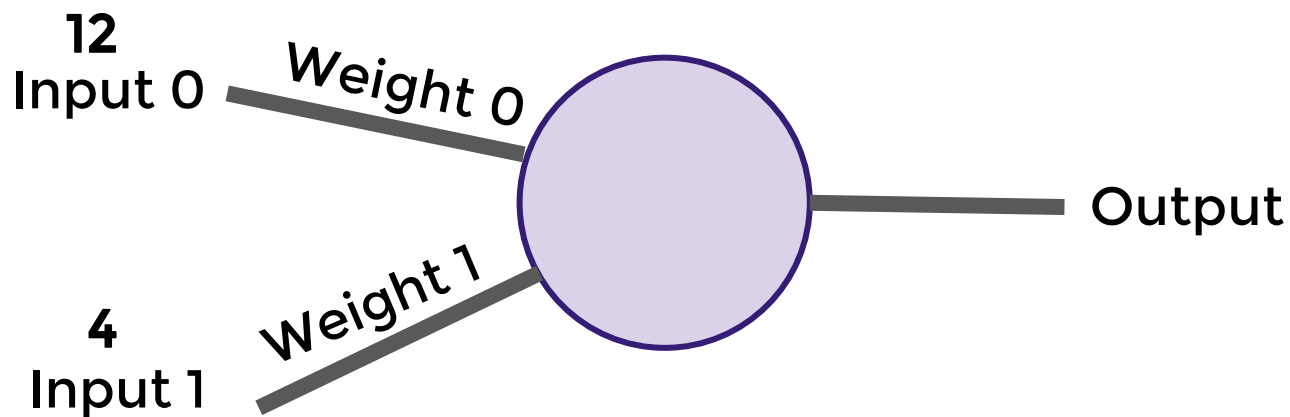
Input of ANN

Inputs will be values of features

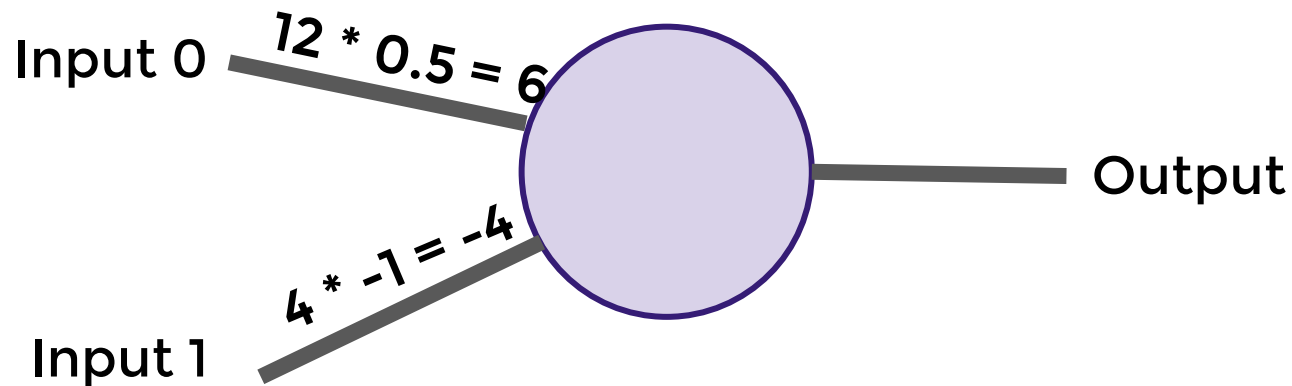


ANN Weights

- Inputs are multiplied by a random weight to start

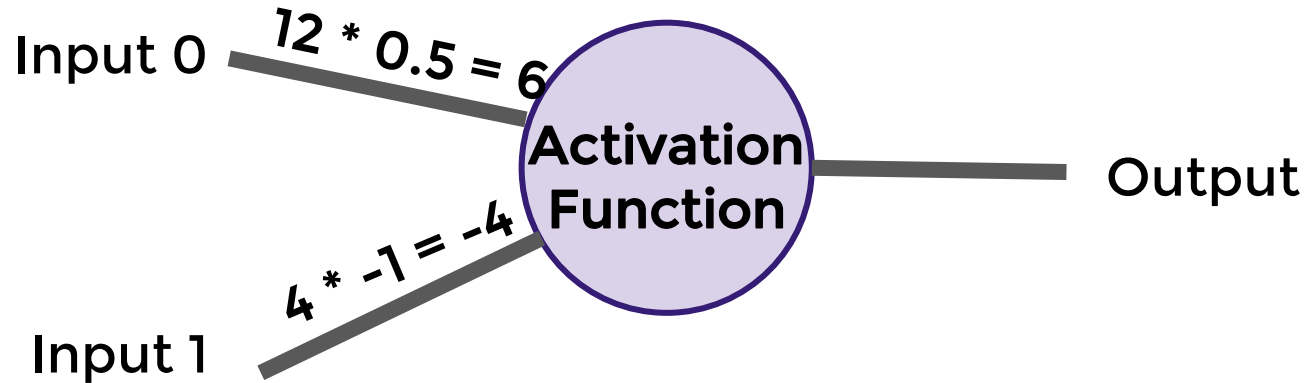


ANN Weights Example



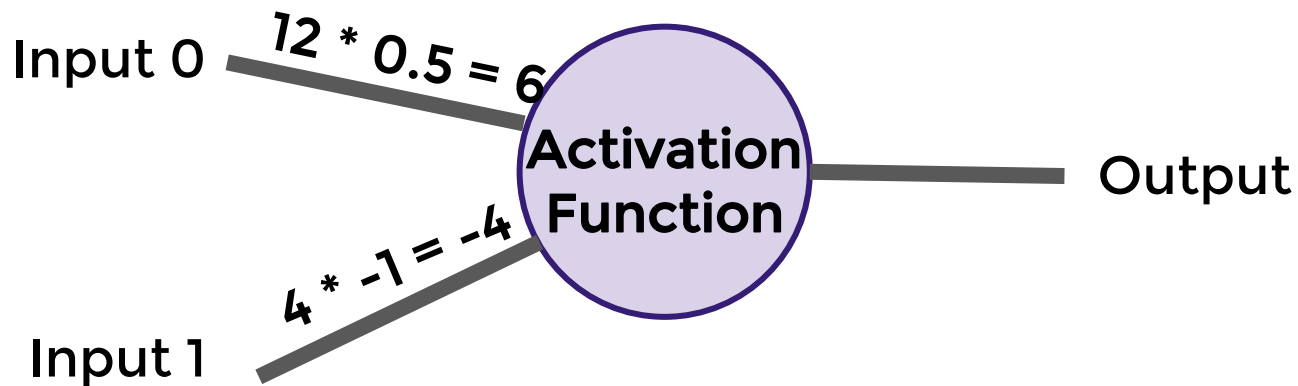
Activation Function

- Then these results are passed to an activation function.



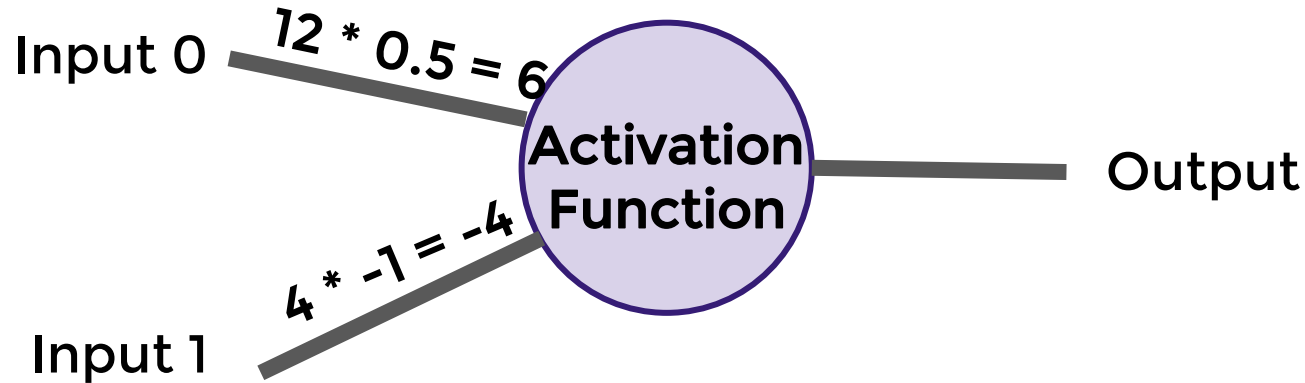
Activation Function

- Many activation functions to choose from, we'll cover this in more detail later!



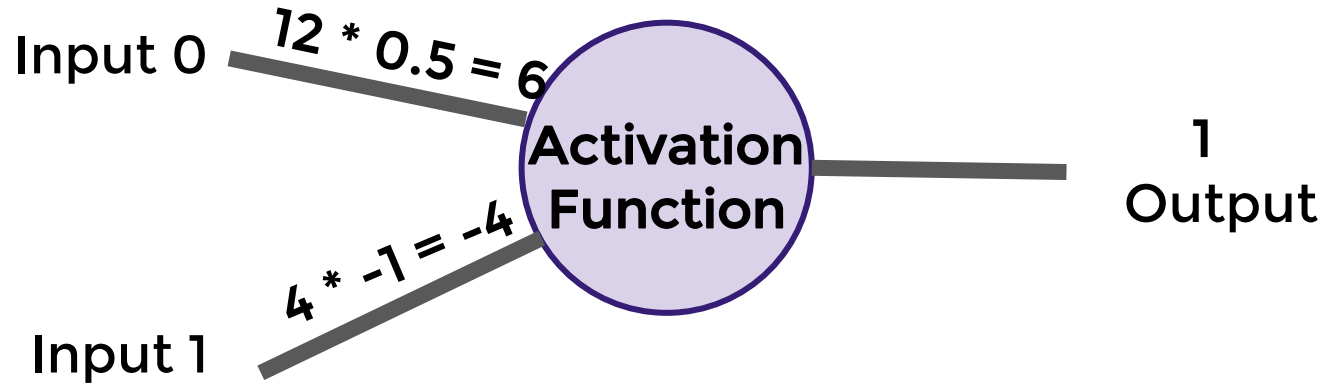
Simple Activation Function

- If sum of inputs is positive return 1, if sum is negative output 0.



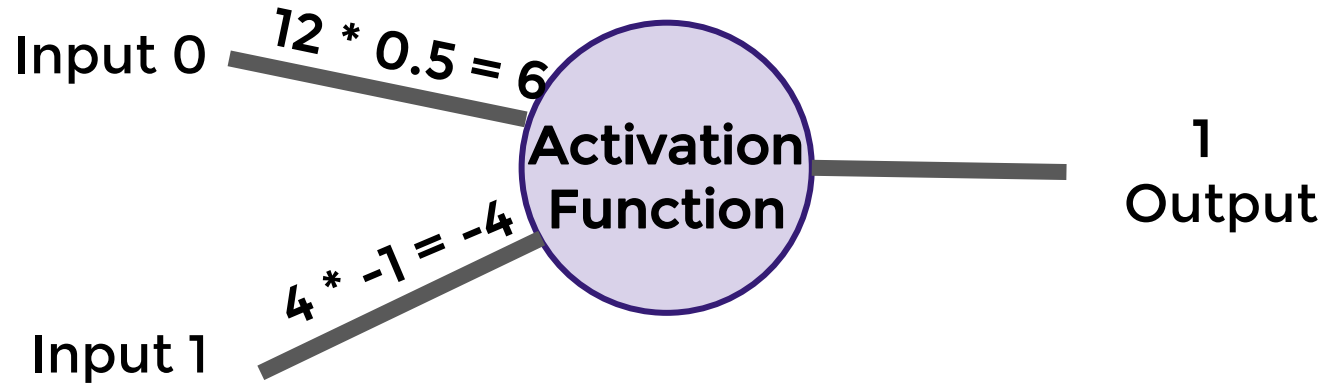
Simple Activation Function

- In this case $6 - 4 = 2$ so the activation function returns 1.



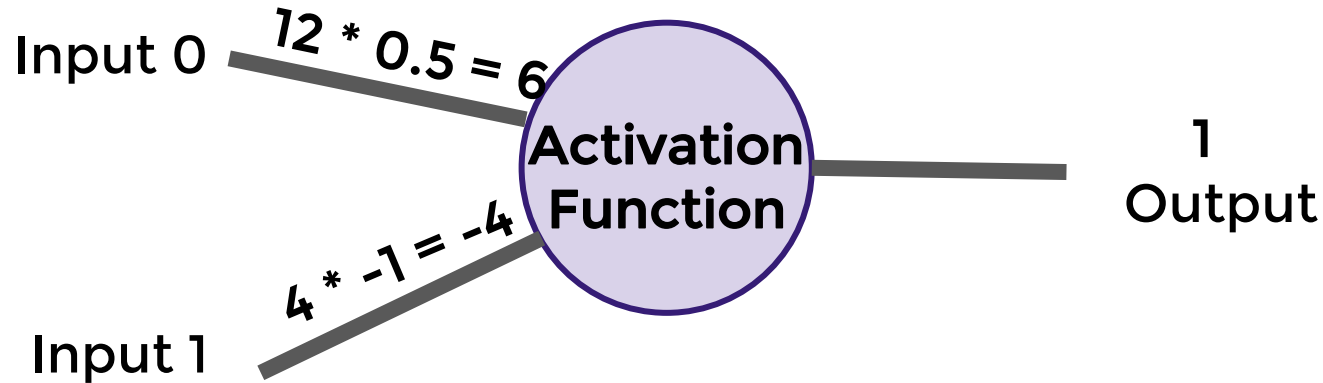
Simple Activation Function

- There is a possible issue. What if the original inputs started off as zero?



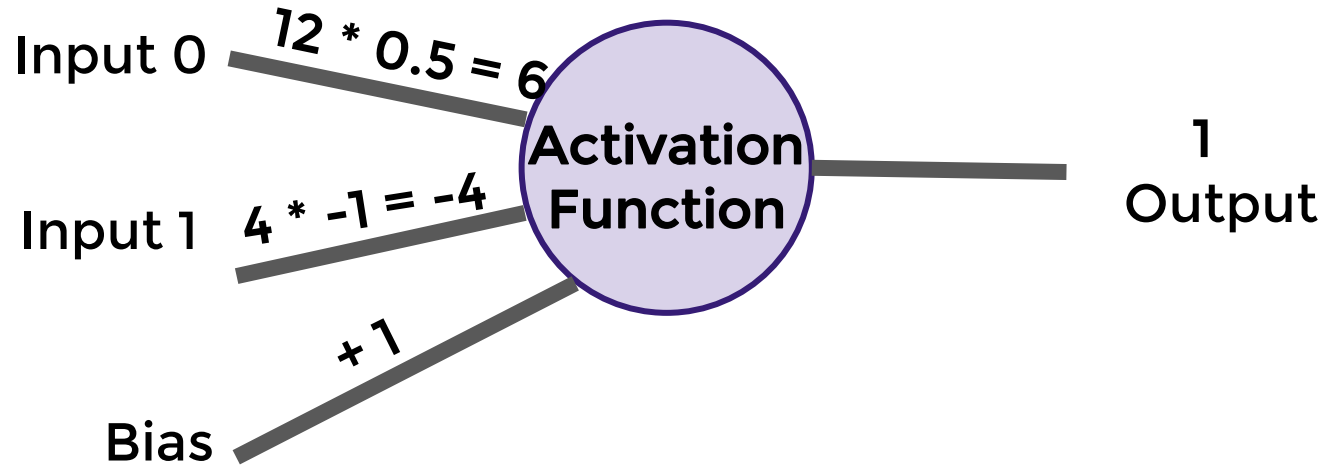
Simple Activation Function

- Then any weight multiplied by the input would still result in zero!



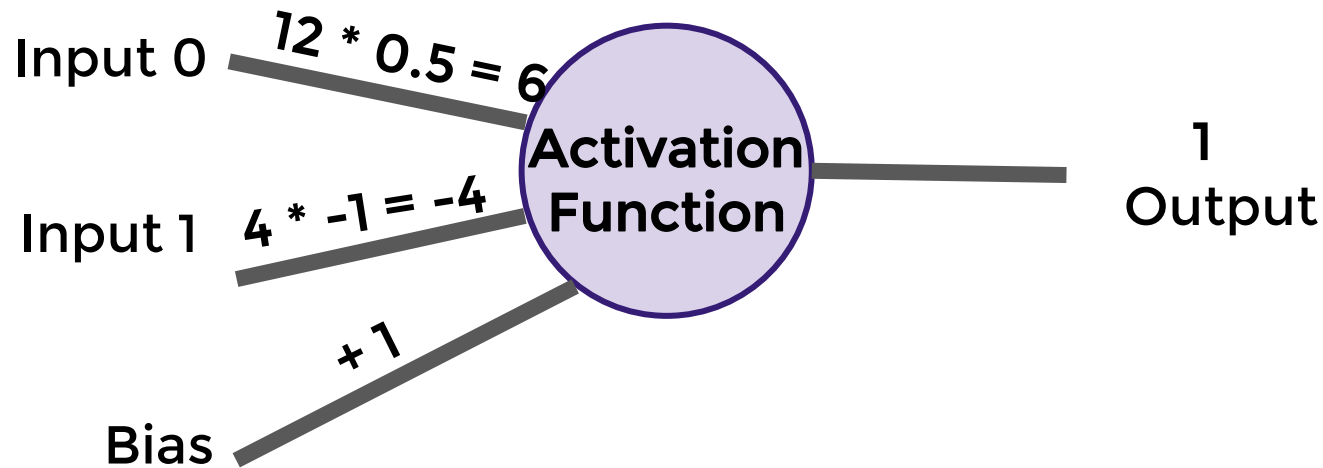
Simple Activation Function

- We fix this by adding in a bias term, in this case we choose 1.



Simple Activation Function

- So what does this look like mathematically?



Activation Function

- Let's quickly think about how we can represent this perceptron model mathematically:

$$\sum_{i=0}^n w_i x_i + b$$

Activation Function

- Once we have many perceptrons in a network we'll see how we can easily extend this to a matrix form!

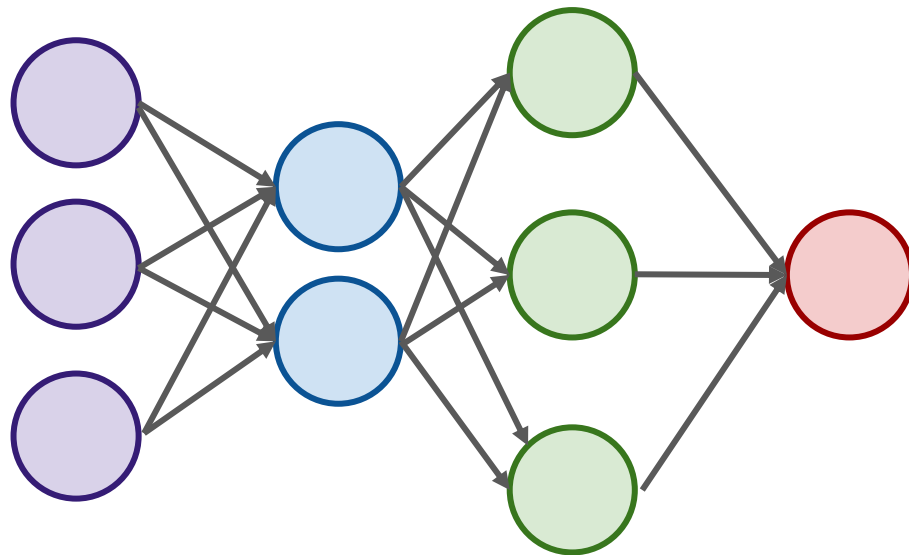
$$\sum_{i=0}^n w_i x_i + b$$

Introduction to Neural Networks

From Single Perceptron to ANN

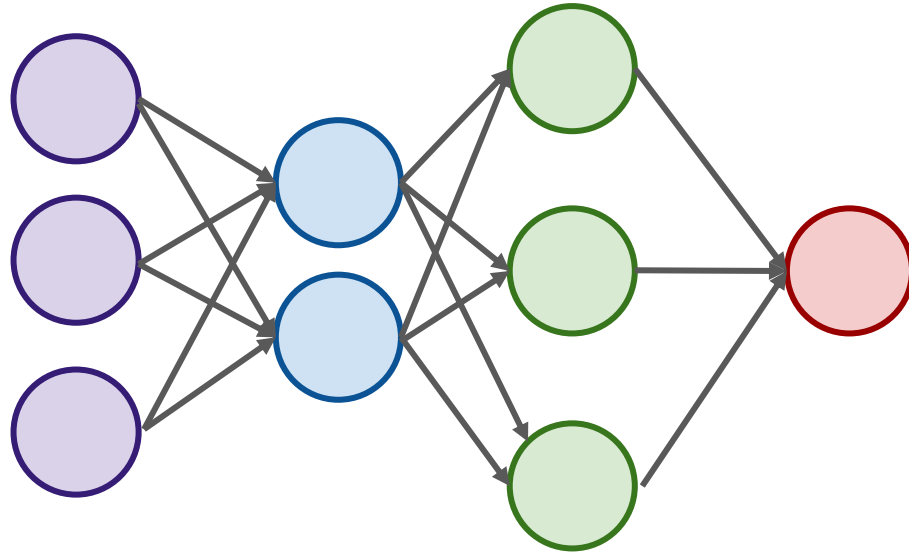
- We've seen how a single perceptron behaves, now let's expand this concept to the idea of a neural network!
- Let's see how to connect many perceptrons together and then how to represent this mathematically!

Multiple Perceptrons Network



Layers

- Input Layer. 2 hidden layers. Output Layer



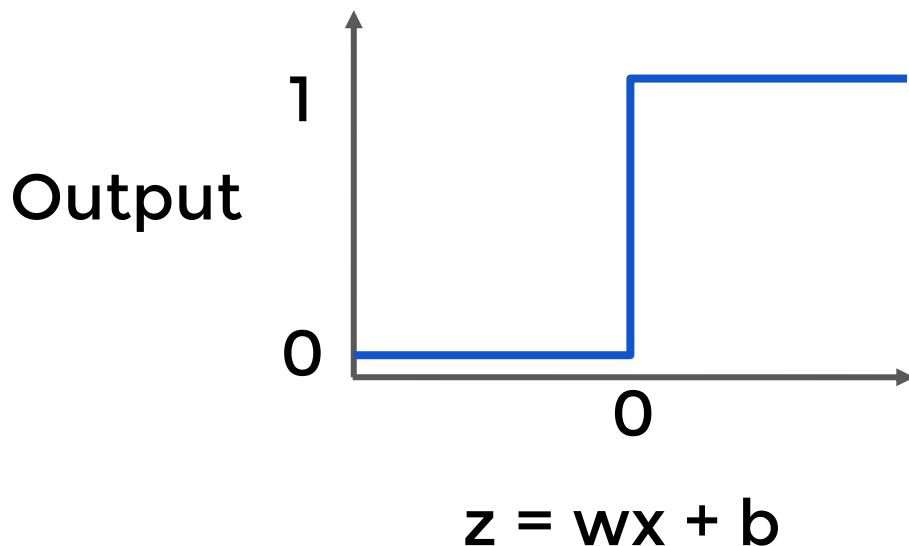
Layers

- Input Layers
 - Real values from the data
- Hidden Layers
 - Layers in between input and output
 - 3 or more layers is “deep network”
- Output Layer
 - Final estimate of the output

- As you go forward through more layers, the level of abstraction increases.
- Let's now discuss the activation function in a little more detail!

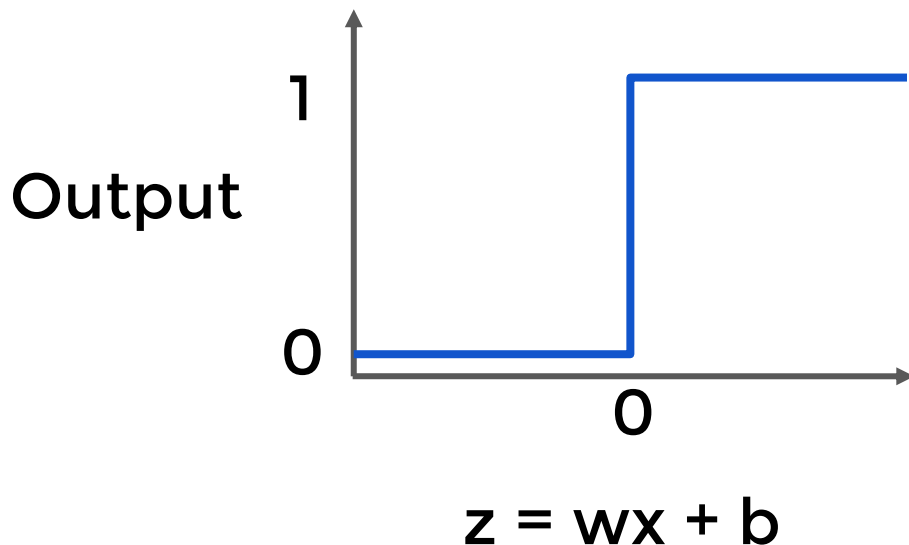
Activation Function

- Previously our activation function was just a simple function that output 0 or 1.



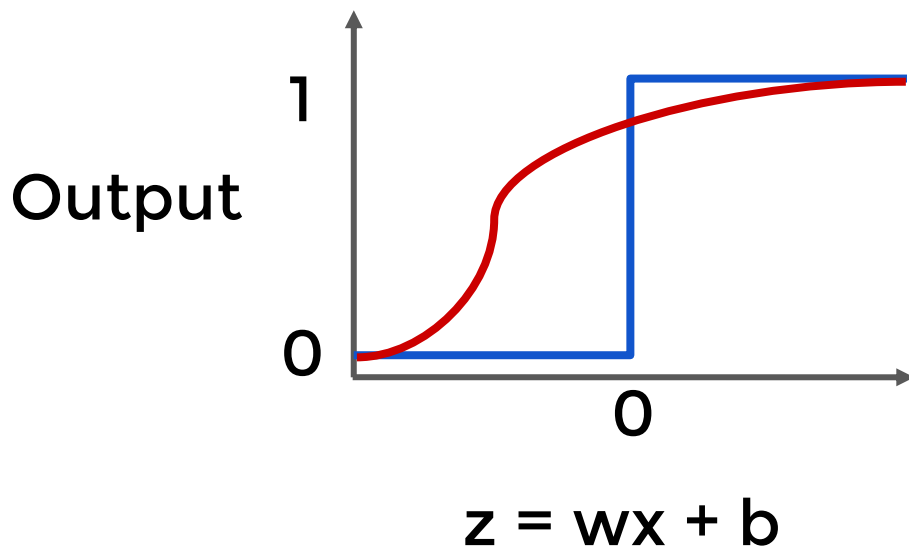
Activation Function

- This is a pretty dramatic function, since small changes aren't reflected.



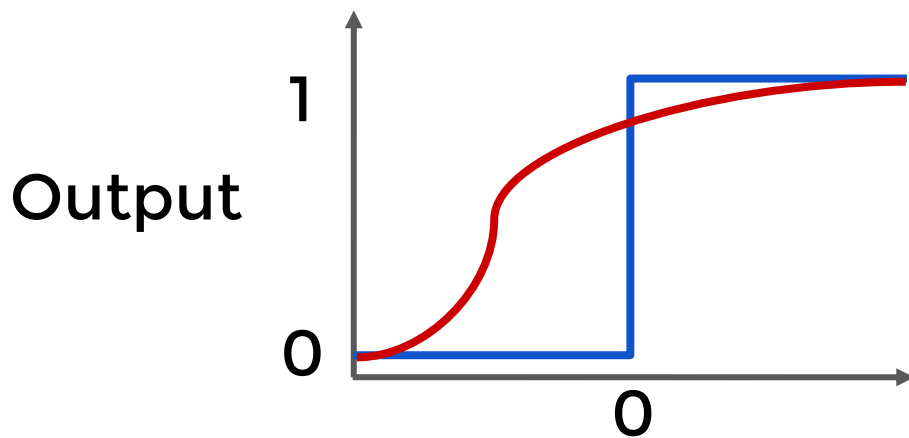
Activation Function

It would be nice if we could have a more dynamic function, for example the red line!



Sigmoid Activation Function

Lucky for us, this is the sigmoid function!

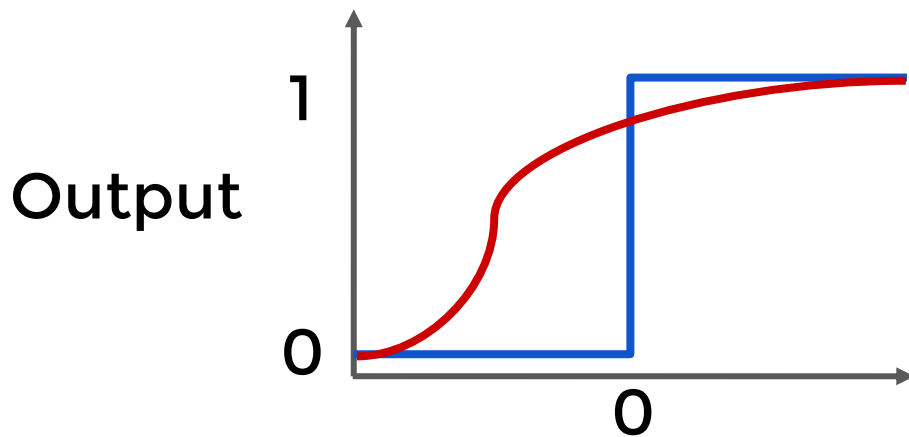


$$f(x) = \frac{1}{1 + e^{-(x)}}$$

$$z = wx + b$$

Activation Function

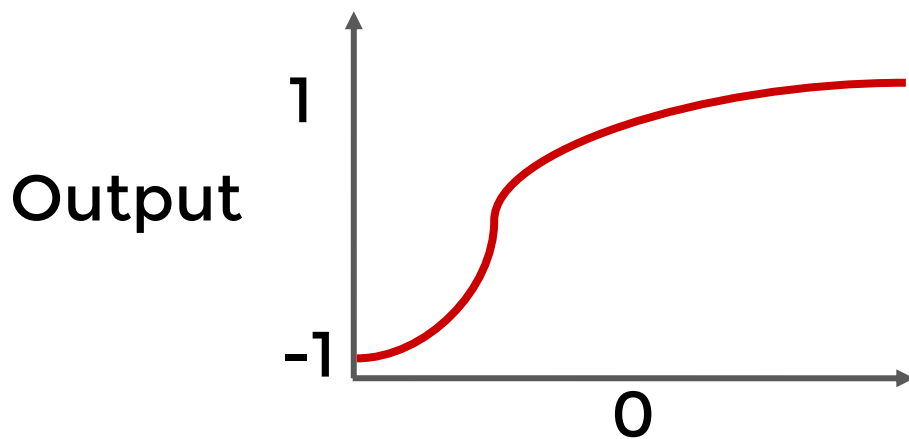
Changing the activation function used can be beneficial depending on the task!



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

$$z = wx + b$$

Hyperbolic Tangent: $\tanh(z)$



$$z = wx + b$$

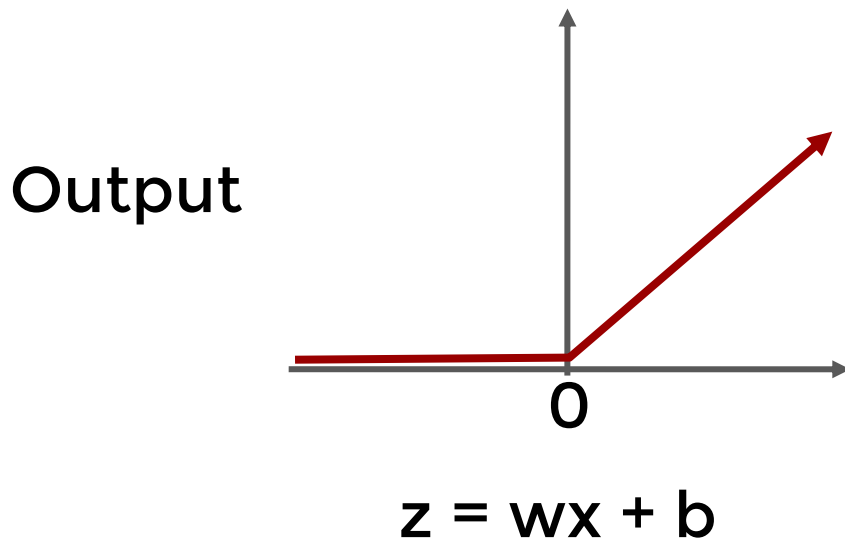
$$\cosh x = \frac{e^x + e^{-x}}{2}$$

$$\sinh x = \frac{e^x - e^{-x}}{2}$$

$$\tanh x = \frac{\sinh x}{\cosh x}$$

Rectified Linear Unit (ReLU)

This is actually a relatively simple function:
 $\max(0, z)$



ReLu

- ReLu tends to have the best performance in many situations.
- Deep Learning libraries have these built in for us, so we don't need to worry about having to implement them manually!

Backpropagation

