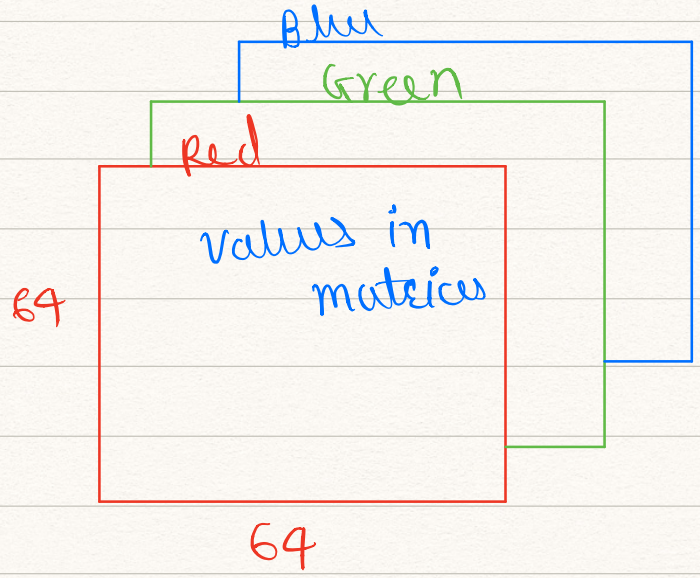
We can use for loop to go through entire training set but it will take lot of time complexity.

→ In Neural Networks we don't use for loops to go through training sets

Binary Classification :

(e.g.) Cat or No Cat in image

(64 × 64) px



Generate feature vector by putting all values in (n × 1) matrix

64 × 64 × 3

Here : (x, y)

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

m training Example: $\{(x_1, y_1), (x_2, y_2), \dots\}$

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

m columns

$$Y = \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix}_{1 \times m}$$

Logistic Regression:

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

$x \in R^{n_x}$

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

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

↳ not good

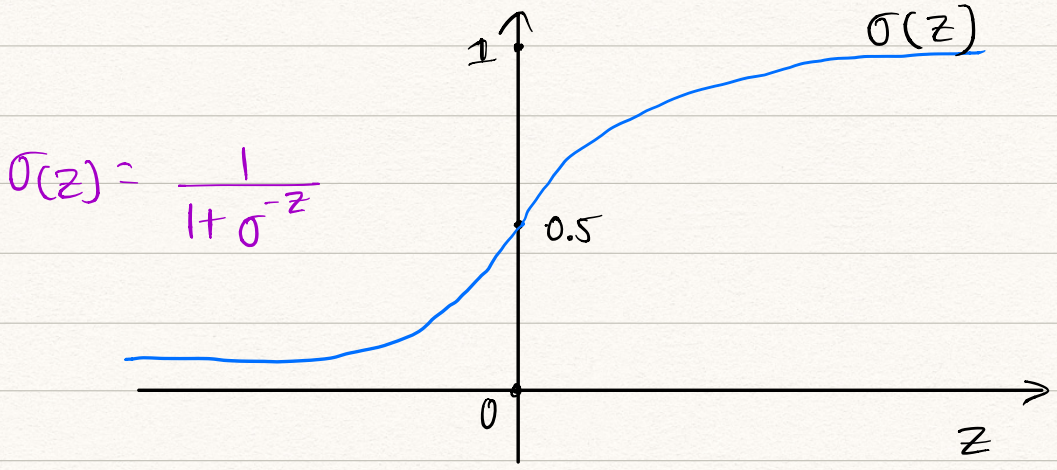
can't be binary (0 or 1)

So apply **sigmoid function**

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

$\downarrow$

z



In logistic Regression:

We try to ting w & b so that $\hat{y}$ become good estimate of chance of y being equal to 1.

Another Notation:

$x_0 = 1$, $x \in R^{n_x+1}$

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

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{n_x} \end{bmatrix}$ $\theta_0 \rightarrow b$, $\theta_1, \ldots, \theta_{n_x} \rightarrow w$

Logistic Regression Cost Function:

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

optimization problem becomes non convex. might have local optimum so gradient decent might not find global optimum.

Loss function

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

↓ small as possible

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

large $\Rightarrow \hat{y} \rightarrow$ large

if $y=0$

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

large $\rightarrow \hat{y} \rightarrow$ small

Cost Function:

overall

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

$$= \frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)}) \right]$$

Gradient Descent:

We want to find w, b for which J(w,b) is minimum.

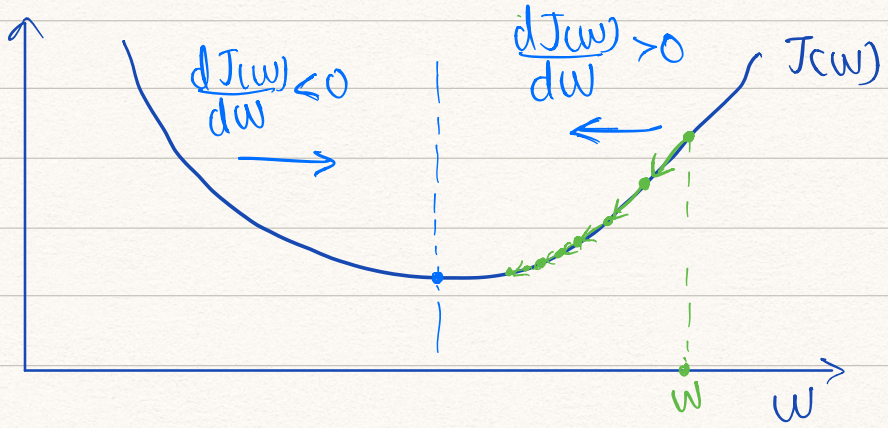
↓

convex function

→ It will start at some initial point and then go down on the steepest decent in direction of $\nabla J(w,b)$.

→ It will converge to global minimum in case of convex function.

Gradient Descent for 1 D:



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

$\alpha \rightarrow$ learning rate

$\frac{dJ(w)}{dw} \Rightarrow$ in code "dw"

Coding term: $w := w - \alpha\, dw$

Actual in 2D:

$$w := w - \alpha \frac{dJ(w,b)}{dw}$$ → partial derivative

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

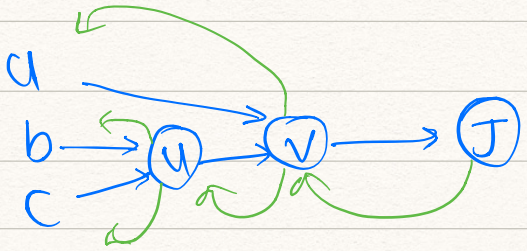
Computation Graph:

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

$u = bc$

$v = a+u$

$J = 3v$



$u = bc$ $\quad v = a+bc$ $\quad J = 3v$

$\frac{dJ}{dw} = 3$ $\quad \frac{dJ}{da} = 3$

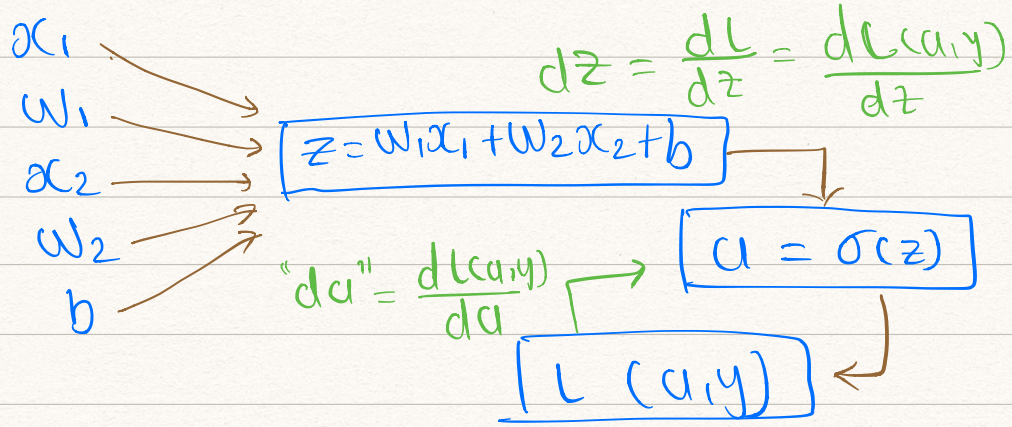
Logistic Regression Gradient Descent:

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

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

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

computational Graph:



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

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

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

$$db = dz$$

$$w_1 := w_1 - \alpha\, dw_1$$
$$w_2 := w_2 - \alpha\, dw_2$$
$$b := b - \alpha\, b$$

Logistic Regression on m examples:

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

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

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

$$dw_1^{(i)} \longrightarrow (x^{(i)}, y^{(i)})$$

Algorithm:

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

for loop:

For $i=1$ to m

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

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

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

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$ ↕ $n=2$

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

$J /= m$

$dw_1 /= m$

$dw_2 /= m$

$db /= m$

dw_1, dw_2, db → accumulator

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

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

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

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

Vectorization: get rid of for loop.

Logistic Regression Cost Function (Adv.)

If $y=1$ $\quad p(y|x) = \hat{y}$

$y=0$ $\quad p(y|x) = 1-\hat{y}$ } $p(y|x)$

$$p(y|x) = \hat{y}^{y} (1-\hat{y})^{(1-y)}$$ → function with above Boundary

conditions.

$$\log P(y|x) = \underbrace{y \log(\hat{y}) + (1-y)\log(1-\hat{y})}_{L(y,\hat{y})}$$

Cost Function:

$$P\left(\begin{matrix}\text{labels in} \\ \text{training set}\end{matrix}\right) = \prod_{i=1}^{n} P(y^{(i)}|x^{(i)})$$

we have to maximize it.

Put log & maximize it.

$$\log P(\,) = \sum_{i=1}^{n} \underbrace{\log P(y^{(i)}|x^{(i)})}_{-L(\hat{y}^{(i)}, y^{(i)})}$$

Maximum likelihood Estimate:

choose parameter that maximize $\log P(\cdot)$

$$= -\sum_{i=1}^{n} L(\hat{y}^{(i)}, y^{(i)})$$

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

↓so get rid of Negative sign.

$\frac{1}{m}$ → for better scale

By minimizing cost function, we are carrying out max. likelyhood estimation over logistic Reg. model under the assumption that our training examples are IRD (Identical, Randomly distri.)