Introduction to Neural Networks

Deep Learning

- This series of lectures will cover key theory aspects:
    - Neurons and Activation Functions
    - Cost Functions
    - Gradient Descent
    - Backpropagation

Deep Learning

- Once we build a general high level understanding we will code out all these topics manually with Python, without the use of a deep learning library.
- Then we can move on to using TensorFlow!

Deep Learning

- Understanding a high level overview of these key elements will make it much easier to understand what is happening when we begin to use TensorFlow!
- Tensorflow has direct connections to these concepts in its syntax!

Let’s get started!

Introduction to the Perceptron

Deep Learning

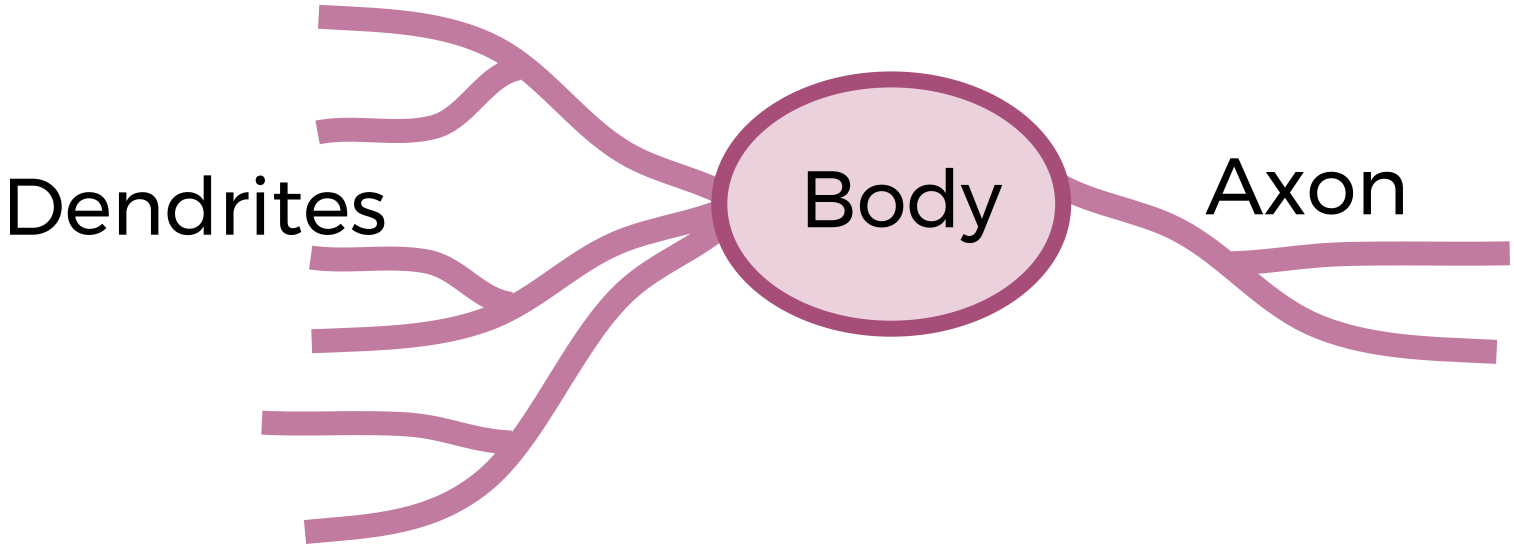
- Before we launch straight into neural networks, we need to understand the individual components first, such as a single “neuron”.

Deep Learning

- 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!

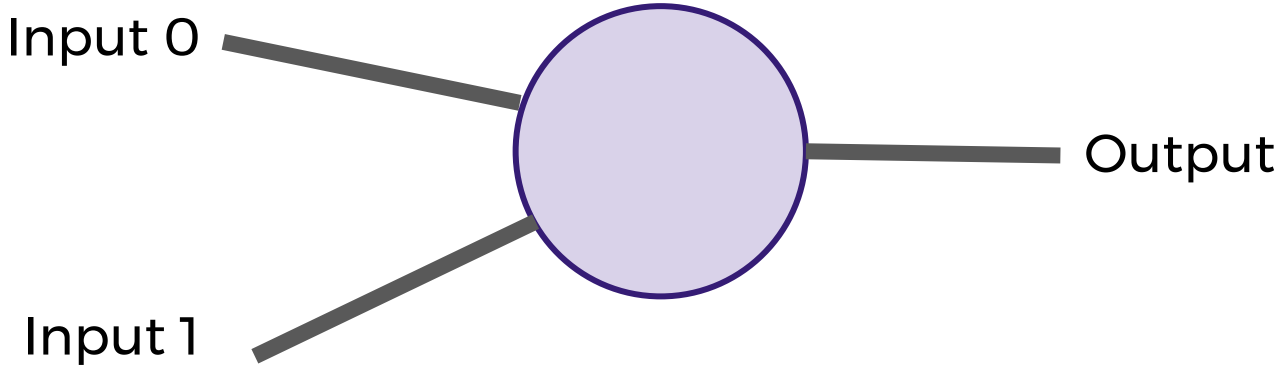
Deep Learning

- The biological neuron:



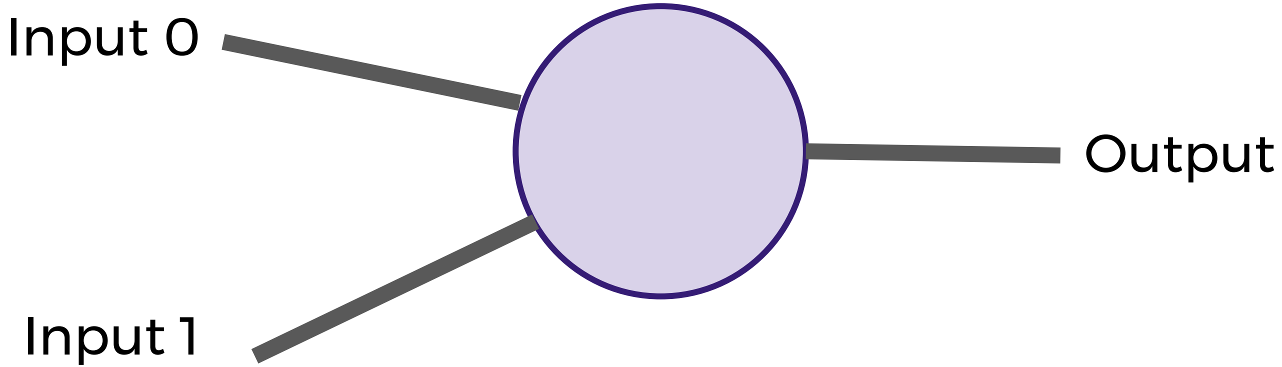
Deep Learning

- The artificial neuron also has inputs and outputs!



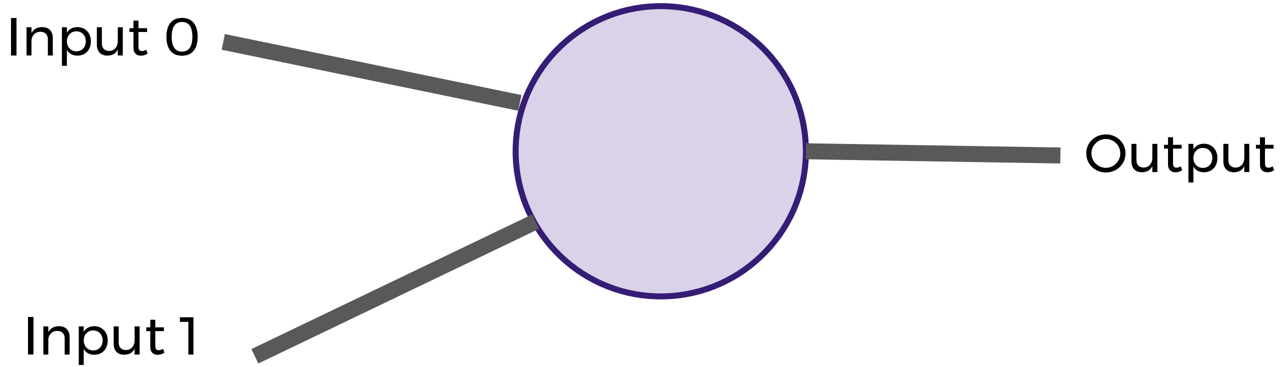
Deep Learning

- This simple model is known as a perceptron.



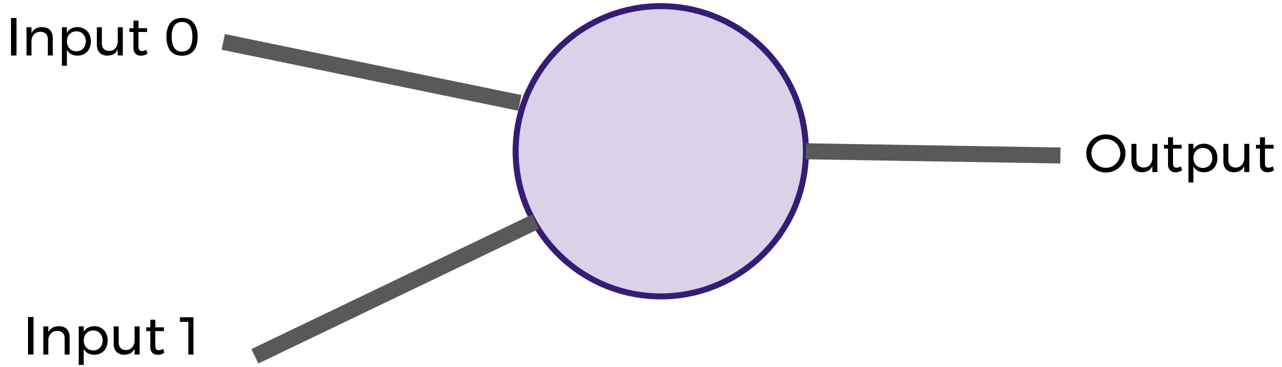
Deep Learning

- Simple example of how it can work.



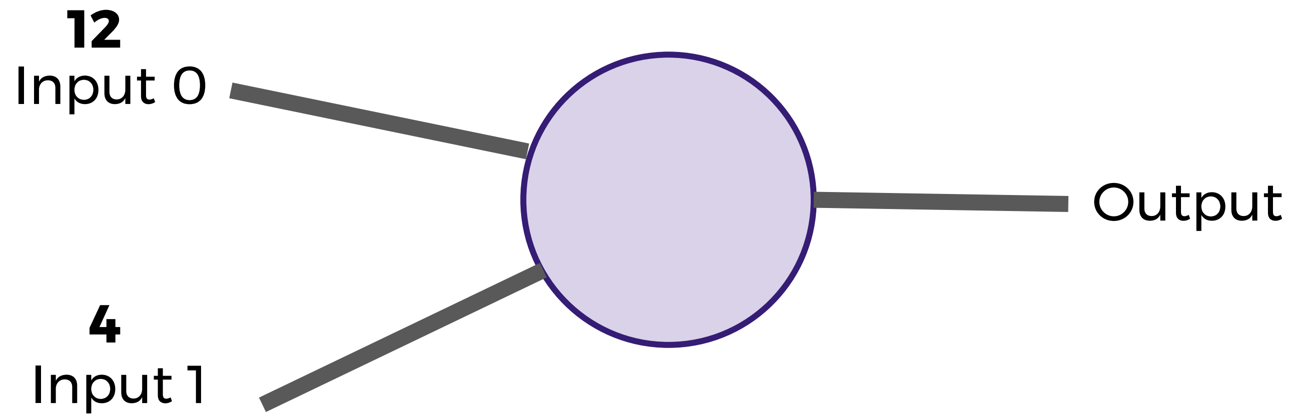
Deep Learning

- We have two inputs and an output



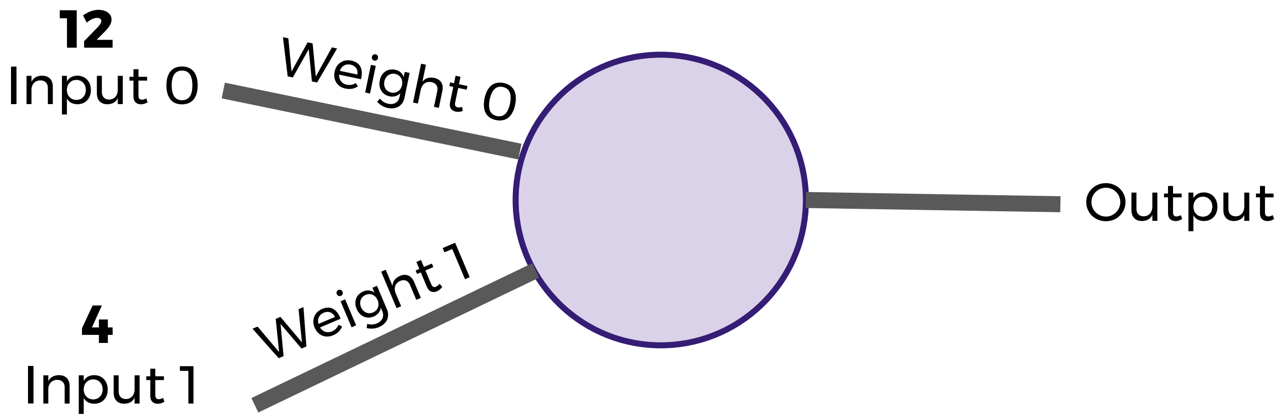
Deep Learning

- Inputs will be values of features



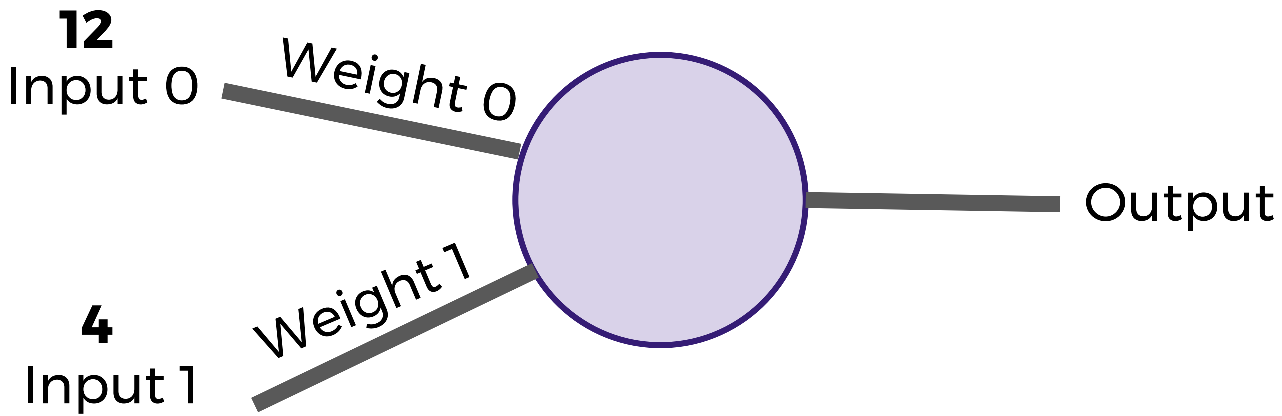
Deep Learning

- Inputs are multiplied by a weight



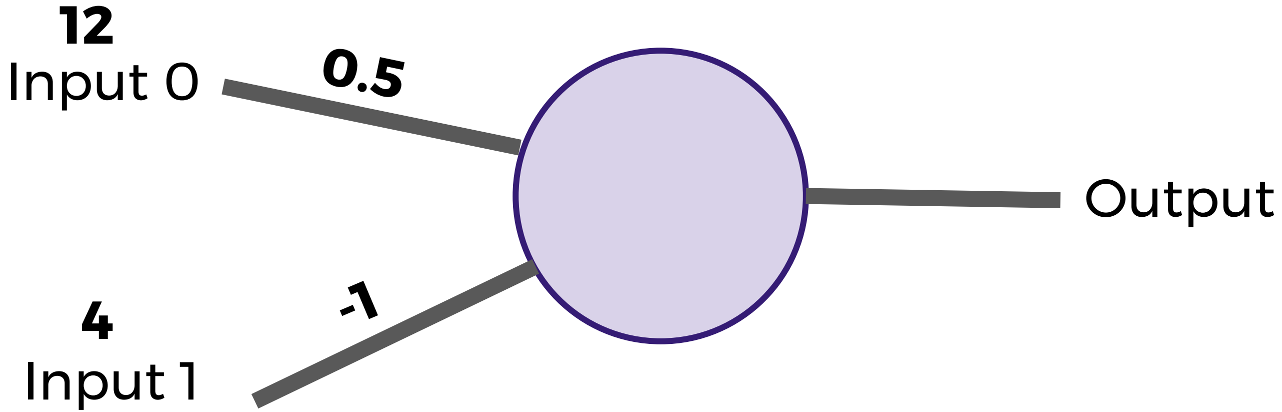
Deep Learning

- Weights initially start off as random



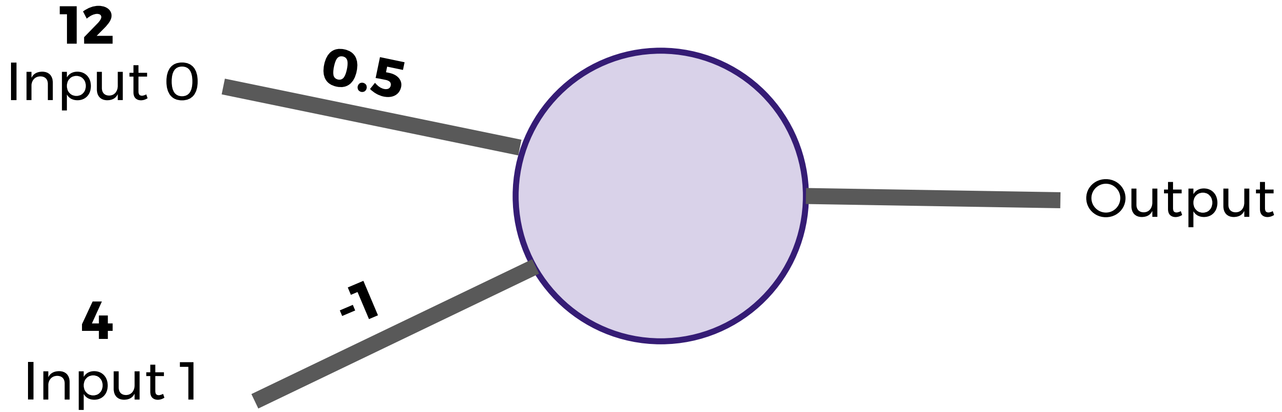
Deep Learning

- Weights initially start off as random



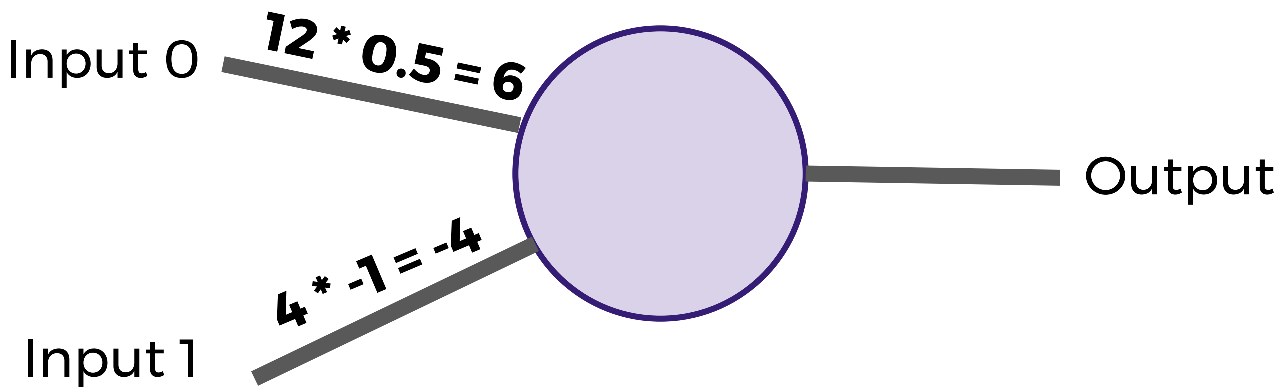
Deep Learning

- Inputs are now multiplied by weights

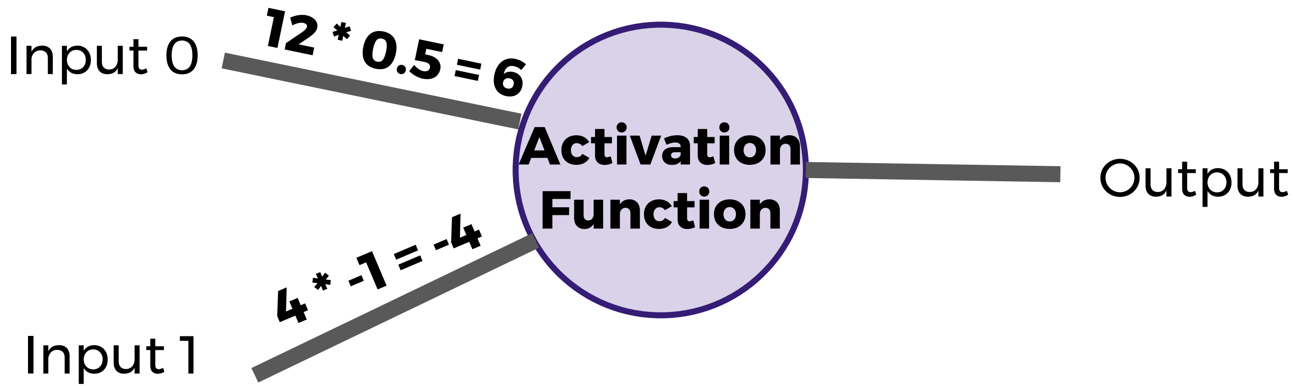


Deep Learning

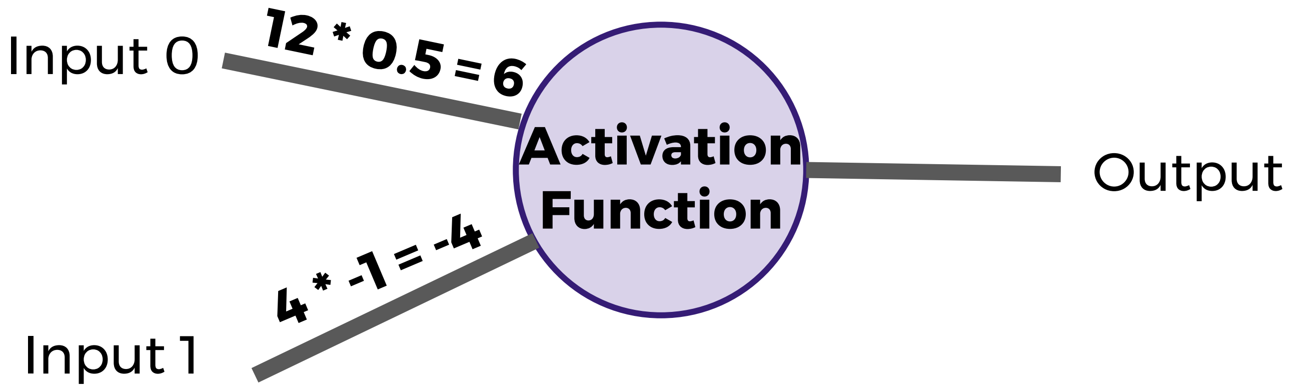
- Inputs are now multiplied by weights



- Then these results are passed to an activation function.

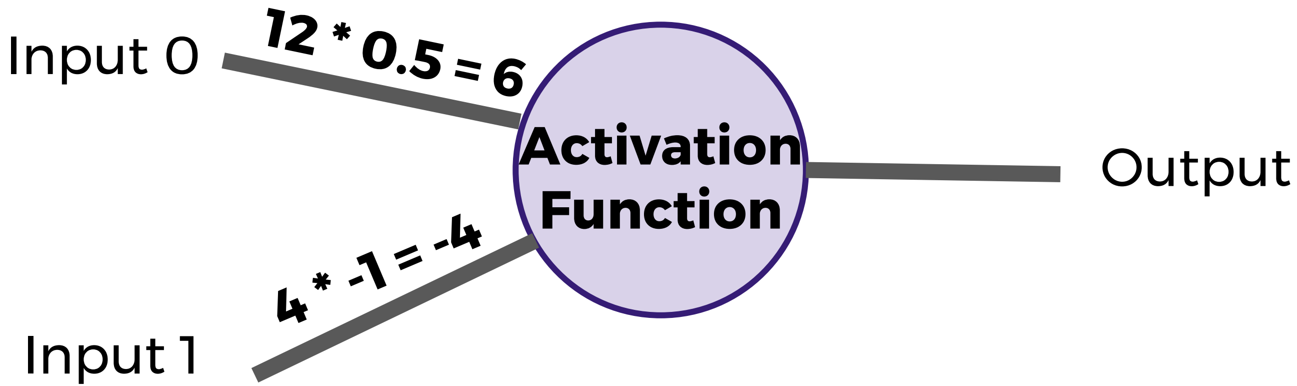


- Many activation functions to choose from, we'll cover this in more detail later!



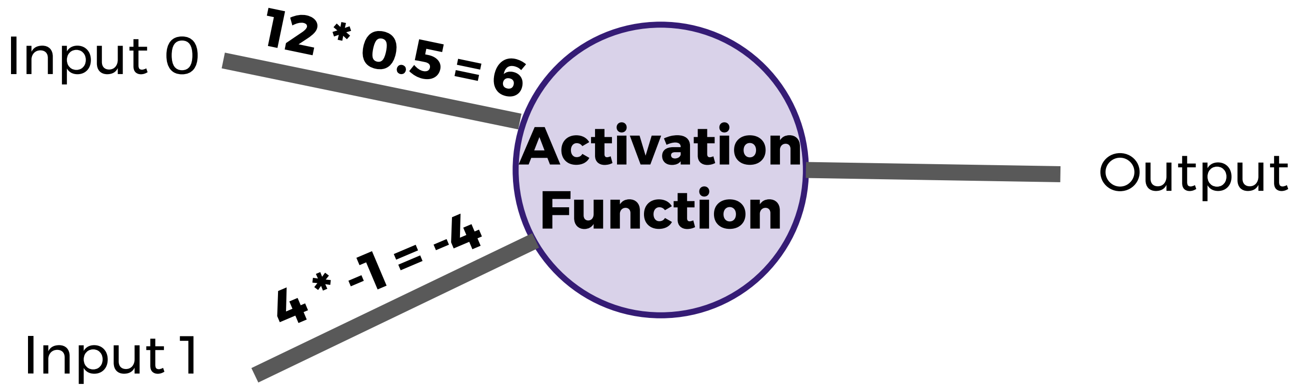
Deep Learning

- For now our activation function will be very simple...



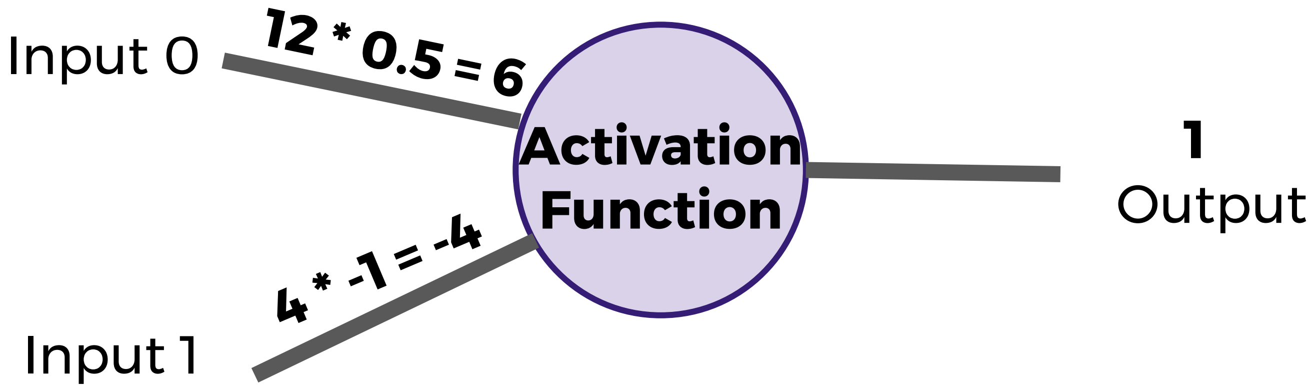
Deep Learning

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

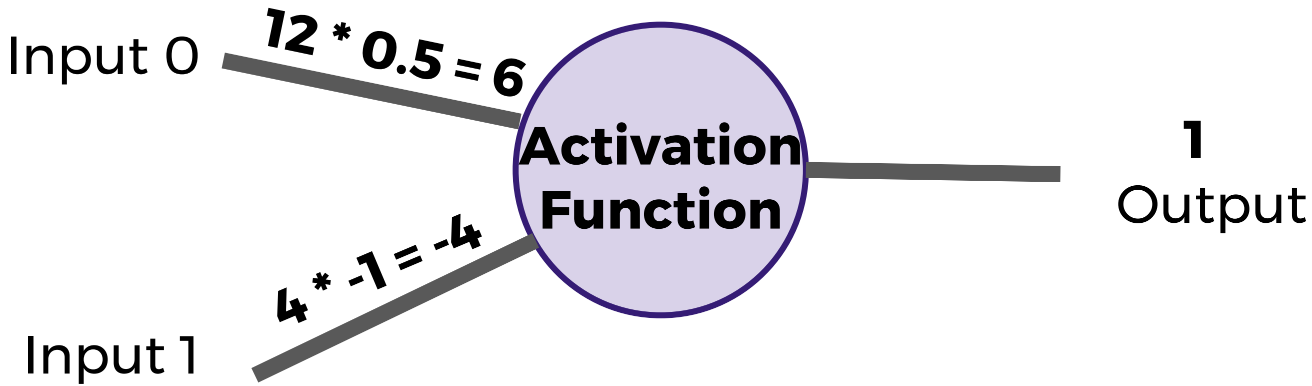


Deep Learning

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

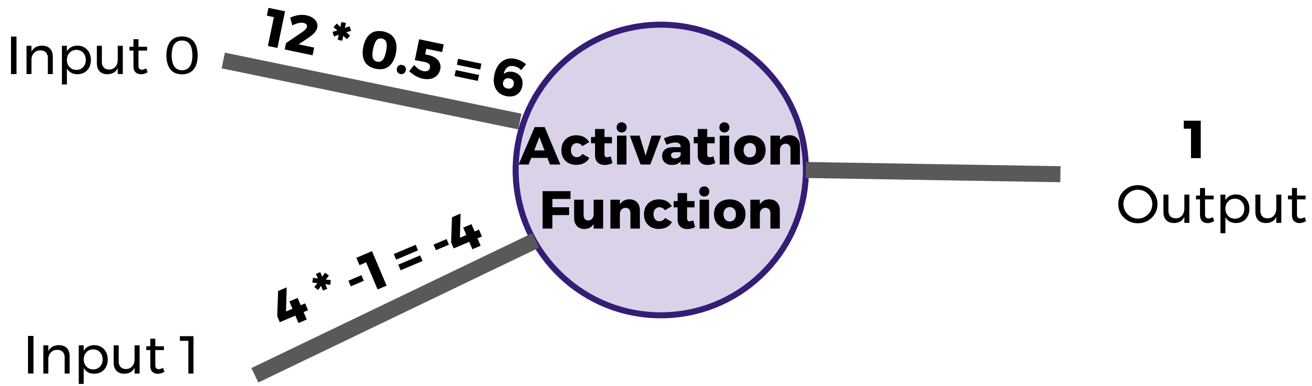


- There is a possible issue. What if the original inputs started off as zero?



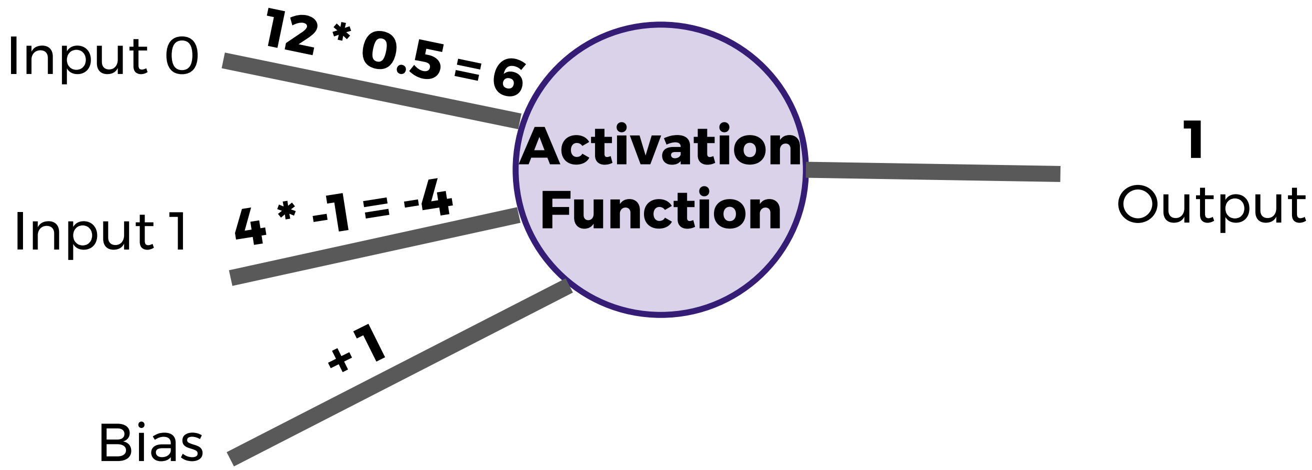
Deep Learning

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



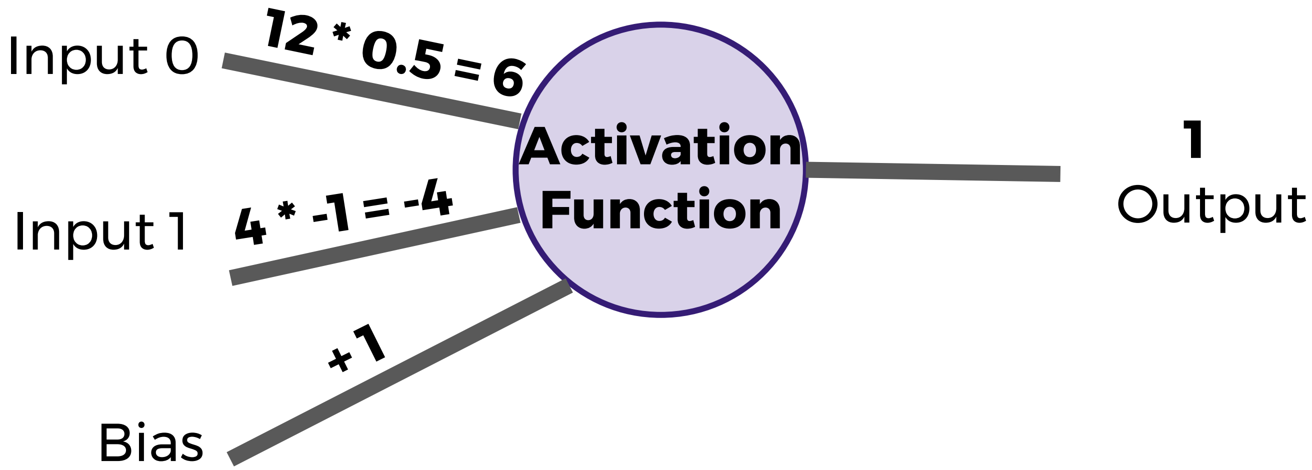
Deep Learning

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



Deep Learning

- So what does this look like mathematically?



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

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

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

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

Deep Learning

- Review
  - Biological Neuron
  - Perceptron Model
  - Mathematical Representation

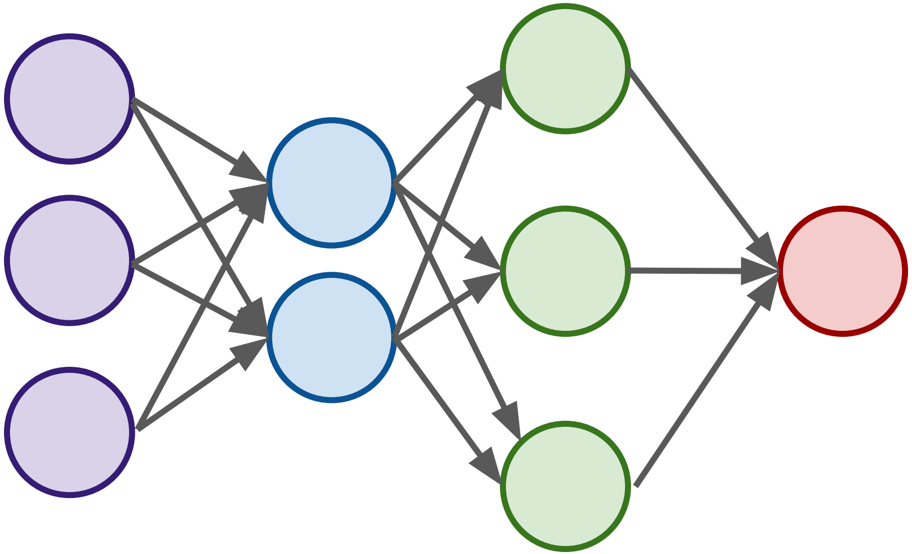
Introduction to Neural Networks

Deep Learning

- 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!

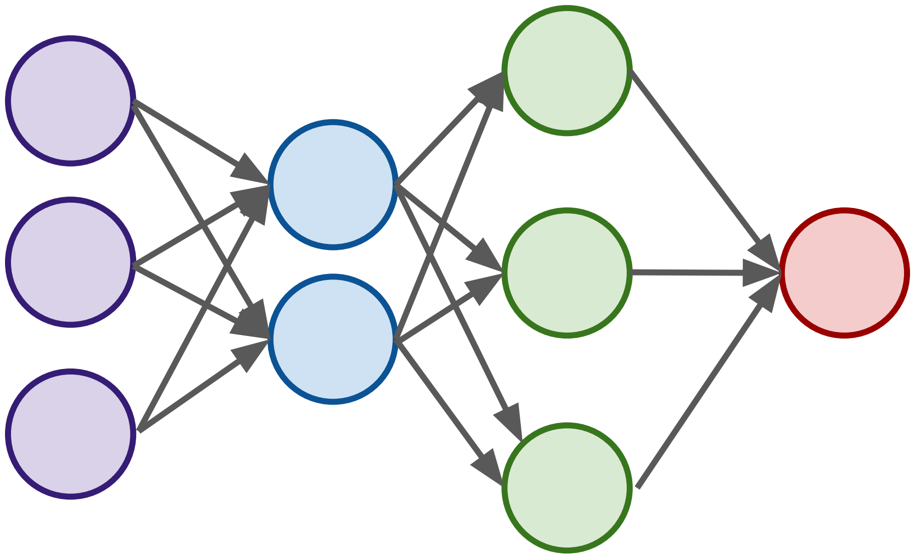
Deep Learning

- Multiple Perceptrons Network



Deep Learning

- Input Layer. 2 hidden layers. Output Layer



Deep Learning

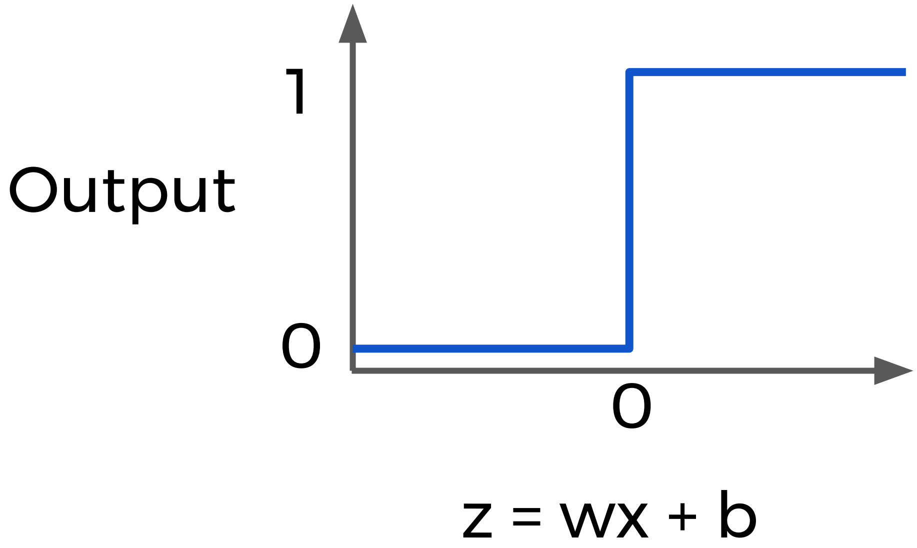
- Input Layers
  - Real values from the data
- Hidden Layers
  - Layers in between input and output
  - 3 or more layers is “deep network”
- Output Layer
  - Final estimate of the output

Deep Learning

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

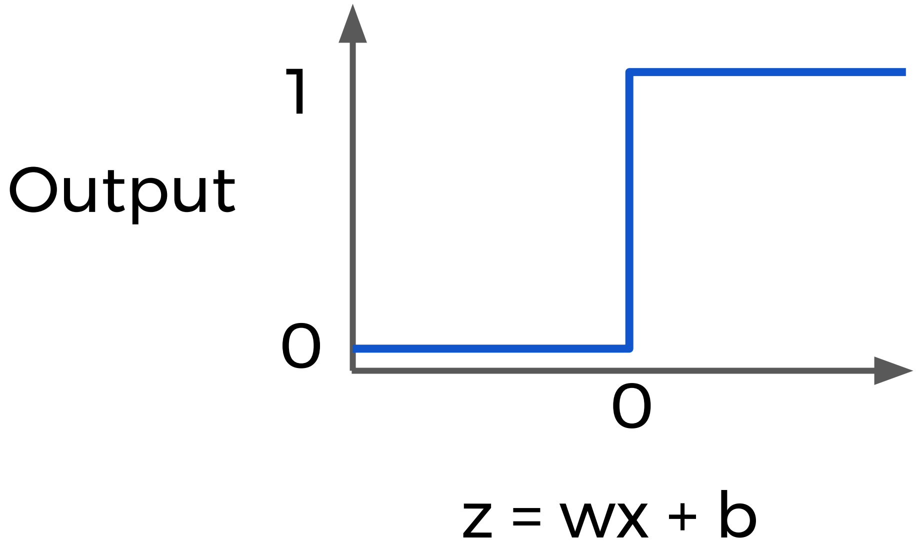
Deep Learning

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



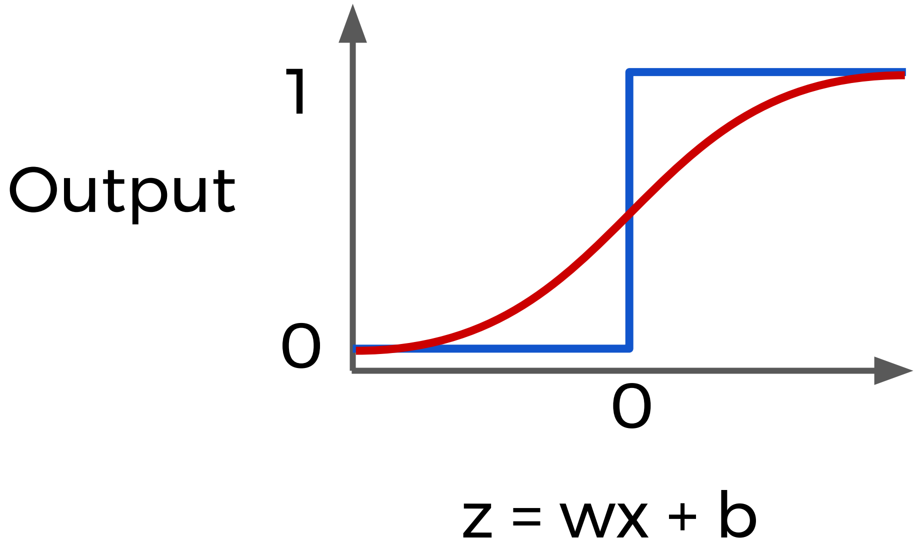
Deep Learning

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

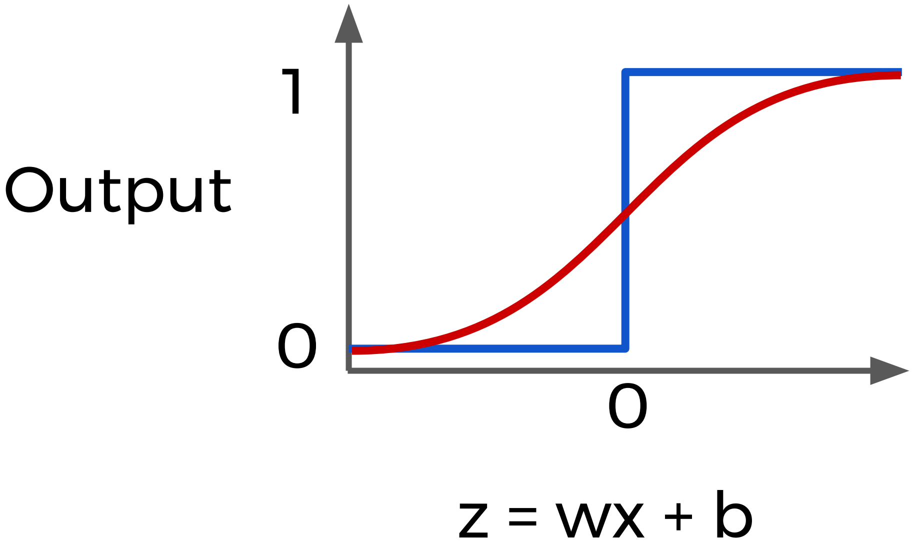


Deep Learning

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

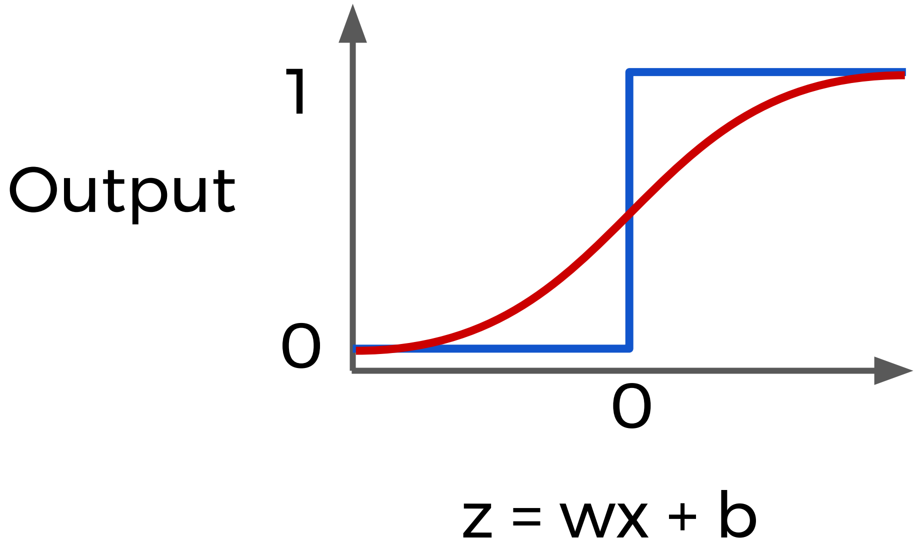


- Lucky for us, this is the sigmoid function!



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

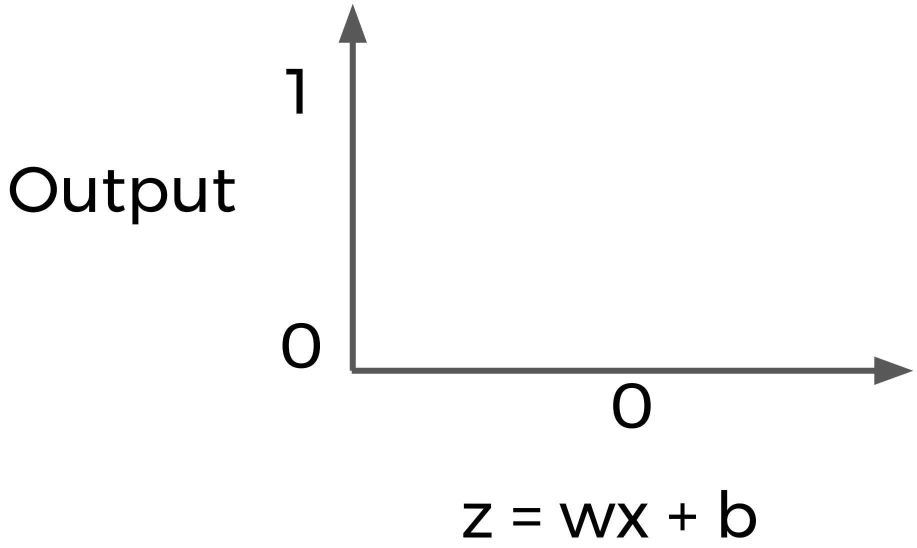
- Changing the activation function used can be beneficial depending on the task!



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

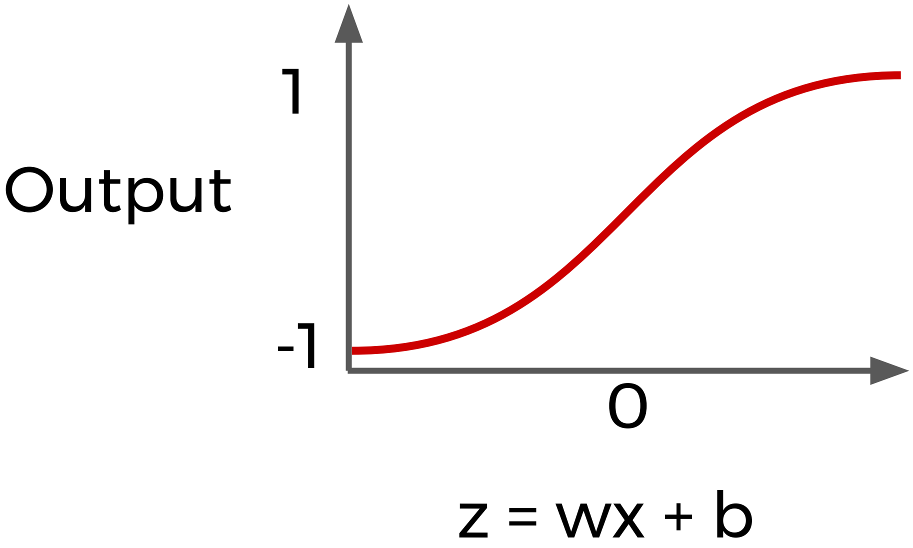
Deep Learning

- Let's discuss a few more activation functions that we'll encounter!



Deep Learning

- Hyperbolic Tangent: tanh(z)



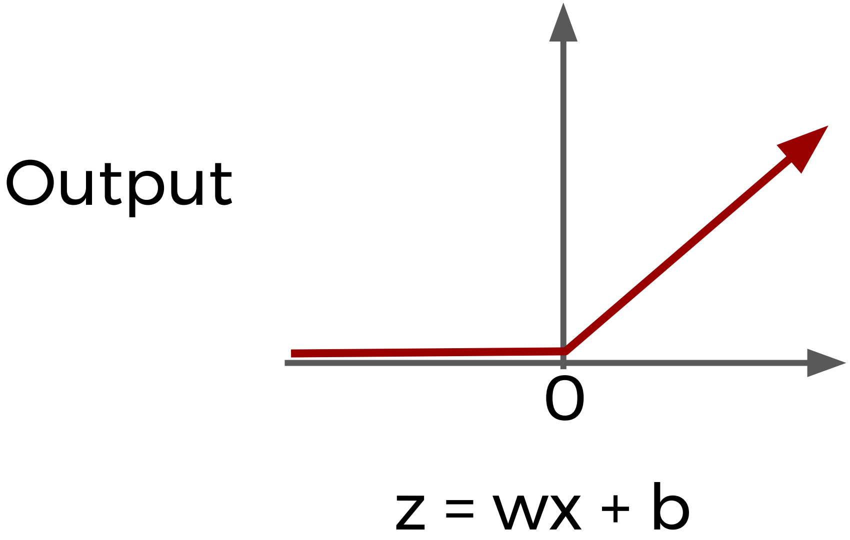
$$\cosh x = \frac{\mathbf{e}^{x} + \mathbf{e}^{-x}}{2}$$

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

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

Deep Learning

- Rectified Linear Unit (ReLU): This is actually a relatively simple function: max(0,z)



Deep Learning

- ReLu and tanh tend to have the best performance, so we will focus on these two.
- Deep Learning libraries have these built in for us, so we don't need to worry about having to implement them manually!

Deep Learning

- As we continue on, we'll also talk about some more state of the art activation functions.
- Up next, we'll discuss cost functions, which will allow us to measure how well these neurons are performing!

Cost Functions

- Let’s now explore how we can evaluate performance of a neuron!
- We can use a cost function to measure how far off we are from the expected value.

Deep Learning

- We’ll use the following variables:
    - y to represent the true value
    - a to represent neuron’s prediction
- In terms of weights and bias:
    - $w*x + b = z$
    - Pass z into activation function $\sigma(z) = a$

Deep Learning

- Quadratic Cost
  - $C = \Sigma(y-a)^2 / n$
- We can see that larger errors are more prominent due to the squaring.
- Unfortunately this calculation can cause a slowdown in our learning speed.

Deep Learning

- Cross Entropy
    - $C = (-1/n) \sum (y \cdot \ln(a) + (1-y) \cdot \ln(1-a))$
- This cost function allows for faster learning.
- The larger the difference, the faster the neuron can learn.

Deep Learning

- We now have 2 key aspects of learning with neural networks, the neurons with their activation function and the cost function.
- We’re still missing a key step, actually “learning”!

Deep Learning

- We need to figure out how we can use our neurons and the measurement of error (our cost function) and then attempt to correct our prediction, in other words, “learn”!

Deep Learning

* In the next lecture we'll briefly cover how we can do this with Gradient Descent!

Gradient Descent and Backpropagation

PIERIAN DATA

Deep Learning

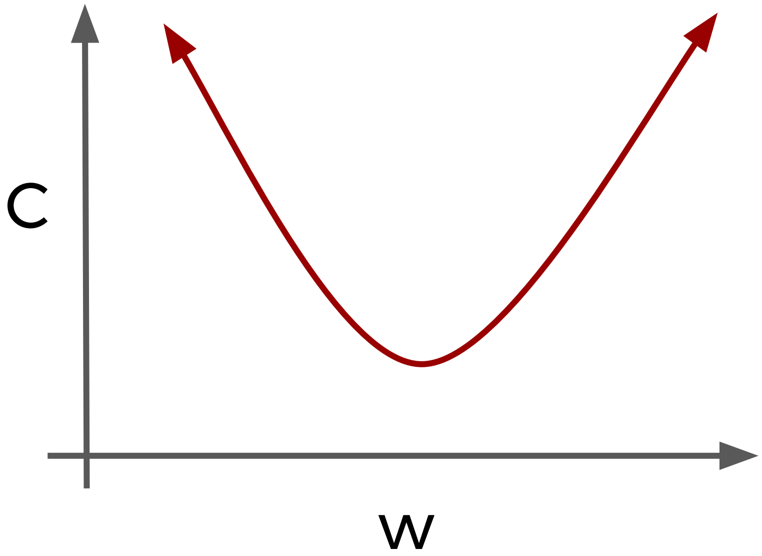
- If you've dabbled in machine learning before, you may have already heard of Gradient Descent!
- Let's quickly go over it with a high level overview!

Deep Learning

- Gradient descent is an optimization algorithm for finding the minimum of a function.
- To find a local minimum, we take steps proportional to the negative of the gradient.

Deep Learning

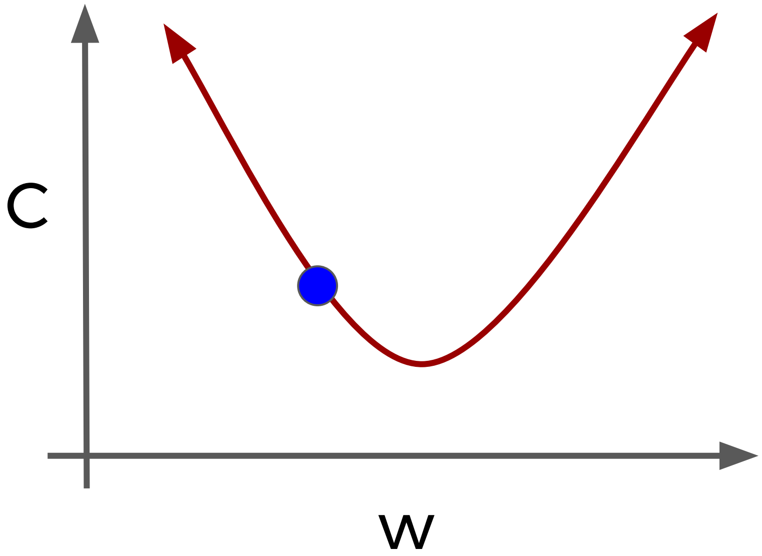
- Gradient Descent (in 1 dimension)



PIERIAN DATA

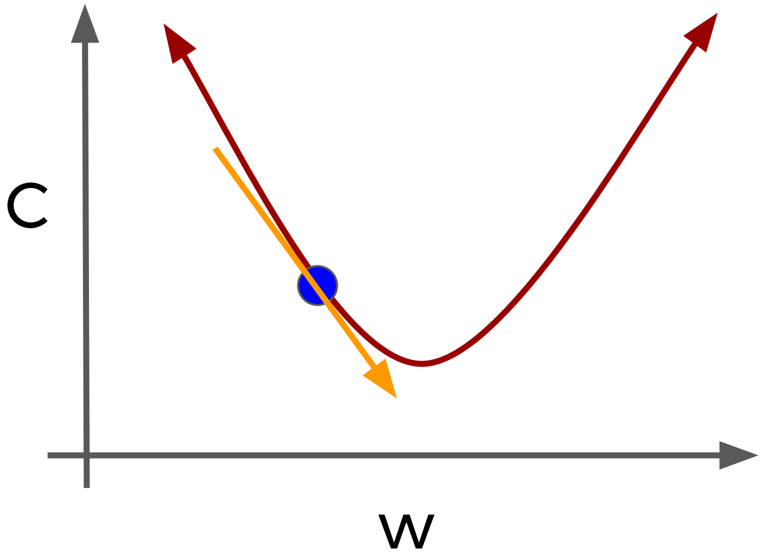
Deep Learning

- Gradient Descent (in 1 dimension)



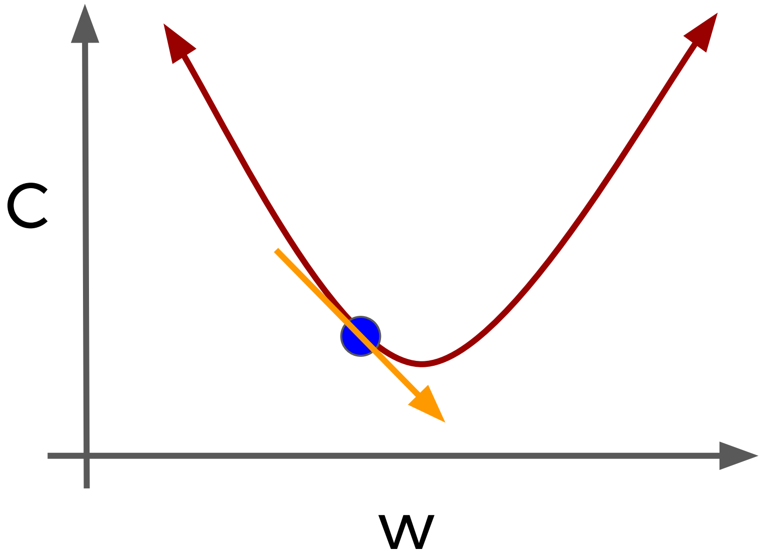
Deep Learning

- Gradient Descent (in 1 dimension)



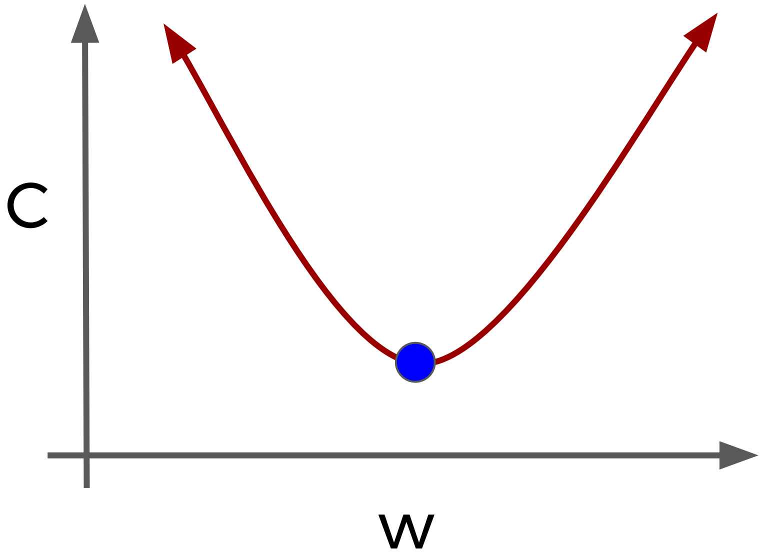
Deep Learning

- Gradient Descent (in 1 dimension)



Deep Learning

- Visually we can see what parameter value to choose to minimize our Cost!



- Finding this minimum is simple for 1 dimension, but our cases will have many more parameters, meaning we'll need to use the built-in linear algebra that our Deep Learning library will provide!

Deep Learning

- Using gradient descent we can figure out the best parameters for minimizing our cost, for example, finding the best values for the weights of the neuron inputs.

Deep Learning

- We now just have one issue to solve, how can we quickly adjust the optimal parameters or weights across our entire network?
- This is where backpropagation comes in!

Deep Learning

- Backpropagation is used to calculate the error contribution of each neuron after a batch of data is processed.
- It relies heavily on the chain rule to go back through the network and calculate these errors.

Deep Learning

- Backpropagation works by calculating the error at the output and then distributes back through the network layers.
- It requires a known desired output for each input value (supervised learning).

- The implementation of backpropagation will be further clarified when we dive into the math example!
- For now let's finish off our high level discussion with TensorFlow's playground!

TensorFlow Playground

- Go to:
    - playground.tensorflow.org

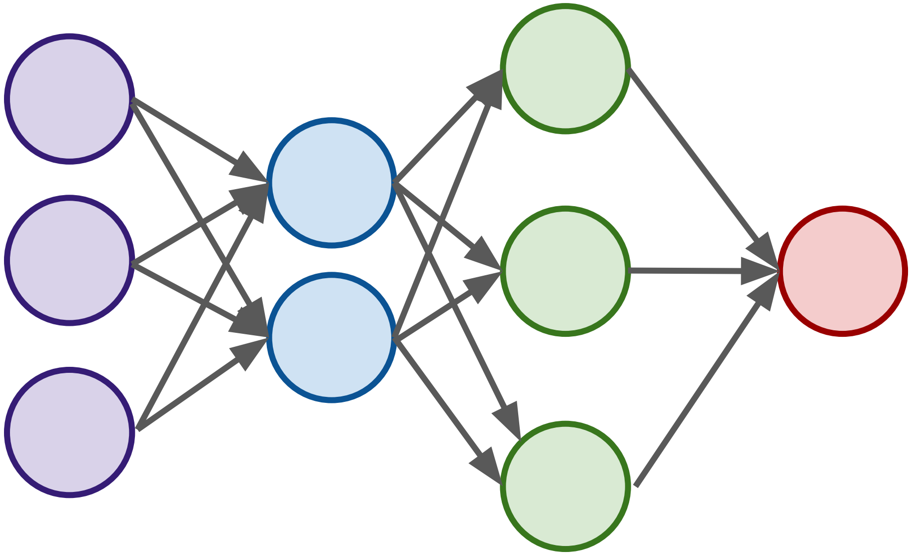
Types of Networks

Deep Learning

- Let's discuss high level overviews of the various types of neural networks
  - Dense Networks
  - Convolutional Networks-CNN
  - Recurrent Neural Networks-RNN
  - Generative Adversarial Networks-GAN

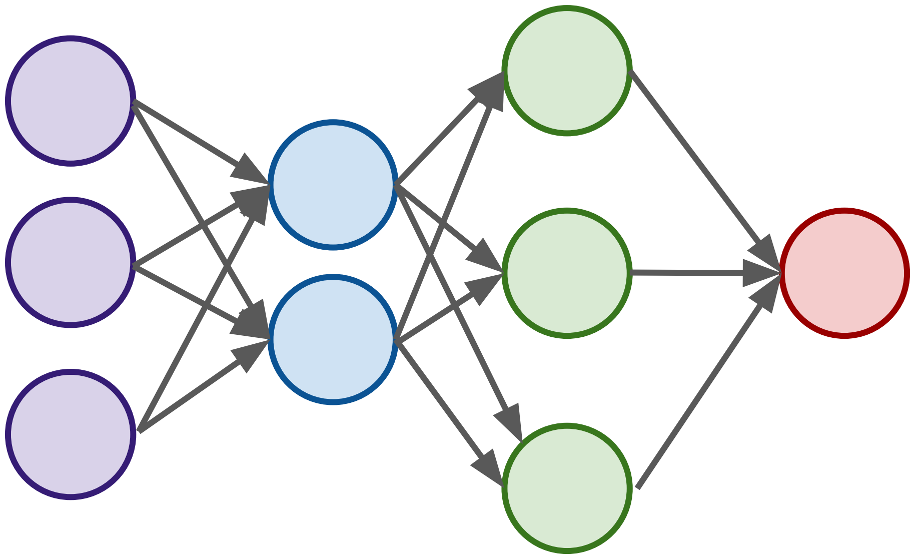
Deep Learning

- Dense Networks



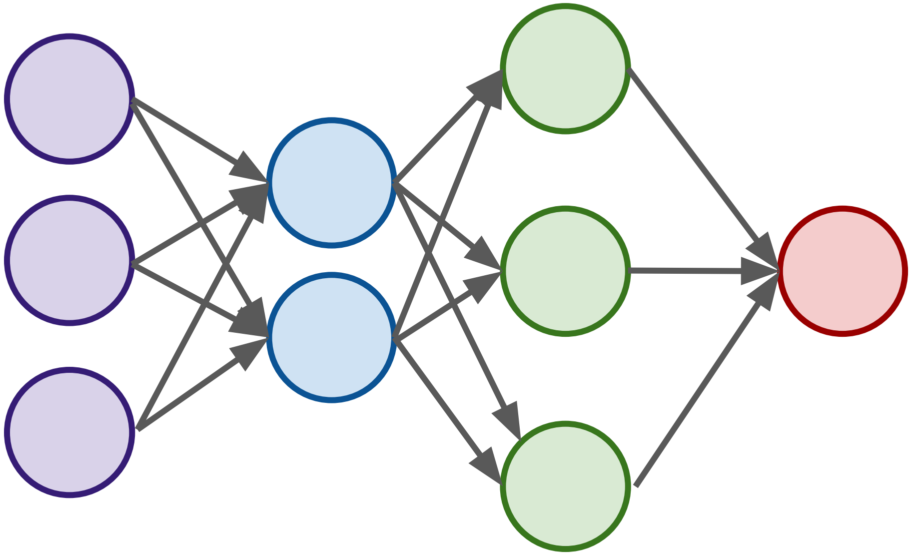
Deep Learning

- Convolutional Neural Networks



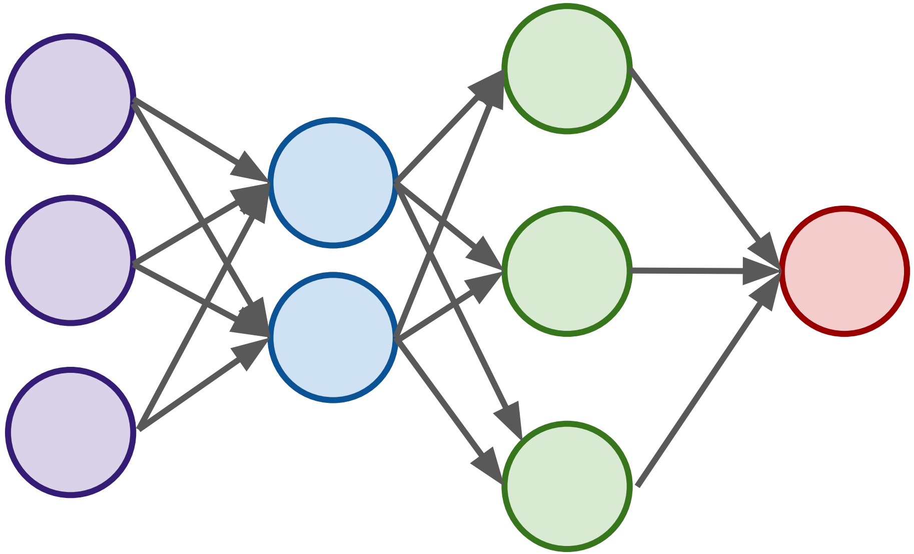
Deep Learning

- Recurrent Neural Networks



Deep Learning

- Generative Adverserial Neural Networks

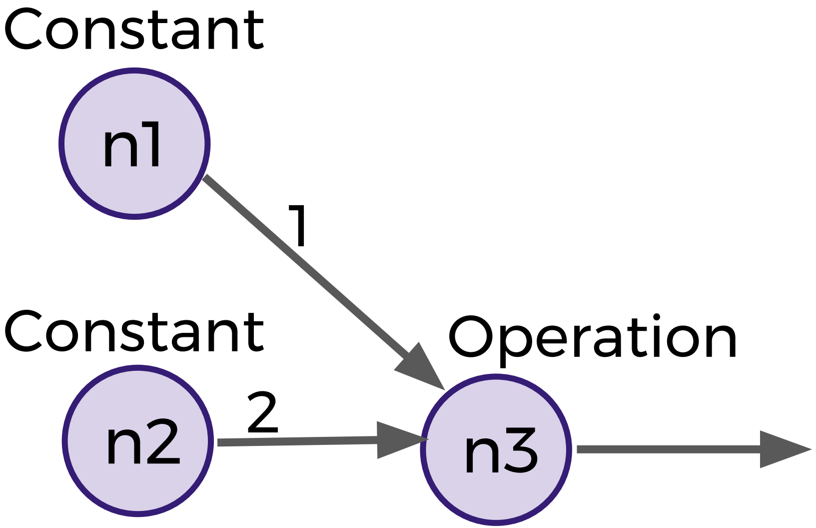


Manual Neural Network
Part 2 - Operation

Deep Learning

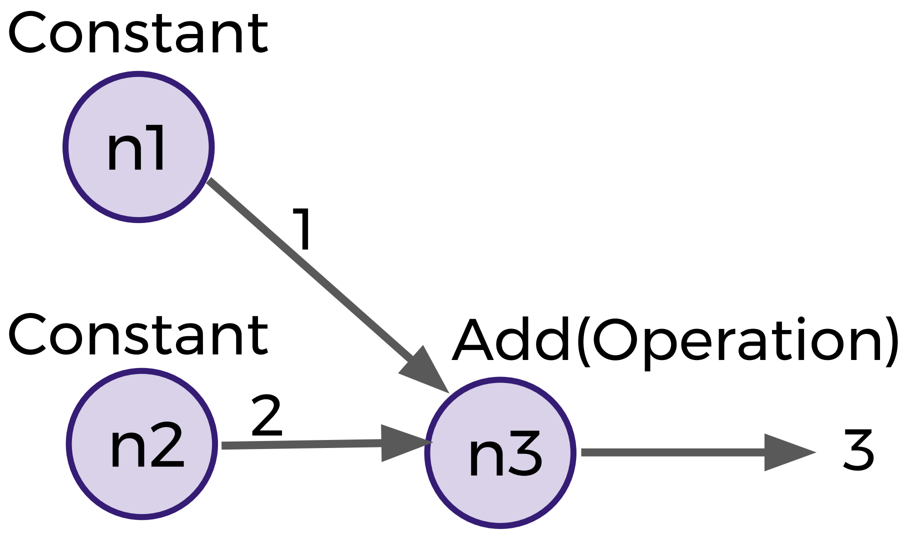
- Operation Class
    - Input Nodes
    - Output Nodes
    - Global Default Graph Variable
    - Compute
        - Overwritten by extended classes

- Graph - A global variable



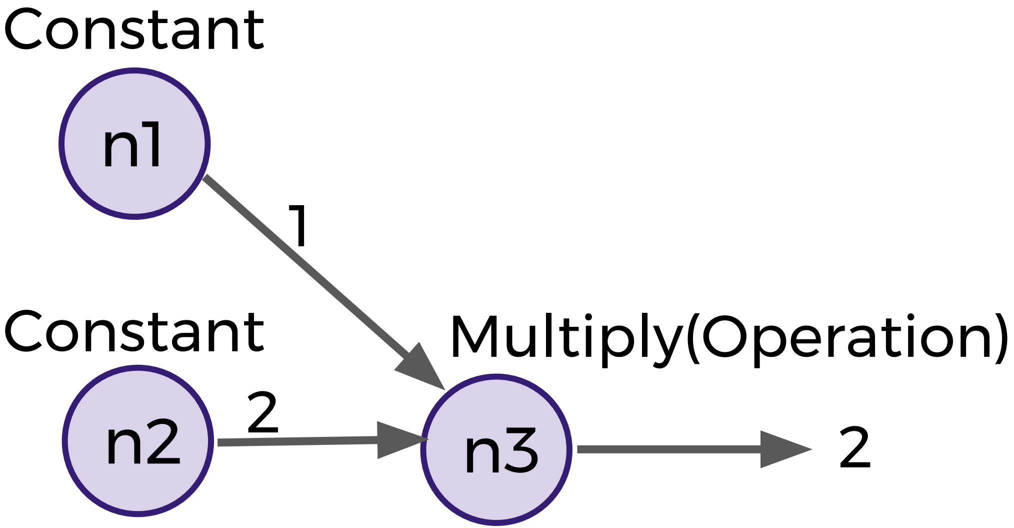
Deep Learning

- Graph



Deep Learning

- Graph



Manual Neural Network Variables,Placeholders, and Graphs

Deep Learning

- Placeholder - An “empty” node that needs a value to be provided to compute output.
- Variables - Changeable parameter of Graph
- Graph - Global Variable connecting variables and placeholders to operations.

Let’s get started!

Manual Neural Network Session

- Now that the Graph has all the nodes, we need to execute all the operations within a Session.
- We’ll use a PostOrder Tree Traversal to make sure we execute the nodes in the correct order.

Manual Neural Network Classification

Deep Learning

- y = mx + b
- y = -1x + 5
- Remember that both y and x are features!
- Feat2 = -1*Feat1 + 5
- Feat2 + Feat1 - 5 = 0
- FeatMatrix[1, 1] - 5 = 0