## 5 intro to ANN:

**part 1)**

the artificial neuron network is called Perceptron.

Like real biological neuron system, each Perceptron has inputs and outputs.

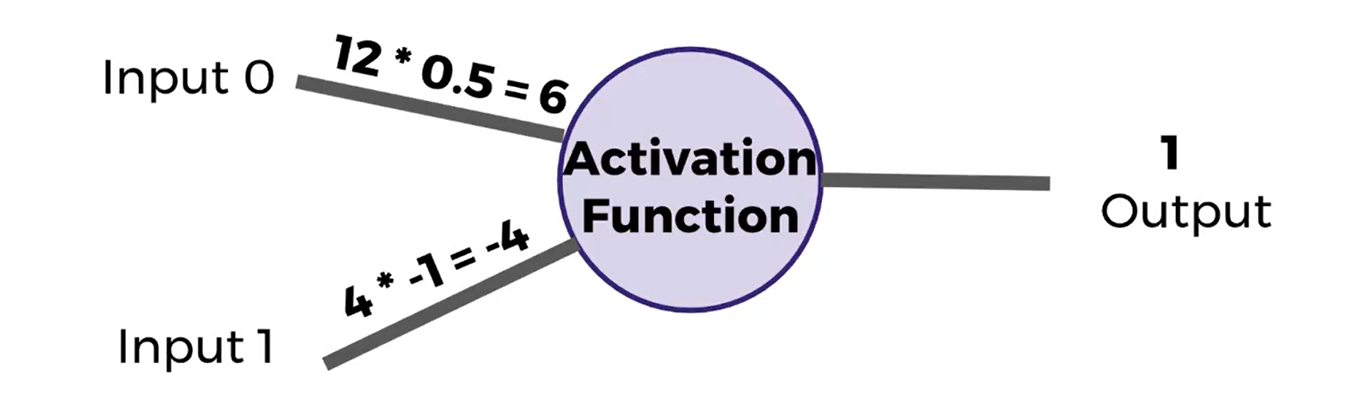
The process is simple:

-first we assign features to inputs.

- then we assign weights for each input. Wights are assigned by random numbers.

- then each input values gets multiplied by the associated weight.

- the node is the Activation function.



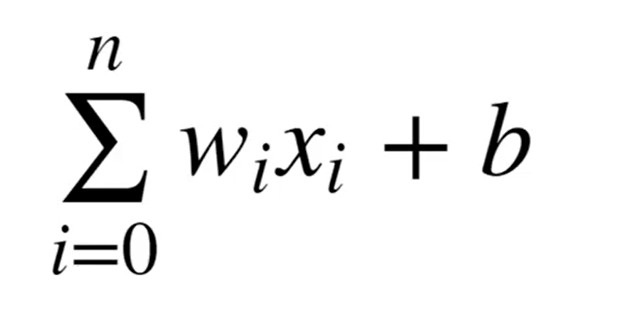
- the activation function returns 0 if the sum of the inputs is negative and returns 1 if the sum of the inputs is positive.

- in this case 6-4=2 ===> then the output will be 1 because 2 is positive.

- if one of the inputs ends up being zero then everything including output will be zero. In order to prevent this we need to add another input called Bias.

- the perceptron model is:

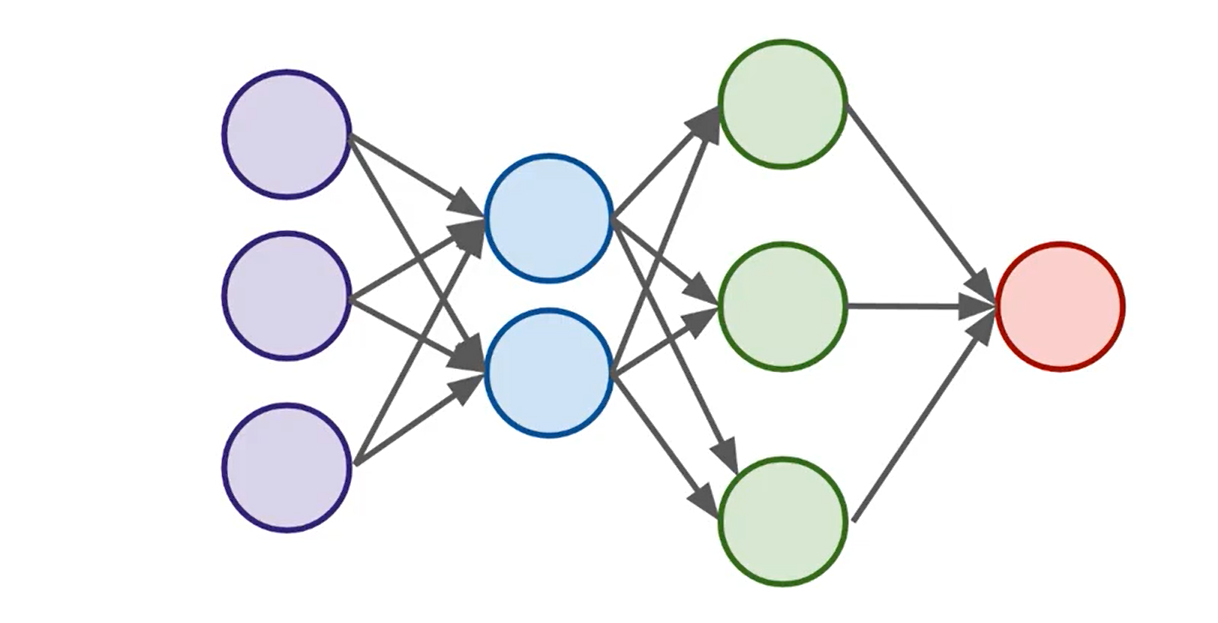
each input times the weight and after summing them up we add bias to it.



**Part 2)**

now lets connect many perceptrons together and see how it works mathematically.

It looks like this :



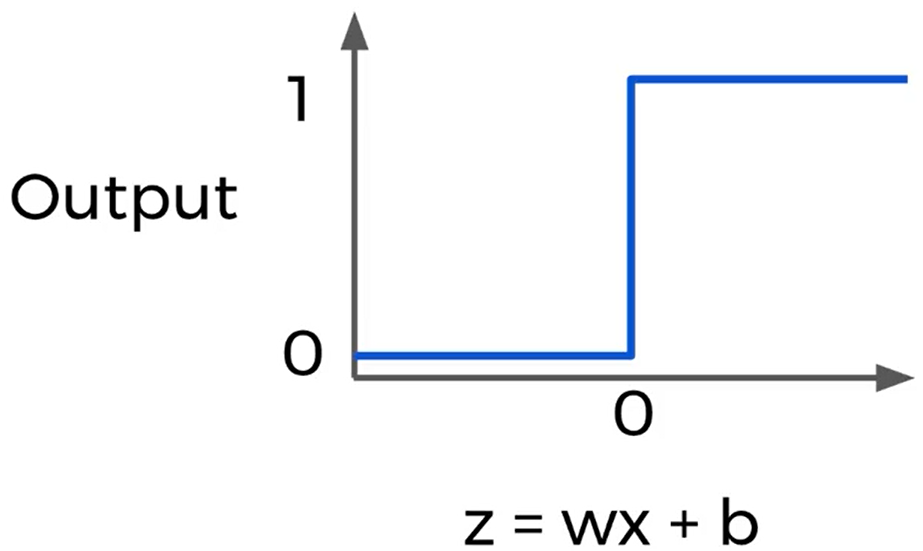
the purple ones are input layer, blue and green hidden layers, and red ones output layer.

- input layers get the actual data.

- if we have 3 or more layers we consider it deep network.

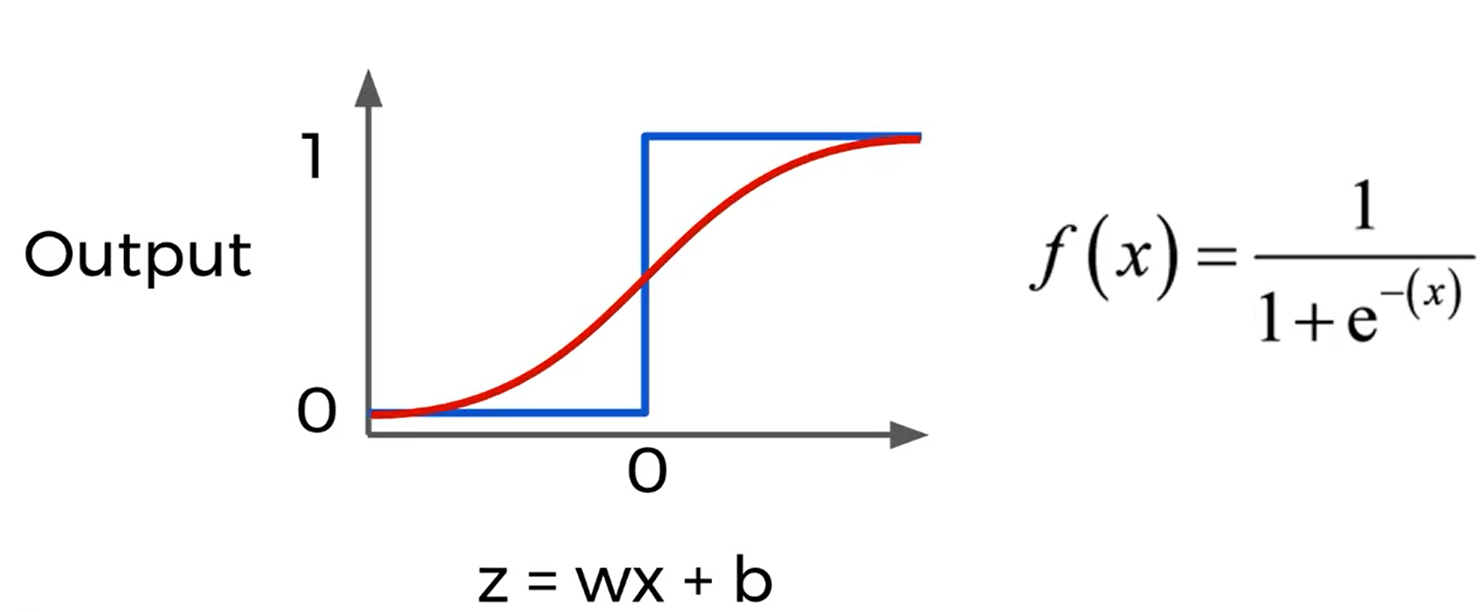
- output layer is the final estimation of the values.

**Activation function works like this:**



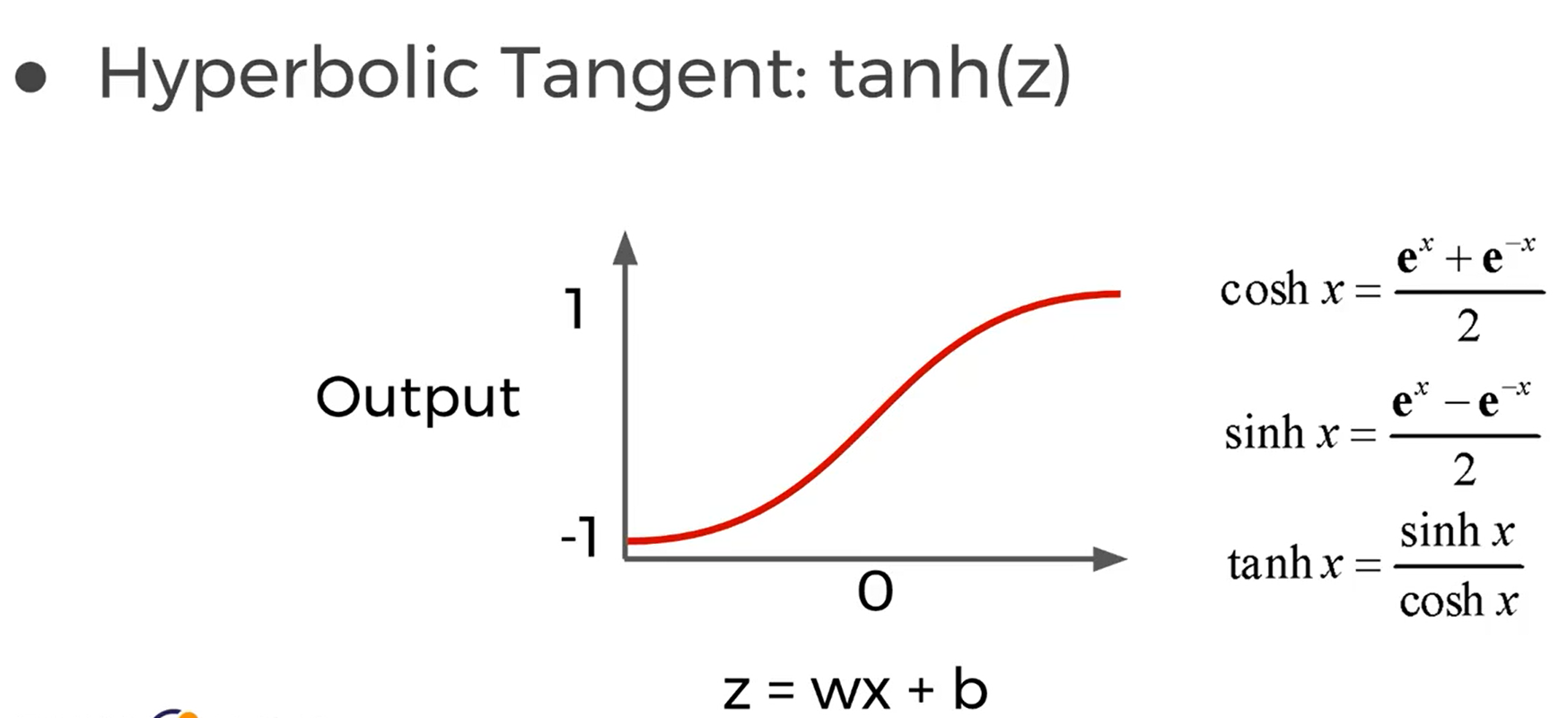
because the function is discrete, we need to make it exponential to give us more info for values between 0-1 not looking like steps.

We can use Sigmoid function:



sigmoid is only one of the activation functions we need to use.

There are others such as:

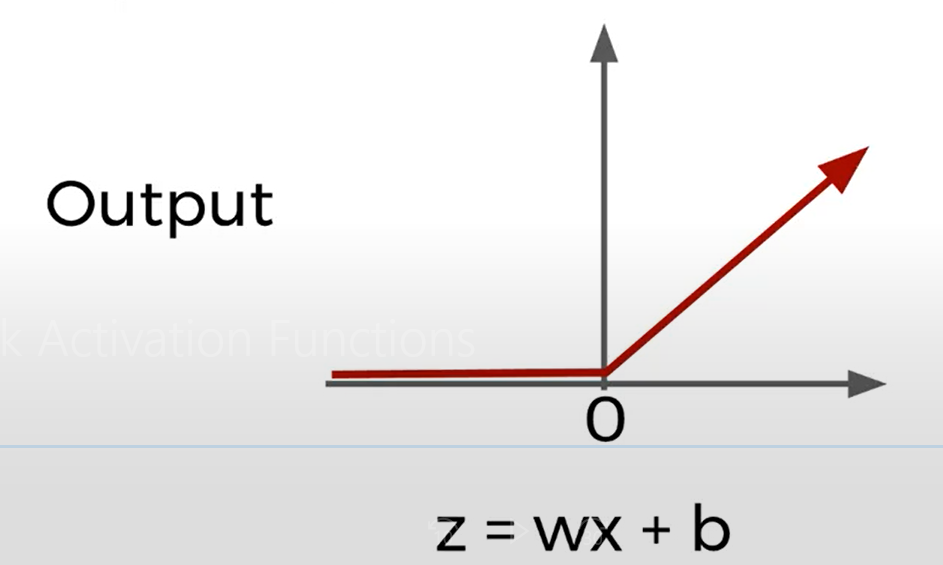


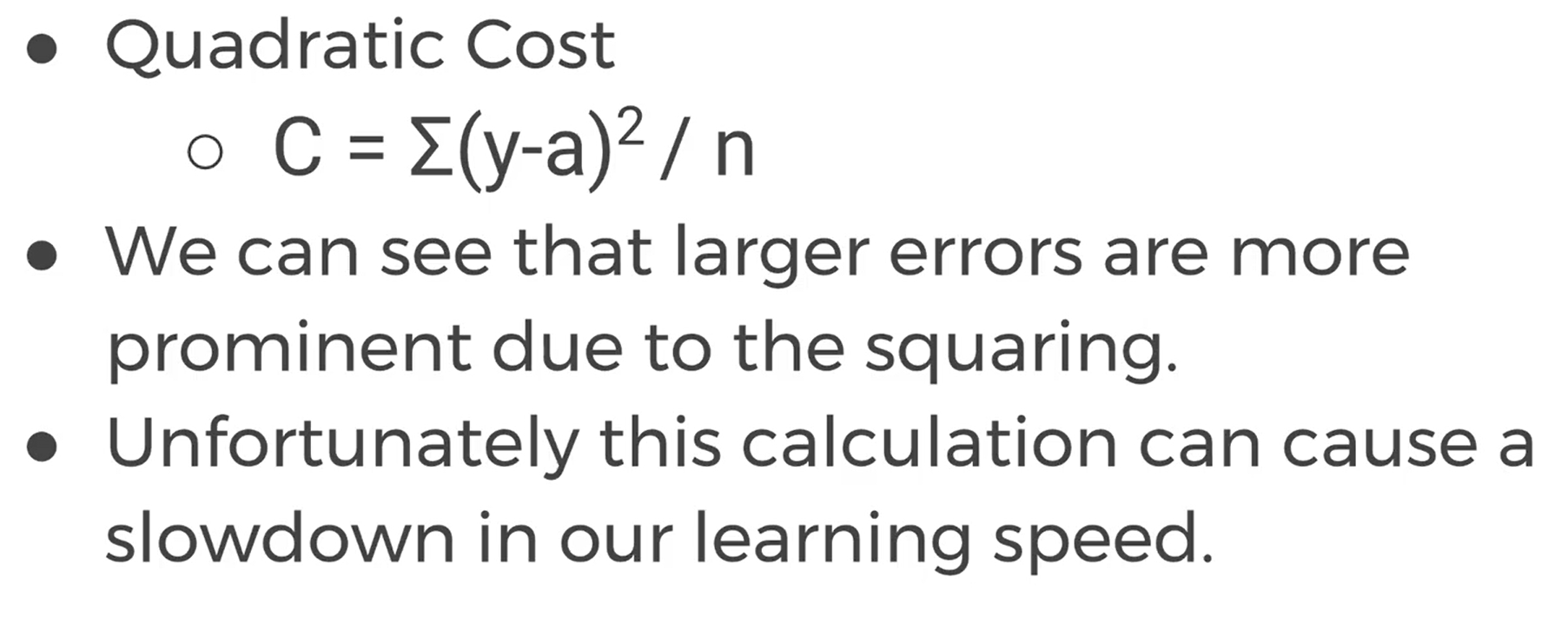
sigmoid function has y between (0-1) while in hyperbolic tangent we have it between (-1,1)

Another activation function is called ReLU:

(Rectified Linear Unit)

- its so simple yet so common to use. It says between the value of z and 0 return the max. (for negative values it always returns 0)



these 2 functions are built-in with deep learning libraries and we just use them.

**Part 3)**

the cost function shows us how off we are from the actual value.

In order to explain cost function there are some variables:

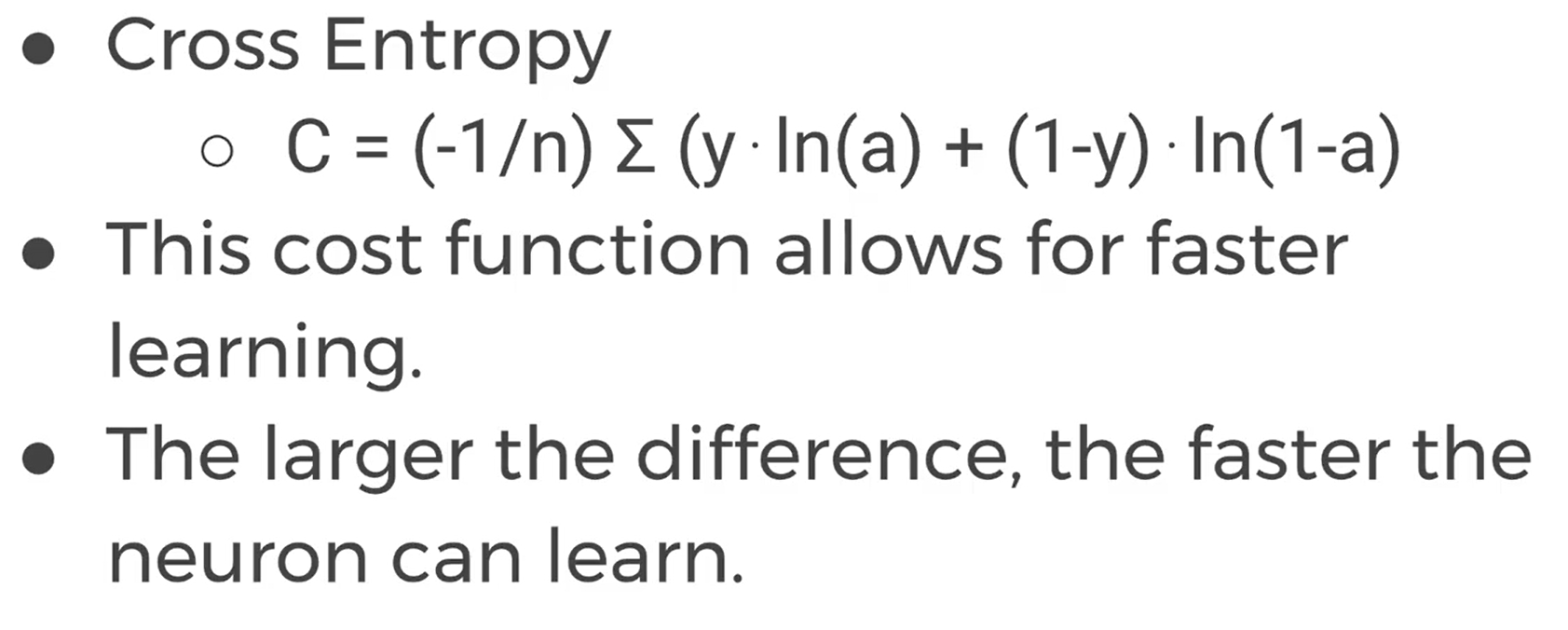
y: actual vlaue

a: neuron’s prediction

W\*x + b = z ==> sigmoid(z) = a

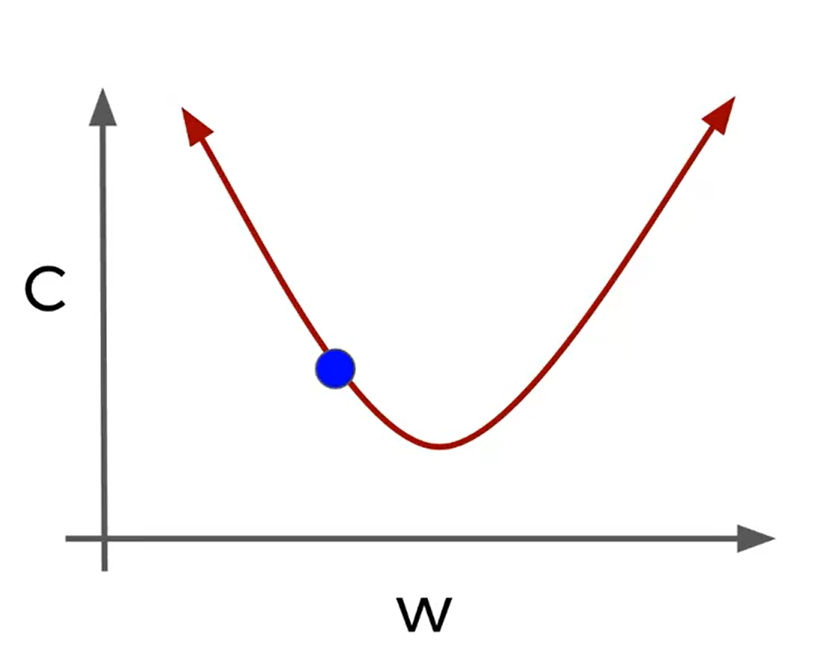
we have different cost functions:

another type of cost function is:

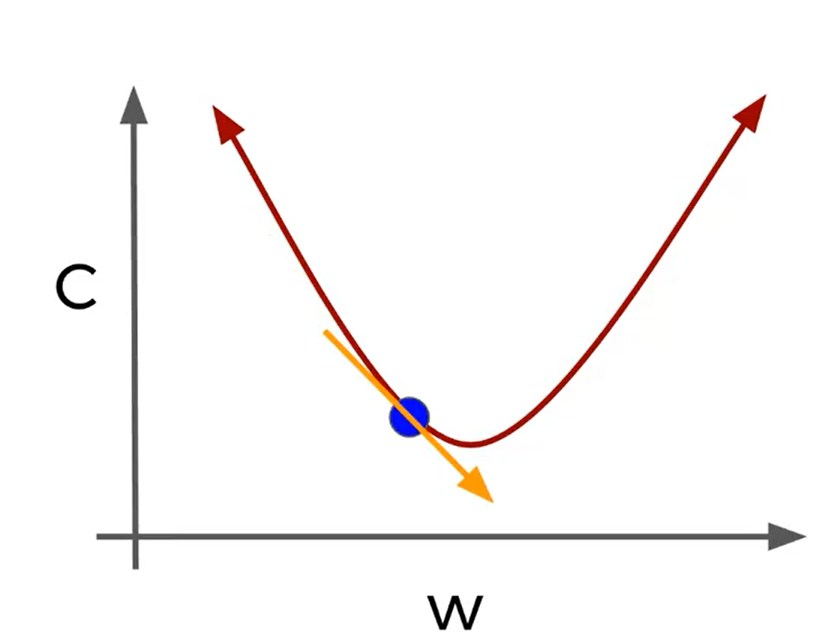


**Gradient Descent:**

in order to learn and improve from the cost function we need to use a technique called Gradient descent. It is an optimization algorithm to find the minimum of a function.



C is the cost and w is weight. First we chose a random weight( the blue point)

the algorithm keeps decreasing the weight until the c gets minimum.

For one dimension topics we can easily visualize it but when we get big data with tens of dimensions we can’t use simple algebra and visualization, so we need to use linear algebra libraries.

Based on gradient descent we can minimize the cost but we need another technique to adjust all weights and re run the model. We should use Backpropagation technique.

Backpropagation calculates the error contribution of each neuron.

**Part 4)**

Here is a great website for learning Tensorflow concepts:

[https://playground.tensorflow.org](https://playground.tensorflow.org/)

**Part 5)**

Manually making the neural network in Python:

the format for classes in python:

class <class\_name>():

def \_\_init\_\_(self):

(all similar to php and \_\_construct() method.)

when we want to extend a class (inherit):

class ExtendedClass(<other\_class\_name>):

def \_\_init\_\_(self):

-when we extend a class we don’t access to its init method anymore but we can access all other methods from the new class.

- the way to inherit also the init function of the first class is to add this to the second class:

super().\_\_init\_\_()

**part 6)**

we want to manually make a simple ANN manually,.

First we need to make Operation class:

- input nodes

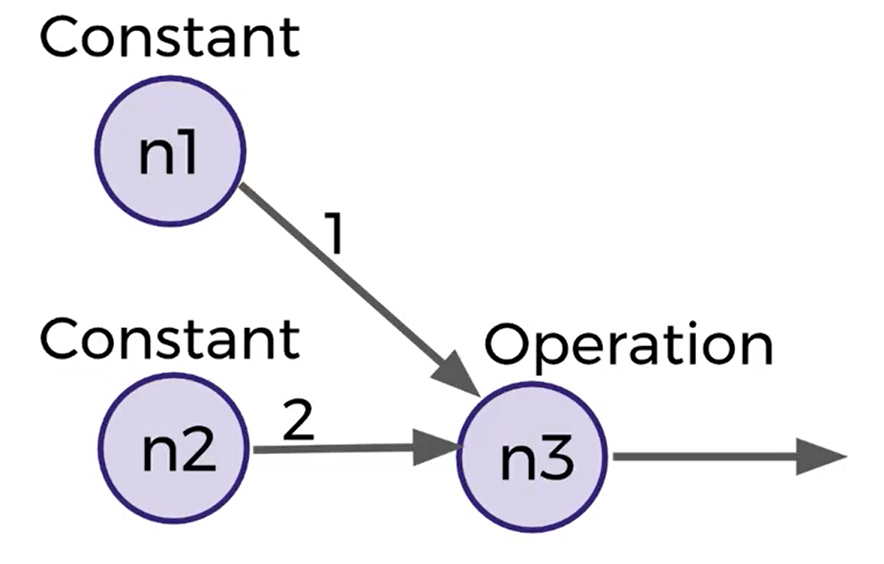
- output nodes

- global default graph variable

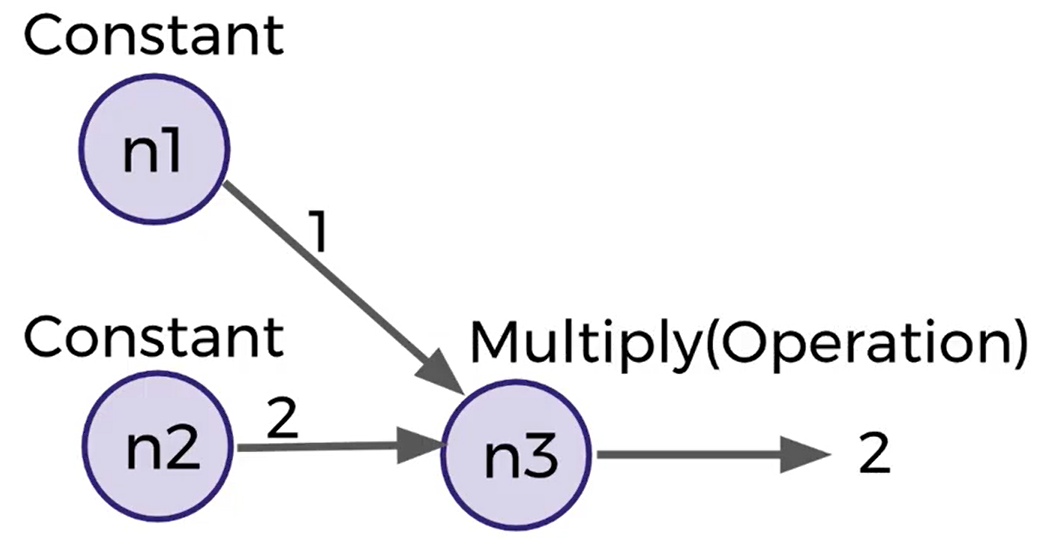
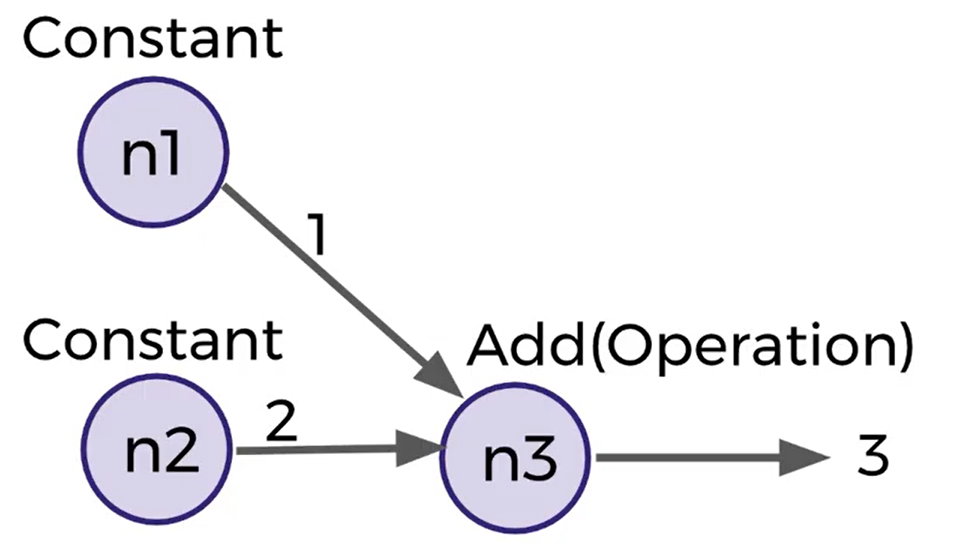
- compute

A graph is a global variable that defines a list of nodes.

In this example we will have the following graph:



when we make this skeleton, it will be inherited by different operation classes (add, subtract, etc.)



matmul in tf means matrix multiplication.

.dot() function is used in numpy for multiplying matrices.

**Part 7)**

**Variables, placeholders and graphs.**

**Placeholder:** is an empty node that needs a value to be provided to compute output.

**Variables:** changeable parameters for Graph.

**Graph:** global variable connecting variables and placeholders to operations.

**Part 7)**

after defining all the nodes and variables, etc. we need to execute all the operations within a session.

In order to make sure all the operations are done in correct order, we use **PostOrder Tree Traversal**.

Copy-pasted some code from the resource.

**Part 8)**

classification, activation function

numpy.linspace() returns evenly distributed numbers between 2 numbers.

Numpy.linspace(-10,10,100) returns 100 numbers evenly distributed between -10,10.

sklearn has a very nice library called make\_blob that makes fake datasets.

from sklearn.datasets import make\_blobs

data = make\_blobs(n\_samples=50, n\_features=2, centers=2, random\_state=75)