Week 2

Binary classification

As the name suggests, this classification gives an output in binary form,i.e., 1 or 0.
This is achieved using feature vectors and classifier.

Feature Vector - It is an n-dimensional matrix containing all pixel values from the matrices of the image with red, green and blue pixels. Here, n is the number of pixel values in total. Each matrix (red, green or blue) has same size as the image.
Eg. Image of size 64*64 will give a feature vector with size (64*64*3, m), i.e., (n, m).
Here, n = no of total reshaped or un-rolled pixel values. [Total =r+g+b]
m = Training data size

Training set : matrix denoted by Capital X that contains total 'm' feature vectors;

X is the training set where x1, x2,... xm are feature vectors
Y is also introduced as stacking data in columns is conventional for implementing neural networks

Logistic Regression

It is an algorithm that is used when we want output labells to be 0 or 1,i.e., binary

Parameters of the algorithm are W and b; W is n-dimensional vector and b is a real number.

Parameters: $\boxed{w} \in \mathbb{R}^{n_x}$, $\boxed{b} \in \mathbb{R}$.

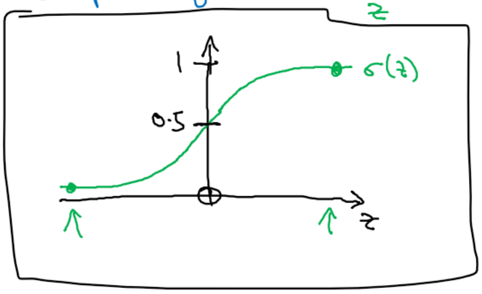
Output $\hat{y} = \sigma(\underbrace{w^T x + b}_{z})$

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

If z large $\sigma(z) \approx \frac{1}{1+0} = 1$

If z large negative number

$$\sigma(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{1+\text{Bignum}} \approx 0$$



y hat is output, and formula for is given in the image in terms of w and b ;Wt is w transpose
here, sigmoid function is used as it helps to get an output between 0 and 1. Note y is different for each training example

Loss function

this function judges a models performance.

$\boxed{\text{Loss}}$ (error) function: $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y}-y)$

$$\boxed{\mathcal{L}(\hat{y}, y)} = -(y \log \hat{y} + (1-y)\log(1-\hat{y}))$$

If $y=1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$ ← Want $\log \hat{y}$ large, want $\hat{y}$ large.

If $y=0$: $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$ ← Want $\log 1-\hat{y}$ large .... want $\hat{y}$ small

$\boxed{\text{Cost}}$ function: $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log \hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$

The 1st formula besides the title in the image is actually a squared function [2 got erased]
This formula isnt feasible. The next formula below the title is used.

It computes error for single training eg., for entire training set, cost function is used.
Cost function is the avg of loss functions of the entire training set.

$$z = w^T x + b$$
$$\hat{y} = a = \sigma(z)$$
$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

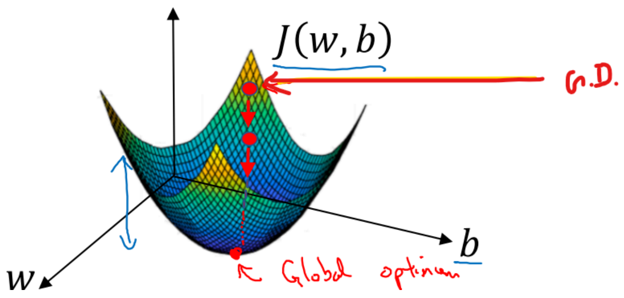
Cost function [J(w,b)]

Gradient Descent

Recap: $\hat{y} = \sigma(w^T x + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$

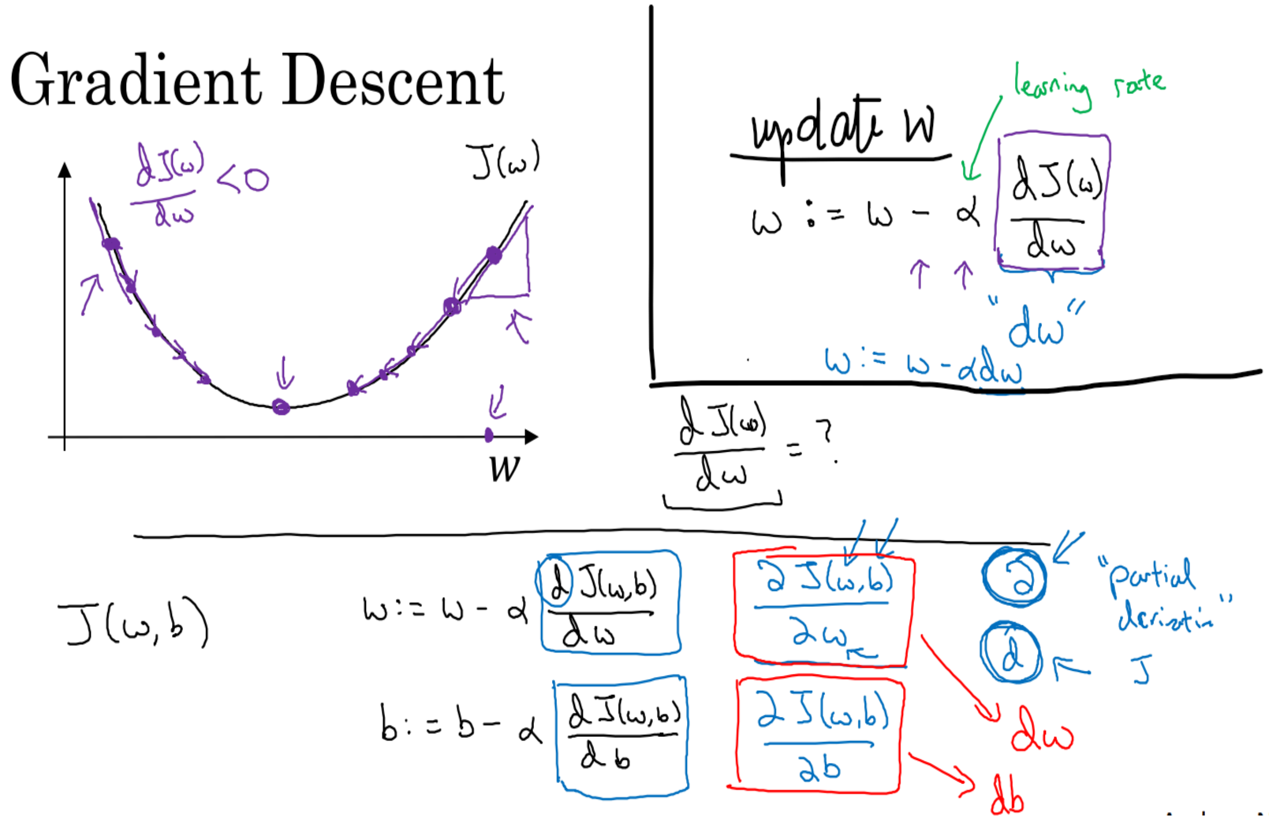
$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\boxed{\hat{y}^{(i)}}, \boxed{y^{(i)}}) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Want to find w, b that minimize $J(w, b)$



The formula given above the graph is of cost function, its avg of loss function

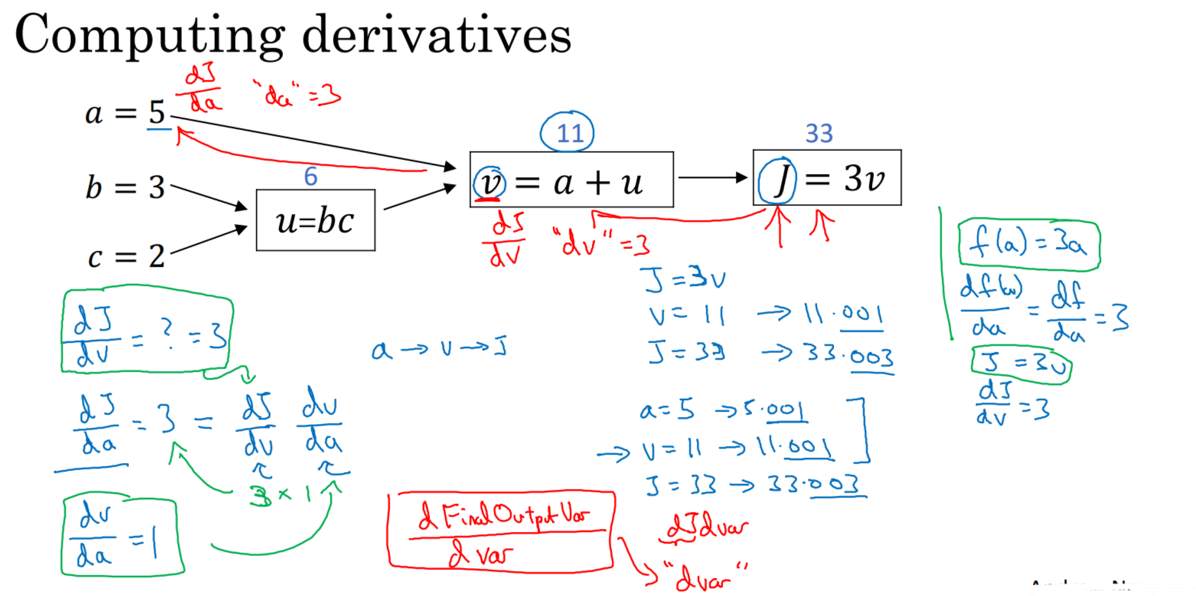
The graph represents a convex function whose height equals the cost function and its Global optimum gives values of w and b that help to minimise the J(w,b)
The red coloured point that is drawn down is due to iterations of gradient descent
G.d. is used to reduce the initial points of w and b to the global optimum



We also need to update w; w updated w is denoted by “w:”
the variable in code fore the derivative term
The Variable in code for the derivative present in the w formula is dw
dw or dj/dw is the slope [height/width = j/w] which if positive initially makes gradient descent to decrease.
J is a function of w and b as per the original formula, here, if J is supposed to be a function of b too for that we use partial derivative of J

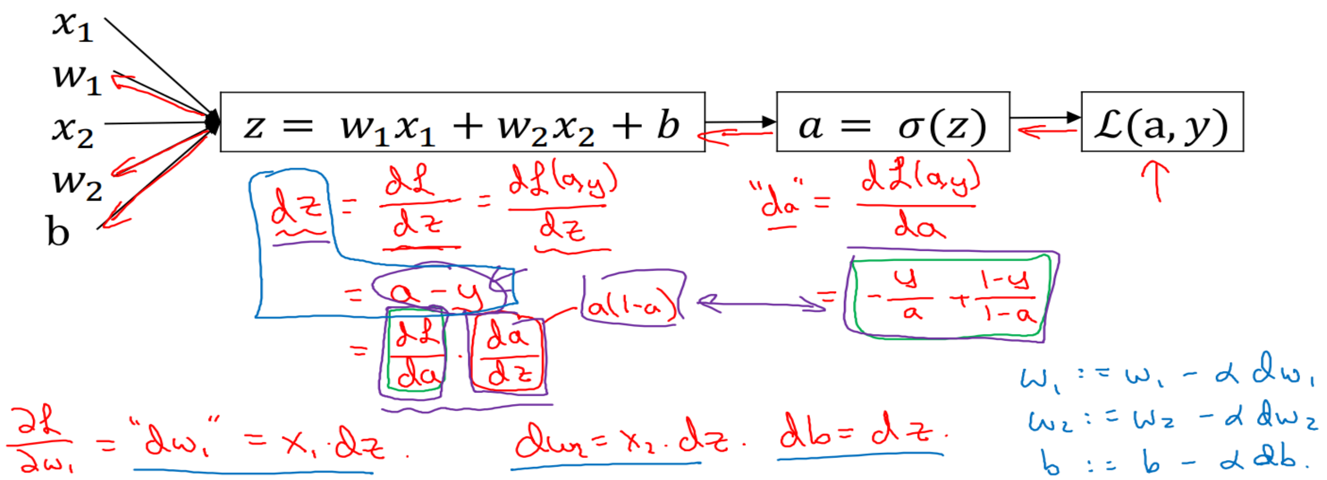
Computational graphs

Helps to find the derivative of the final output variable at every step, with respect to some intermediate variable.



d var is used as variable in code to represent the derivatives of final output var with respect to various intermediate quantities
Suppose 2 parameters are present, that means 2 vals of x and w, and 1 val of b [b is constt]

Below is the computation graph for loss function



Remember derivative of loss wrt z is a-y
The derivatives of loss remain constt, like formulae do.

In the below pic it is shown which variables vary for m no. of training examples [subscript i associated all variables]

$$\underline{J(w,b)} = \underbrace{\frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y^{(i)})} \qquad (x^{(i)}, y^{(i)})$$

$$\rightarrow a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b) \qquad \underline{dw_1^{(i)}},\ \underline{dw_2^{(i)}},\ \underline{db^{(i)}}$$

$$\underline{\frac{\partial}{\partial w_1} J(w,b)} = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)})}_{\underline{dw_1^{(i)}}\ -\ (x^{(i)}, y^{(i)})}$$

The above figure gives the formula of loss for m training egs. and its derivative for some parameter w1

Below we initialise J, w and b to 0 and the following figure shows how to implement the formulas for the m training examples

$J=0;\ dw_1=0;\ dw_2=0;\ db=0$

For $i=1$ to m

$$z^{(i)} = w^T x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$J \mathrel{+}= -\left[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})\right]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$$

$$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$$

$$db \mathrel{+}= dz^{(i)}$$

$n=2$

dw_3 ⋮ dw_n

$$J \mathrel{/}= m$$

$$dw_1 \mathrel{/}= m;\ dw_2 \mathrel{/}= m;\ db \mathrel{/}= m.$$

$$dw_1 = \frac{\partial J}{\partial w_1}$$

$$w_1 := w_1 - \alpha\, dw_1$$

$$w_2 := w_2 - \alpha\, dw_2$$

$$b := b - \alpha\, db$$

Vectorization

this is applicable only works for small datasets.

In the above fig, 2 for loops run -
1 for the Training egs
Another for the parameters w1 and w2

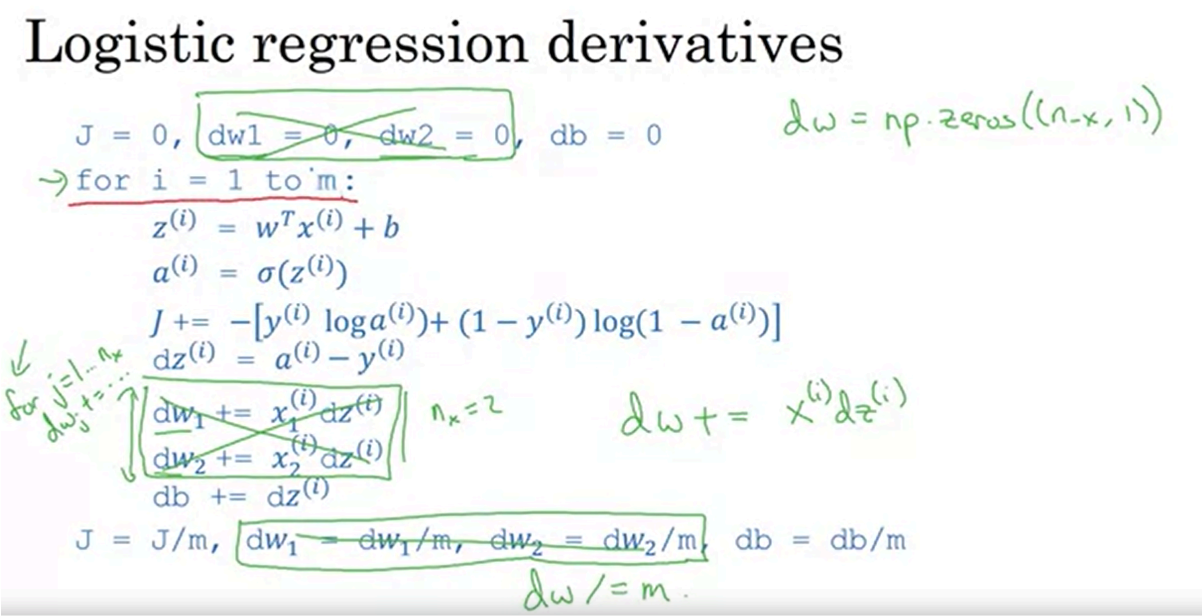
Even though in neural networks we use *vectorisation* instead of for loops

After W(s)are found, we update them
W(s) are accumulative variables which's why there are no subscripts with them
derivative dz_i helps calculate the loss function for each training eg

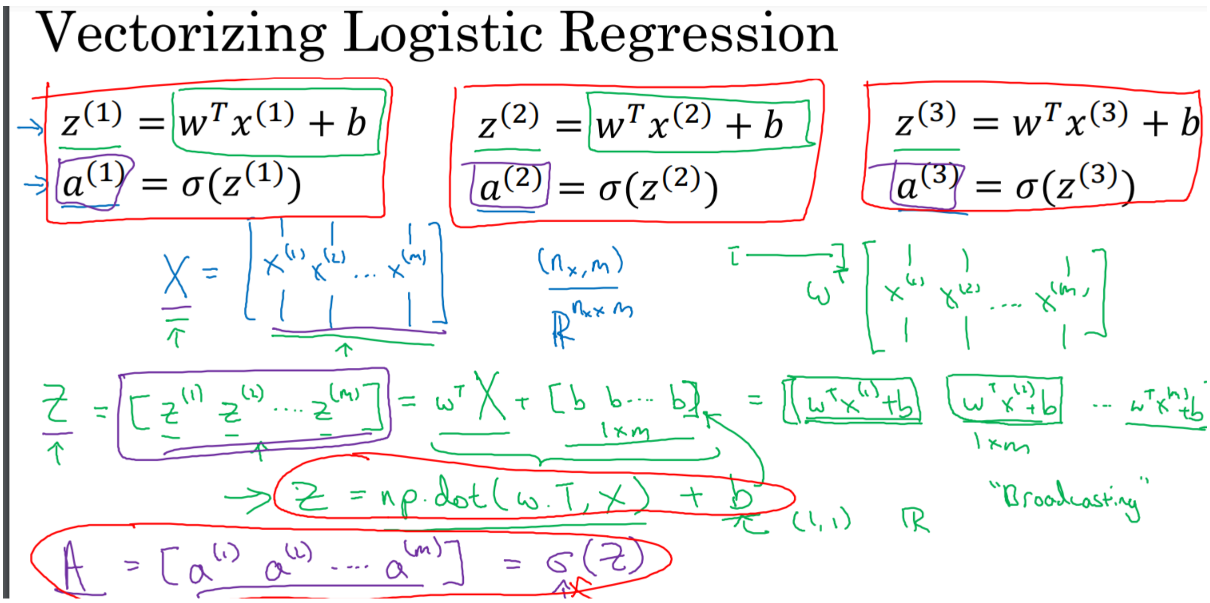
Vectorisation

Colab nbk for vectorisation function, diff bw this function and normal for loop [execution time]

Untitled14.ipynb



The above fig is for calculating derivatives by vectorisation
The original version was by non-vec, we then cancelled the areas,where np functions and other vectorisation tactics can be used
Modifications-
Initialisations replaced with np.zeros
The for loop for j=1..n_x and w[j]+= x_1[i]*dz[i]
is replaced by dw+=x[i]*dz[i]
So, now only 1 for loop present [was 2 earlier]
At the end, dw/=m replaces dw_1/=m and dw2 too



Instead of operating individually on z1, z2,z3 ,lly,for a(s)
We use the Z that equals dot product of wT and x and adds b

Refer the pdf directly for rest - codes and syntax