Logistic Regression as a Neural Network

Binary classification

Results 0 or 1 in the output

- For classifying images,

Dim of img = $64 \times 64 \times 3$ (RGB)

$n_x = 12288$

For m training examples,

$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots (x^{(m)}, y^{(m)})\}$

m_{test} and m_{TRAIN}

→ $X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$ n_x

$\longleftarrow m \longrightarrow$

X.shape = (n_x, m)

→ $Y = \begin{bmatrix} y^{(1)}, y^{(2)}, \ldots y^{(m)} \end{bmatrix}$

$\longleftarrow m \longrightarrow$

Y.shape = $(1, m)$

* Logistic Regression

Ex: to check whether a given image is a cat image or not

For given x,
Here x is a feature vector.

We want $\hat{y} = P(y=1|x)$
Probability of given image being a cat image.

Parameters : w, b

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

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

° Cost function

LF: $L(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$

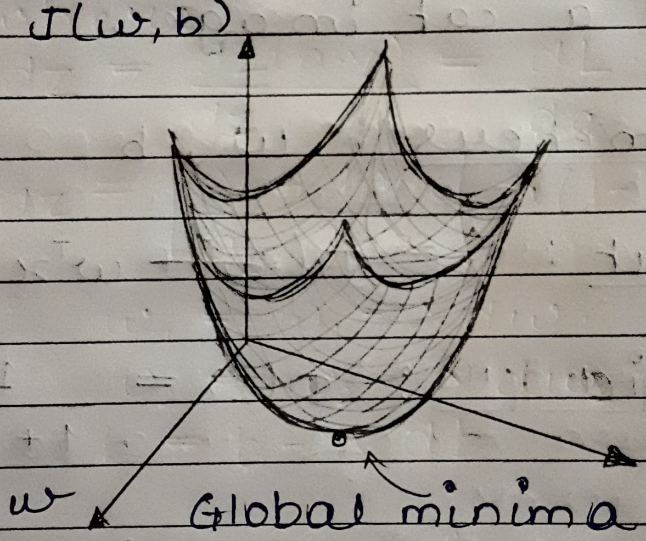
If $y = 1$, $L(\hat{y}, y) = -\log \hat{y}$
∴ want $\hat{y}$ large

If $y = 0$, $L(\hat{y}, y) = -\log(1-\hat{y})$
∴ want $\hat{y}$ to be small

CF : $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$

⟹ Loss function computes error for a single training example while cost function is the average LF error for all training examples.

- Gradient Descent



GD makes J(w,b) take small steps until it reaches the global minima.

$w = w_0 - \alpha \frac{dJ(w)}{dw}$ ... repeat

α ⇒ learning rate

dw ⇒ slope at each step

Computation graphs

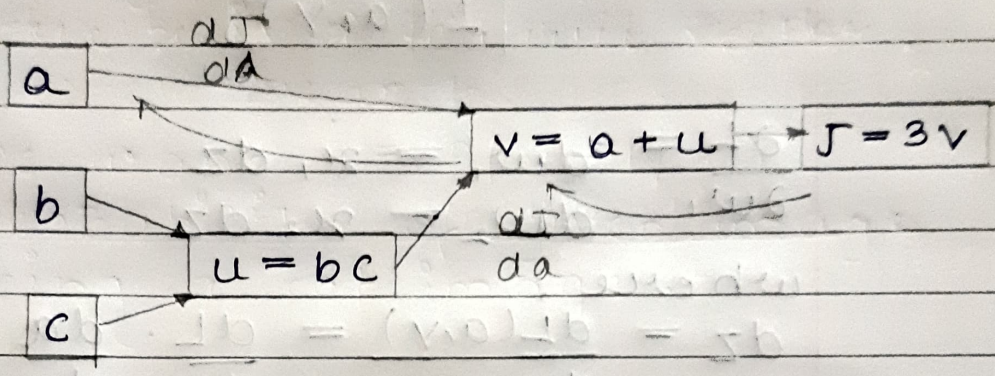
$J(a,b,c) = 3(a+bc)$

We do,

$u = bc$

$v = a+u$

$J = 3v$



Derivatives of comp graphs

$$\frac{dJ}{dv} = 3 \qquad \frac{dv}{da} = 1$$

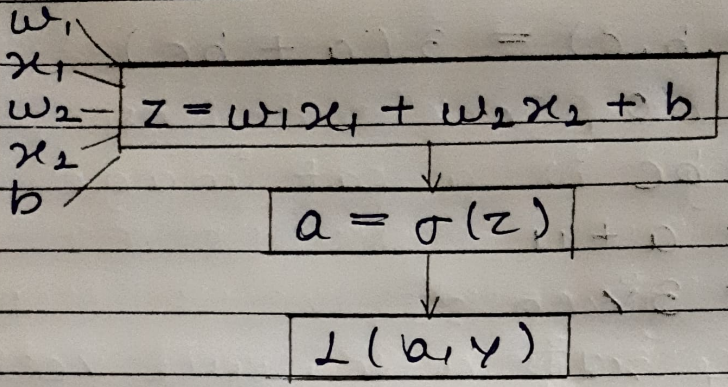
$$\frac{dJ}{da} = \frac{dJ}{dv} \cdot \frac{dv}{da} = 3$$

similar for b, c

~~find output var~~ / ~~dvar~~ $\frac{d(FOV)}{dVAR}$ = "dVar"

for the code

Log Reg Derivatives



$$\frac{\partial L}{\partial w_1} \quad dw_1 = x_1 \, dz$$
$$dw_2 = x_2 \, dz$$

where,

$$dz = \frac{dL(a,y)}{dz} = \frac{dL}{da} \cdot \frac{da}{dz}$$

$$dz = \left\{ \frac{-y}{a} + \frac{1-y}{1-a} \right\} \cdot a(1-a) = a - y$$

$$\left\{ \begin{aligned} w_1 &= w_1 - \alpha \, dw_1 \\ w_2 &= w_2 - \alpha \, dw_2 \\ b &= b - \alpha \, db \end{aligned} \right\} \quad *$$

→ For m training examples,

we use $J = 0, dw_1 = 0 \ldots dw_n = 0, db = 0$

Since we compute CF (avg), $J /= m$, $dw_1 /= m \ldots dw_n /= m$ and $db /= m$

We keep updating using previous formula*

Note: Blunt method uses 2 for loops, one to go thru m training exs & 2nd one thru each features. This is solved using 'vectorization.'