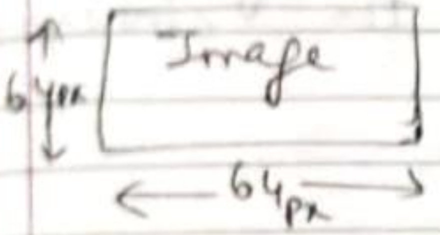
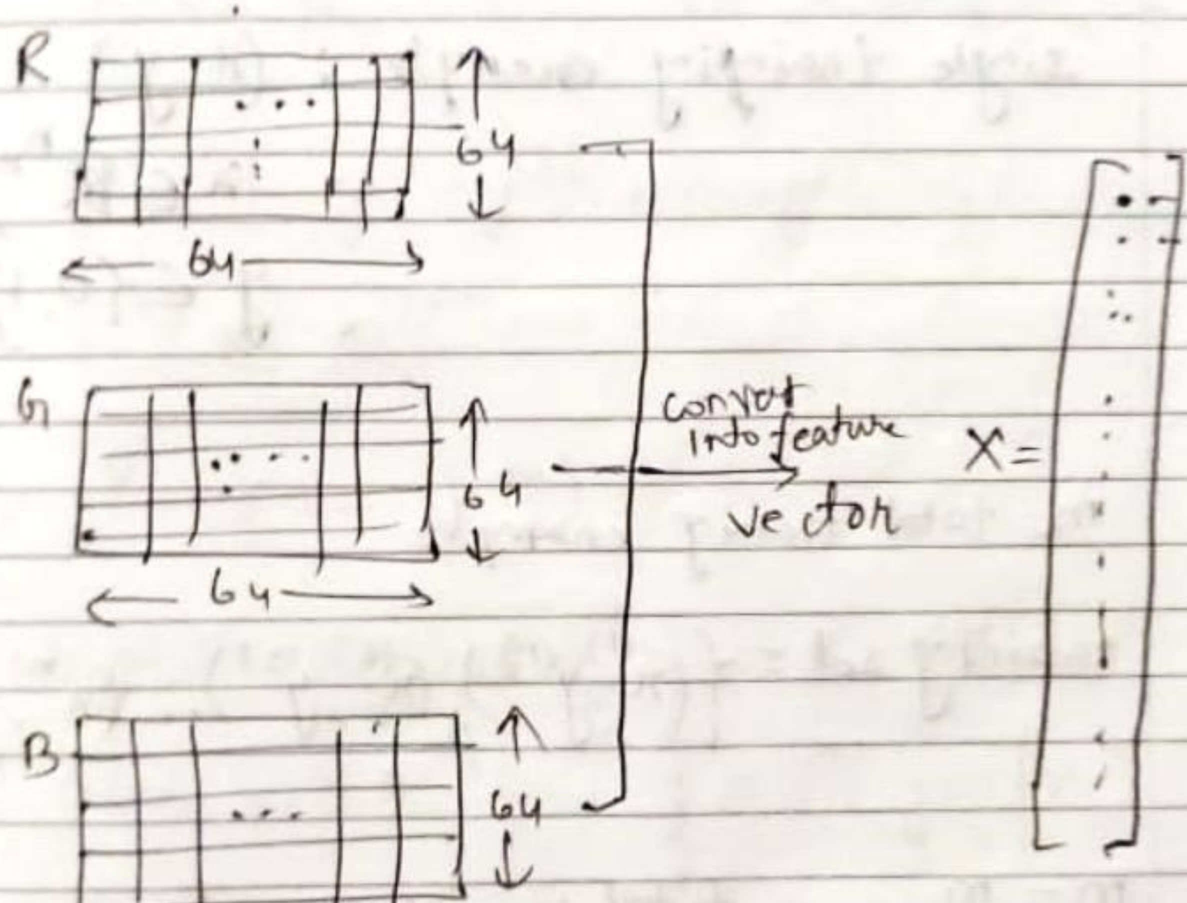
Binary classification :-



$\longrightarrow$ cat (1) or Not cat (0)



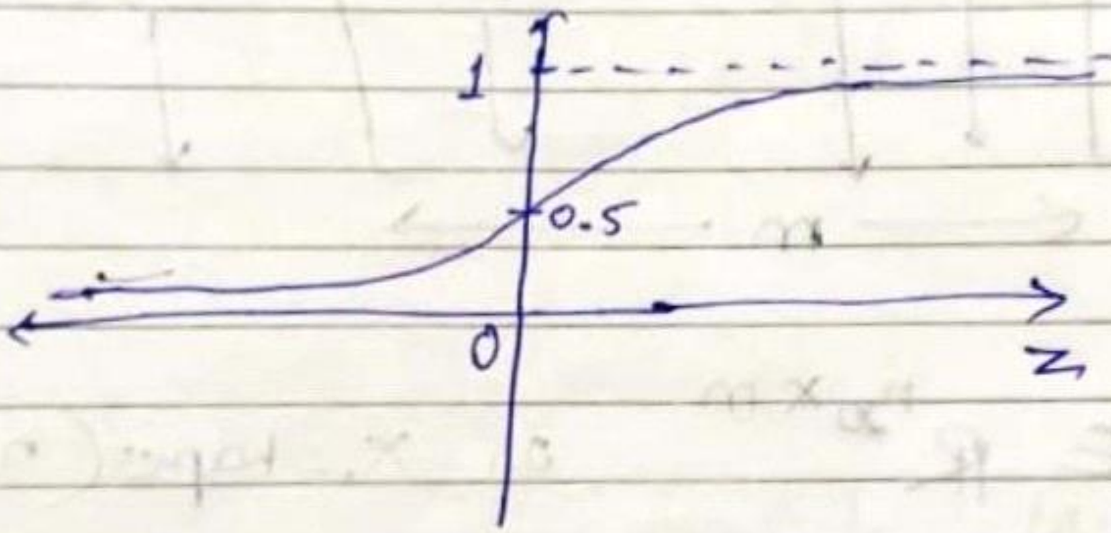
where $len(x) = 64 \times 64 \times 3$

$n_x = 12288$

$RGB = (64, 64, 3)$

Date :
Page no.

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



sigmoid function

Another alternative to b vector is

$x_0 = 1$ so that now $x \in \mathbb{R}^{n_x+1}$

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

$$\text{where } \theta = \begin{bmatrix}\theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n\end{bmatrix} \begin{matrix} \to b \\ \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} w \end{matrix}$$

will not use here in this class

set of all training examples

$$X = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} \; \begin{matrix} \uparrow \\ n_x \\ \downarrow \end{matrix}$$

$$\longleftarrow m \longrightarrow$$

$X \in \mathbb{R}^{n_x \times m}$ or $X.\text{shape} = (n_x, m)$

$Y = [y^{(1)}, y^{(2)} \dots y^{(m)}]$

$Y \in \mathbb{R}^{1 \times m}$

$\therefore$ Y.shape $= (1, m)$

Logistic Regression :-

Given $x \in \mathbb{R}^{n_x}$, $\hat{y} = P(y=1 \mid x)$

Parameter θ or $w \equiv$ parameter vector

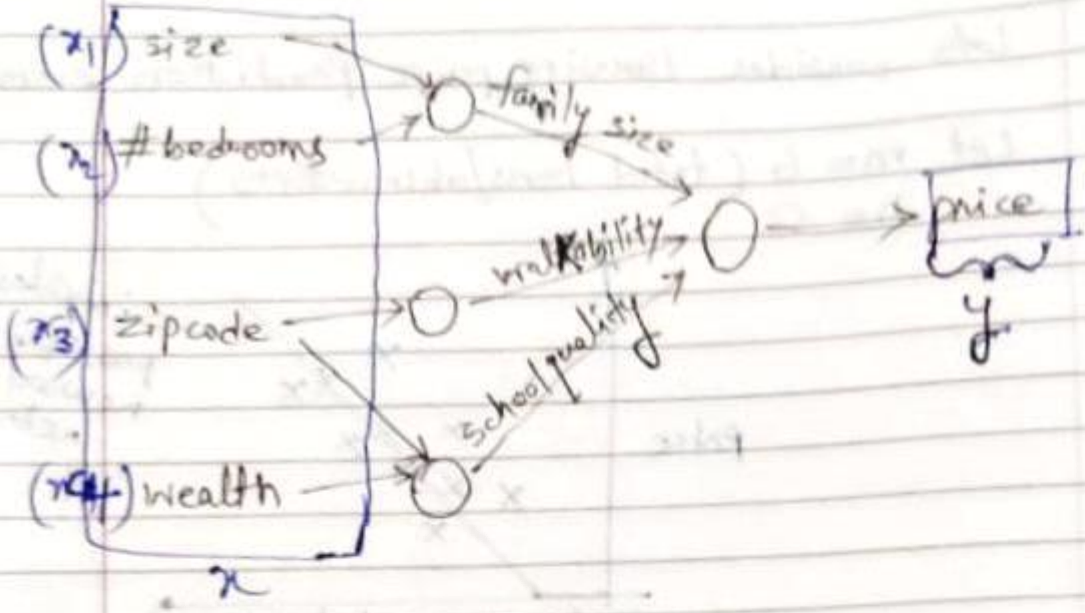
$w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

Output $\hat{y} = w^T x + b$ $\rightarrow$ will not work good as $-\infty \leq \hat{y} \leq \infty$

but we want $\hat{y} \in [0,1]$

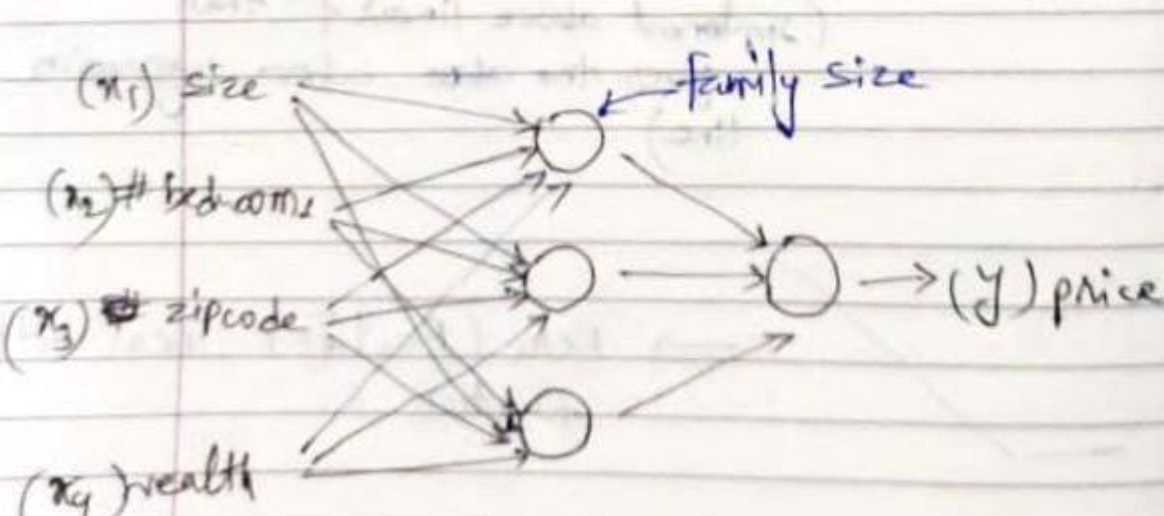
So

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



All we need to give is input x & output y in training set and all the intermediate levels / hidden layers will be taken care automatically.

↓ turns out to become



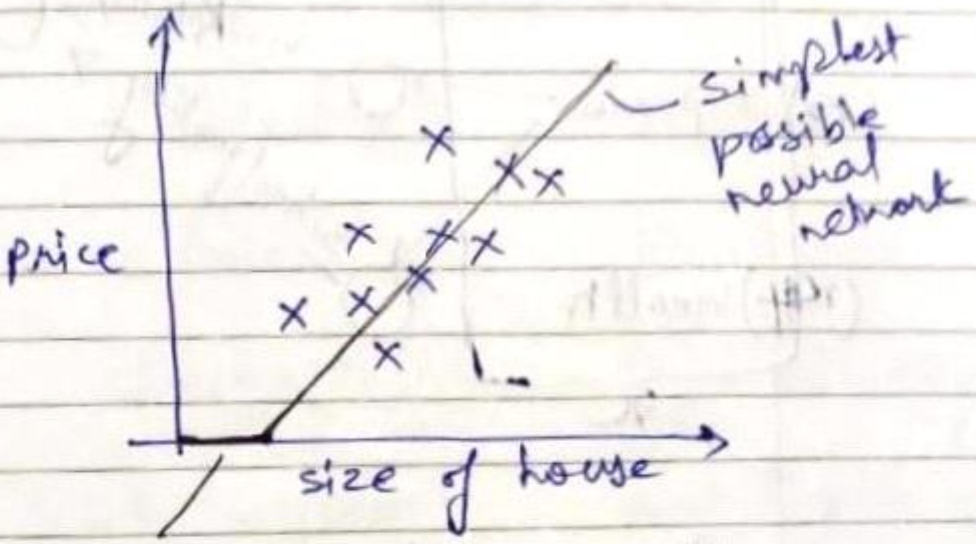
Although family size is just dependent on size of house & # bedrooms we will say well neural network

Q What is Neural Network?

Lets consider Housing price Prediction example.

Let $m = 6$ (total rows/observations)

$n = 2$



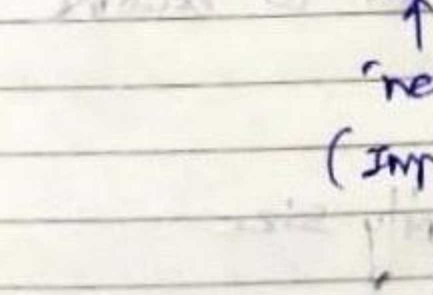
~~size~~ ~~size~~
prices cannot be negative

size $\longrightarrow$ ○ $\longrightarrow$ price

↑

'neuron'

(Implement above linear f^n that gives the ~~abo~~ above regression line)



$\longrightarrow$ Relu (Reitified linear Unit) function

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

Note: $a^{(i)} = \hat{y}^{(i)} = h_\theta(x^{(i)})$

Date :

Page no.

$w_1, w_2, x_1, x_2 \in \mathbb{R}^{m \times 1}$

Lets initialize

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

Let total features ($n_x = 2$) i.e. x_1, x_2

For $i = 1$ to m

{

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

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

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

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

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

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

may need to write the for loop if lets say $n_x = k$ features

$$\boxed{\begin{array}{l} \text{for } j = 1 \text{ to } k \\ \{ \\ \quad dw_j = dw_j + x_j^{(i)} dz^{(i)} \\ \} \end{array}}$$

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

$$J = J/m \longrightarrow \text{gives } \frac{1}{m} \sum_{i=1}^{m} L(a^{(i)}, y^{(i)})$$

$$dw_1 = dw_1/m \longrightarrow \text{gives } \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_1} (a^{(i)}, y^{(i)})$$

$$dw_2 = dw_2/m$$

$$db = db/m$$

}

Date :
Page no.

you decide whatever this node "family size" should be

(*) Supervised learning with neural network :-

	Input (x)	Output (y)	Application
(1)	Home features	Price	Real estate
(2)	Ad, User Info	Click on ad? (0/1)	Online Advertising
(3)	Image	Object (1,...,1000)	Photo-tagging
(4)	Audio	Text transcript.	Speech Recognition
(5)	English	Hindi	Machine translation
(6)	Image/Radar Info	Position of other cars	Autonomous driving

→ Slightly different types of Neural networks are used for different applications.

For above examples

(1) Real state (Home features → price) } Standard Neural
(2) Online Advertising (Ad → Click or Not) } Network
(3) ~~Ima~~ Photo tagging (Image → lots of Images) — CNN
(4) Speech Recognition ⟶ RNN
(5) Machine translation ⟶ RNN
(6) ~~Image~~ Autonomous driving ⟶ Hybrid NN

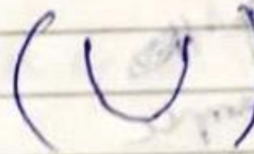
Gradient Descent

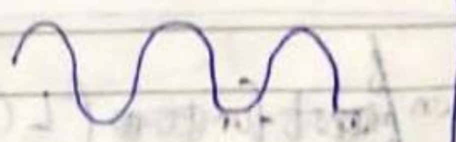
$$\hat{y} = \sigma(w^T x + b), \quad \sigma(z) = \frac{1}{1+e^{-z}}$$

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

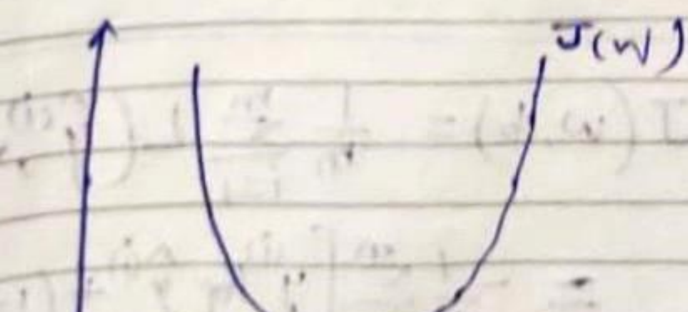
$$= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log \hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})$$

Find w, b that minimizes $J(w,b)$

$J(w,b) \rightarrow$ convex function ()

[Non-convex f^n: ]

Lets ignore b for now for simplicity



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

$= \frac{1}{1+e^{-\theta^T x}}$

Logistic regression cost function :-

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

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

want $\hat{y}^{(i)} \approx y^{(i)}$

Prediction for ith ~~training the~~ training example

label for ith training example

$\boxed{\hat{y} \equiv h_\theta(x)}$

Loss function ~~cost function~~ $(L(\hat{y}, y)) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$

for single training example

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

for entire training set

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

will try to find w & b that minimizes cost f^n J

So in binary classification our goal is to learn a classifier that can input an image ~~representat~~ represented by feature vector X and predict the corresponding label y is 1 or 0.

Notation we will use :-

single training example : (x, y)

$x \in \mathbb{R}^{n_x}$,

$y \in \{0, 1\}$

m = total training examples

training set = $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \dots (x^{(m)}, y^{(m)})\}$

$m = m_{train}$ = total training examples

m_{test} = total test examples

Now we have to come up with a cost function that is applicable to logistic regression & is convex f?.

→ cost function for logistic regression:-

$$\boxed{\text{cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y=1 \\ -\log(1-h_\theta(x)) & \text{if } y=0 \end{cases}}$$

cost$(h_\theta(x), y)$ refers to the cost/penalty the learning algo has to pay when it predicts $h_\theta(x)$ & actual label is y.

So **FINAL vectorized Logistic Regression Implementation**

$$Z = w^T X + b \quad (\text{i.e. } np.dot(w^T, X) + b)$$

$$A = \sigma(Z)$$

$$dZ = A - Y$$

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

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

So now Gradient descent code will be

$$w := w - \alpha dw$$

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

↳ with this we just implemented a single iteration of gradient descent without using a single 'for' loop.

→ But to implement multiple iterations of gradient descent we still need a 'for' loop even in vectorized implementation, for the number of iterations.

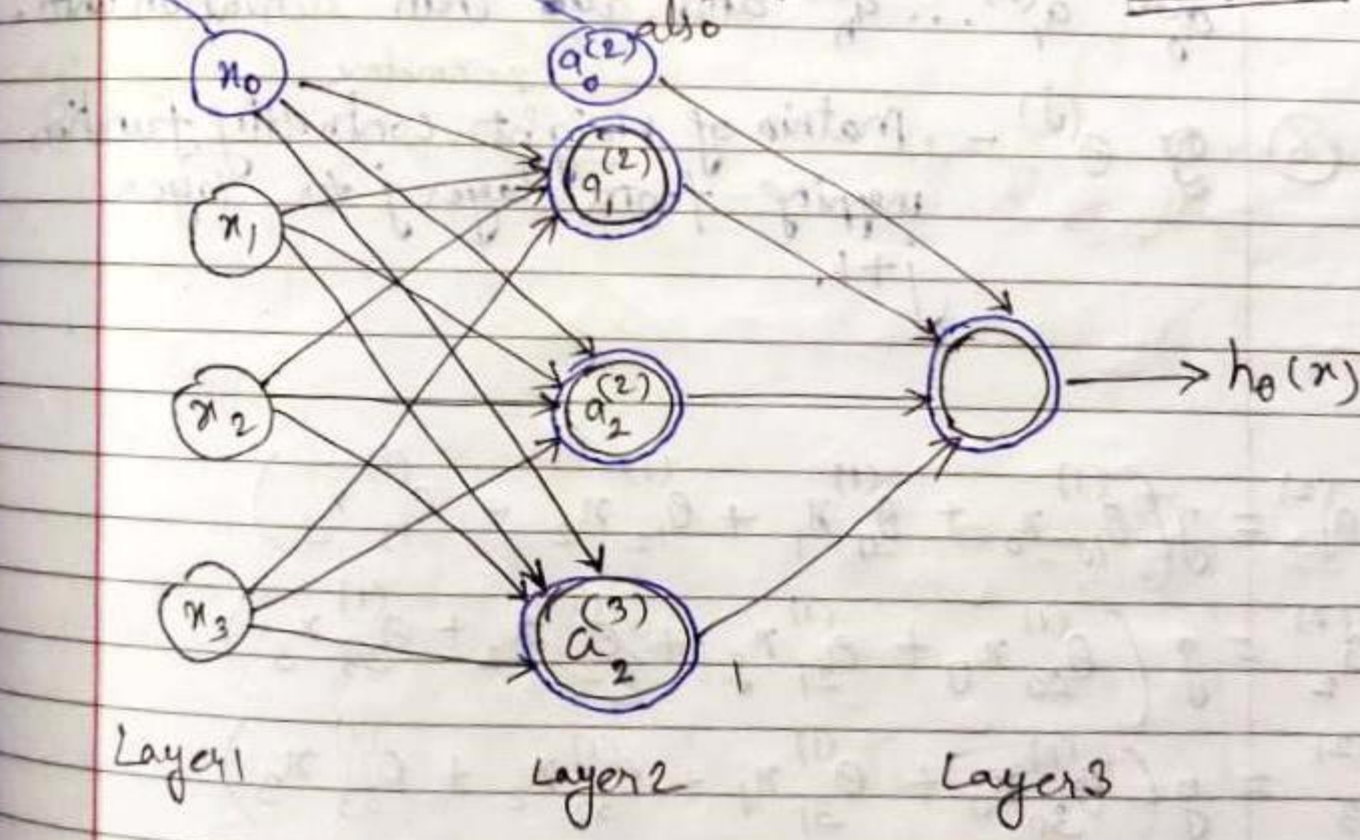
bias unit that usually outputs value 1.

Date
DELTA Pg No

→ ~~Sometimes~~ when we talk about neural networks sometimes we will say that this is a neuron with a ~~simo~~ sigmoid/logistic activation function.

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

usually we call this as parameters but in neural networks literature we also refer this as "weights"



In neural networks

① The first layer (Layer 1) is called input layer

② The final layer (layer 3) is called output layer because that layer has the neuron that ~~computes~~ ~~the~~ outputs the final value computed by the hypothesis.

Neural network:- Much better way to learn complex ~~hypothesis~~ non linear hypothesis even when your input feature space is large

Neuron model:-

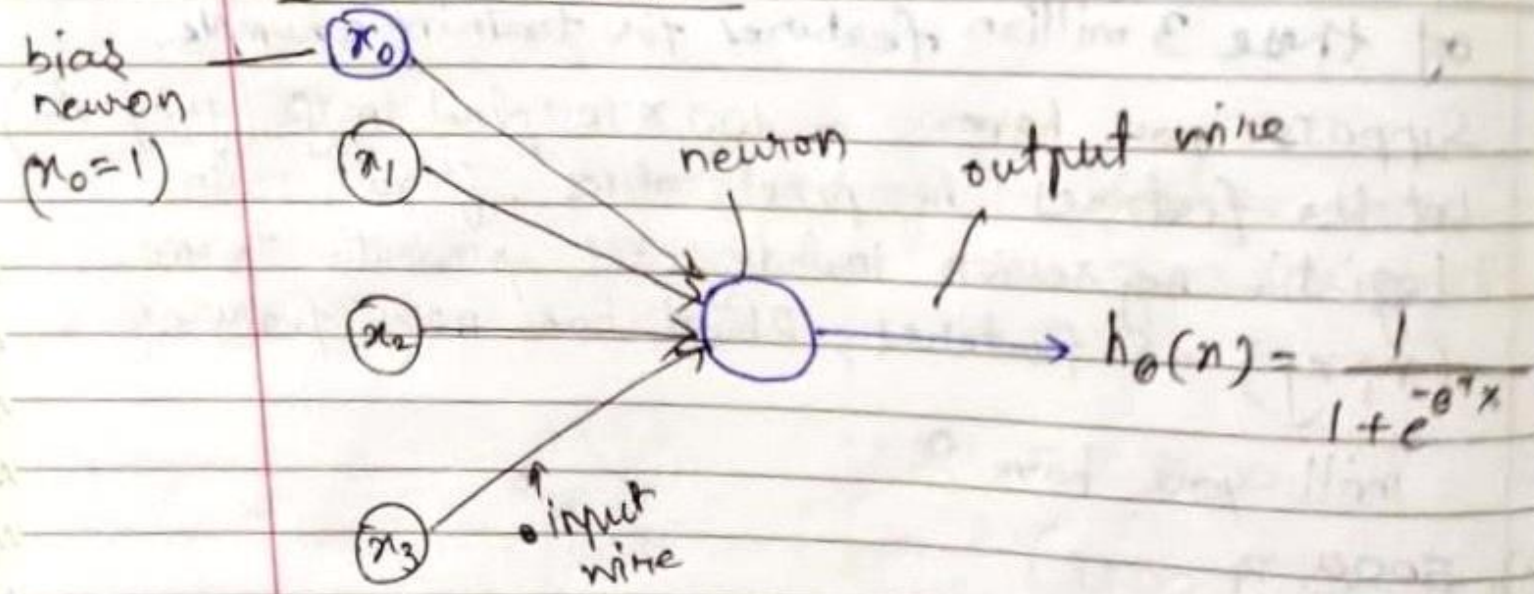
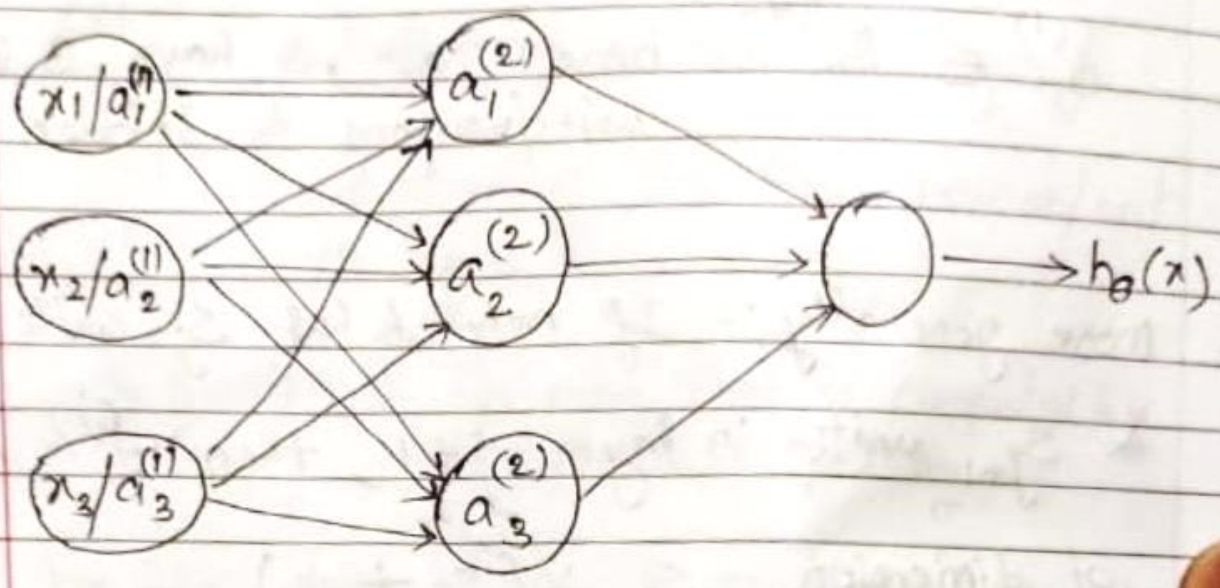


Diagram with single neuron

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Forward Propagation : Vectorized Implementation



Let

$$z_1^{(2)} = \theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3$$

$$z_2^{(2)} = \theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3$$

$$z_3^{(2)} = \theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3$$

$$\Rightarrow a_1^{(2)} = g(z_1^{(2)})$$

$$a_2^{(2)} = g(z_2^{(2)})$$

$$a_3^{(2)} = g(z_3^{(2)})$$

[So these $z_j^{(i)}$ values are just ~~we~~ linear combination of the input values x_0, x_1, x_2, x_3 that go into a particular j^{th} neuron of i^{th} Level]

We have 3 input units/neurons
3 hidden units

Date
DELTA Pg No.

(3) Layer 2 is called the hidden layer. We can have more than 1 hidden layers also.

(4) $a_i^{(j)}$ — Activation of neuron i in layer j

eg. $a_1^{(2)}$ — activation of the first neuron/unit in layer 2.

Activation ⇒ The value that is computed by and that is output by

(5) We label these intermediate or "hidden" layer nodes $a_0^{(2)}, a_1^{(2)} \ldots a_n^{(2)}$ and call them "activation units".

(6) $\Theta^{(j)}$ = Matrix of weight (parameters) controlling function mapping from layer j to layer j+1.

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3\right)$$

$$h_\Theta(x) = a_1^{(3)} = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)$$

clearly from the curve we can observe that

since we draw for

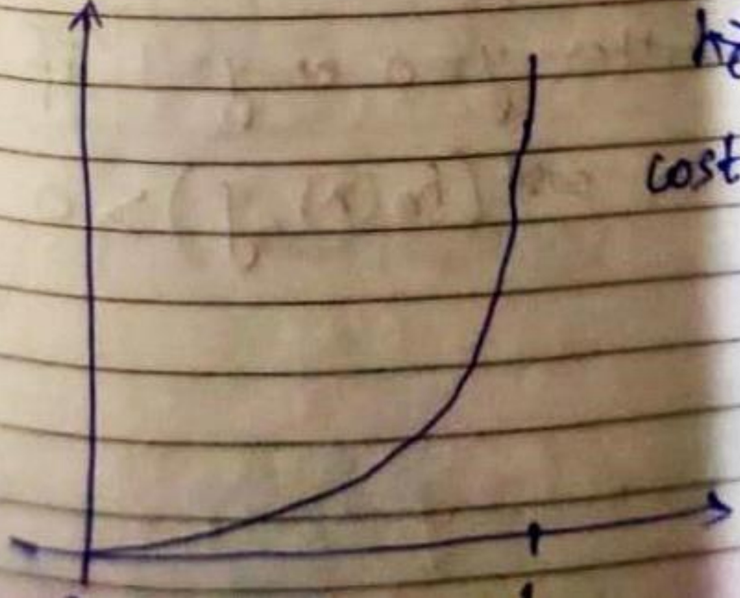
$$cost(h_\theta(x), y) = -\log(h_\theta(x)) \text{ for } y=1$$

so when $h_\theta(x) = 1$ & $y = 1 \Rightarrow cost(h_\theta(x), y) = 0$

$h_\theta(x) \Rightarrow 0$ & $y = 1 \Rightarrow cost(h_\theta(x), y) \simeq \infty$

so if we go and say the patient that he has malignant tumor but it is actually benign then the learning algo has to pay $\simeq \infty$ cost/penalty for wrong prediction

→ If $y = 0$:-



$\cancel{h_\theta(x) = 0}$

$$cost(h_\theta(x), y) = -\log(1 - h_\theta(x))$$
for $y = 0$