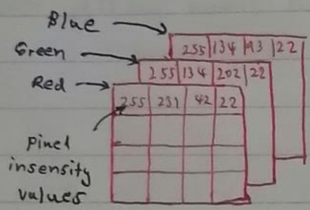
Week 2

Neural Networks Basics

~~Supervised Learning~~

Binary Classification

* Logistic regression is an algorithm for binary classification.
* Let's assume we have a image in our computer. That image always stored in three separated metrices. each metrices represent by one color (Red, Green, Blwe)



← Image with cat

1 (cat) vs 0 (non cat)

y

* if you store ~~image~~ 64x64 pixels image, then you will have three 64x64 pixels metrices.

* Now we have to convert those pixel insensity values into a feature vector. There for we have to assign those values to ~~out~~ input feature vector x.

$$x = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ \vdots \\ 255 \\ 134 \\ \vdots \end{bmatrix}$$

pixel x pixel x no. of metrics

$64 \times 64 \times 3 = 12288$

$n_x = 12288$

* In binary classification, our goal is larn a classifier that can input an Image represented by feature vector x and predict whether given label is 1 or 0. (In my case cat or not cat)

* A single training example is represented by (x, y)

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

x is an dimentional feature
y is the label either 0 or 1,

* Your training set comprice : m training examples
* first farining example : $(x^{(1)}, y^{(1)})$
* Second training example : $(x^{(2)}, y^{(2)})$
* m (final) training example : $(x^{(m)}, y^{(m)})$

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

* To define those training set create metrics using X

$$X = \underbrace{\begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}}_{m \text{(coloums)}} \; n_x \text{(height)}$$

* In dimensional metrics
$X \in \mathbb{R}^{n_x \times m}$

* In Python we use command
X.shape = (n_x, m)

* In output (Y) $\longrightarrow$ $Y = [\, y^{(1)} \; y^{(2)} \; \cdots \; y^{(m)} \,]$
$Y \in \mathbb{R}^{1 \times m}$
Y.shape = $(1, m)$

Logistic Regression

Given x, want $\bar{y} = P(y=1|x)$

* Let's assume x is a picture and you want $\bar{y}$ to tell you what is is the chance that this is a cat picture

* Given that parameter of logistic regression will be w which also an x dimensional vector $\cancel{\in x \in \mathbb{R}^{n_x}}$ and together with b which is just a real number

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

* Then, given x, w and b how can we genarate output $\bar{y}$

* What if we use

$$\text{Output } \bar{y} = w^T x + b$$

* Above fomula represent linear regression. but it is not good algorithm for binary classification because we wat $\bar{y}$ hat to be $(y=1)$.

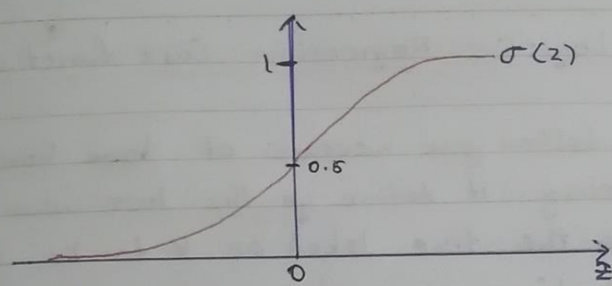
$\therefore$ $\bar{y}$ should be in between 1 an 0 $(0 \leq \bar{y} \leq 1)$

* $w^T x$ can give more than 1 or negative value.

* Therefor, In Logistic regression we use σ (sigmoid function) to put $\bar{y}$ in between 1 and 0.

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

$$\bar{y} = \sigma(z)$$



sigmoid function

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

Ex:- $X_0 = 1$, $x \in \mathbb{R}^{n_x+1}$

$$\bar{y} = \sigma(\theta^T x)$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{matrix} \} b \\ \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} w \end{matrix}$$

Logistic Regression Cost function

* ~~Cost~~ Loss function tells you messure of ~~[illegible]~~ how well our algorithm is doing. it define ~~ya~~ the loss when your algorithm outputs $\bar{y}$ and the true label as Y to be the squre error or one half a square error.

Loss(error) function : $L(\bar{y}, y) = \frac{1}{2}(\bar{y}-y)^2$

Loss function

$$L(\bar{y},y) = -(y \log \bar{y} + (1-y)\log(1-\bar{y})) \qquad 0 \le y \le 1$$

~~y=0~~ y=1

$L(\bar{y},y) = -\log \bar{y}$ y=1. because we want $-\log \bar{y}$ to be small as possible

y=0

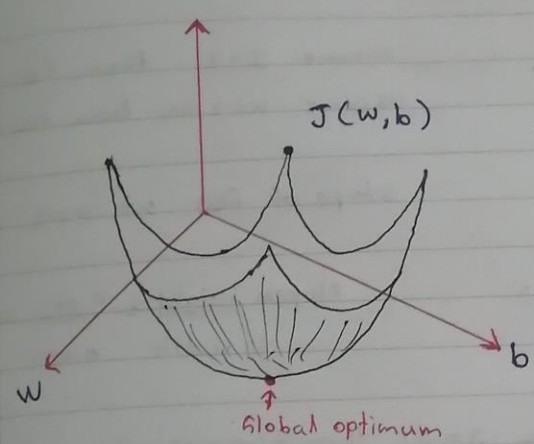
$L(\bar{y},y) = -\log(1-\bar{y})$ y=1. becare we want $-\log(1-\bar{y})$ to be large as possible

* Cost function tells you messure of how ~~e~~ well our entire training set doing. Cost function denote by J.

~~Cost function : $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\bar{y}^{(i)}, y^{(i)})$~~

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\bar{y}^{(i)}, y^{(i)}) = \frac{-1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log \bar{y}^{(i)} + (1-y^{(i)})\log(1-\bar{y}^{(i)})\right]$$

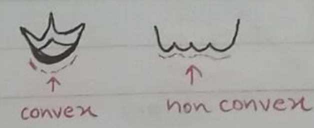
Gradient Descent



w -: single real number

b -: single real number

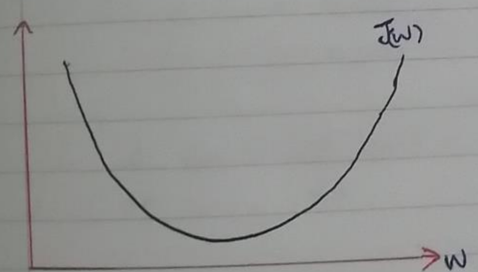
J(w,b) -: height of the surface



convex -: has a one optima

nonconvex -: have multiple optimas

Let's remove "b" for a moment



Repeat {

$$w : w - \alpha \frac{dJ(w)}{dw}$$

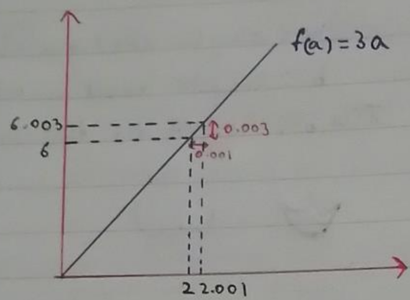
}

$J(w, b)$

$$w : w - \alpha \frac{dJ(w,b)}{dw}$$

$$b : b - \alpha \frac{dJ(w,b)}{db}$$

Derivatives



assume $a=2$ then $f(a)=6$

assume $a=2.001$ then $f(a)=6.003$

$\therefore$ slope of $f(a)$ at $a=2$ is

$$\text{slope} = \frac{\text{height}}{\text{width}} = \frac{0.003}{0.001} = \underline{\underline{3}}$$

for anothe example assum $a=5$ then $f(a)=15$

assum $a=5.001$ then $f(a)=15.003$

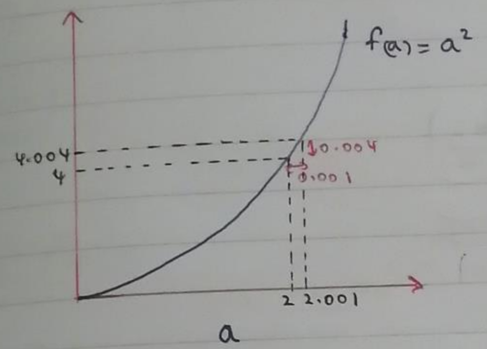
Then slope of $f(a)$ at $a=5$ is

$$\text{slope} = \frac{0.003}{0.001} = \underline{\underline{3}}$$

$$\therefore \frac{d\,f(a)}{da} = 3 = \frac{d}{da} f(a)$$

* if you increase value of "a" ~~slope~~ by 0.001, the value of f(a) goes up by three times as much. Therefor, we can tell this function has same slope everywhere.

More Derivative Examples



$a = 2 \quad \therefore f(a) = 4$

$a = 2.001 \quad \therefore f(a) \approx 4.004$
(4.004001)

slope of $f(a)$ at $a = 2$ is

$$\text{slope} = \frac{0.004}{0.001} = 4 = \frac{d}{da} f(a)$$

for anothe example $\quad a = 5 \quad \therefore f(a) = 25$

$a = 5.001 \quad \therefore f(a) \approx 25.010$

$$\therefore \frac{d}{da} f(a) = \frac{0.01}{0.001} = 10 \quad , \text{ when } a = 5$$

$$\therefore \frac{d}{da} f(a) = \frac{d}{da} (a^2) = 2a$$

* slope will change time to time in these functions.

Computation Graph

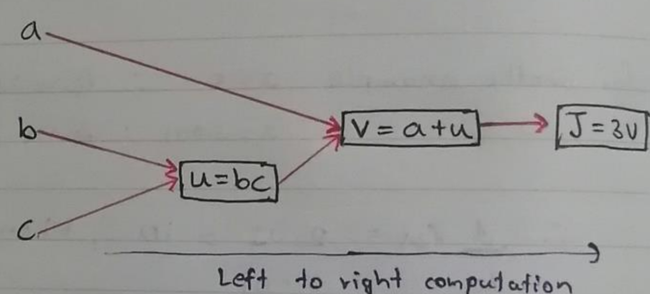
Example

$$J(a,b,c) = \underbrace{3(\underbrace{a+\underbrace{bc}_{u}}_{v})}_{J}$$

$u = bc$

$v = a + u$

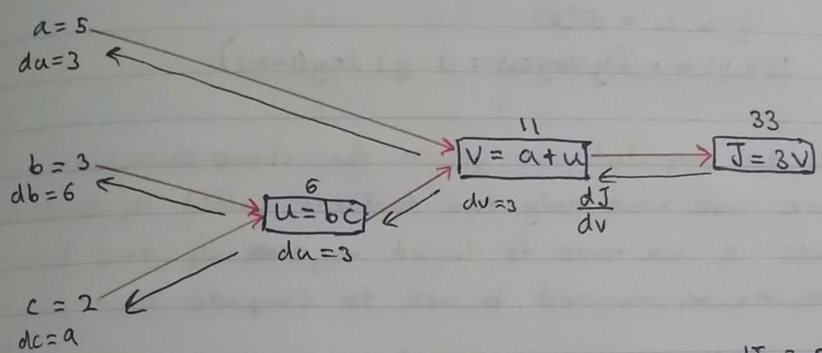
$J = 3v$



* In here "J" is our cost function. because we are trying to minimize it.

* we saw in here is left to right computation. (forward propagation)

Derivatives with a Computation Graph



u = 6 ⟶ 6.001

v = 11 ⟶ 11.001

J = 33 ⟶ 33.003

$$\frac{dJ}{du} = \underbrace{\frac{dJ}{dv}}_{3} \cdot \underbrace{\frac{dv}{du}}_{1}$$

$$\frac{dJ}{dv} = \frac{0.003}{0.001} = 3$$

$$\frac{dv}{du} = \frac{0.001}{0.001} = 1$$

$$\therefore \frac{dJ}{du} = 3$$

b = 3 ⟶ 3.001

$u = \overset{3\times 2}{bc} = 6$ ⟶ 6.002

J = 33 ⟶ 33.006

$$\frac{dJ}{db} = \underbrace{\frac{dJ}{du}}_{3} \cdot \underbrace{\frac{du}{db}}_{2}$$

$$\frac{dJ}{du} = \frac{0.006}{0.002} = 3$$

$$\frac{du}{db} = \frac{0.002}{0.001} = 2$$

$$\therefore \frac{dJ}{db} = 6$$

c = 2 ⟶ 2.001

u = bc = 6 ⟶ 6.003

J = 33 ⟶ 33.009

$$\frac{dJ}{dc} = \underbrace{\frac{dJ}{du}}_{3} \cdot \underbrace{\frac{du}{dc}}_{3}$$

$$\frac{dJ}{du} = \frac{0.009}{0.003} = 3$$

$$\frac{du}{dc} = \frac{0.003}{0.001} = 3$$

$$\therefore \frac{dJ}{dc} = 9$$

* we saw in here is right to left computation. (backward propagation)

Logistic Regression Gradiant Descent

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

* Let's write computational graph for above example.
* We assume we have only two features called x_1 and x_2.
* To compute z we have to input w_1, w_2 and b.
* Those x_1, x_2, w_1, w_2 and b use to compute z.

x_1, w_1, x_2, w_2, b $\rightarrow$ $\boxed{z = w_1x_1 + w_2x_2 + b} \rightarrow \boxed{\hat{y} = a = \sigma(z)} \rightarrow \boxed{L(a,y)}$

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

$dz = a - y$

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

$da = -\frac{y}{a} + \frac{1-y}{1-a}$

* In logistic regression, what we want to do is to modify the parameters "w's" (w_1, w_2) and "b", in order to reduce loss $(L(a,y))$

* To reduce loss, first thing to do is going backwards to compute the derivative of this loss with respect to $da_{[1]}$

$$da = \frac{dL(a,y)}{da}$$
$$= [-y\log(a)]\frac{dL(a,y)}{da} + [(1-y)\log(1-a)]\frac{dL(a,y)}{da}$$
$$= -\frac{y}{a} + \frac{1-y}{1-a}$$

$$dz = \frac{dL}{dz}$$

$$= \frac{dL}{da} \cdot \frac{da}{dz}$$

$$= \left[\frac{-y}{a} + \frac{1-y}{1-a}\right] \times [a(1-a)]$$

$$dz = a-y$$

$$\frac{da}{dz} = \sigma(z) \times (1-\sigma(z))$$
$$= a \times (1-a)$$
$$= a(1-a)$$

next we assume how much we have to change ws and b

$\therefore \frac{dL}{dw_1} = dw_1 = x_1 dz$ | $dw_2 = x_2 dz$ | $db = dz$

$\therefore w_1 = w_1 - \alpha dw_1$

$w_2 = w_2 - \alpha dw_2$

$b = b - \alpha db$

Gradient Descent on m Examples

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

$$\rightarrow a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

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

Example

$J=0$, $dw_1=0$, $dw_2=0$, $db=0$

For $i=1$ to m

[1]

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

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

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

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

[2]

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

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

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

$J \mathrel{/}= m$ ← $J = J/m$

$dw_1 \mathrel{/}= m$

$dw_2 \mathrel{/}= m$

$db \mathrel{/}= m$

weaknesses above code

* to ~~impleme~~ implemet logistic regression this way, we have to write two for loops. first ~~loop for~~ for loop is given for loop [1] above which is use to iterate untill ~~m~~ m training examples. The second for lopp use for over all the features over here. [2]

* those for loops make your algorithm slow. As a solution deep learning proposed "vectorization". It can use to get rid of from for loops.

Vectorization

* This is ure to get rid of from enplicit for loop from code
* In deep learning we have to work with large datsets. Therefor, it is must to run our codes efficiently. wectorization help us to reduce run time those codes.

$$z = w^T x + b$$

Non - vectorized

```
z = 0
for i in range(n-x)
    z += w[i] * x[i]
z += b
```

vectorized

$$z = \underbrace{np.dot(w,x)}_{w^T x} + b$$

More Vectorization Examples

* If you want to compute a vector "u" as the product of matrix "A" and another vector "v"

$$u = Av$$

* Then definition of our matrin is

$$u_i = \sum_j A_{ij} v_j$$

* If we use non vectorized method

```
u = np.zero (n,1)
for i ...
    for j ...
        u[i] += A[i][j] * v[j]
```

* If we use vectorized method

```
u = np.dot (A,v)
```

Vectorizing Logistic Regression

* Let's assume you have M training examples

∴

first training example	second training example	Third training example
$z^{(1)} = w^T x^{(1)} + b$	$z^{(2)} = w^T x^{(2)} + b$	$z^{(3)} = w^T x^{(3)} + b$
$a^{(1)} = \sigma z^{(1)}$	$a^{(2)} = \sigma z^{(2)}$	$a^{(3)} = \sigma z^{(3)}$

* Likewise you have to done for m training examples.

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

$$Z = [z^{(1)}\ z^{(2)} \ldots z^{(m)}] = w^T X + [b\ b \ldots b] = [\boxed{w^T x^{(1)} + b}\ \boxed{w^T x^{(2)} + b} \ldots \boxed{w^T x^{(m)} + b}]$$

$$Z = np.dot(w.T, X) + b$$

* Python automaticaly take real number b and expands it out to 1xm row vector. This operation is called as "broadcasting"

* Likwise

$$A = [a^{(1)}\ a^{(2)} \ldots a^{(m)}] = \sigma(Z)$$

Vectorizing Logistic Regression's Gradient Output

* You have remembered in previous lessons we use ~~dz~~ $dz = a - y$. Let's try apply it to vectorization. Let's assume again you have m training examples.

$$dz^{(1)} = a^{(1)} - y^{(1)} \qquad dz^{(2)} = a^{(2)} - y^{(2)} \quad \cdots \quad dz^{(m)} = a^{(m)} - y^{(m)}$$

* We can vectorized the into lik this.

$$dZ = [dz^{(1)} \; dz^{(2)} \cdots dz^{(m)}]$$
$$A = [a^{(1)} \; a^{(2)} \cdots a^{(m)}]$$
$$Y = [y^{(1)} \; y^{(2)} \cdots y^{(m)}]$$

$$\therefore \; dZ = A - Y = [a^{(1)} - y^{(1)} \;\; a^{(2)} - y^{2} \cdots a^{(m)} - y^{(m)}]$$

* Now we have get rid of from one for loop which calculate dz. Now we have second for ~~lo~~ loop which contains below

$dw = 0$	$db = 0$
$dw \mathrel{+}= x^{(1)} dz^{(1)}$	$db \mathrel{+}= dz^{(1)}$
$dw \mathrel{+}= x^{(2)} dz^{(2)}$	$db \mathrel{+}= dz^{(2)}$
$\vdots$	$\vdots$
$dw \mathrel{+}= x^{(m)} dz^{(m)}$	$db \mathrel{+}= dz^{(m)}$
$dw \mathrel{/}= m$	$db \mathrel{/}= m$

* Now we have to vectorize above for loop contains.

$$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)}$$

$$= \frac{1}{m} \text{ np.sum}(dZ)$$

$$dw = \frac{1}{m} X \, dZ^T$$

$$= \frac{1}{m} \begin{bmatrix} | & | & | & | \\ x^{(1)} & \cdots & \cdots & x^{(m)} \\ | & | & | & | \end{bmatrix}_{(n \times m)} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}_{(1 \times m)}$$

$$= \frac{1}{m} \left[x^{(1)} dz^{(1)} + \cdots + x^{(m)} dz^{(m)} \right]_{n \times 1}$$

* Now Let's compare for loop version and vectorized version

$J=0$, $dw_1=0$, $dw_2=0$, $db=0$

for $i=1$ to m:

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

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

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

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

$dw_1 = x_1^{(i)} dz^{(i)}$

$dw_2 = x_2^{(i)} dz^{(i)}$

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

$J = J/m$

$dw_1 = dw_1/m$

$dw_2 = dw_2/m$

$db = db/m$

for item in range(1000)

$Z = w^T X + b$

$= \text{np.dot}(w.T, X) + b$

$A = \sigma(Z)$

$dZ = A - Y$

$dw = \frac{1}{m} X \, dZ^T$

$db = \frac{1}{m} \text{ np.sum}(dZ)$

$w := w - \alpha dw$

$b := b - \alpha db$

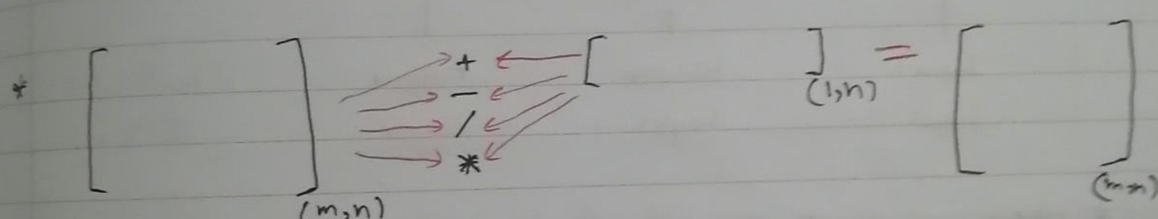
but

* you want one for loop run the vectorized model

Broadcasting in Python

* Broadcasting technique uses to make our Python code run faster.

	Apples	Beef	Eggs	Potatoes	
Carb	56	0	4.4	68	$= A_{(3,4)}$
Protein	1.2	104	52	8	
Fat	1.8	135	99	0.9	

* 

*