Logistic Regression Gradient Descent

we're trying to study Logistic Regression thru computation Graphs

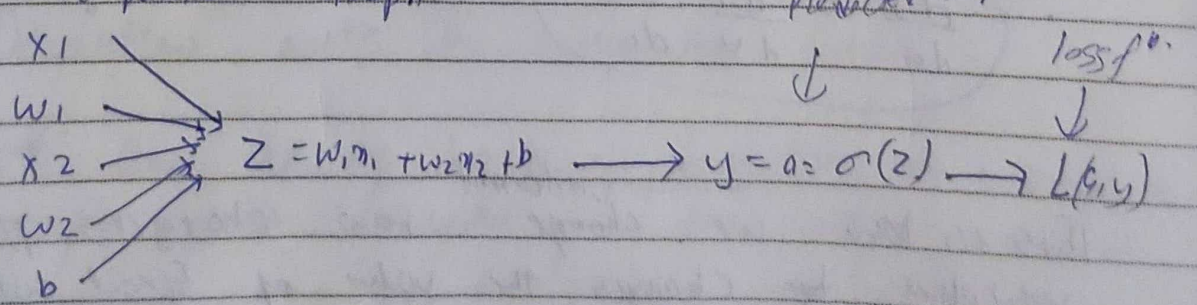
In LR.

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

prediction $\hat{y} = a = \sigma(z)$

$$L(a, y) = -(y \log(a) + (1-y) \log(1-a))$$

Computation Graph



now go backward to compute derivative of Loss wrt to the initial variables

basically how much J changes on changing a.

remember our loss fn

1st step we enter $da = \frac{d\,L(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$

2nd step (in going backward)

$$dz = \frac{dL(a,y)}{dz} = \frac{dL(a,y)}{da} \cdot \frac{da}{dz} \quad \text{(chain rule!!)}$$

$\frac{da}{dz} = a(1-a)$ ← remember our a fn.

$$dz = a - y$$

3rd step in going backward

$$\frac{dL}{dw_1} = dw_1 = x_1 dz \; ; \; dw_2 = x_2 dz$$

$x_1 (a-y)$; $db = dz$

$(a-y)$ $x_2(a-y)$

then we apply Gradient Descent as

$w_1 := w_1 - \alpha dw_1$

$w_2 := w_2 - \alpha dw_2$

$b := b - \alpha db$

→ All of this was only for 1 training example,
Now we'll see how we apply Gradient Descent for multiple training example.

Gradient Descent on m examples

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

we notice

so ↑ overall % Cost function is average of individual Loss fn from each training example

So

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1} L(a^{(i)}, y^{(i)})}$$

↓

we calculated this part

So now we just have to average them !!

So :- for model with 2 input parameters and 1 middle layer

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

1 step of gradient descent

For $i=1$ to m :

Forward propagation

$$z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$
$$J \mathrel{+}= -\left[y^{(i)} \log(a^{(i)}) + (1-y^{(i)}) \log(1-a^{(i)})\right]$$

Backward Propagation

$$dz^{(i)} = a^{(i)} - y^{(i)}$$
$$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$$
$$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$$
$$db \mathrel{+}= dz^{(i)}$$

↑ we only have $n=2$.

outside of for loop

$$J /= m;\quad dw_1 /= m;\quad dw_2 /= m;\quad db /= m$$

Now dw_1, dw_2, db represent the respective changes we need to make to reduce Cost function

So we do $w_1 := w_1 - \alpha dw_1$

$w_2 := w_2 - \alpha dw_2$

$b := b - \alpha db$

This completes one step of Gradient descent

✡ Now we repeat this process until we find Principal minima using Gradient Descent

Now, Guess what — this is very bad.
this is computationally stupid.

In 1 step of Gradient descent we running
$O(mn)$ ⊛ Time Complexity { and then we repeat Gradient descent n no. of times to reach minima
m training n updating n parameters

running nested for loops in Deep Learning algo's is considered a bad practice.

So we need to find a different way
↳ Vectorization