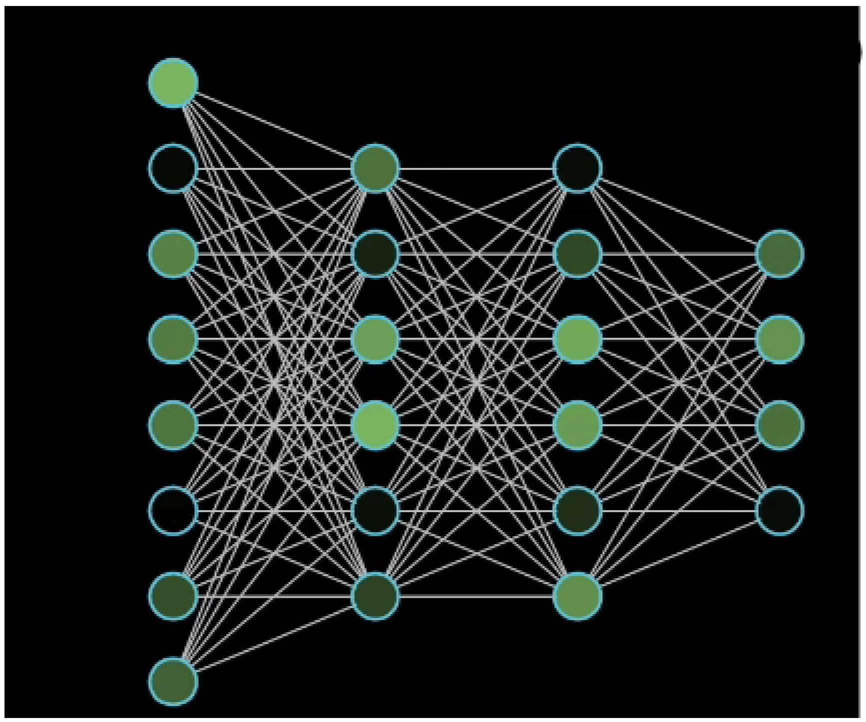
NEURAL NETWORKS

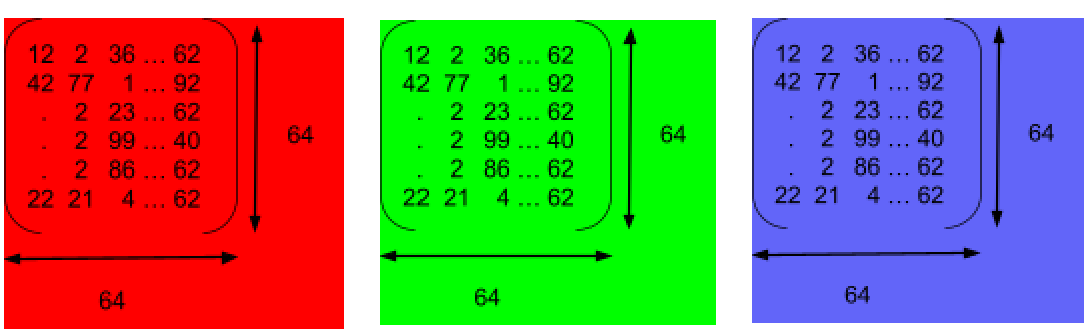


Neural networks are layered structures with the next layer extracting and giving more information towards the output and the final layer spits out the output.

The middle layers are called hidden layers.

BINARY CLASSIFICATION

When we use neural networks for image recognition, it takes the input of the image as its RGB (Red Blue Green) matrix.
So if the image for example is a 64x64 image then:



This is how the three matrices are read.
Then the entries in these matrices are put in columns
Let that matrix be x
Then:

x= $\begin{bmatrix} 12 \\ 2 \\ 36 \\ . \\ . \\ . \\ 62 \\ 42 \\ 77 \\ 1 \\ . \\ . \\ . \\ 92 \\ . \\ . \\ . \end{bmatrix}$ R G B The dimension of this matrix is n_x x 1

Where n_x in this case is 64 x 64 x 3 = 12288

This x is the feature vector of the image and this is used to classify whether the output is 1 (required image) or 0.

Notations are input is x and output is y

x$\in R^{n_x}$ and y$\in$(1,0)

Let there be m training examples { $(x_1, y_1), (x_2, y_2), (x_3, y_3) \dots (x_m, y_m)$ }
Where x and y are the inputs and outputs of the training example respectively.
So now we take the training examples in a more compact notation,
We'll write

X= $\begin{bmatrix} | & | & | & | & & | \\ x_1 & x_2 & x_3 & x_4 & \cdots & x_m \\ | & | & | & | & & | \end{bmatrix}$

Here we take the RGB matrix (Feature matrix) of all the inputs in one matrix

This matrix will therefore have m columns and n_x rows.

*Note: We did not take them in rows as it is easier to use them when in columns.

So here $X \in R^{n_x \times m}$

So in python the command X.Shape will give us the the dimensions
.i.e (n_x, m)

To make neural networks easier the outputs are also stacked in columns and the matrix Y is given by:

Y= $[y_1 \ y_2 \ y_3 \ldots \ y_m]$
This is a 1 x m matrix

Therefore, $Y \in R^{1 \times m}$

LOGISTIC REGRESSION
Let there be an input image with feature matrix x

Now, we need to find out the probability $\hat{y}$ which is the output of our program, the estimate of y.

So ŷ gives us the change of what is the probability that the given input is the required image.

We know that x$\in R^{n_x}$ which is the input

Let there be a parameter w$\in R^{n_x}$ and b$\in$R

Therefore we can write the output in terms of parameter and the input as

ŷ = $w^T x$+ b

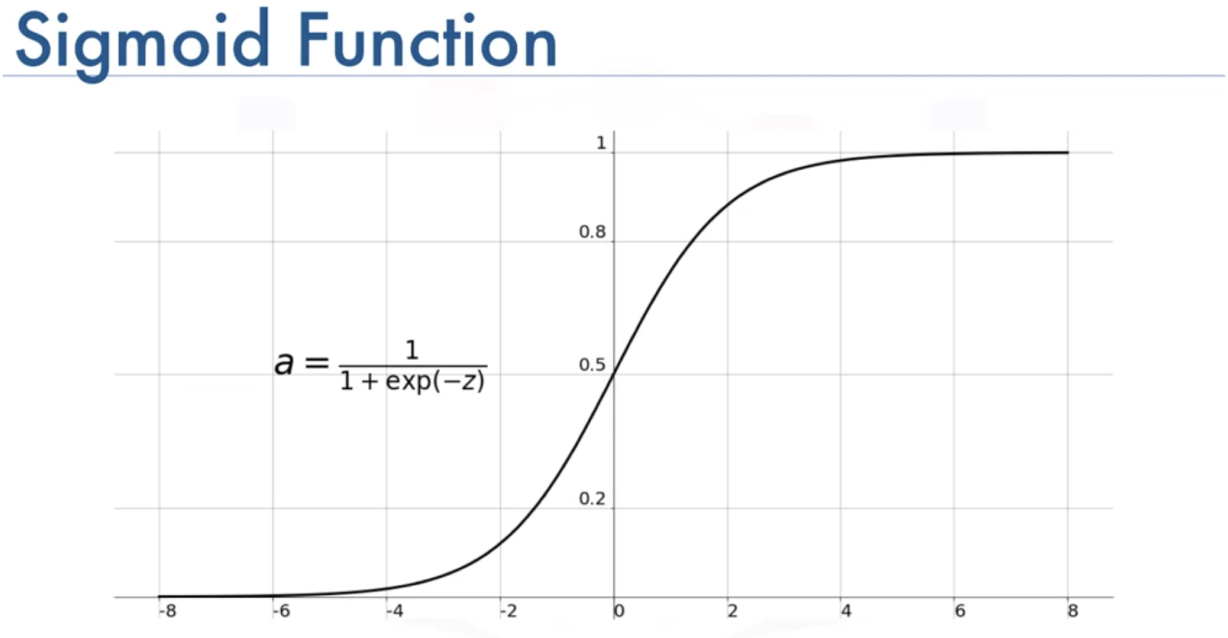
This is a type of linear regression and is not a good algorithm for binary classification as we want ŷ but

$w^T x$+ b can be very large or negative

So we use a sigmoid function:

ŷ = $\sigma(w^T x + b)$

σ(z) = $1/(1 + e^{-z})$



When we use parameters in neural networks we should keep the parameters w and b separate rather than write them in one matrix as that is a good practice.

LOGISTIC REGRESSION COST FUNCTION

To determine the parameters for our function we need a Loss function that gives us the smallest value

Let there be ŷ and y that are the probability of required image and the output respectively.

So defining the loss function as

L(ŷ, y) = $-\ (ylog(\hat{y})\ +\ (1-y)log(1-\hat{y}))$

This is a function which gives the minimum value of ŷ.

So if the output is 1 .i.e y = 1

Then the function spits out only $-\ (log(\hat{y}))$

And here for the function to be minimum $log(\hat{y})$ should be maximum and for that ŷ should be maximum, but since it is a sigmoid function therefore maximum value of ŷ is 1.

If the output is 0 .i.e y = 0

Then the function spits out only $-\ log(1-\hat{y})$

And here for the function to be minimum $log(1-\hat{y})$ should be maximum and for that ŷ should be minimum, but since it is a sigmoid function therefore minimum value of ŷ is 0.

This Loss function is only for one training example

For a whole training set, the Cost function is taken

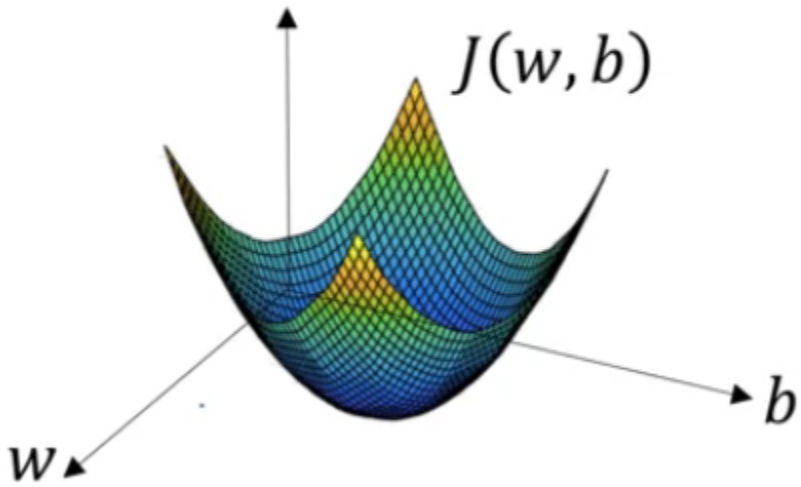
The cost function is given by:

$$\frac{1}{m}\sum_{i=1}^{m} L(\hat{y}_i\ ,\ y_i)$$

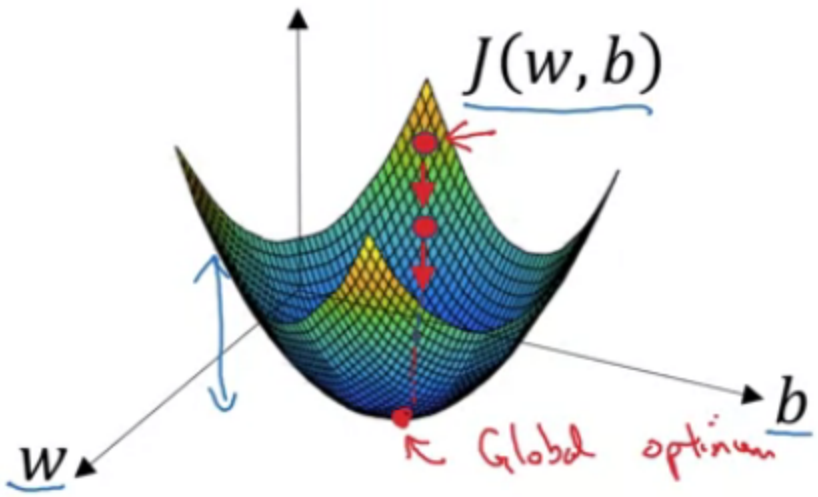
J(w, b) =

This is the cost function.
It is the average of the Loss function of all the training examples.
This is used for the whole training set.

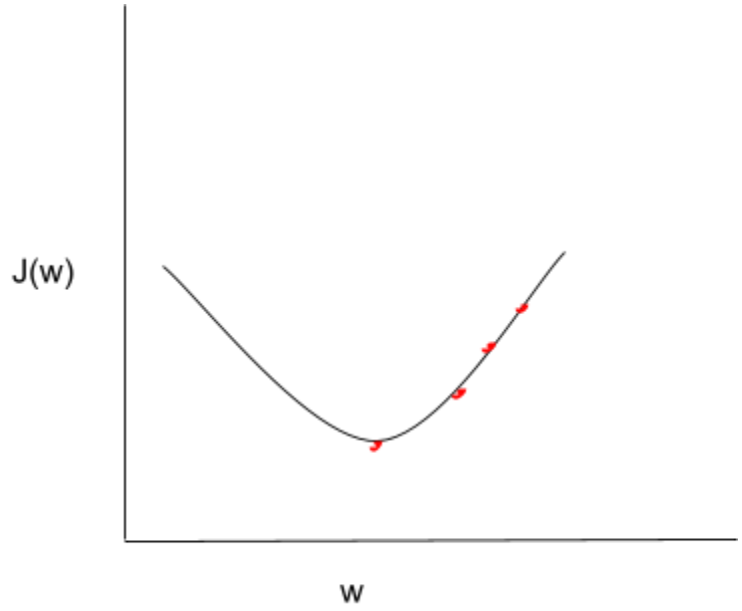
GRADIENT DESCENT
We want to find w and b such that the function J(w, b) { cost function }
Becomes minimum.
It being a convex function can be represented by:



Gradient descent is used to find the minimum of a convex function.
Any value on the curve is taken and by making small iterations
Downwards we can slowly reach the minima.



To make it more clear let J(w) be a convex function of only w, Then:



We can continuously update w as

$$w := w - \alpha \frac{\mathrm{d}J(w)}{\mathrm{d}w}$$

α here is the learning rate .i.e the rate at which we move down the slope of the graph to reach minima.

For the function J(w, b) in 2 variables the updated values of w and b are given by:

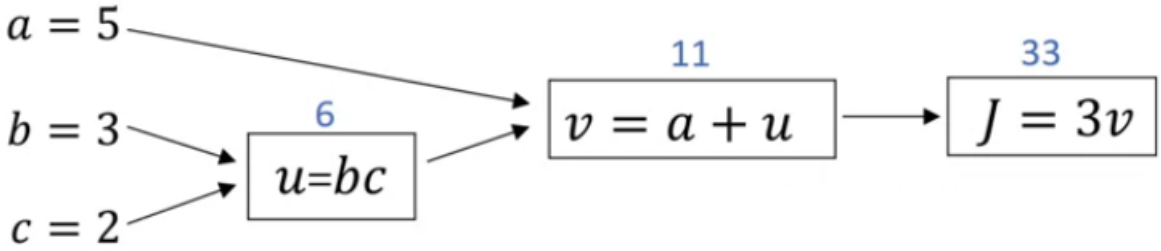
$$w := w - \alpha \frac{\partial J(w, b)}{\partial w}$$ = w - α(dw)

$$b := b - \alpha \frac{\partial J(w, b)}{\partial b}$$ = b - α(db)

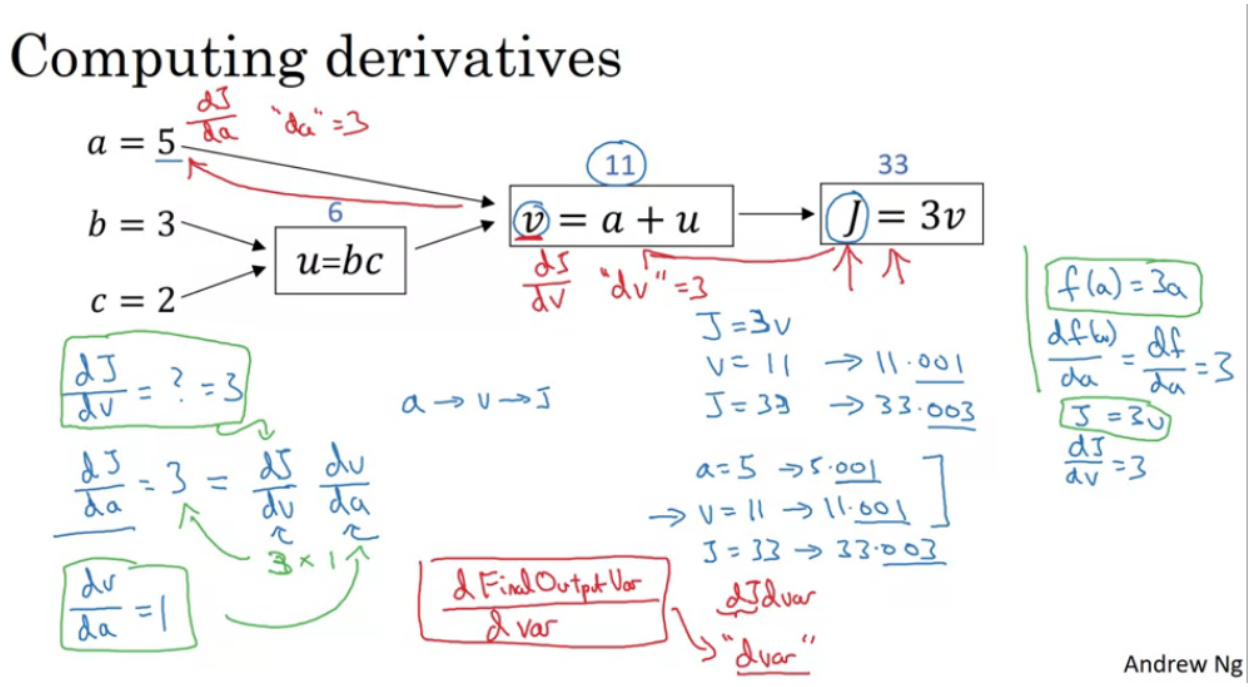
DERIVATIVES WITH COMPUTATIONAL GRAPH

Computational graph is basically the chain rule in derivative in which a group of variables can be considered to be one variable so as to make a chain and go ahead grouping all the variables together to form the given function.

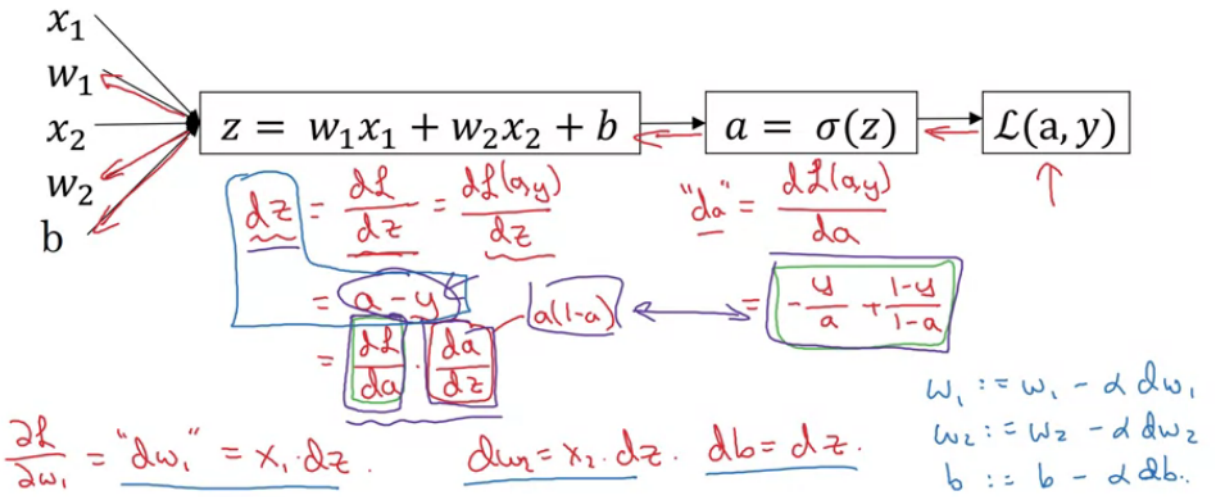
J(a, b, c) = 3(a + bc)



Back propagation gives us the derivative of the function w.r.t to the function preceding it.



LOGISTIC REGRESSION DERIVATIVE



To find the derivative of the logistic regression function we go backwards And find derivatives of various functions to different variables.

VECTORIZATION

For calculating the matrix product and the parameters various for loops are to be implemented which in turn take a very large time for the program to run. Therefore vectorization technique is used in which we take the inputs in the form of a matrix and perform the operations.

While writing in code the python library **numpy** is used

(numpy.dot(W.T, X) + b) This function takes the transpose of the parameter matrix W (by .T) and also the training or testing set of data X and multiples them element wise and later adds b to each row.
This removes the necessity to use a for loop in our program.

Basically numpy library is to be used to the fullest while writing the code.

BUILDING BLOCKS OF DEEP NEURAL NETWORKS

Forward Propagation

- For a hidden layer l in our neural network, we can have as seen before $W^{[l]}$, $b^{[l]}$, $a^{[l]}$. Thus
  $\mathbf{Z^{[l]} = W^{[l]}.a^{[l-1]} + b^{[l]}}$, where $a^{[l-1]}$ is the input from the previous layer and $\mathbf{a^{[l]} = g^{[l]}(Z^{[l]})}$ is the output for the next layer.
- For the first iteration $a^{[0]} = X$ i.e. our input matrix.
- Here, we note that it is useful to cache the value of $Z^{[l]}$ for back propagation.

Backward Propagation

- Here, for a layer l going backward from layer l+1, we input $da^{[l]}$ and output $da^{[l-1]}$, $dW^{[l]}$, and $db^{[l]}$.

dW =

- It is convenient to similarly cache $dW^{[l]}$ and $db^{[l]}$ along with $Z^{[l]}$.
- Thus, using these derivative terms that are outputted, we can update $\mathbf{W^{[l]} = W^{[l]} - \alpha dW^{[l]}}$ and $\mathbf{b^{[l]} = b^{[l]} - \alpha db^{[l]}}$.

Once all layers are finished in forward and backward propagation, that completes one iteration of gradient descent.

HYPER-PARAMETERS AND PARAMETERS

- Parameters are all variables that we have already discussed so far such as W,b,etc. Hyperparameters are values that determine the final value of the parameters that we end up using, for example
  - learning rate alpha
  - iterations
  - number of hidden units/layers
  - choice of activation function
- Basically hyper parameters are changed a lot based on our experiments with the code till we find the best value for it.