ROC curve helps us in deciding the threshold.
→ under the ROC curve tells us the accuracy → Area

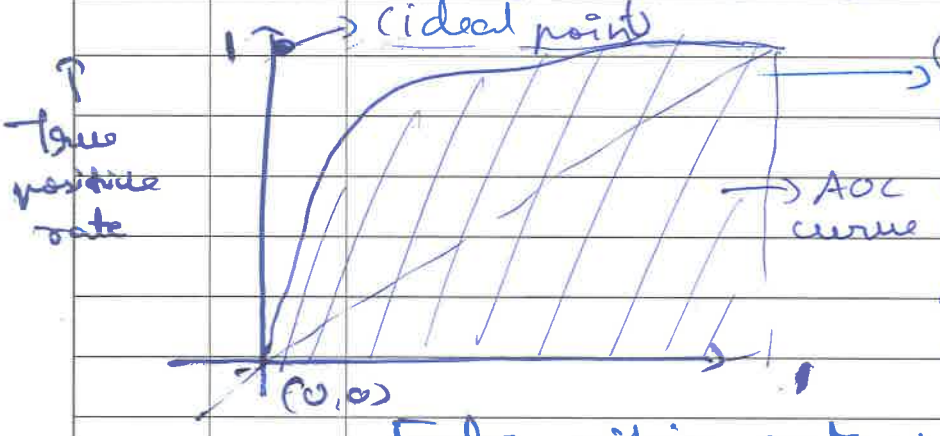
INDEX

Name _____ Std _____ Sec _____

Roll No. _____ Subject _____ School/College _____

Sl. No.	Date	Title	Page No.	Teacher Sign/ Remarks

AUC Curve and ROC curve (Receiver operator characteristic)



Any point on this lines tells us that the number of +ve cases correctly classified as +ve by total +ve cases is equal to the -ve cases classified as +ve by total -ve cases. The farther the point is from the line, the better it is

(True positive rate :- No. of positive cases correctly classified as positive by total no. of positive cases at that threshold)

(False positive rate :- No. of negative cases classified as positive by total no of negative cases.)

$$\text{True positive rate} = \text{sensitivity} = \frac{TP}{TP + FN}$$

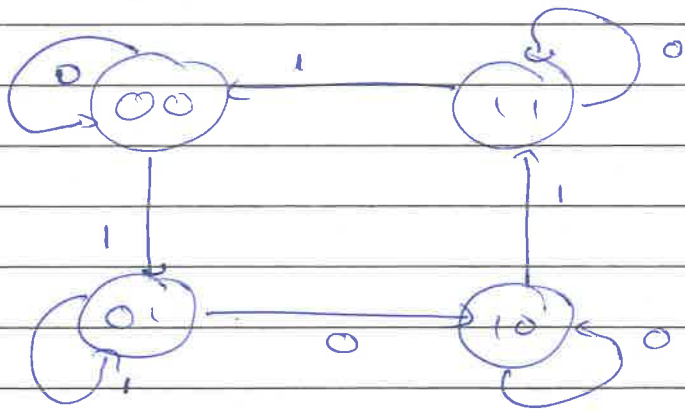
$$\text{False positive rate} = 1 - \text{specificity} = 1 - \left(\frac{(TN)}{TN + FP}\right) = \frac{FP}{TN + FP}$$

→ No. of -ve cases correctly classified as -ve by total number of -ve cases

The greater is the AUC for a ROC curve, better is the model.

DLD

Design procedure for sequential circuits.

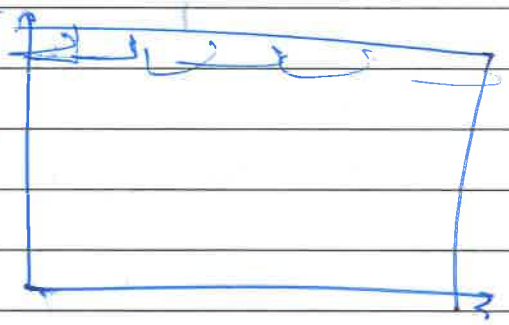


Q) We wish to design a synchronous sequential circuit whose state diagram is shown. The type of flip flop to be used is D.

Q_A	Q_B	X	Q_A^+	Q_B^+	T_A	T_B
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	0	0	0
1	0	1	1	1	0	1
1	1	0	1	1	0	0
1	1	1	0	0	1	1

A	B	T
0	0	0
0	1	1
1	0	1
1	1	0

100



100.

P_1 P_2 P_3 P_4 — — — P_{100}

P_1 P_2 P_3 P_4 — — — —

10000

$y = F_0\theta_0 + S\cdot\theta_2$

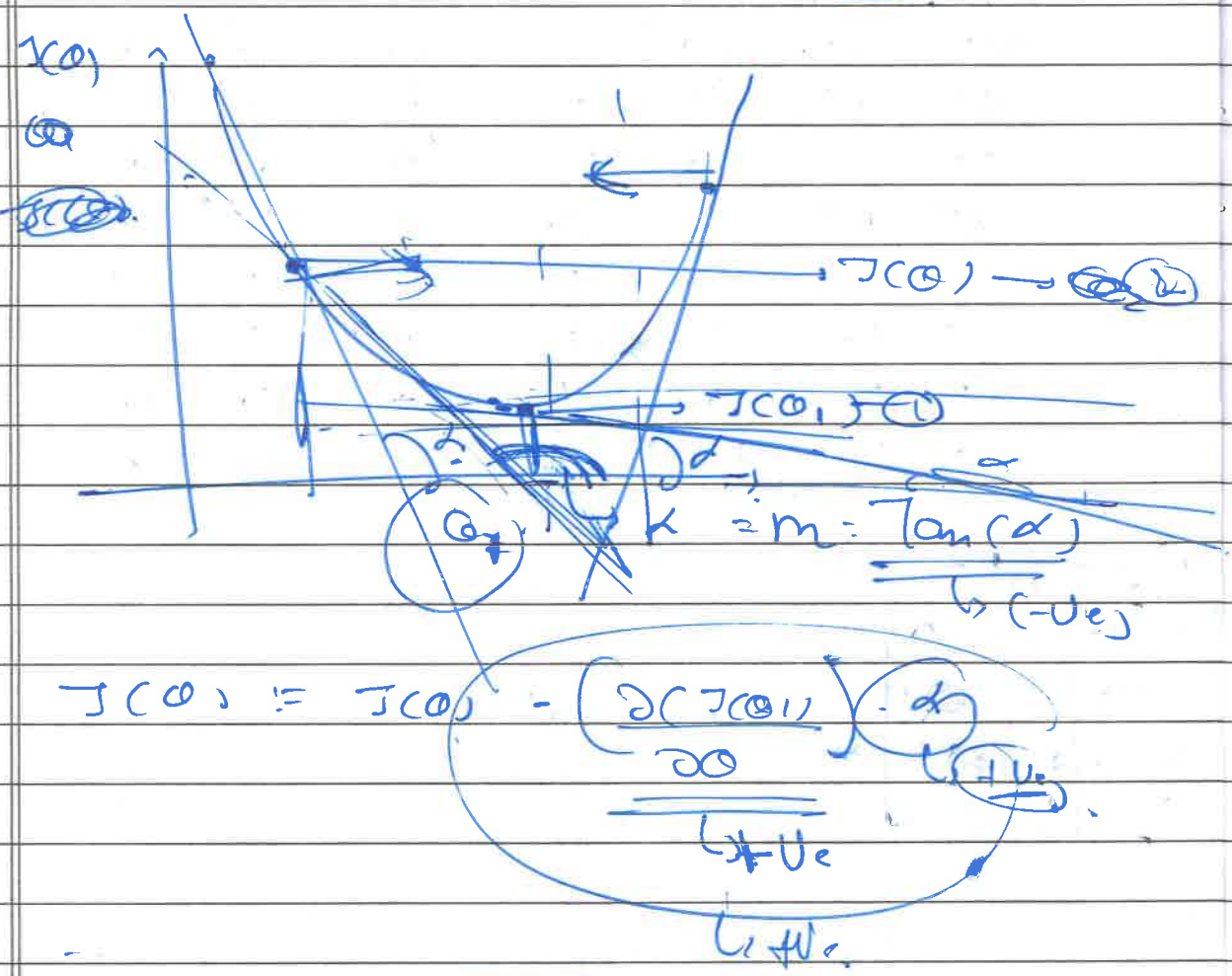
papergrid

Date:

	Floor (F)	size (S)	Predicted (y)	Actual (A)
(1)	F_1	S_1	y_1	A_1
(2)	F_2	S_2	y_2	A_2
(3)	F_3	S_3	y_3	A_3
⋮				
(n)	F_n	S_n	y_n	A_n

$$\text{Cost} = \frac{(y_1 - A_1)^2}{2} + \frac{(y_2 - A_2)^2}{2} \cdots \frac{(y_n - A_n)^2}{2}$$

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y - A)^2$$



$$J(\theta_1) = J(\theta) - \left(\frac{\partial J(\theta)}{\partial \theta}\right)\cdot\alpha$$

(+Ve)

(+Ve)

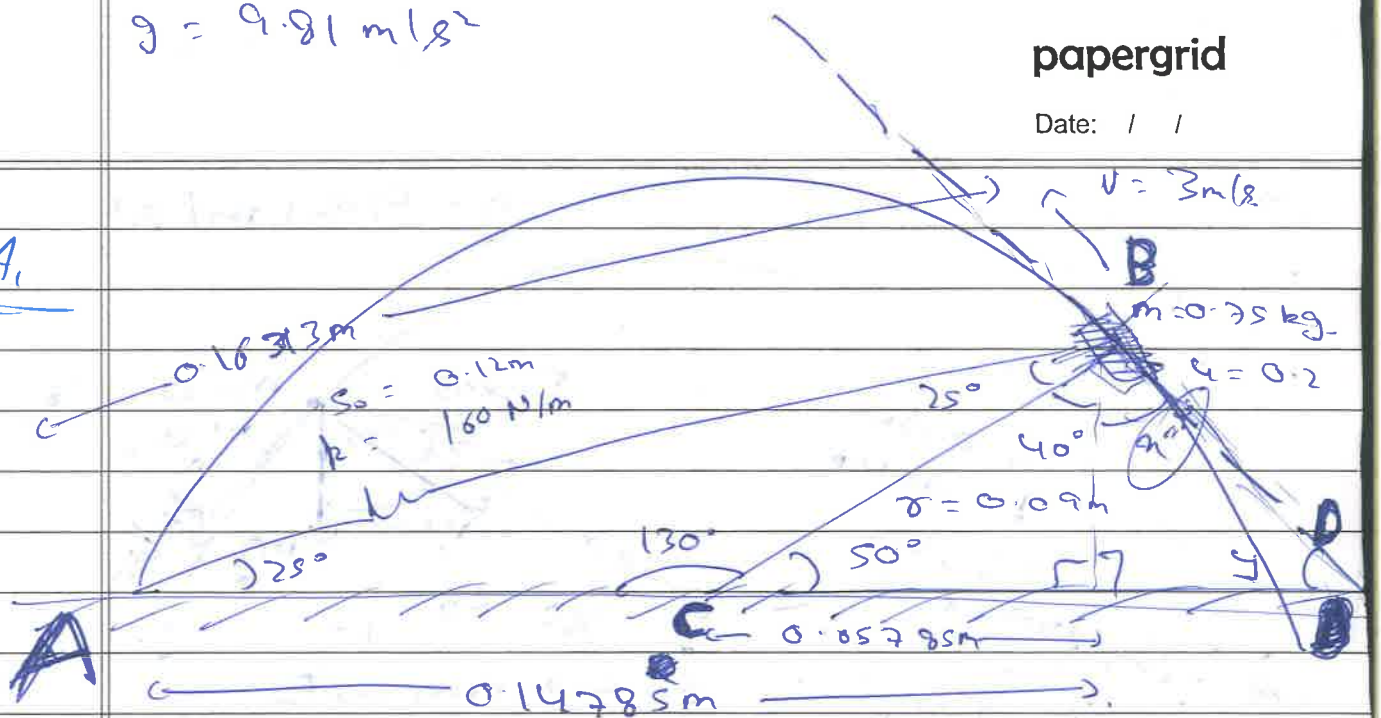
$y = F\theta_0 +$

$y = F^2\theta_0 + F\theta_2 + \sqrt{F}_2$

$+ S \cdot$ ——

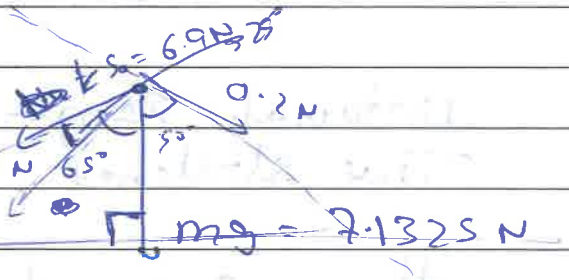
$+ F = S$

$g = 9.81\ m/s^2$



Spring force = (0.04313) × (160)
= 6.9 N.

F.B.D :-



In △ ABD :- $y = x^2 + 3 + 2$

$25 + 65 + x + y = 180$

$\Rightarrow 90 + x + y = 180$

$\Rightarrow x + y = 90$ ①

In △ BCD :-

$N = 6.9 \cdot \cos 25$

$N = 6.2535$

$F = 1.2507$

$a = \frac{v^2}{r} = \frac{(3)^2}{(0.09)} = 100\ m/s^2$

$2x + 1 = 0$

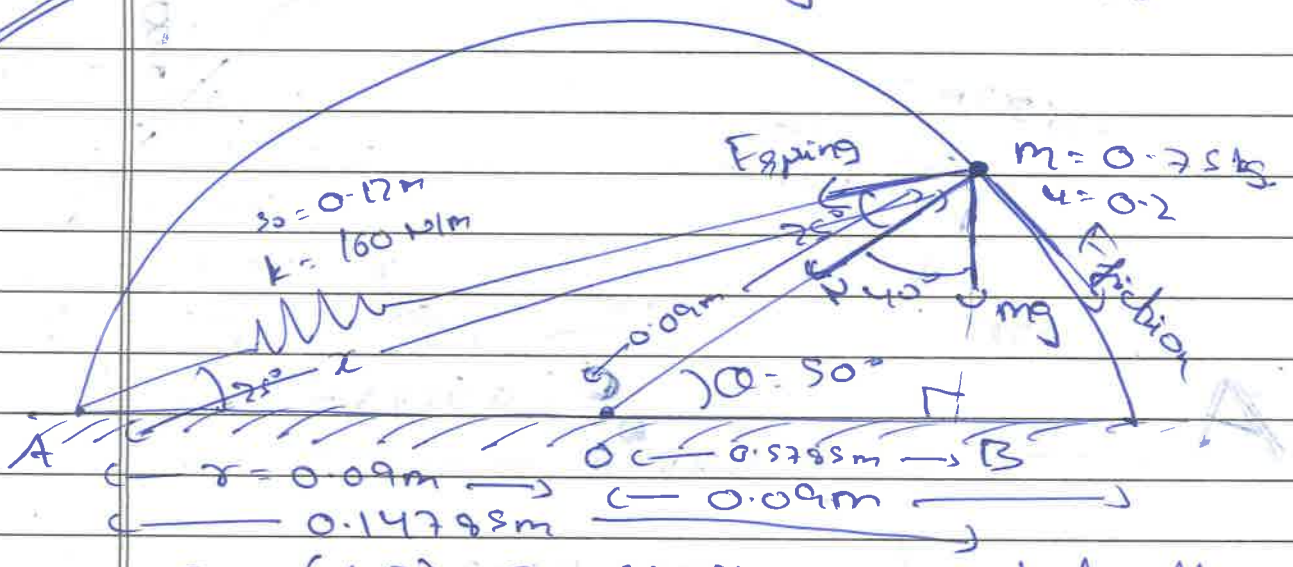
$x = -\frac{1}{2}$

$\frac{dy}{dx} = 0$

$y = \left(-\frac{1}{2}\right)^2 + 3 - \frac{1}{2}$

Q11.

$g = 9.81 \text{ m/s}^2$



$x = (AB) \cdot \sec(25°)$ = current length of spring

$\Rightarrow x = (0.147885) \cdot \sec(25°)$

$\Rightarrow x = 0.16313\ m$

natural length = $s_0 = 0.12\ m$

Thus stretched length $= (0.16313 - 0.12)\ m$
$= (0.04313)\ m$

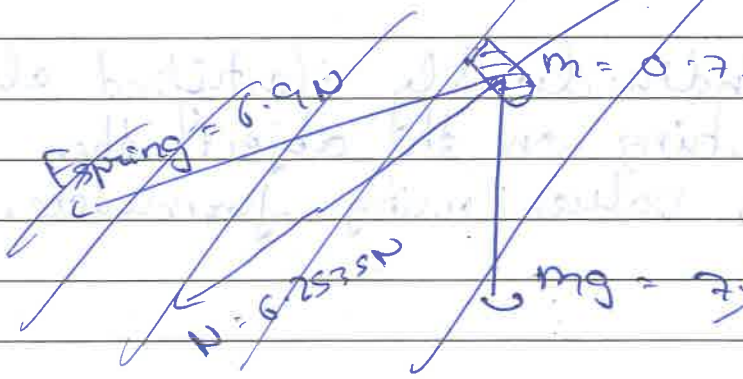
thus spring force $= (0.04313) \cdot k$
$= (0.04313)(160)$
$= 6.9\ N$

mg (gravitational force) $= (0.75)(9.81)$
$= 7.1325\ N$

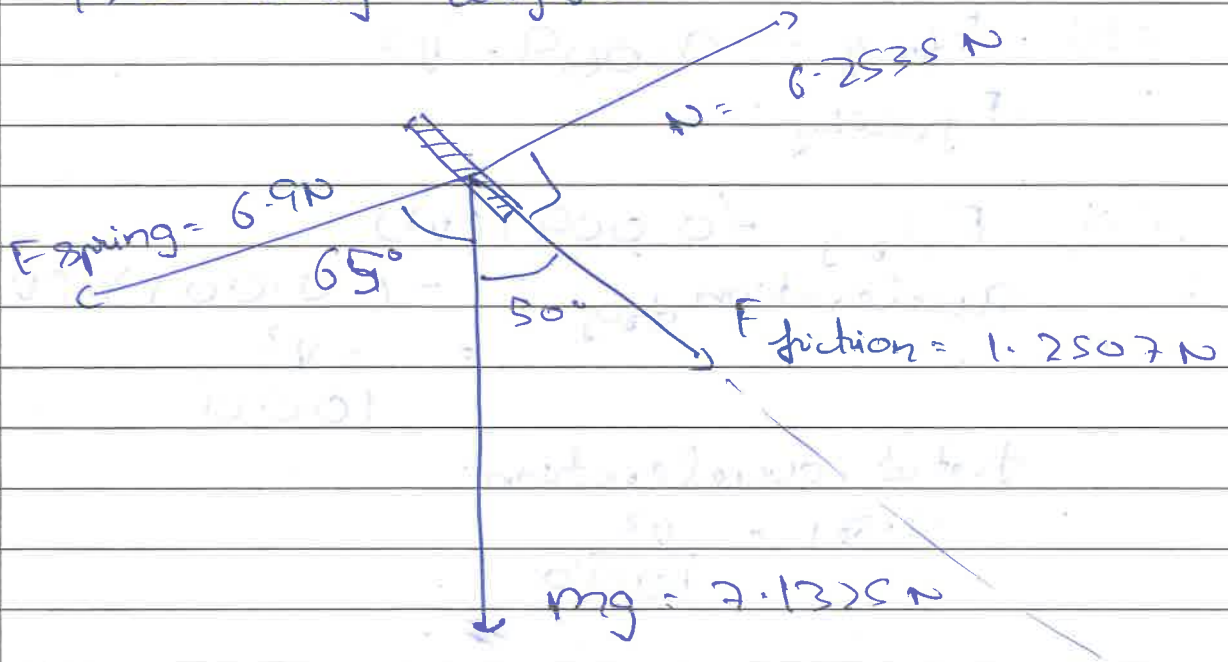
Normal force = (spring force) $\cdot \cos(25°)$
$= 6.2535\ N$

Frictional force = (Normal) $\cdot$ (friction co-efficient)
$= (6.2535) \cdot (0.2)$
$= 1.2507\ N$

1a) Free body diagram:-



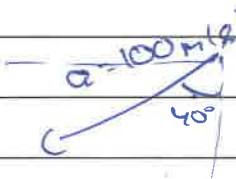
1a) Free body diagram



1b) N = 6.2535 N

1c) acceleration = $\frac{v^2}{r} = \frac{(30)^2}{(0.09)} = \frac{9}{0.09}$

= 100 m/s²



= (100 sin 40 $\hat{i}$ - 100 cos 40 $\hat{j}$)

$\Rightarrow \vec{a} = (64.27987\,61\,\hat{i} - 76.6\,\hat{j})\ m/s^2$

(d) First individually identified all the forces acting on the object then calculated their values using formulaes and F.B.D.

(2)

(2(a)) ~~$F_{drag} = 0.009 \cdot v^2$~~

~~$F_{gravity} =$~~

(2(c)) $F_{drag} = -(0.009)(v^2)$

$a_{drag} = -(0.001)\cdot(v^2)$

$= \frac{-v^2}{1000}$

total acceleration :-

$9.81 - \frac{v^2}{1000}$

~~acceleration~~ =

The bird will keep accelerating until its total acceleration becomes 0.

then until :-

$9.81 = \frac{v^2}{1000}$

$\Rightarrow v^2 = 9810$

$\Rightarrow v = 99.045\ m/s$

$\hookrightarrow v_t$ (maximum velocity)

Q2d) First found out the drag force using the final velocity, acceleration due to gravity. Then found out the work done by multiplying drag force with distance travelled. Then to calculate maximum velocity found the velocity at which the total downward acceleration would become 0.

Q3a)

$V_d = R \cdot \omega$

$= (1.5) \cdot (2.5)$

$\Rightarrow V_d = 3.75 \text{ m/s}$

$\alpha_{AD} = \omega^2 R$

A	T	I	S	H	A	K
0	1	2	3	4	5	6

$T = O(n)$ → linear search

0	1	2	3	4	5	6
10	20	30	40	50	60	70

∴ bottom = 0

mid = int(len(a)/2) top = 6

60 > 40 → 60 > mid

bottom = mid + 1

bottom = 50

mid = (bottom + top)/2 = 60

60 > 60

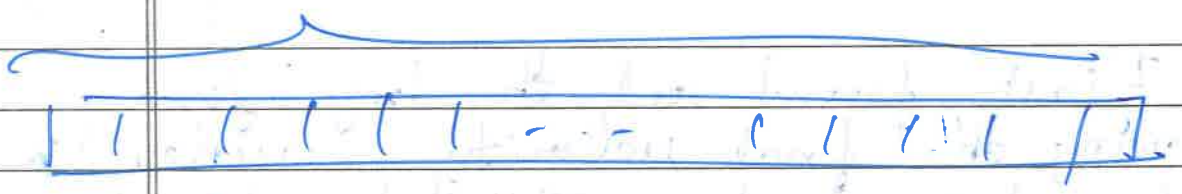
$O(\log(n))$

top = bottom → 0.

papergrid

Date: / · /

n



len = n → 0th

len = $\frac{n}{2^1}$ → 1st

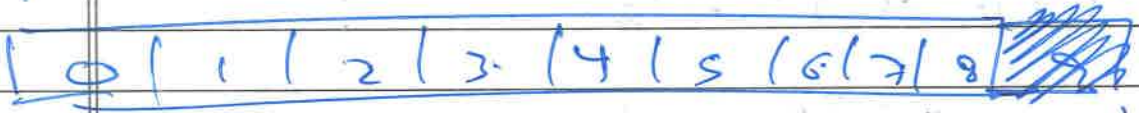
len = $\frac{n}{2^2}$ → 2nd.

$\frac{n}{2^x} = 1$

$2^x = n$

$x = \log_2(n)$

(9)



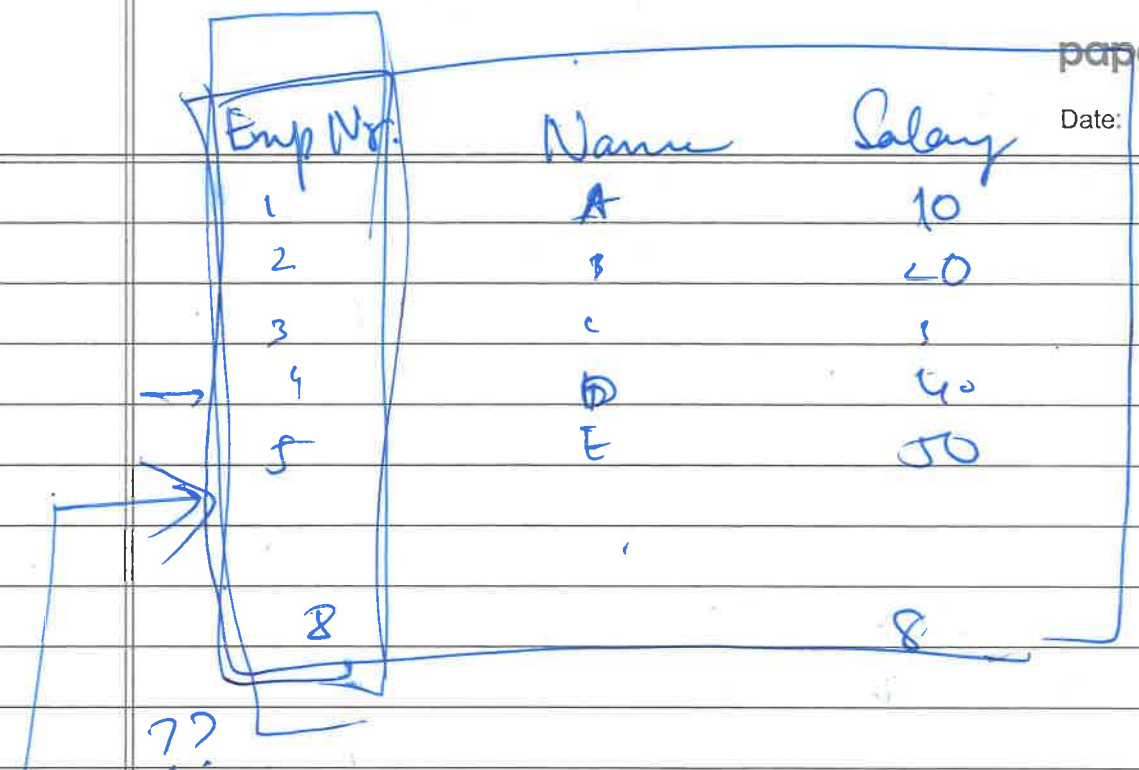
bottom = 0 ; top = 8 ; mid = 4

bottom = 5 ; top = 8 ; mid = 6

bottom = 7 ; top = 8 ; mid = 7

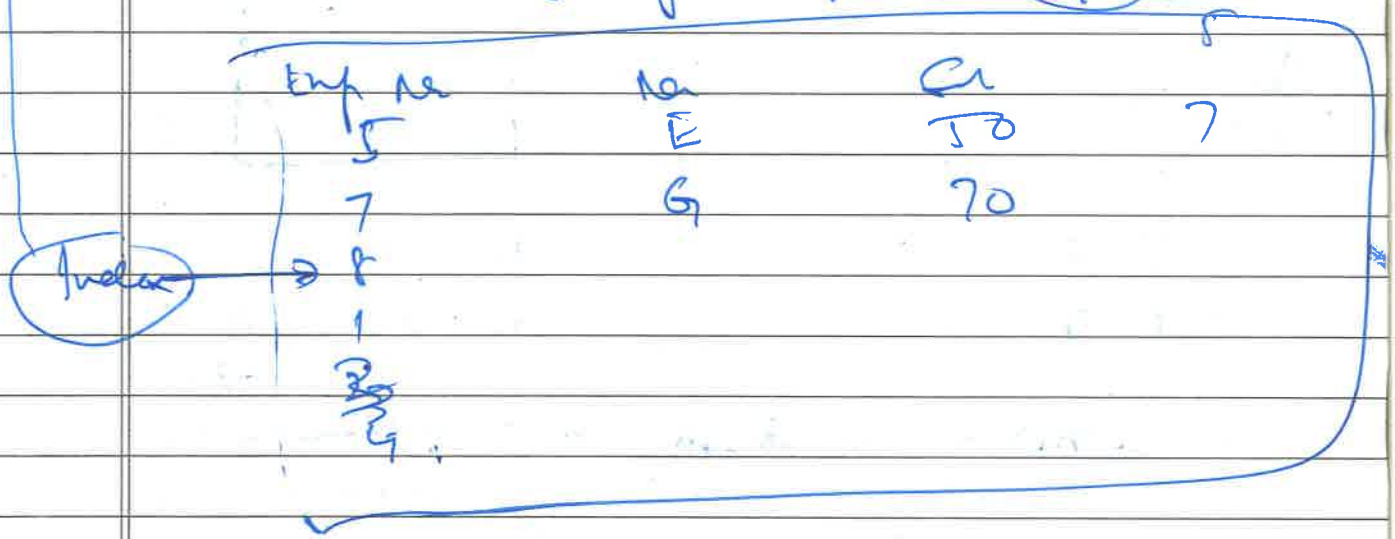
bottom = 8 ; top = 8

$\log_2 8 = 3$



Search

→ Select Salary of Emp No = (key) 1 to 5



number of Searches = n

$h \log(n)$ =

Key = k	Value
5	50
7	70
8	80
1	10
3	30
9	90
2	20

Hash table (function)

key → (7) → value → 70

key	Value	
5		0
7		1
8		2
1		3
3		4
9		5
2	50	6

len = 7

(i) linear probing → index

$F(2k+3)$ → $F((2k+3) \% \text{len})$

$F((10+3) \% 7)$

6

hash(Selq)

5000 ⟶ Mem address
10000 (5000)

when Selq 5000 ⟶ 5000

Column db

~~Row~~ RDBMS.

1, 3, 7

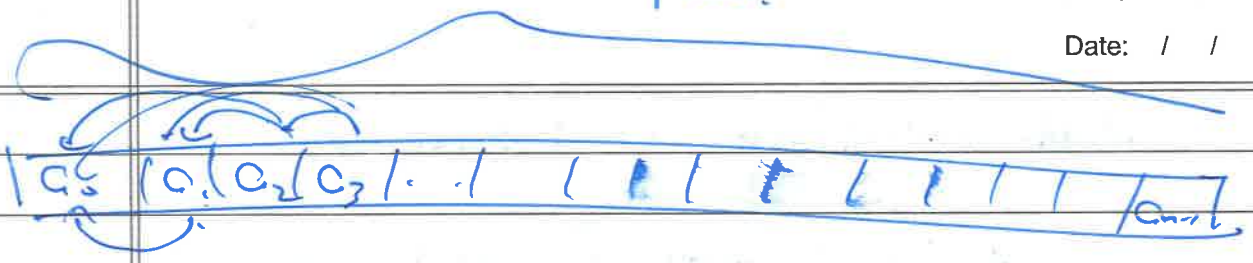
f(fnp) ⟶ malloc(address)

1 ⟶
2 ⟶ 10 (madd 1L)
5 2L
10 5L

→ f(Selq) → 5000 → (1, 5, 15) 5L

madd 10 bytes
(

n samples.



1 $\frac{(n)(n+1)}{2}$

2

3

.

.

.

4

multiple features (variables)

Size (feet^2) (x_1)	Number of bedrooms (x_2)	No. of floors (x_3)	Age of home (years) (x_4)	Price ($1000) ($y$)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
..	..	..	..	..
..	..	..	..	..

(m = no. of data)

notations:

n = number of features

$x^{(i)}$ = input (features) of i^{th} training example

$x_j^{(i)}$ = value of feature j in i^{th} training example.

$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$$

hypothesis :-

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

Eg. $h_\theta(x) = 80 + 0.1(x_1) + 0.01(x_2) + 3(x_3) - 2(x_4) \ldots$

Date: / /

=> $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots \theta_n x_n$

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

$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 \cdots \theta_n x_n$

$= \theta^T x$

$$= \begin{bmatrix} \theta_0 & \theta_1 & \cdots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Cost function :-

$$J(\theta_0, \theta_1, \theta_2, \theta_3 \ldots \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$J(\theta)$

Gradient descent :- (for linear regression)

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} [J(\theta_0, \theta_1 \ldots \theta_n)]$$

($J(\theta)$)

} (simultaneously update for every $j = 0, 1, 2, 3 \ldots n$)

Gradient descent :- (for multiple regression)..

{p.7.0}

repeat until convergence {

$$\theta_j := \theta_j - \alpha \left(\frac{1}{m}\right) \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update θ_j for $j = 0, \ldots, n$)

}

$$\rightarrow \theta_0 := \theta_0 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

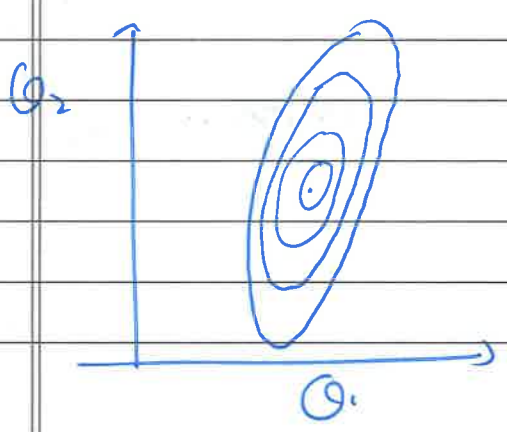
$$\rightarrow \theta_1 := \theta_1 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\rightarrow \theta_2 := \theta_2 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

#1 Feature scaling

idea:- make sure features are on similar scale

Eg. x_1 = size (0 - 2000 feet^2)
x_2 = number of bedrooms (1-5)



→ $x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$

→ $x_2 = \frac{\text{number of bedrooms}}{5}$

Get every feature into approximately $-1 \leq x_i \leq 1$ range

if we get :-

$-0.0001 \leq x_4 \leq 0.0001$

this is too small.

mean normalization :-

replace x_i with $x_i - \mu_i$ to make features have approximately 0 mean.

Eg. $x_1 = \frac{\text{size} - 1000}{2000}$

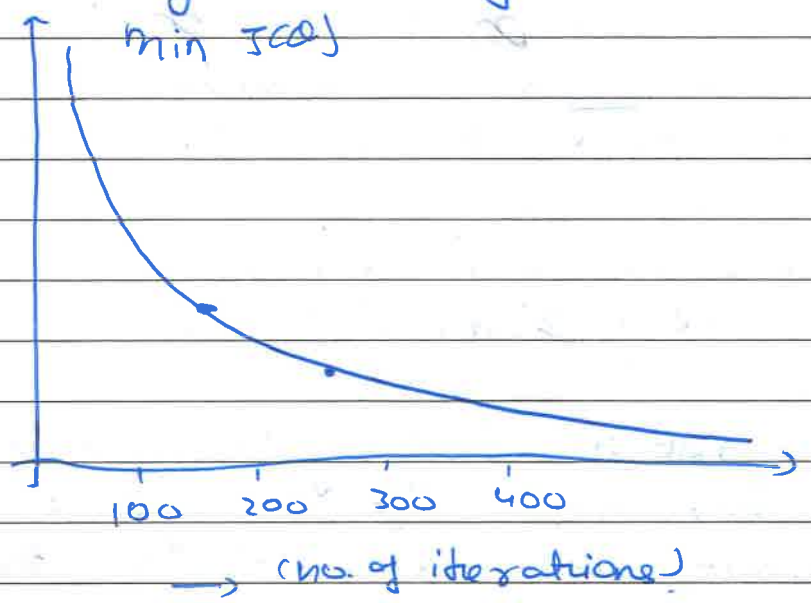
$x_2 = \frac{\text{bedroom} - 2}{5}$

then we can replace

x_i with $\frac{x_i - \mu_i}{s_i}$ → (avg. value of x_i in training set)

(s_i → range → max - min)

making sure gradient descent is working correctly :-

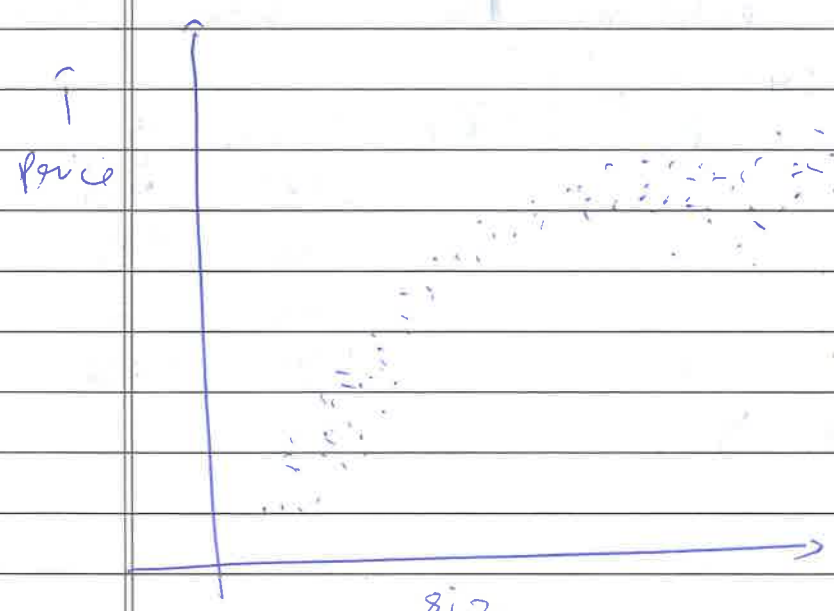


#1 Polynomial regression :-

let y be the price of the house and it depends upon length & breadth thus

$$y = \theta_2 (\text{length}) + \theta_3 (\text{breadth}) + \theta_0$$

So we could say that in a way the price depends upon the size (area) we are given :-



So we could in a way formulate

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$

So let $\sqrt{x_1}$ be x_2

So we have :-

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

Here feature scaling is important

So if we have the range of x_1 as 1000 then as $\sqrt{1000} \approx 33.33$ we could do :-

$x_1 = \frac{x_1}{1000}$ & $x_2 = \frac{x_2}{33.33}$

#1 Normal equation :-

$\theta = (X^T X)^{-1} X^T y$

	Size (feet^2)	Number of bedrooms	Number of floors	Age of home	Price ($1000)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$m = 4$

$$X = \begin{bmatrix} \longleftarrow x^{(1)} \longrightarrow \\ \longleftarrow x^{(2)} \longrightarrow \\ \longleftarrow x^{(3)} \longrightarrow \\ \longleftarrow x^{(4)} \longrightarrow \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}_{(m \times (n+1))}$$

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}_{(m \times 1)}$$ (m - dimensional vector)

normal equation	gradient descent
→ No need to choose alpha	→ Need to choose alpha
→ No need of iterations	→ Needs many iterations
→ $O(n^3)$	→ $O(kn^2)$
→ slows when n is large	→ works well when n is large

When implementing normal equation in octave we want to use (pinv) rather than (inv)

If $X^T X$ is non-invertible, common causes may be :-

(i) Two features may be very closely related (linearly dependent)

(ii) Too many features ($m \leq n$). In this case delete some features or use regularization.

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$ $X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

Unvectorized implementation :-

```
prediction = 0.0;
for j = 1:n+1,
    prediction = prediction + theta(j) * x(j)
end;
```

Vectorized implementation :-

```
prediction = theta' * x
```

→ It is a classification model.

Date: / /

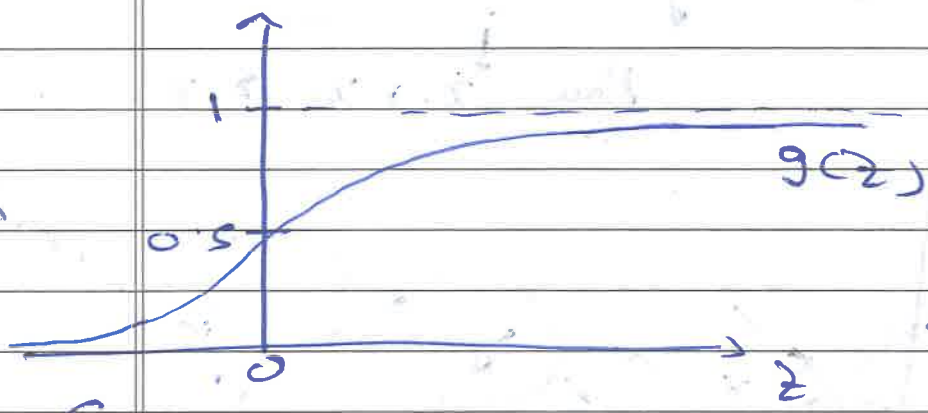
Logistic regression:-

want $0 \le h_\theta(x) \le 1$

$h_\theta(x) = g(\theta^T x)$

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

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



Interpretation of hypothesis output

$h_\theta(x)$ = estimated probability that $y=1$ on input x.

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumor size} \end{bmatrix}$

$h_\theta(x) = 0.7$

Patient has 70% chance of malignant tumor

If $h_\theta(x) \ge 0.5 \rightarrow y=1$

If $h_\theta(x) < 0.5 \rightarrow y=0$

→ we notice that $g(z) \ge 0.5$ when $z \ge 0$ then $h_\theta(x) \ge 0.5$ when $\theta^T(x) \ge 0$

Date: / /

let $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

let $\theta_0 = -3$; $\theta_1 = 1$; $\theta_2 = 1$

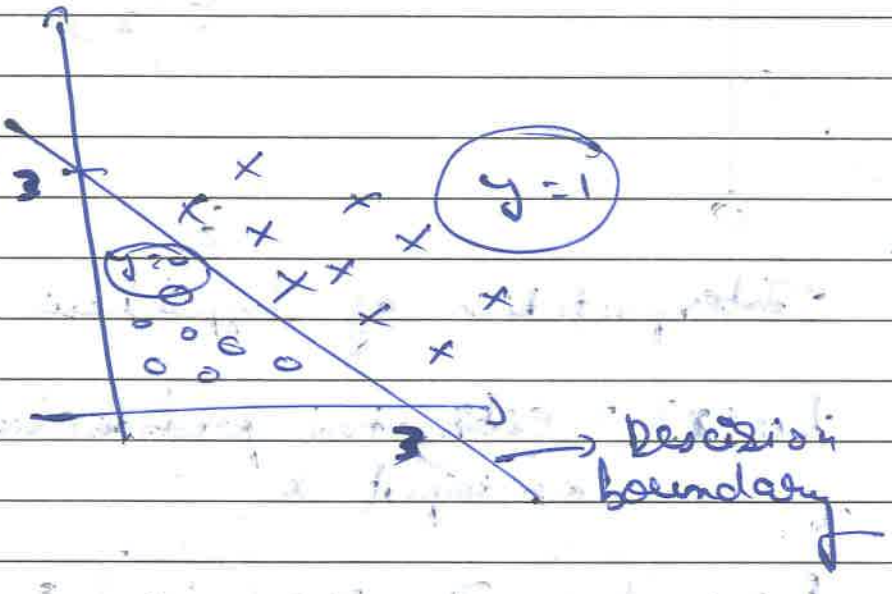
then

$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

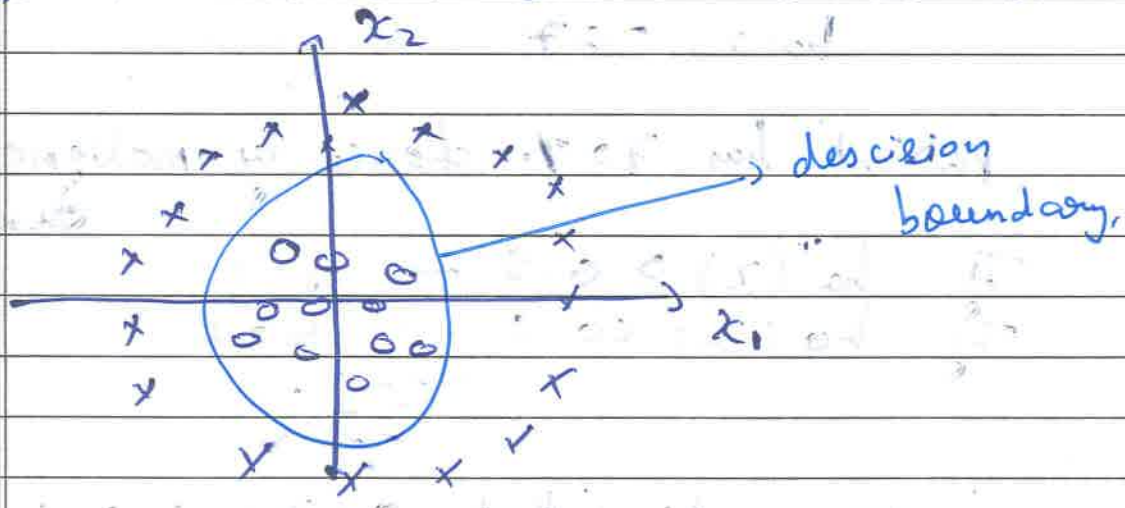
then predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

$\downarrow$

then $x_1 + x_2 \geq 3$



Q. Non-linear decision boundaries



$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$

predict "y = 1" if $-1 + x_1^2 + x_2^2 \geq 0$

$x_1^2 + x_2^2 \geq 1$

$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

Training set:-

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

m examples:- $x \in \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$ $x_0 = 1$ $y \in \{0, 1\}$

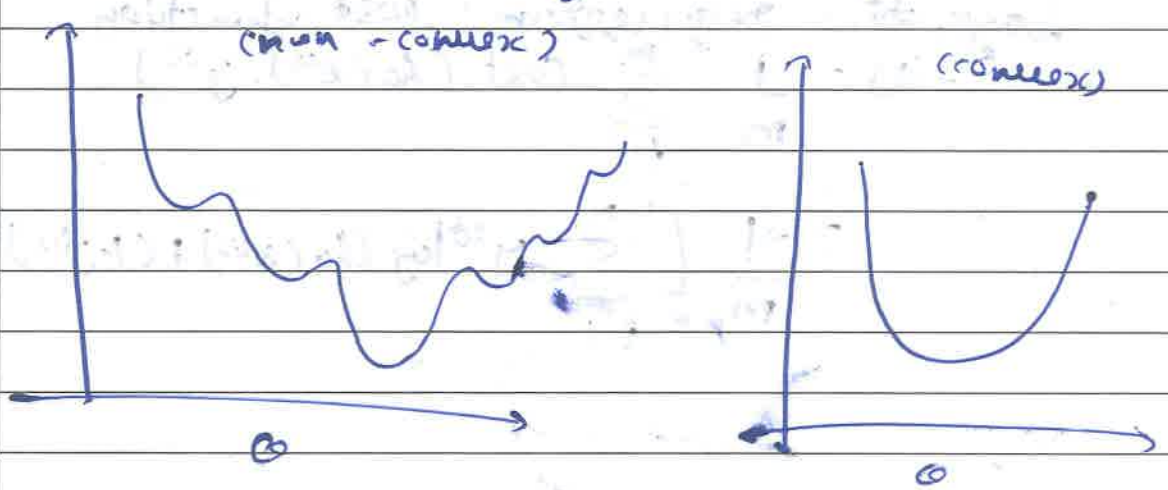
$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$

now we have to decide parameter θ.

cost function:-

linear regression:- $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \underbrace{\frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2}_{\text{cost}(h_\theta(x^{(i)}), y)}$

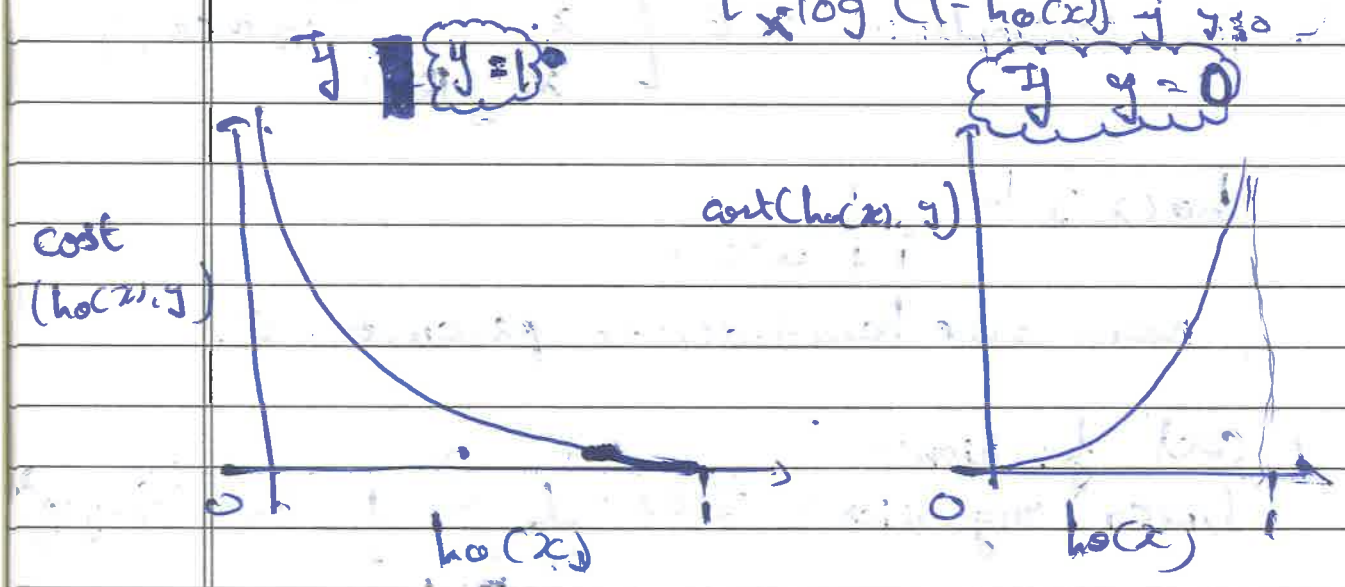
$\text{cost}(h_\theta(x), y) = \frac{1}{2} (h_\theta(x) - y)^2$



If we have a non-convex function then we won't be able to get to the global minimum through the gradient descent algorithm. And we get a non-convex graph if we choose $h_\theta = \frac{1}{1+e^{-\theta^T x}}$ for our cost function. Then we change our cost function.

Logistic regression cost function:-

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



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

Logistic regression cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$= \frac{-1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1 - h_\theta

To fit parameters θ :-

$$\min_{\theta} J(\theta)$$

To make a prediction given by new x :-

output $h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$

We want to minimise $J(\theta)$.

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} (J(\theta))$$

}

(simultaneously update for all θ_j).

Solving :-

Repeat {

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

}

Algorithm looks identical to linear regression.

optimization algorithms :-

- → Gradient descent
- → Conjugate gradient
- → BFGS
- → L-BFGS

Advantages :-
- No need to manually pick α
- Often faster than gradient descent

Disadvantages :-
- More complex.

Example :-

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

$$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

$$\frac{\partial (J(\theta))}{\partial \theta_1} = 2(\theta_1 - 5)$$

$$\frac{\partial (J(\theta))}{\partial \theta_2} = 2(\theta_2 - 5)$$

```
function [jVal, gradient]
        = costFunction (theta)
jVal = (theta(1)-5)^2 + (theta(2)-5)^2
        = costFunction (theta)
jVal
```

```
function [jVal, gradient]
        = costFunction (theta)
jVal = (theta(1)-5)^2 + (theta(2))^2;
gradient = zeros (2,1);
gradient(1) = 2*(theta(1)-5);
gradient(2) = 2*(theta(2)-5);
options = optimset ('GradObj', 'on', 'MaxIter', 100);
initialTheta = zeros(2,1);
```

```
options = optimset ('GradObj', 'on', 'MaxIter', 100);
initialTheta = zeros (2,1);
[optTheta, functionVal, exitFlag]
        = fminunc (@costFunction, initialTheta,
                   options);
```

$$\text{theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

function [jval, gradient] = costFunction (theta)

jval = [code to compute $J(\theta)$];
gradient(1) = [code to compute $\frac{\partial}{\partial \theta_1} J(\theta)$];

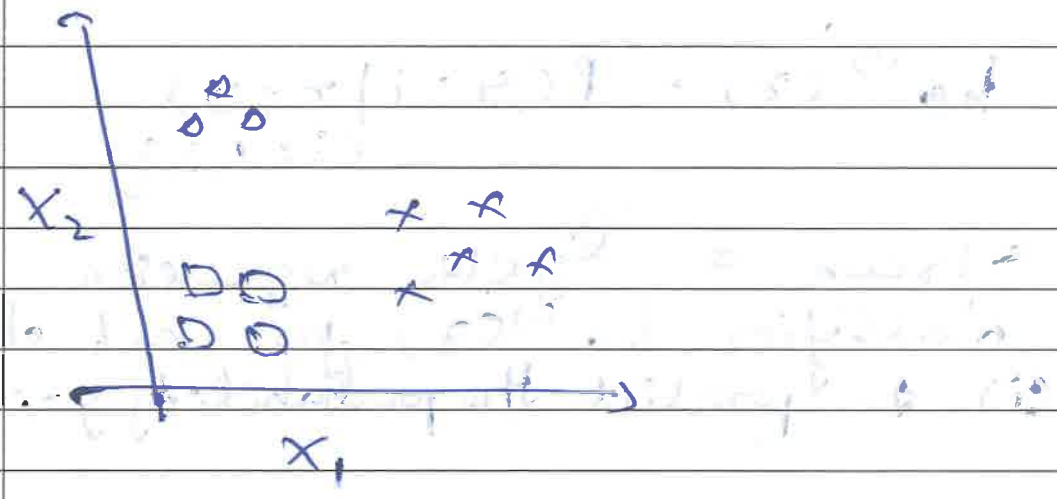
gradient(2) = [code to compute $\frac{\partial}{\partial \theta_2} J(\theta)$];

gradient(n) = [code to compute $\frac{\partial}{\partial \theta_n} J(\theta)$];

Multiclass - classification.

Eg: E-mail folderings / tagging: work, friends, family, hobby.

Eg: medical diagram:- cold, flu, not ill.

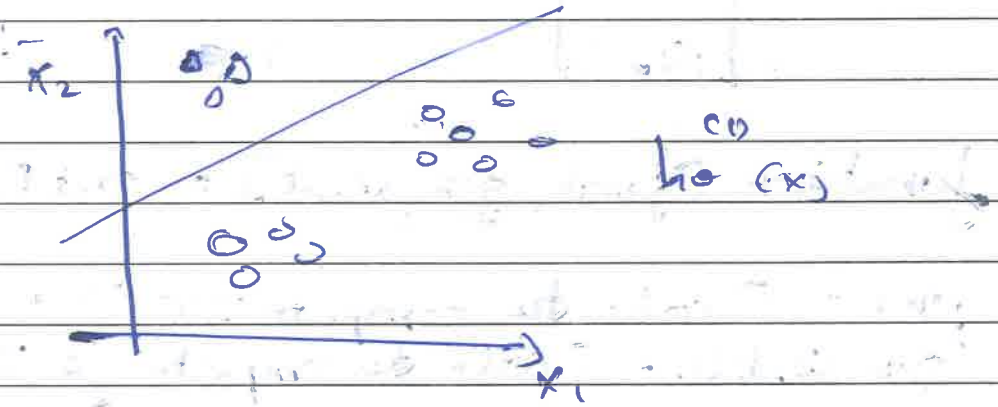


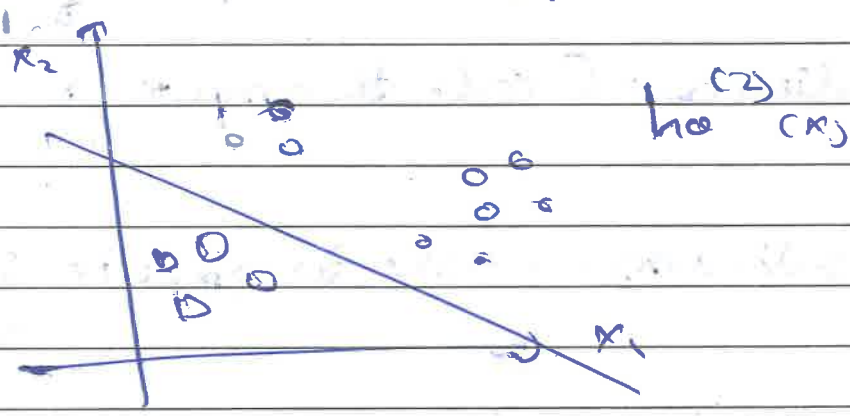
class 1 -> △△△
class 2 -> □□□
class 3 -> ×××

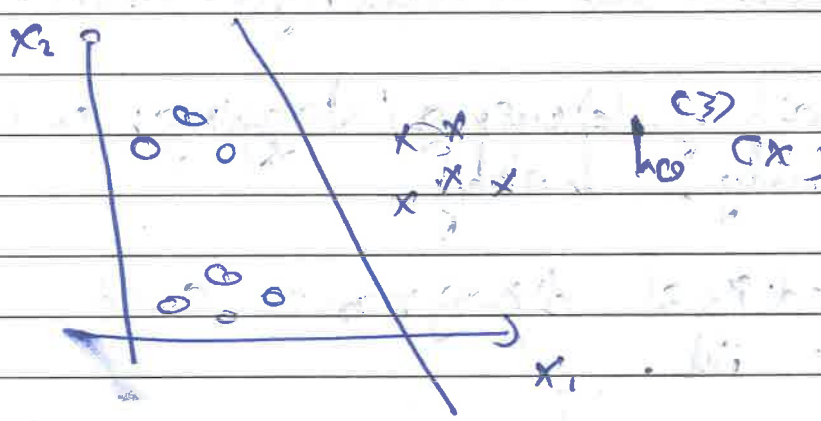
papergrid
Date: / /

we will solve this by making 3 binary classification.

then :-







$h_\theta^{(i)}(x) = P(y = i \mid x ; \theta)$
$(i = 1, 2, 3)$

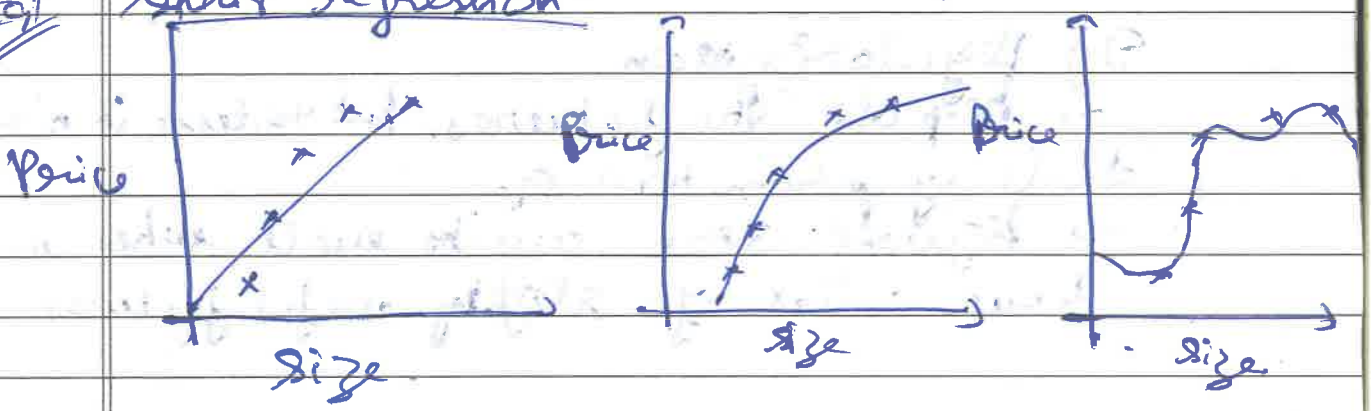
Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class (i) to predict the probability $y = i$

Regularization:

The problem of overfitting.

Sometimes logistic & linear regression work poorly because of overfitting.

Eg: linear regression



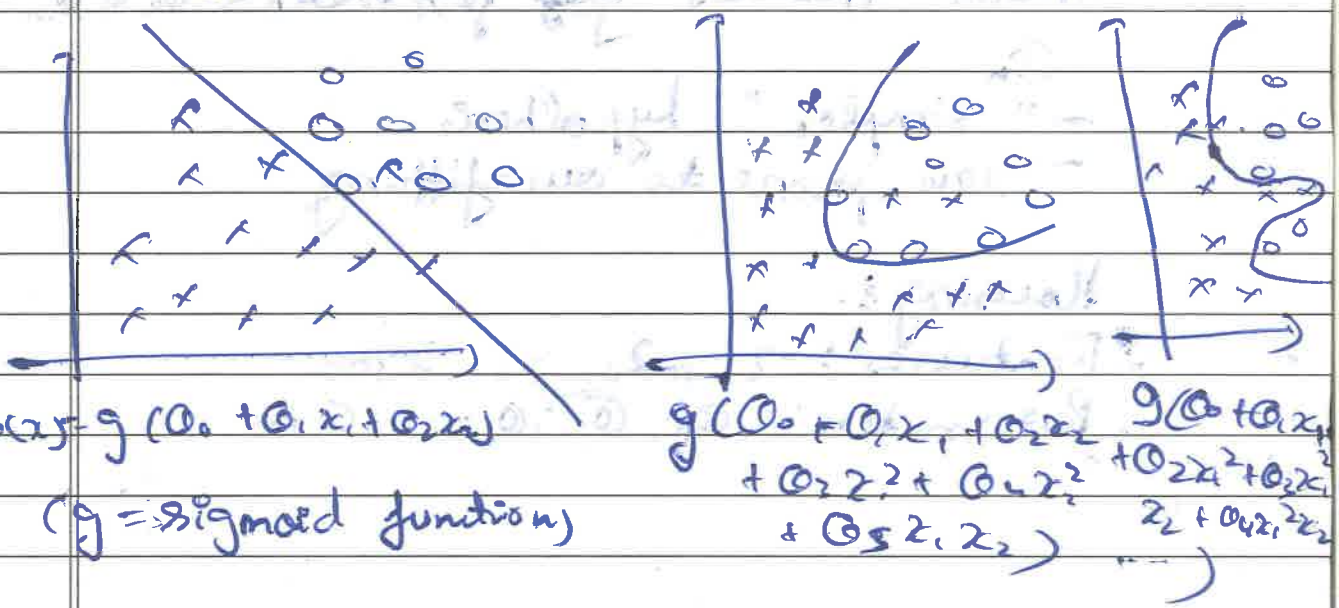
$\rightarrow \theta_0 + \theta_1 x$ "Underfit" "High bias"

$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2$ "Just right"

$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$ "over-fitting"

Overfitting:- "If we have too many features, the learned hypothesis may fit the training set very well but fail to generalize to new examples.

Eg: logistic regression



$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = Sigmoid function)

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3 + \theta_4 x_1^2 x_2 + \theta_5 x_1 x_2^2 + \ldots)$

Addressing overfitting

1.) Reduce number of features.
- -> Manually select features to keep
- -> Model selection algorithm

2.) Regularization.
- -> keep all the features, but reduce the magni. / size of parameters θ_j.
- -> Regularization works well when we have a lot of slightly useful features.

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Suppose we penalize and make θ_3, θ_4 very small.

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\theta_3^2 + 1000\theta_4^2$$

$\hookrightarrow \theta_3 \approx 0$ & $\theta_4 \approx 0$

Small values for parameters $\theta_0, \theta_1, \theta_2, \theta_3, \theta_4$
- "Simpler" hypothesis
- less prone to overfitting.

Housing:-
- Features: $x_1, x_2, \dots x_{100}$
- Parameters:- $\theta_0, \theta_1, \theta_2 \dots \theta_{100}$

Note:- we do not penalize θ_0 (the constant term). we could penalize though. that won't make much difference. but we don't.

Date: / /

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

λ → regularization parameter.

This extra term would decrease the values of θ.

Regularized linear regression.

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

Gradient descent:

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\cdot(x_j^{(i)}) + \lambda\frac{\theta_j}{m}\right]$$

$$\Rightarrow \theta_j := \theta_j\left(1 - \frac{\alpha\lambda}{m}\right) - \alpha\cdot\frac{1}{m}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

TP-T04

Usually, $1 - \alpha\left(\frac{\lambda}{m}\right) < 1$ then

$$\theta_j\left(1 - \frac{\alpha\lambda}{m}\right) < \theta_j$$

As we know:-

$$\theta_j := \theta_j\left(1 - \frac{\alpha \cdot \lambda}{m}\right) - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

this term would become a little less than θ_j

And this term is the term earlier

Normal equation:-

$$X = \begin{bmatrix} \leftarrow (x^{(1)}) \rightarrow \\ \leftarrow (x^{(2)}) \rightarrow \\ \leftarrow (x^{(3)}) \rightarrow \\ \vdots \\ \leftarrow (x^{(m)}) \rightarrow \end{bmatrix} \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$$\theta = (X^T X + \lambda \cdot L)^{-1} X^T y$$

where $L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}_{(n+1)\times(n+1)}$ excluding θ_0

where n is the number of parameters.

Regularized logistic regression.

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^3 x_2 + \theta_5 x_1^2 x_2^3 + \dots)$$

cost function:-

$$J(\theta) = -\left[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

gradient descent:-

Repeat {

$$\theta_0 := \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

$$+ \frac{\alpha\lambda}{m}\,\theta_j$$

Advanced optimization:-

function [jVal, gradient] = costFunction(theta)

jVal = [code to compute $J(\theta)$];

gradient(1) = [code to compute $\frac{\partial}{\partial\theta_0} J(\theta)$];

gradient(2) = [code to compute $\frac{\partial}{\partial\theta_0} J(\theta)$];

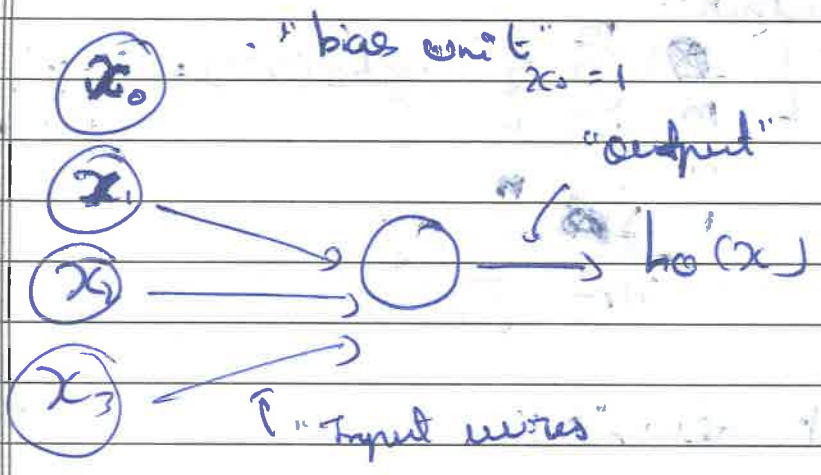
⋮

gradient(n+1) = [code to compute $\frac{\partial}{\partial\theta_n} J(\theta)$];

Neural networks.

→ Origins :- Algorithms that try to mimic the brain.

→ Stole of the art technique for many applications.



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

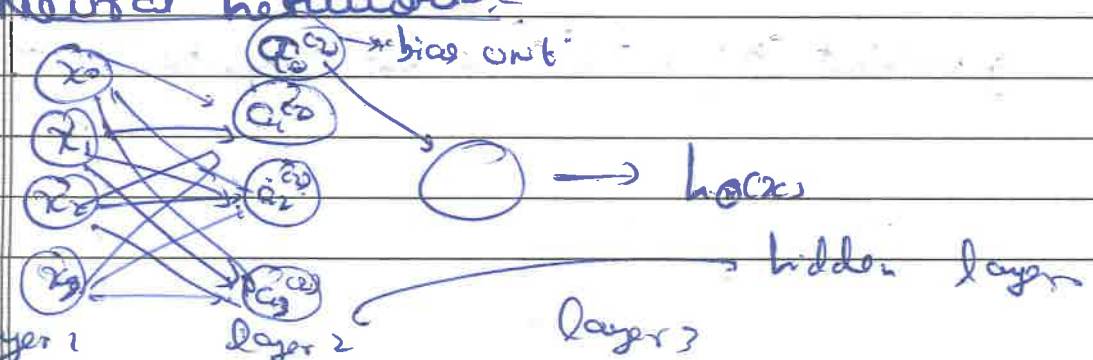
Sigmoid (logistic) activation function

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

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

↳ these are called as weights = parameters.

Neural network:-



$a_i^{(j)}$ = "activation" of unit i in layer j

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

$\Theta^{(1)}$

$z_1^{(2)}$

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

$z_2^{(2)}$

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

$z_3^{(2)}$

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

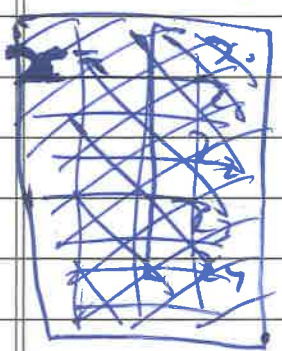
$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimensions $s_{j+1} \times (s_j + 1)$

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

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

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



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$z^{(2)} = \Theta^{(1)} x$ ⟹ is equal to $a^{(1)}$

$a^{(2)} = g(z^{(2)})$

now add $a_0^{(2)} = 1$

$z^{(3)} = \Theta^{(2)} a^{(2)}$

$h_\Theta(x) = a^{(3)} = g(z^{(3)})$

this process is called forward propagation - vectorized implementation

~~so at the end we get:-~~

~~$h_\Theta(x) = g(\Theta_{1}^{(2)} a_0^{(2)} + \Theta_1^{(1)}$~~

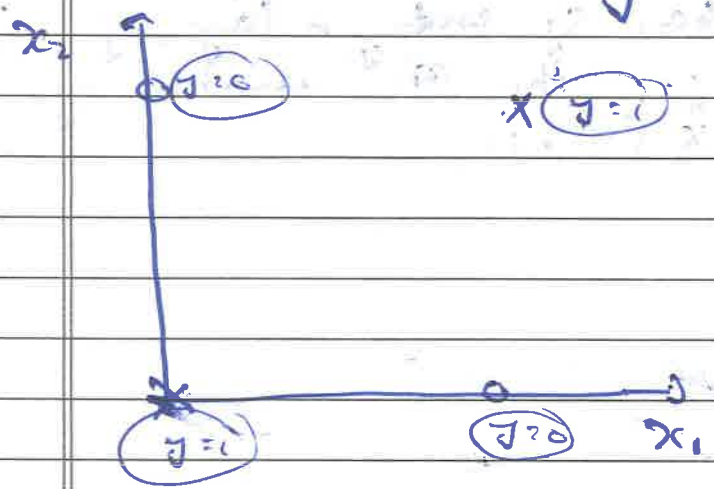
so at the end we get :-

$h_\Theta(x) = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(1)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$

Non-linear classification example :-

XOR

→ x_1, x_2 are binary (0 or 1)

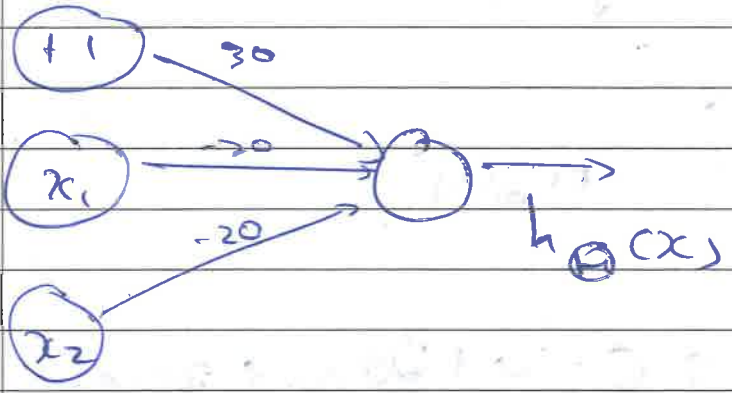


Simple example: AND

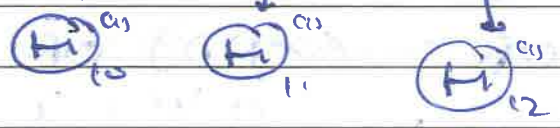
$x_1, x_2 \in \{0, 1\}$

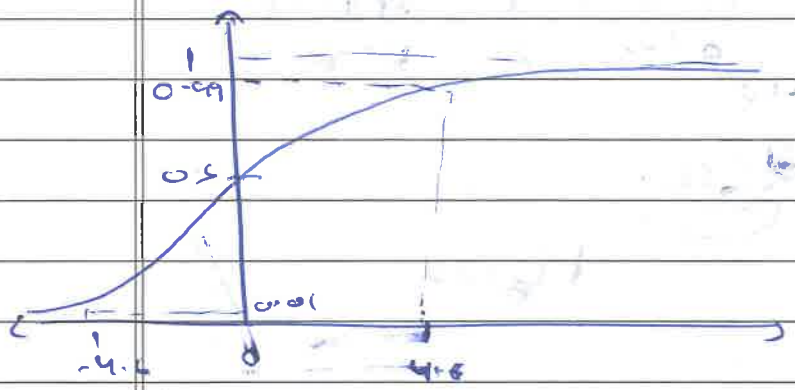
$y = x_1$ AND x_2

→ Simple example :- AND



$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

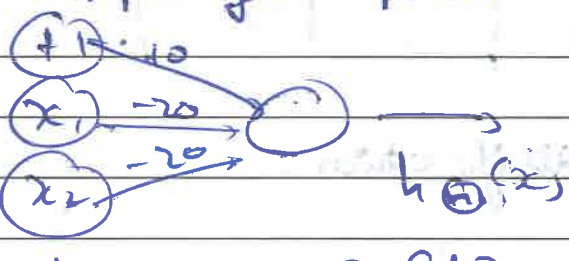




Truth table :-

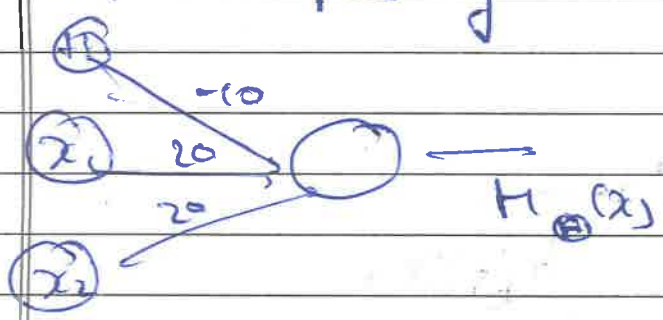
x_1	x_2	$x_1 \wedge x_2 = h_\Theta(x)$
0	0	0
0	1	0
1	0	0
1	1	1

→ Example for NOR



$h_\Theta(x) = g(10 - 20x_1 - 20x_2)$

→ Example of OR



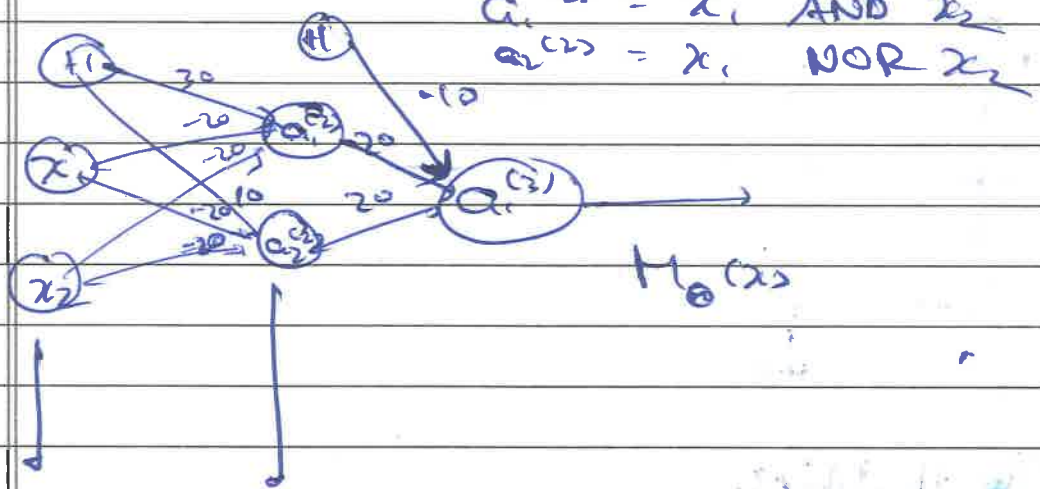
$H_\Theta(x) = g(-10 + 20x_1 + 20x_2)$

→ Now putting it all together for XNOR

~~$a_1^{(2)} = x_1$ AND x_2~~

$a_1^{(2)} = x_1$ AND x_2

$a_2^{(2)} = x_1$ NOR x_2



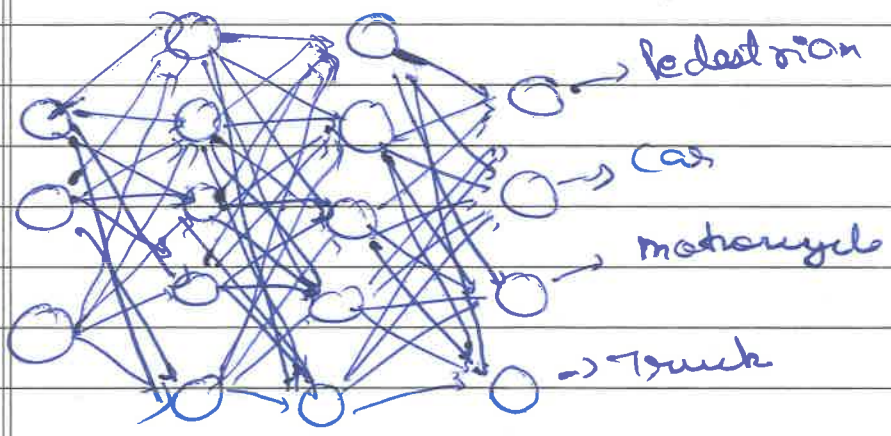
Input layer

Hidden layer

x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

→ Multiclass classification:
one-vs-all.

(Pedestrian) - (Car) - (Motorcycle) - (Truck)



want $h_\Theta(x) = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$

(Pedestrian) (Car) (motorcycle)

etc.

Neural network (Classification)

$\{(x^1, y^1), (x^2, y^2), (x^3, y^3), (x^4, y^4) \cdots (x^m, y^m)\}$

→ training set.

L = Total no. of layers in network

s_l = no. of units (not counting bias unit) in layer l.

If there are k output units
then $s_L = k$

Cost function for logistic regression

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)})\log(h_\Theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\Theta(x^{(i)}))$$

$$+ \frac{\lambda}{2m}\sum_{j=1}^{n}\Theta_j^2$$

- For neural network as we have multiple output nodes:-

$$J(\Theta) = \frac{-1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \Big[(y_k^{(i)}) \log (h_\Theta(x^{(i)})_k) + (1 - y_k^{(i)}) (1 - \log((h_\Theta(x^{(i)}))_k) \Big]$$

$$+ \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{j,i}^{(l)})^2$$

↳ Here i does not refer to the training example

As we need Θ for $\min J(\Theta)$ we need to compute

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} (J(\Theta))$$

"Backpropagation" is a neural network terminology for minimizing our cost function, just like we were doing with gradient descent in logistic and linear regression".

~~$\delta_j^{(4)}$ is the error for $a_j^{(4)}$ (unit j of layer 4) and is calculated as:~~

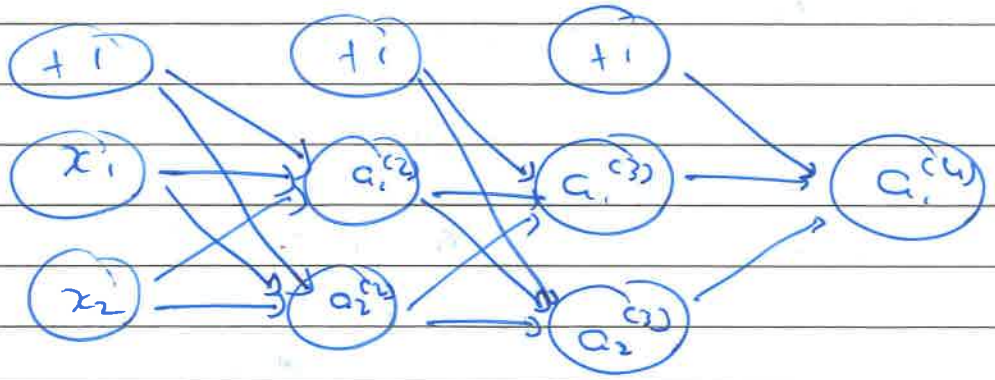
~~$\delta_j^{(4)} = a_j^{(4)} - y_j^{(4)}$~~

~~↳ Predicted value~~

{p.7.0}

Intuitively, $\delta_j^{(l)}$ is the "error" for $a_j^{(l)}$ (unit j in layer l).

Eg:-



$$\delta_1^{(4)} = -y^{(i)} + a_1^{(4)}$$

So first we would do forward propagation:-

$a^{(1)} = x$

$z^{(2)} = \Theta^{(1)} a^{(1)}$

$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

$z^{(3)} = \Theta^{(2)} a^{(2)}$

$a^{(3)} = g(z^{(3)})$ (add $a_0^{(3)}$)

$z^{(4)} = \Theta^{(3)} a^{(3)}$

$a^4 = h_\Theta(x) = g(z^{(4)})$

now we compute $\delta^{(4)} = a^{(4)} - y$

Now we would do back propagation:-

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$$

The delta values of layer l are calculated by multiplying the delta values in the next layer with the theta matrix of layer l.

We then element wise multiply that with a function called g' or g-prime, which is the derivative of the activation function g evaluated with the input values given by $z^{(l)}$.

The g-prime derivative terms can also be written out as:-
$g'(z^{(l)}) = a^{(l)} .* (1 - a^{(l)})$

Then we calculate $\Delta_{ij}^{(l)}$

First set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j)
$\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
Hence we update our new Δ matrix.

$$D_{i,j}^{(l)} := \frac{1}{m}\left(\Delta_{i,j}^{(l)} + \lambda \cdot \Theta_{i,j}^{(l)}\right), \text{ if } j \neq 0$$

$$D_{i,j}^{(l)} := \frac{1}{m}\Delta_{i,j}^{(l)} \text{ if } j = 0$$

The capital-delta matrix D is used as an "accumulator" to add up our values as we go along and eventually compute our partial derivative.
Then we get $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$

"Unroll" into vectors for use in octave

{p-7.0}

function [jval, gradient]
= costFunction(theta)

optTheta = fminunc(@costFunction,
initialTheta, options)

Neural Network (L=4)

→ $\Theta^{(1)}$, $\Theta^{(2)}$, $\Theta^{(3)}$ - matrices (Theta1, Theta2, Theta3)

→ $D^{(1)}$, $D^{(2)}$, $D^{(3)}$ - matrices (D1, D2, D3)

gradient

Example.

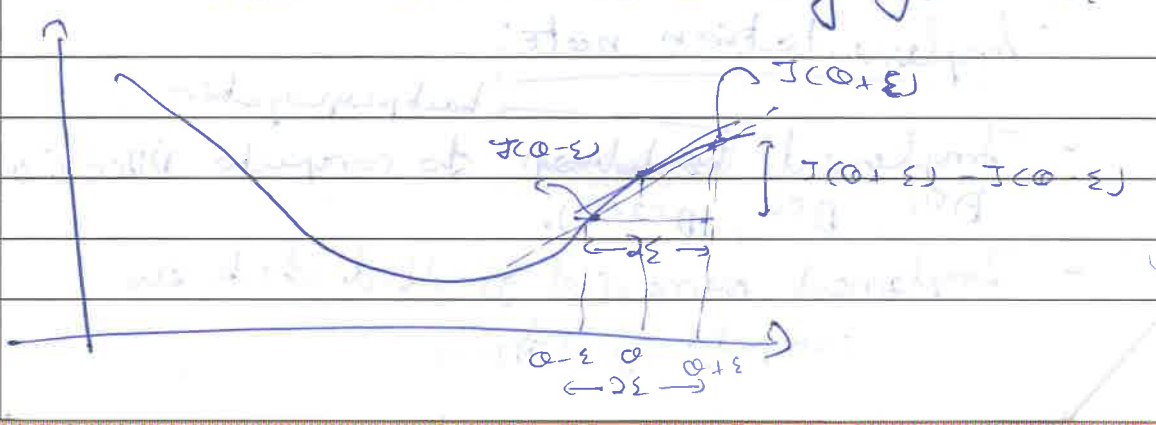
$s_1 = 10$, $s_2 = 10$, $s_3 = 1$

→ $\Theta^{(1)} \in \mathbb{R}^{10 \times 11}$ $\Theta^{(2)} \in \mathbb{R}^{10 \times 11}$, $\Theta^{(3)} \in \mathbb{R}^{1 \times 11}$

→ $D^{(1)} \in \mathbb{R}^{10 \times 11}$, $D^{(2)} \in \mathbb{R}^{10 \times 11}$, $D^{(3)} \in \mathbb{R}^{1 \times 11}$

```
thetaVec = [Theta1(:); Theta2(:); Theta3(:)];
DVec = [D1(:); D2(:); D3(:)];
Theta1 = reshape(thetaVec(1:110), 10, 11);
Theta2 = reshape(thetaVec(111:220), 10, 11);
Theta3 = reshape(thetaVec(221:231), 1, 11);
```

#1 Numerical estimation of gradient



$$\frac{d}{d\theta}(J(\theta)) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$$

Parameter vector θ

$\theta \in \mathbb{R}^n$ (E.g. θ is "unrolled" version of $\theta^{[1]}, \theta^{[2]}, \theta^{[3]}$)

$\theta = [\theta_1, \theta_2, \theta_3 \cdots \theta_n]$

$$\frac{\partial}{\partial \theta_1}(J(\theta)) \approx \frac{J(\theta_1+\varepsilon, \theta_2, \theta_3, \cdots \theta_n) - J(\theta_1-\varepsilon, \theta_2 \cdots \theta_n)}{2\varepsilon}$$

$$\frac{\partial}{\partial \theta_2}(J(\theta)) \approx \frac{J(\theta_1, \theta_2+\varepsilon, \theta_3 \cdots \theta_n) - J(\theta_1, \theta_2-\varepsilon, \theta_3 \cdots \theta_n)}{2\varepsilon}$$

$$\vdots$$

$$\frac{\partial}{\partial \theta_n}(J(\theta)) \approx \frac{J(\theta_1, \theta_2 \cdots \theta_n+\varepsilon) - J(\theta_1, \theta_2 \cdots \theta_n-\varepsilon)}{2\varepsilon}$$

```
for i = 1 : n
    thetaplus = theta;
    thetaplus(i) = thetaplus(i) + EPSILON;
    thetaminus = theta;
    thetaminus = thetaminus(i) - EPSILON;
    gradApprox(i) = (J(thetaplus) - J(thetaminus)) ;
                    (2 * EPSILON)
end i
```

check that gradApprox ≈ DVec

Implementation note:-

- Implement ~~backprop~~ backpropagation to compute DVec (unrolled $D^{[1]}, D^{[2]}, D^{[3]}$).
- Implement numerical gradient check to compute gradApprox.

- make sure they give similar values
- Turn off gradient checking. Using backdrop code for learning.

↳ make sure to disable gradient-checking code before training your classifier because it is very slow

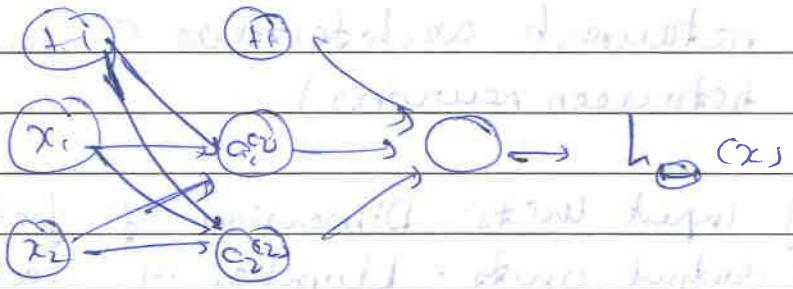
#1 Random initialization.

Initial value of Θ

For gradient descent and advanced optimization method, need initial value of Θ.

Zero initiation would not work unlike logistic regression.

because:-



after each update we will get $a_1^{(2)} = a_2^{(2)}$, also $\delta_1^{(2)} = \delta_2^{(2)}$

After each update, parameters corresponding to inputs going into each of two hidden units are identical

{P-7.0}

To solve this we do:-
Random initialization: (symmetry breaking)

⇒ Initialize each $\Theta_{ij}^{(l)}$ to random value in $(-\alpha, \alpha)$
i.e $-\alpha \leq \Theta_{ij} \leq \alpha$

Eg. ~~random (10×11) matrix between (0 & 1)~~

Theta1 = rand(10,11) * (2 * INIT_ALPHA) - INIT_ALPHA

↳ $[-\alpha, \alpha]$

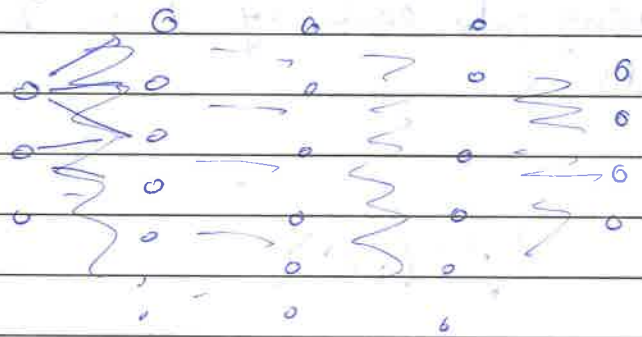
Theta2 = rand(1,11) * (2 * INIT_ALPHA) - INIT_ALPHA

Putting it all together

(i) Pick a network architecture (connectivity pattern between neurons)

No. of input units: Dimension of features $x^{(i)}$
No. of output units: Number of classes.
Reasonable default :-
1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

Eg.)



(ii) Randomly initialize weights.

(iii) Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$.

(iv) Implement code to compute cost function $J(\Theta)$.

(v) Implement backpropagation to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

→ for $i = 1 : m$

{

perform forward propagation & backpropagation using example $(x^{(i)}, y^{(i)})$

(get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$).

$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

}

compute $\frac{\partial}{\partial (\Theta_{jk}^{(l)})} J(\Theta)$

(vi) Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} (J(\Theta))$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$.

→ Then disable gradient checking code.

(vii) Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ.

& by the way $J(\Theta)$ → non convex

(Then we can end up in a local minimum instead).

(But mostly here that is not a problem.)

#1 Debugging a learning algorithm.

Suppose you have implemented regularized linear regression to predict housing prices.

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

Suppose after we test hypothesis, we come across large errors in its prediction. What could we do next :-

(i) Try getting more training examples.
(ii) Try smaller sets of features.
(iii) Try getting additional features.
(iv) Try adding polynomial features. (x_1^2, x_2^2, x_1x_2, etc.)
(v) Try decreasing λ.
(vi) Try increasing λ.

Machine learning diagnostic :-

A test that you can run to gain insight what is/ isn't working with a learning algorithm, and gain guidance as to how best to improve its performance.

Diagnostics can take time to implement, but doing so can be a very good use of your time.

{P.T.O}

Evaluating your hypothesis.

→ Split data into train set and test set.
→ learn parameter θ from training data.
→ For classification - misclassification error :-
(aka 0/1 misclassification error :-)

$$err(h_\theta(x), y) = \begin{cases} 1 & \left(\text{if } h_\theta(x) \geq 0.5, y = 0 \text{ or } \text{if } h_\theta(x) < 0.5 \text{ & } y = 1\right) \\ 0 & \text{otherwise} \end{cases}$$

The average test error :-

$$\text{Test Error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err\left(h_\theta(x_{test}^{(i)}), y\right)$$

→ model selection :-

1. $h_\theta(x) = \theta_0 + \theta_1 x \rightarrow \theta^{(1)} \rightarrow J_{test}(\theta^{(1)})$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow \theta^{(2)} \rightarrow J_{test}(\theta^{(2)})$
3. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \rightarrow \theta^{(3)} \rightarrow J_{test}$

10. $h_\theta(x) = \theta_0 + \theta_1 x \cdots \theta_{10} x^{10} \rightarrow \theta^{(10)} \rightarrow \theta^{(10)} \rightarrow J_{test}$

Suppose we get the best result with degree 5 : $\theta_0 + \theta_1 x \cdots \theta_5 x^5$

alternatively :-
→ we could split dataset as :-
train set, cross-validation, test set
(60 %) (20 %) (20 %)

{p. 7.0}

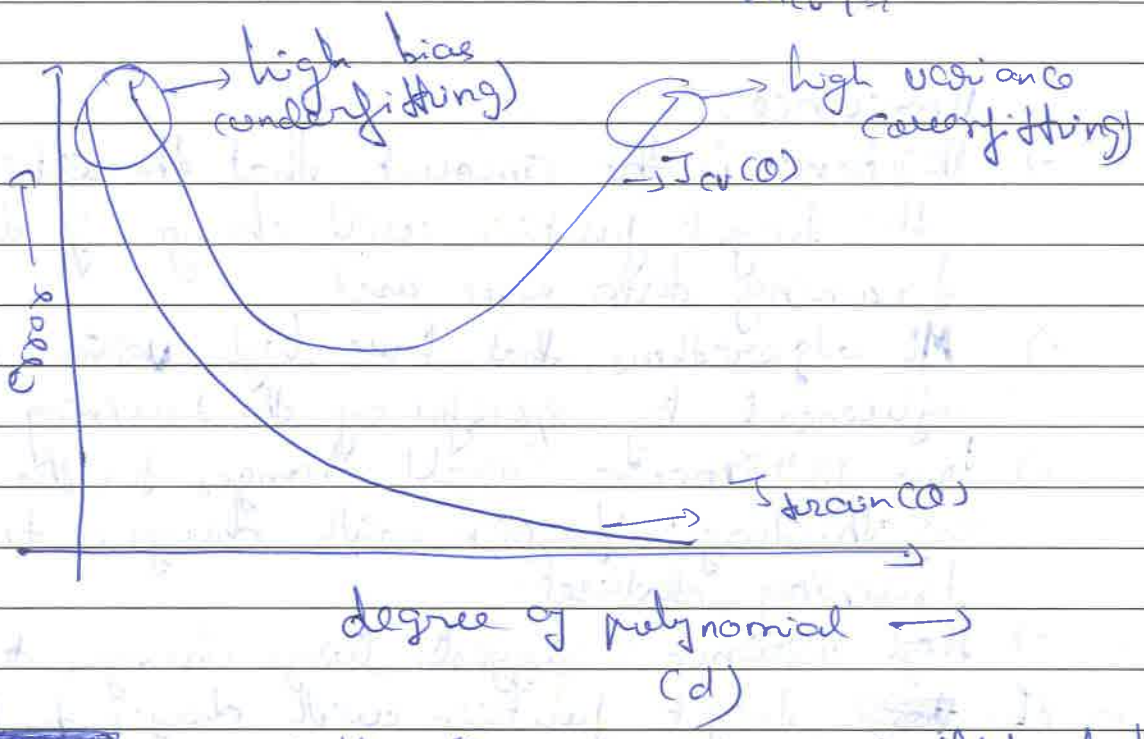
Train / Validation / Test

1. Optimize the parameters in Θ using the training set for each polynomial degree.
2. Find the polynomial degree d with the least error using the cross validation set.
3. Estimate the generalization error using the test set with $J_{test}(\Theta^{(d)})$, (d = theta from polynomial with lower error)

Bias / Variance.

Training error:- $J_{train}(\Theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})^2$

Cross validation error:- $J_{cv}(\Theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\Theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$



underfitting :- $J_{train}(\Theta)$ will be high
$J_{cv}(\Theta) \approx J_{train}(\Theta)$

overfitting: $J_{train}(\Theta)$ will be low
$J_{cv}(\Theta) \gg J_{train}(\Theta)$

Date: / /

Bias & Variance :-

Bias :-

- Low Bias :- Suggests less assumptions about the form of the target function.
- High Bias :- Suggests more assumptions about the form of the target function.

- Higher the Bias less flexibility.
- Bias are the simplifying assumptions made by a model to make the target function easier to learn.
- Eg of low bias :- Decision Tree, K-nearest neighbours and support vector machines.
- Eg of high bias :- linear regression, logistic regression.

Variance :-

- Variance is the amount that the estimate of the target function will change if different training data was used.
- ML algorithms that have high variance are strongly influenced by specifics of the training data.
- low variance :- Small changes to the estimates of the target function with changes to the training dataset.
- High variance :- Suggests large changes to the estimate of the target function with changes to the training dataset.
- Eg. of low variance :- linear regression, logistic regression.
- Eg of high variance :- Decision trees, k-neighbors classifier, SVM's.

-> Ideally we want low bias & low variance.

-> Bias would increase then variance would decrease & vice versa.

-> linear ML algorithms :- High bias & low variance.

-> non-linear ML algorithms :- High variance & low bias.

Linear regression with regularization.

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \cdots \theta_4 x^4$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

when λ -> large = 1000 -> underfitting
$\theta_j = 0$

when λ -> small = 0.01 -> overfitting.

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

{p.7.0}

Choosing the regularization parameter λ.

model :- $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^{d} \theta_j^2$$

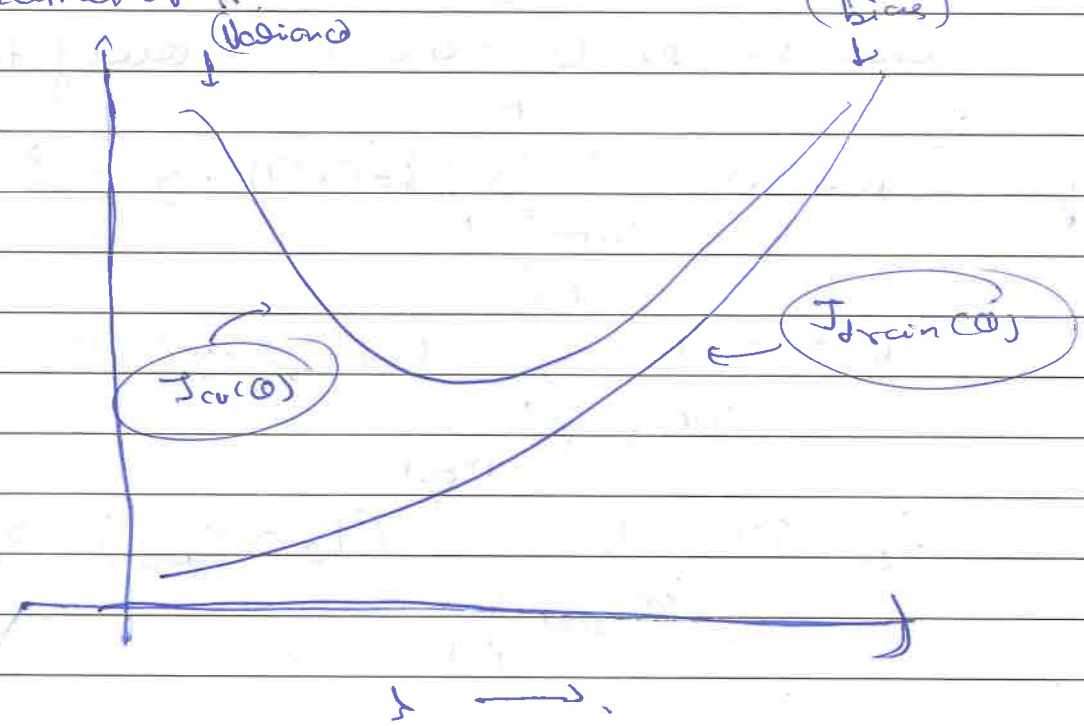
1. Try $\lambda = 0 \rightarrow \min_\theta J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$
2. Try $\lambda = 0.01 \rightarrow \min_\theta J(\theta) \rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$
3. Try $\lambda = 0.02$
4. Try $\lambda = 0.04$

:

12. Try $\lambda = 10 \rightarrow \min_\theta J(\theta) \rightarrow \theta^{(12)} \rightarrow J_{cv}(\theta^{(12)})$

Let say we pick $\theta^{(5)} \rightarrow$ Test error $\rightarrow J_{test}(\theta^{(5)})$

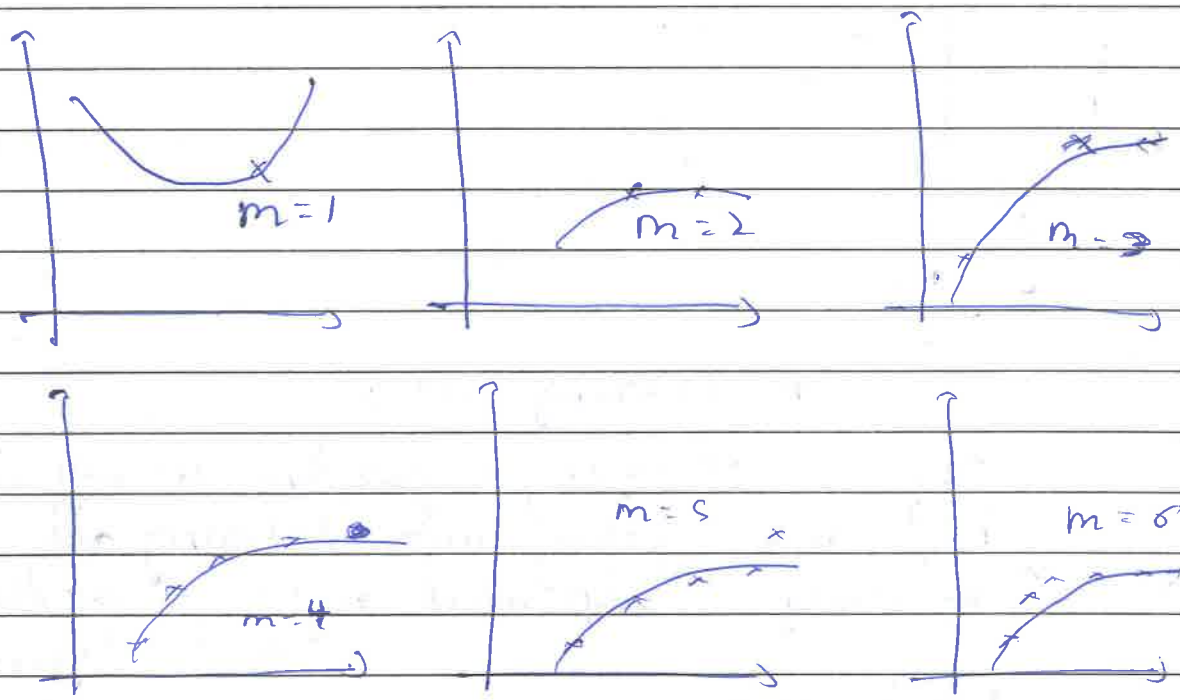
Bias / Variance as a function of the regularization parameter λ.



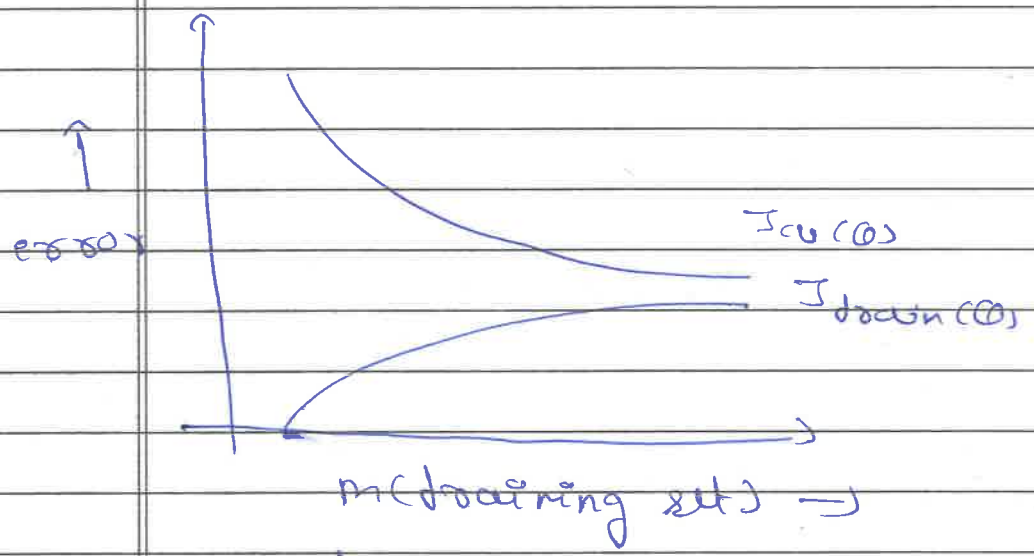
{p.T.O}

Learning curves

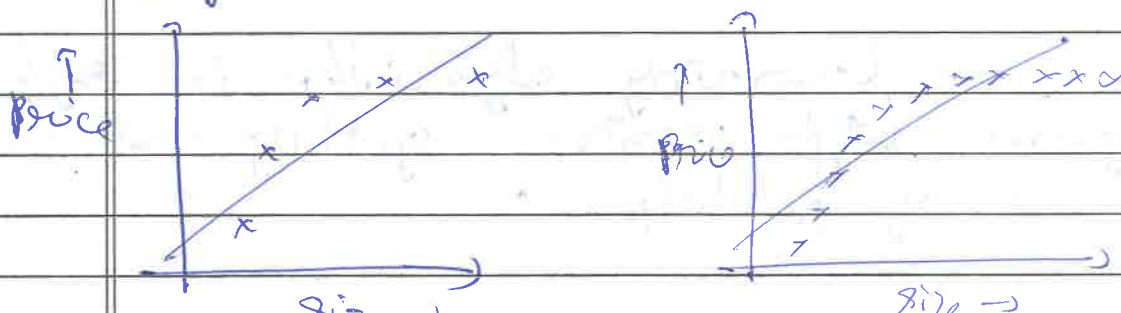
$h(\theta) = \theta_0 + \theta_1 x + \theta_2 x^2$



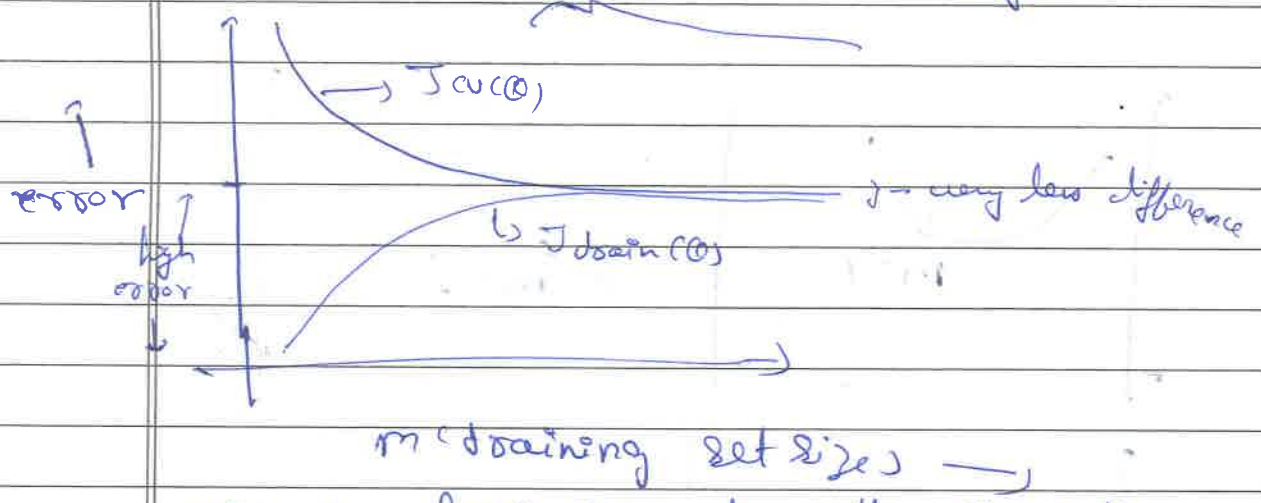
So we can see that as m increases the error would increase for J_{train} (error)



High bias :-

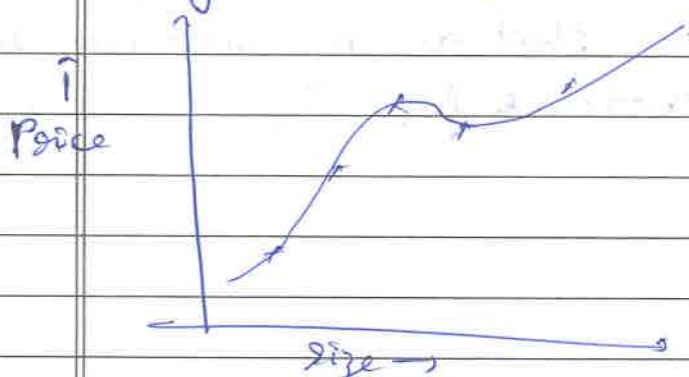


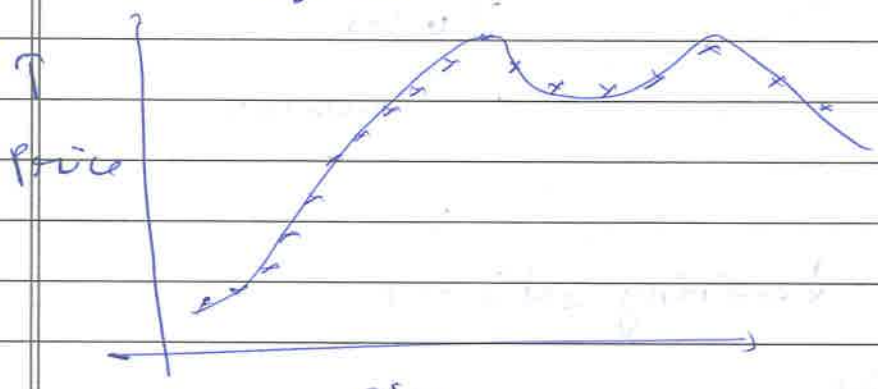
Date: / /



If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much as we can see that the curve is flattening out.

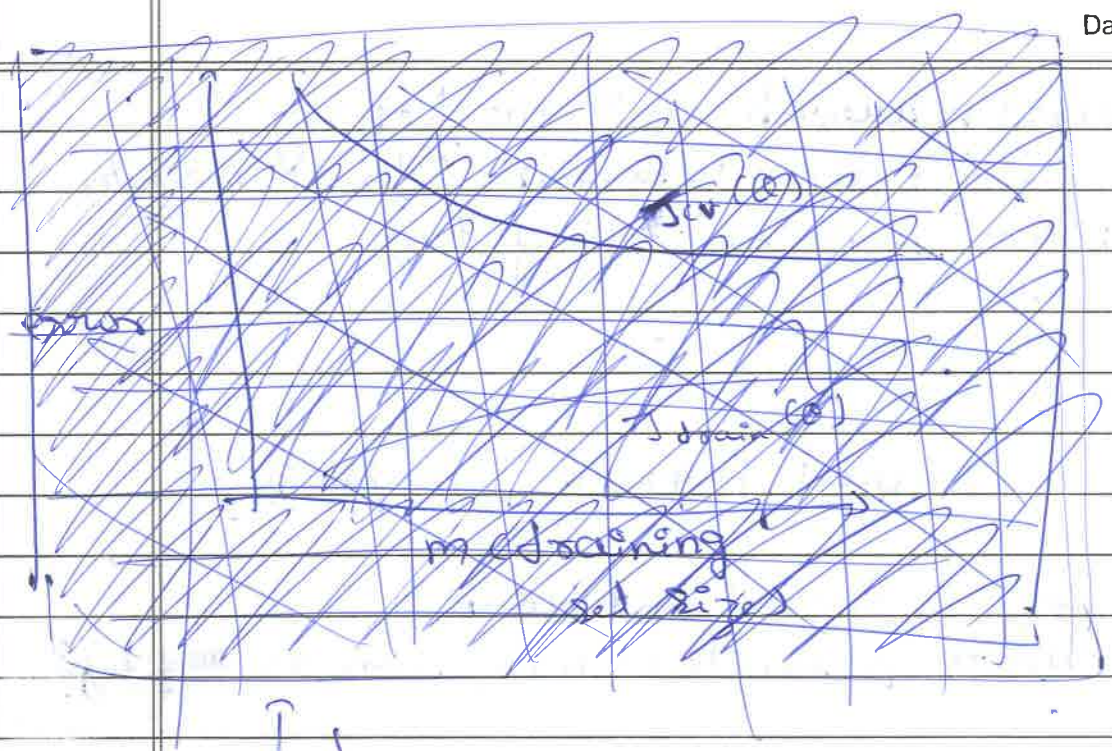
High variance

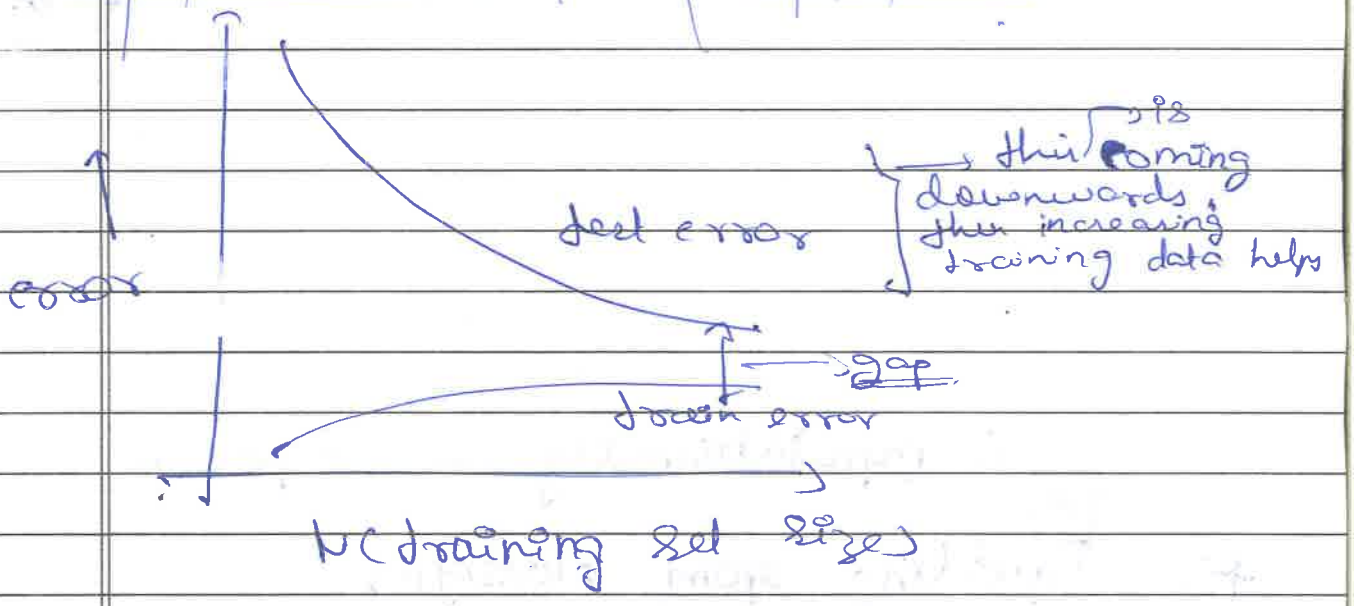




If a learning algorithm is suffering from high variance, getting more data is likely to help.

Date: / /





#11 Debugging a learning algorithm.

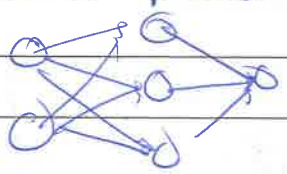
Get more training examples → fixes high variance
Try smaller set of features → fixes high variance.
Try getting additional features → fixes high bias
Try adding polynomial features (x_1^2, x_2^2, x_1x_2, etc) → fixes high bias
Try decreasing λ → fixes high bias
Try increasing λ → fixes high variance.

{R T-0}

Date: / /

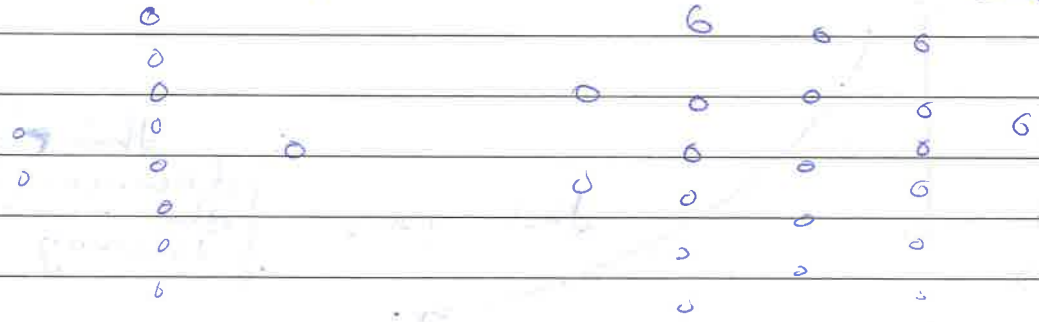
Neural networks and overfitting:-

→ "small" neural network (fewer parameters, more prone to underfitting)



(computationally cheaper)

→ "large" neural network
(more parameters: more prone to overfitting)



(computationally more expensive)

Building spam classifier

Supervised learning.
x = features of email
y = spam (1) or not spam (0).
Features x: choose 100 words indicative of spam / not spam.

Eg. deal, buy, discount
indicate spam

always, now
not spam.

Eg. of spam e-mail:-

From: — —

To:- — ———

Subject:- Buy now!

Deal of the week! buy now! — —

— —

$x = \begin{bmatrix} \\ \vdots \\ \\ \vdots \\ \\ \vdots \\ \\ \end{bmatrix}$ andrew buy deal [redacted] now [redacted] discount

Note:- In practice, take most frequently occuring n words (10,000 to 50,000) in training set, rather than manually pick 100 words.

How to spend your time to make it have low error

→ collect lots of data

→ Develop [redacted] sophisticated features based on email routing information (from email header)

→ Develop sophisticated algorithm to detect misspellings (e.g. m0rtgages, w4tches.. discounts)

→ Develop sophisticated features for [redacted] message body, eg should "discount" and "discounts" be treated as the same word?

Recommended approach:-
→ Start with a simple algorithm that you can implement quickly.
Implement it and test it on your cross-validation data.
→ Plot learning curves to decide if more data, more features, etc are likely to help
→ Error analysis:- Manually examine the examples (in cross validation set) that your algorithm made errors on. See if you spot any systematic trend in what type of examples it is making error on.

↳ Error Analysis:-

m_{cv} = 500 examples in cross validation set.
Algorithm misclassifies 100 emails.
Manually examine 100 errors and categorize them based on:-
(i) What type of email it is
↳ Pharma, replica, steal-password

(ii) What cues (features) you think would have helped the algorithm classify them correctly

(type) (no. of emails of that type misclassified)
Pharma : 12
Replica : 4
Steal password : 53
others :- 31

Deliberate misspellings :- 5
(morgage, medicine, etc)
Unusual email routing :- 16
unusual (spamming) punctuation :- 32

The recommended approach to perform error analysis using the cross validation data used to compute $J_{cv}(\theta)$ rather than the test data which is used to compute $J_{test}(\theta)$ is because if we develop new features by examining the test set, then we may end up choosing features that work well specifically for the test set, so $J_{test}(\theta)$ is no longer a good estimate of how well we generalize to new examples.

Cancer classification example.
Train logistic regression model $h_\theta(x)$. ($y=1$ if cancer $y=0$ otherwise)

Now suppose we get 1% error on test set.
But if only 0.5% patients then it seems that we have diagnosed double the number of people as sick as the original number.

And suppose we do some changes so that we get 99.2% accuracy, this also doesn't mean necessarily that we have done better. it might just mean that now our algorithm just predicts ($y=0$) more often.

to 1

— they use bring precision & recall into picture

papergrid

Date: / /

#1 Precision / Recall.

(Predicted) class	(1) Actual (0) class: True	False
(1)	True +Ve	False +Ve
(0)	False -Ve	True -Ve

Precision :- $\frac{\text{True positives}}{\text{Predicted +ves.}}$

$= \frac{\text{True positive}}{\text{True positives + False positives}}$

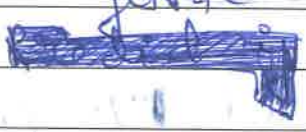
Recall :- $\frac{\text{True positives}}{\text{actual positives}}$

$= \frac{\text{True positive}}{\text{True positives + False negatives}}$

F, R
0.4
0.1
0.0

#2 Trading off precision and recall.

Logistic regression :- $0 \leq h_\theta(x) \leq 1$



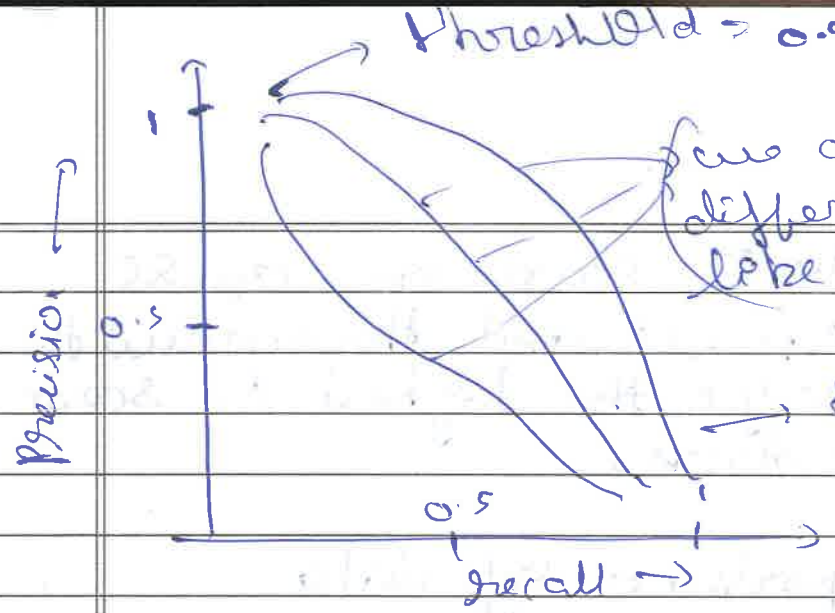
Now if we want that we predict $h_\theta(x) = 0$ only if we are absolutely sure because if they have cancer and we tell them that they dont then they might die so we may do :-

predict 1 if $h_\theta(x) \geq 0.7$

predict 0 if $h_\theta(x) < 0.7$

This would lead to :-

Higher recall, lower precision.



F1 Score (F Score)

How do compare precision / recall numbers

F1 Score		Precision (P)	Recall (R)
0.444	Algorithm 1	0.5	0.4
0.175	Algorithm 2	0.7	0.1
0.0392	Algorithm 3	0.02	1.0

predict y = 1 all the time

If we get considerably low value of recall but a good value of precision that wouldnt be very nice.

OR (if we would get considerably low value of precision but a good value of recall that also wouldnt be very nice)

Thus we come up with F1 score where

$$F_1 \text{ score} = \frac{2 \cdot (P) \cdot (R)}{(P + R)}$$

Evaluate F_1 score on cross validation set and the one with which we get the highest F_1 score should be chosen.

The importance of data

If we use a learning algorithm with many parameters (e.g. logistic regression/ linear regression with many features; neural network with many hidden units).

Then we would get low bias algorithm.

→ $J_{train}(\theta)$ will be small.

But then if we choose a very large training set, then it would be unlikely to overfit.

Because then we might get

$\Rightarrow J_{train}(\theta) \approx J_{test}(\theta)$

Thus $J_{test}(\theta)$ would also become small.

For almost all algorithms the accuracy increases with increase in the training set size.

But if we don't have enough features then even the dataset might not be able to help us.

Eg. If we have the size the house → just to predict its price.

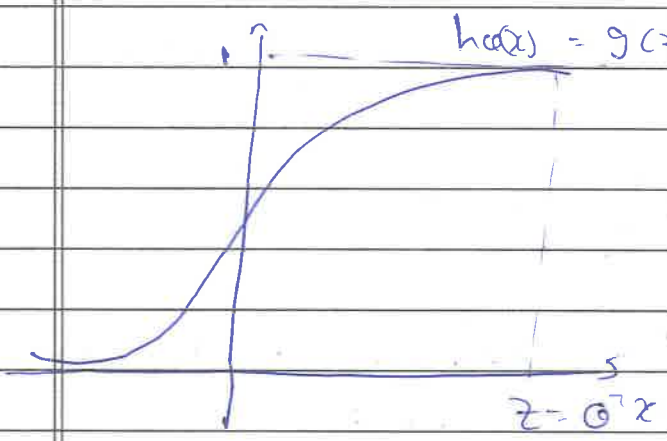
Then how much every examples we may help. It would not help much.

We could also think that if a human expert were be presented with the parameters then would he be able to predict the output and if not probably we need more features. Eg. like a human expert cannot predict the price with land size so we need more parameters

Support vector machines

→ Alternate view of logistic regression

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

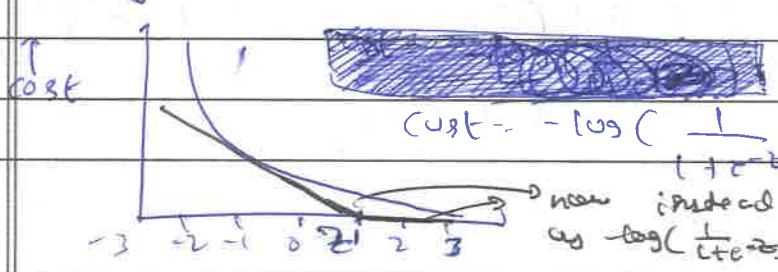


If $y = 1$, we want $h_\theta(x) \approx 1$, $\theta^T x \gg 0$
If $y = 0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

Cost of 1 example : $-(y \log(h_\theta(x)) + (1-y) \log(1 - h_\theta(x))$

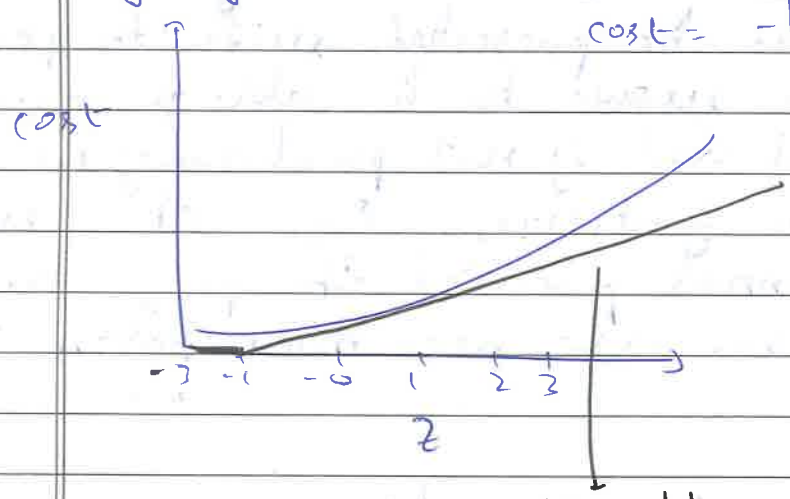
$$= -y \log\left(\frac{1}{1+e^{-\theta^T x}}\right) - (1-y) \log\left(1 - \frac{1}{1+e^{-\theta^T x}}\right)$$

If $y = 1$ (want $\theta^T x \gg 0$):



↳ $y = 0$ (want $\theta^T x << 0$):

$$cost = -\log\left(1 - \frac{1}{1+e^{-z}}\right)$$



Use this as cost function instead of $-\log\left(1 - \frac{1}{1+e^{-z}}\right)$

⇒ For support vector machines:

$$cost = \frac{1}{m} \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)}) + \frac{\lambda}{2m} \sum_{j=0}^{n} \theta_j^2$$

we remove m for support vector machines

Before for regularization we did $A + \lambda B$ where $A \to \frac{1}{m}\sum_{i=1}^{m} y\, cost \ldots$ & $B = \frac{1}{2m}\sum_{j=1}^{n} \theta_j^2$

but now we would do:-

$CA + B$. for SVM's

where C plays role like $1/\lambda$

thus

$$cost = \min_{\theta} C\left[\sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{j=1}^{m} \theta_j^2$$

Hypothesis $h_\theta(x)$ $\begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$

→ large margin intuition

Date: / /

```
>> [a, b] = SquareAndCube(5)
>> a
   a = 25
>> b
   b = 125
```

Function J = costFunctionJ(X, y, theta)
% X is the "design matrix" containing our training examples.
% y is the class labels.
m = size(X, 1); % number of training examples.
predictions = X * theta; % predictions of hypothesis on all m examples
sqrError = (Predictions - y).^2;
J = 1/(2*m) * sum(sqrError);

```
>> indices = 1:10;
>> for i = indices,
       disp (i);
   end;
>> i = 1;
>> while i <= 5,
       v(i) = 100;
       i = i + 1;
   end;
>> i = 1;
>> while true,
       v(i) = 999;
       i = i + 1;
       if i == 6,
           break;
       end;
   end;
>> v(1) = 2;
>> if v(1) == 1,
>       disp ('The value is one');
> elseif v(1) == 2,
>       disp ('The value is two');
> else
>       disp ('The value is not one or two.');
> end;
The value is two

function y = SquareThisNumber(x)
    y = x.^2

function [y1, y2] = SquareAndCube(x)
    y1 = x.^2;
    y2 = x.^3;
```

```
>> y2 = cos(2 * pi * 4 * t);
>> [redacted] hold on;
>> plot(t, y2, 'r');
>> xlabel('--');
>> [redacted] ylabel('--');
>> title('--'); legend('sin', 'cosine');
>> figure(2);
>> plot(t, y1); [redacted] figure(1);
>> plot(t, y2); figure(2);
>> [redacted] cd 'C:\users\...';
>> print -dpng 'myplot.png'
>> close % This would cause the figure to go away.

>> subplot(1, 2, 1); % divides plot e 1x2 grid, access first element

>> plot(t, y1);
>> subplot(1,2,2); % access 2nd element
>> plot(t, y2);
>> plot(t, y2);
>> [redacted] help axis
>> axis([0.5 1 -1 1])
>> clf;
>> A = magic(5)
>> imagesc(A)
>> imagesc(A), colorbar, colormap gray.
>> V = zeros(10,1)
>> for i = 1:10,
>     V(i) = 2^i;
>     end;
>> V
V =
     2
     4
     8
    16
     1
```

```
>> let A =
        8   1  6
        3   5  7
        4   9  2
>> max ( A, [], 1 ) %. returns maximum
Ans :                  of each column
        8   9  7
>> max ( A, [], 2 ) %. returns maximum
Ans :                  of each row.
        8
        7
        9
>> max ( A(:) )
>> A = magic ( 9 )
>> sum( A, 1 ) %. sums up each
                  column of A.
>> sum ( A, 2 ) %. sums up each row of A
```

now if we want to find the sum of the diagnol

```
>> sum( sum (A .* eye (9)))
Ans = 369
>> eye (3)
     1   0   0
     0   1   0
     0   0   1
>> flipud ( eye (9) )   %. this would flip
                           the matrix vertically
     0   0   1
     0   1   0
     1   0   0

>> t = [0: 0.01: 0.98] ;
>> y1 = sin( 2 * pi * 4 * t) ;
>> plot (t, y1) ;
```

papergrid

Date: / /

```
>> a = [1 1s 2 0.5]
>> a < 3
     1   0   1   1
>> find (a < 3)
     1   3   4
```

↳ index of elements less than 3.

```
>> A = magic (3)
A =
     8   1   6
     3   5   7
     4   9   2
```

(Sum of every row, column and diagonal adds up to the same number)

```
>> sum (A)
    18.500
>> prod (a)
     1s
>> floor (a)
     1   1s   2.0
>> ceil (a)
     1   1s   2   1
>> max ( rand (3), rand (3))
    0.72136     0.78771     0.93872
```

This would return us a 3x3 matrix in which each element is the maximum of 2 randomly generated matrix.

let A = 1 2
3 4
5 6

B = 11 12
13 14
15 16

>> C = [A B]

C = 1 2 11 12
3 4 13 14
5 6 15 16

>> C = [A ; B]

C = 1 2
3 4
5 6
11 12
13 14
15 16

>> A * B
>> A .* B % element wise multiplication of 2 matrices
>> A .^ 2 % element wise squaring of A
>> 1./A % element wise reciprocal
>> log(V)
>> exp(V) % e^V (element wise)
>> abs(V)
>> A' (would return transpose of A)
>> max(V) % returns maximum value of V
>> [val, ind] = max(V) % returns maximum value as well as index of maximum value
value
index

>> V = vector_name (1:10) % now we will have the first 10 elements of vector_name into V.

>> clear % would all the variables.

>> A = [1 2; 3 4; 5 6]

>> A (3, 2) % element in 3rd row 2nd column
6

>> A (2, :) % every element in 2nd row.

>> A (:, 2) % every element in 2nd column.

>> A ([1 3], :) % every element in 1st & 3rd row.

>> A (:, 2) = [10; 11; 12]
So now A would become:

1	10
3	11
5	12

>> A = [A, [100; 101; 102]];
now A would become:

1	10	100
3	11	101
5	12	102

>> A (:) % put all elements of A into a single vector

1
3
5
10
11
12
100
101
102

```
>> A = [ 3 2 ; 1 3 ]
>> size (A)
     1   2
>> size ( A, 1 )  % would return the number
                    of rows.
>> size ( A, 2 )  % would return the number
                    of columns
>> U = [ 1 2 3 4 ]
U =
     1   2   3   4
>> length (U)
Ans = 4
>> length (A)   % returns the longest dimension.
ans = 3
>> pwd % would tell us the current directory
                                    path.
>> [redacted]
>> cd '.....'  % would change directory.
>> ls  % list files of current directory.
>> load ( ' name of the file ' ) % this would
                                   load the file
                                   into octave

>> who  % this would tell us what variables
          we have instantiated.

>> size ( . . )

>> whos  % would give us the detail
           view of all the variables.

>> clear variable name.
```

- onion
- Raw banana
- buffallo

papergrid

Date: / /

~~[illegible]~~, Amt of Rs 100000 is invested in 2 shares. First one interest ~~[illegible]~~ 9% and second 11% per annum. Total interest is 93/4 %. find amt invested in each shares.

1000

C = ones (1,3)
C = 2 * ones (1,3)
C = zeros (1,3)

ones(1,3)

r = rand (3,3)
C = randn (1,3) → number of numbers returned
(normal distribution) ↳ variance = 1
mean = 0

→ 3×3 matrix of random numbers from 0 to 1

w = -6 + sqrt(10) * (randn (1,10000))

here mean would be -6
variance = 1
this would return 10000 numbers

eye(3)

1 0 0
0 1 0
0 0 1

help eye

(1) Water melon 2
(2) green grapes.
(3) buffalo meat.
- 1/2 letre - 2L
(4)
(5) cheese.
(6) raw banana
(7) ginger.
(8) metro shaft