Machine Learning

By Andrew Ng

* Tom Mitchell (1998) : A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measure by P, improves with experience E.

Ex : Checkers : E → Playing 1000's of games
T → Play checkers
P → Winning or loosing Probability.

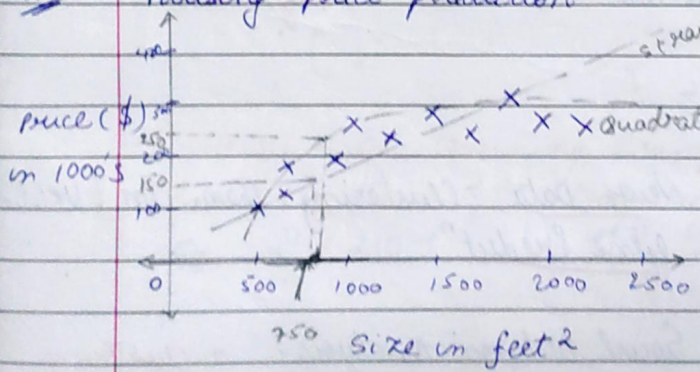
Ex Spam & Non-spam Email - ID *
Classifying : Task T
Labelling emails : Experience E
correctly classified : Performance P

Machine learning
- Supervised Learning
- unsupervised learning

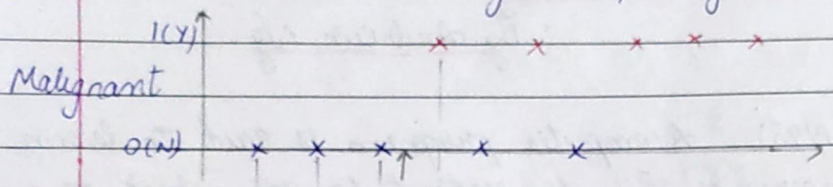
* Supervised learning : "Right answers" given To the host

Ex Housing price prediction :

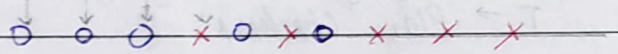
Regression - Predict continuous valued output (price)



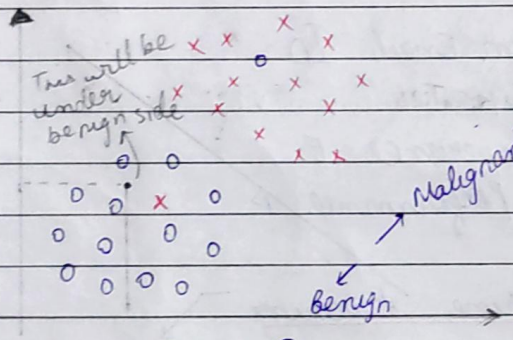
Ex Breast Cancer (Harmful Malignant, Harmless benign) :



Classificat^{n} :
Discrete valued output (0 or 1)
There can be more classification too like
There are 3 types of cancer
↓ ↓ ↓
0 benign type 1 type 2 type 3 cancer



Ex

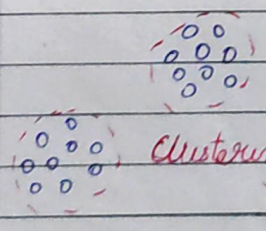


We can also use different parameters like
: Clump thickness
: Uniformity of cell size
: Uniformity of cell shape
∴ It can use ∞ no of features too.

↳ Ex of Supervised learning

* Quest^{n} of Classificat^{n} & regression
Hacking → output Yes or No → classification
Say Produce crops which will sell or not → Regression

* Unsupervised learning :



* News Data : Clustering them in 1 URL link
* Gene Predict^{n}

Ex Social Network Analysis.
Market Segments
Astronomical Data Analysis.
Organising Computer cluster.
} clustering problems

1-10 in English

Ex. Speaker 1 (Voice 1) — Mic 1 ← 2 diff Post^n — Mic 2 — Speaker 2 (Voice 2) → 1-10 in French

Non-Clustering
Cocktail party problem

To differentiate the two voice
we use this single line of code

```
[W,s,v] = svd((repmat(sum(x.*x,1),size(x,1),1).*x)*x');
```

* Cocktail party prob Identify individual voice.

* We will be using Octave as our environment

* Question for unsupervised learning algorithm.
Clustering like → News article group them on same story
→ Database for discovering market segments.

Intro Quiz 1 × 2 ✓ Regres^n 3 × Anom Regres^n 4 × 5 ✓ Defin^n

* Model and Cost function :
We saw that Example of housing System
Training set of housing prices (Portland)

Size in feet^2 (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...
x	y

m = No of Training Ex
x's = "input" variable / features
y's = "output" variable / "target" variable

(x, y) = One training Example
$(x^{(i)}, y^{(i)})$ = i^{th} training example
Ex $x^{(1)} = 2104$, $y^{(2)} = 232$.

Training set

↓

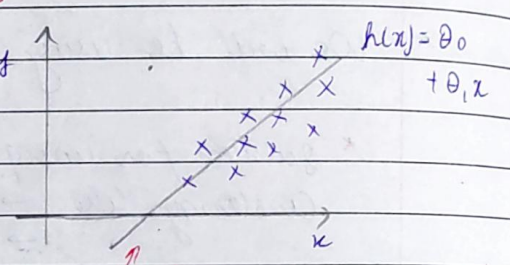
Learning Algorith

↓

Size of house → h → Estimated price predicted

x Hypothesis y

h maps from x's to y's



★ How do we represent h?

$h_\theta(x) = \theta_0 + \theta_1 x$

Short hand

$h(x) = \theta_0 + \theta_1 x$

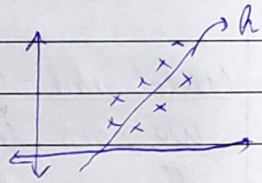
Linear Regression with one variable. } ↗ Name of this model

Univariate Linear Regression }

⇒ Using the data set in Prev page

Hypothesis → $h_\theta(x) = \theta_0 + \theta_1 x$

θ_i's : Parameters



How to chose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training example (x, y)

$$\underset{\theta_0, \theta_1}{\text{minimize}} \; \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

m → training set; $h_\theta(x^{(i)})$ → $\theta_0 + \theta_1 x^{(i)}$; $y^{(i)}$ → outcome provided; 2m → Mathematical convenience

Cost funct^n → $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$ } Squared Error funct^n

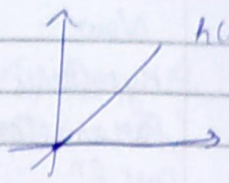
Goal → $\underset{\theta_0, \theta_1}{\text{minimize}} \; \underbrace{J(\theta_0, \theta_1)}_{\text{cost } f^n}$

parameters → θ_0, θ_1

⇒ Simplified hypothesis Equat^n :

$h_\theta(x) = \theta_1 x$ & $\theta_0 = 0$

$\therefore h(x) = \theta_1 x$

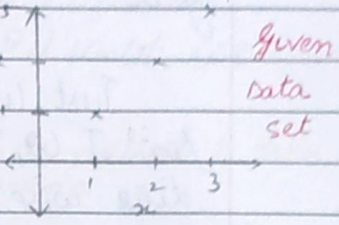


Parameters → θ_1

cost f^n → $J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

$h_\theta(x)$ → θ_1

Goal → minimize $J(\theta_1)$
θ_1

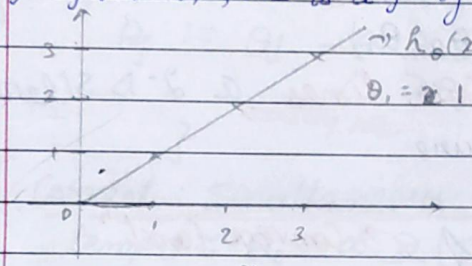


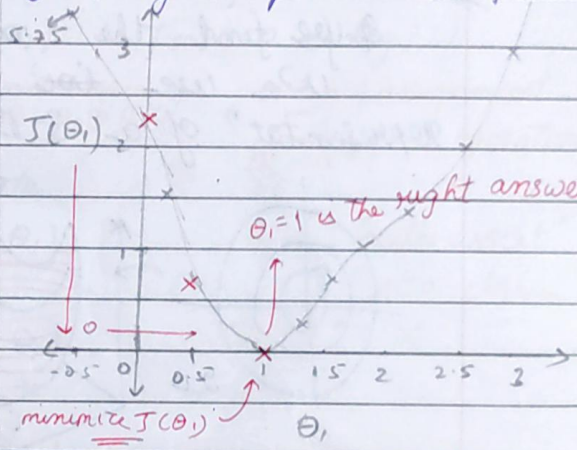
Given data set

$h_\theta(x)$ for fixed θ_1, this is a f^n of x

$J(\theta_1)$ funct^n of the parameter θ_1.

i)





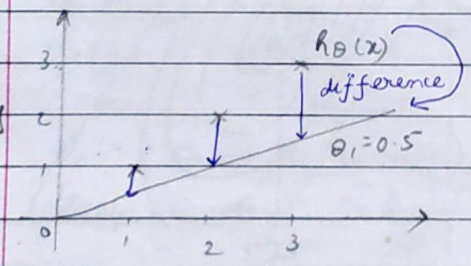
$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

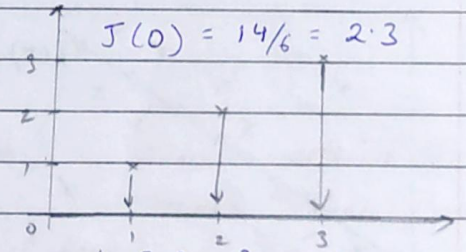
$= \frac{1}{2m} \sum_{i=1}^{m} (\theta_1 x^{(i)} - y^{(i)})^2$

$= \frac{1}{2m} (0^2 + 0^2 + 0^2) = 0^2$

$\therefore J(1) = 0$

ii)





$J(0.5) = \frac{1}{2m} \left[\left(\frac{1}{2} - 1\right)^2 + (1-2)^2 + \left(\frac{3}{2} - 3\right)^2 \right]$

$= \frac{1}{2 \times 3} \left[\frac{14}{4} \right] = 0.58$

$J(0.5) = 0.58$

Now :

Hypothesis $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters θ_0, θ_1

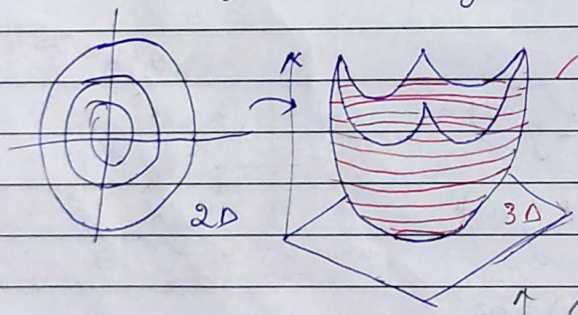
Cost f^n $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

Goal $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Just like for one variable θ_1 we had a 2-D dig for $J(\theta_1)$ and then we found global minimum. Like wise for $J(\theta_0, \theta_1)$ we get a 3-D shape

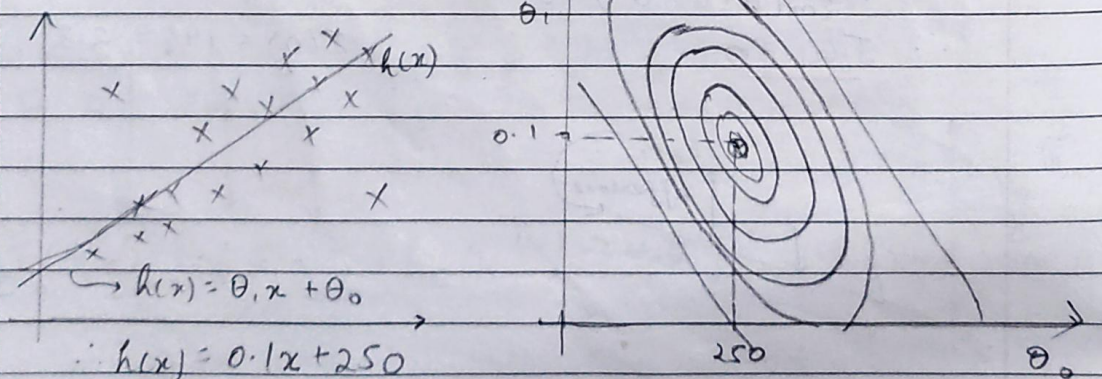
Eg:- $J(\theta_0, \theta_1) = z = x^2 + y^2$ we do the same analysis & we find the $\min J(\theta_0, \theta_1)$

We use Contour lines a 2-D schematic representat^n of a 3-D figure



→ Convex funct^n
Bowl-Shaped

Eg:-



$\hookrightarrow h(x) = \theta_1 x + \theta_0$

$\therefore h(x) = 0.1x + 250$

Best h we can find

★ Gradient Descent :

We Have some f^n $J(\theta_0, \theta_1)$ or $J(\theta_0, \theta_1, \theta_2 \cdots \theta_n)$

We want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$ or $\min_{\theta_0 \cdots \theta_n} J(\theta_0, \cdots \theta_n)$

⇒ Outline :-

Start with some θ_0, θ_1 [say $\theta_1 = \theta_0 = 0$]

keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum.

★ Gradient Descent Note its method to take a small steps towards a global minima.

⇒ Algorithm :-

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

↙ learning rate

Simultaneously update θ_0 & θ_1

$a := b$ } assignment operator

$a := a + 1$

}

✓ Correct : Simultaneous update

$a = b$ } Truth assert"

Just a claim

$\times a = a + 1 \times$

$$temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$\theta_0 := temp0$ } update θ_0 & θ_1

$\theta_1 := temp1$

✗ Incorrect :

$$temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$\theta_0 := temp0$ If we write it here

$$temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

Then updated θ_0 will move to the further steps that we don't want.

So make Sure to update it at end.

$\theta_1 := temp1$

Q) Suppose $\theta_0 = 1$, $\theta_1 = 2$ & we update it using the rule $\theta_j := \theta_j + \sqrt{\theta_0 \theta_1}$ (for $j=0$ & $j=1$) what are the resulting values of θ_0 & θ_1?

⇒ $\therefore \theta_0 := \theta_0 + \sqrt{\theta_0 \theta_1}$

$\theta_0 := 1 + \sqrt{1 \cdot 2} = 1 + \sqrt{2}$

$\theta_1 := 2 + \sqrt{2} < 2 + \sqrt{2}$ //

$\therefore \theta_0 = 1+\sqrt{2}$ & $\theta_1 = 2+\sqrt{2}$

$\theta_0 := \theta_0 + \sqrt{2}$

$\theta_0 := 1 + \sqrt{2}$

$\theta_1 := 2 + \sqrt{2}$

★ Gradient descent algorithm:

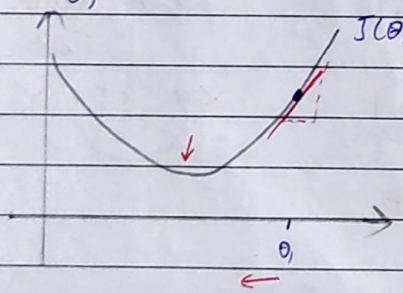
repeat until convergence {

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ (for $j=0$ & $j=1$)

(α: learning rate; $\frac{\partial}{\partial \theta_j}$: derivative)

}

$\min_{\theta_1} J(\theta_1)$ for $\theta_1 \in \mathbb{R}$ [Real]

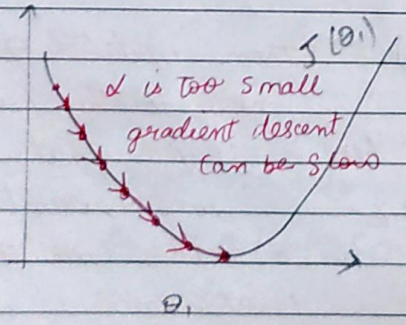


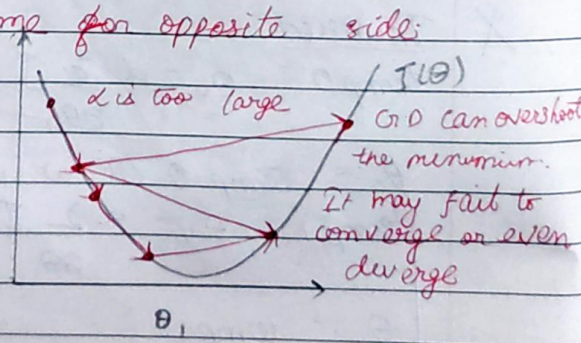
$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$ slope is (+)

$\theta_1 := \theta_1 - \alpha$ (positive number)

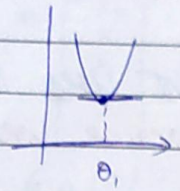
∴ we have to go towards left on scale, which is true.

Same for opposite side:





$\theta := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$

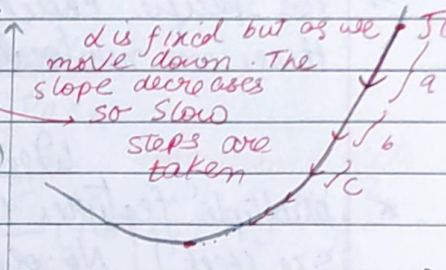
⇒ 

$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$ as slope is zero

$\therefore \theta_1 := \theta_1$ unchanged

⇒ Gradient Descent can converge to a local minimum, even with the learning rate α fixed.

$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$ — α: const; keeps on decreasing



α is fixed but as we move down. The slope decreases so slow steps are taken

As we approach a local minimum, $J(\theta_1)$ g.d will automatically take smaller steps. So no need to decrease α over time

$a > b > c$

Gradient Descent Algorithm

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

for $j = 1$ & $j = 0$

Linear Regression Model

$h_\theta(x) = \theta_0 + \theta_1 x$

$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$$= \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

θ_0; $j = 0$: $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$

θ_1; $j = 1$: $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$

Gradient Descent Algorithm:

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

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

update θ_0 & θ_1 simultaneously

★ Gradient Example

"Batch" gradient Descent

"Batch": Each step of gradient descent uses all the training examples. Just because we look at the whole batch (m)

⇒ Gradient descent can converge even if α is kept fixed (But α cannot be too large, or else it may fail to converge)

⇒ For specific choice of cost f^n $J(\theta_0, \theta_1)$ in linear regression there are no local optima (other than global optimum).

Week - 2

★ Multiple features (variables):

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

$n = 4$ (features) m (examples)

Notation:

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

$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$ (1, 2, 3, 4)

$x_3^{(2)} = 2$

Hypothesis: Now

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

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

(Example: Base price; ↑ beds price; ↑ price floors; ↑ price size; ↓ price age of home)

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

for convenience of notation, define $x_0 = 1$ $[x_0^{(i)} = 1]$

Take dot and

$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$ $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$ $\underbrace{[\theta_0\ \theta_1 \cdots \theta_n]}_{\theta^T\ (n+1)\times 1\ \text{matrix}} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$ we get $h_\theta(x)$

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

$h_\theta(x) = \theta^T x$ → Multivariate linear regression

* Gradient descent for Multiple variables :

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

Parameters = $\theta_0, \theta_1, \cdots \theta_n$ / θ $n+1$ - dimensional vector

Cost fn $= J(\theta_0, \theta_1, \cdots \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

$J(\theta)$

Gradient descent

Repeat {

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \cdots \theta_n)$ } (simultaneously update for every $j = 0, \cdots n$)

$J(\theta)$

Gradient descent

Previously ($n = 1$):

Repeat {

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

$\frac{\partial}{\partial \theta_0} J(\theta)$ $[x_0 = 1]$

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

}

New Algorithm ($n \geq 1$):

Repeat {

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

} (simultaneously update θ_j for $j = 0, 1, 2 \cdots n$)

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

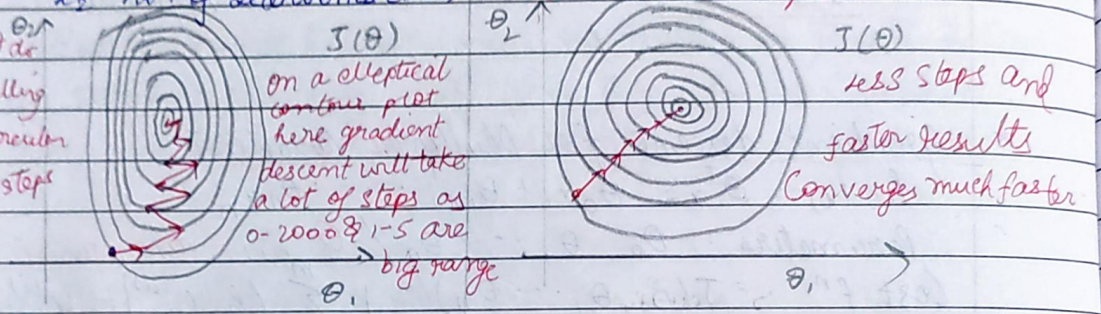
* Gradient Descent in Practice :

1. Feature Scaling :

Idea : Make sure features are on a similar scale.

Ex x_1 = Size (0-2000 feet^2) $\longrightarrow$ $x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$

x_2 = no. of Bedrooms (1-5) $\longrightarrow$ x_2 = no. of bedrooms / 5



∴ We must do feature scaling to make it circular so that less steps are taken

$J(\theta)$ on a elliptical contour plot here gradient descent will take a lot of steps as 0-2000 & 1-5 are big range

$J(\theta)$ less steps and faster results Converges much faster

Ignore θ_0

Feature scaling : Get every feature into approximately $-1 \le x_i \le 1$ range

$x_0 = 1$

Good scaling will work: $\{0 \le x_1 \le 3$ ✓ $\}$ $\{-2 \le x_2 \le 0.5$ ✓ $\}$

$\{-100 \le x_3 \le 100$ ✗ $\}$ $\{-0.0001 \le x_4 \le 0.0001$ ✗ $\}$ Poor scaling / Not approx. to

$\{-3$ to 3 & $-\frac{1}{3}$ to $\frac{1}{3}\}$ works

⇒ Mean normalization :

Replace x_i with $x_i - \mu_i$ to make features have approx. zero mean [Do not apply to $x_0 = 1$]

(0-2000) Ex $x_1 = \frac{\text{size} - 1000}{2000} \longrightarrow -0.5 \le x_1 \le 0.5$

(1-5) $x_2 = \frac{\#\text{bedroom} - 2}{5} \longrightarrow -0.4 \le x_2 \le 0.4$

$\therefore x_1 \longrightarrow \frac{x_1 - \mu_1}{s_1}$ (μ_1 → avg value of x in set)

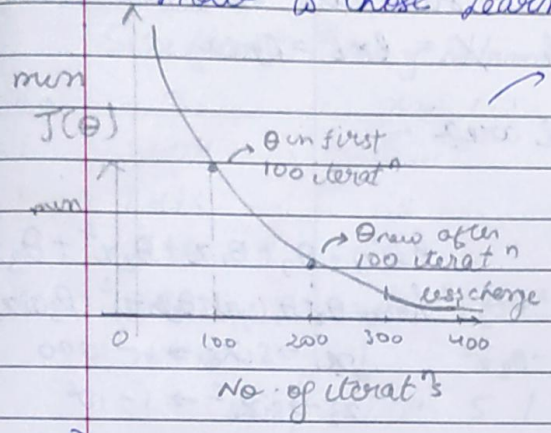
(s_1 → range (max - min) or standard deviation)

2. Learning Rate :

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$; Repeat is called no. of iterat^{n}s

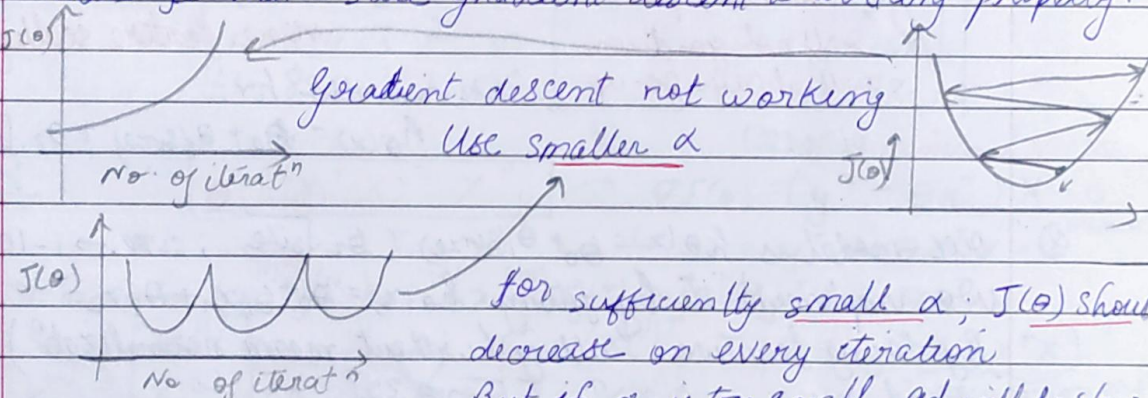
Debugging : How to make sure gradient descent is working correctly

How to chose learning Rate α



Example automatic convergence test
Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration

⇒ Cases to make sure gradient descent is working properly :



For sufficiently small α, $J(\theta)$ should decrease on every iteration
But if α is too small, gd will be slow to converge

⇒ Summary :

If α is too small → slow convergence

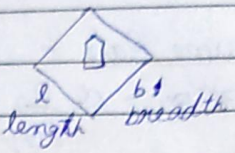
If α is too large → $J(\theta)$ may not decrease on every iteration; may not converge ~~Esta~~

Try α : 0.001, 0.01, 0.1, 1, ...

grad = $\frac{1}{m} \sum h(\theta^{(i)}) - y^{(i)}) x^{(i)}$

★ Features & Polynomial Regression :

Ex Housing price prediction :



So our $h_\theta(x)$ will be

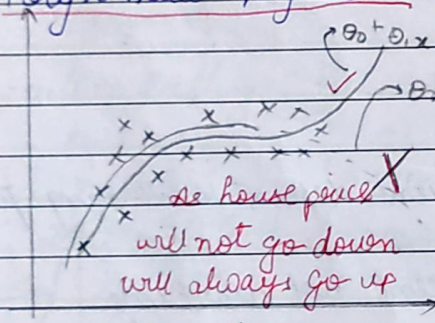
$$h_\theta(x) = \theta_0 + \theta_1 \underbrace{l}_{x_1} + \theta_2 \underbrace{b}_{x_2}$$

So instead using l & b we can substitute them with Area & make a new variable $X = l \times b$ = Area

$$\therefore h_\theta(x) = \theta_0 + \theta_1 X$$

↳ land area

★ Polynomial Regression :



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

$$h_\theta(x) = \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2 + \theta_3(\text{size})^3$$

$x_1 \to \text{size} \Rightarrow 1 - 1000$

$x_2 \to \text{size}^2 \to 1 - 10^6$

$x_3 \to \text{size}^3 \to 1 - 10^9$

Here feature scaling becomes imp.

We can also try :

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

Q) Our model is $h_\theta(x) = \theta_0 + \theta_1(\text{size}) + \theta_2\sqrt{\text{size}}$; $\text{size} \to 1 - 1000\,(\text{feet})^2$

We implement it by fitting $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

By doing feature scaling (without mean normalizat^n)

find new x_1 & x_2 ? [$\sqrt{1000} \approx 32$]

$\Rightarrow x_1 \Rightarrow \frac{\text{size}}{1000}$; $x_2 \Rightarrow \frac{\sqrt{\text{size}}}{32}$

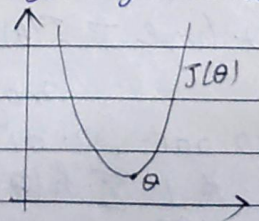
★ Normal Equation : Method to solve for θ analytically

Intuition : If $\theta \in 1D$ ($\theta \in \mathbb{R}$)

$$J(\theta) = a\theta^2 + b\theta + c$$

So we do $\frac{d}{d\theta} J(\theta) = 0$

& solve for θ



But we need to solve it for
$\theta \in R^{n+1}$ $J(\theta_0, \theta_1 \cdots \theta_m) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots = 0$ (for every j) & solve for $\theta_0, \theta_1 \cdots \theta_n$

x_0	Size (feet)^2 x_1	No. of bedroom x_2	No of floors x_3	Age of home (yrs) x_4	Price 1000$ 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

$n = 4$

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$M \times (n+1)$

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

$\rightarrow \nabla J(\theta) = (y^T - \theta X^T) X = 0$; m-dimensional vector

$y^T X - \theta X^T X = 0$ numinia

$y^T X = \theta X^T \cdot X$

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

m-Example : $(x^{(1)}, y^{(1)}), \cdots (x^{(m)}, y^{(m)})$ - n features

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in R^{n+1} \qquad X = \begin{bmatrix} \cdots (x^{(1)})^T \cdots \\ \cdots (x^{(2)})^T \cdots \\ \vdots \\ \cdots (x^{(m)})^T \cdots \end{bmatrix}$$

design matrix

$m \times (n+1)$

Ans If $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$ $\xrightarrow{x^T}$ design matrix $X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_2^{(1)} \\ \vdots & \\ 1 & x_m \end{bmatrix}_{m \times 2}$ $y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$

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

pinv → pseudo inverse
inv → inverse

Page No. 16
Date 23/12/2021

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

Octave: pinv(X' * X) * X' * Y ; X' → X^T ; pinv for inverse

Feature Scaling for normal method is OK. No problem

$0 \le x_1 \le 1$
$0 \le x_2 \le 10^3$
$0 \le x_3 \le 10^{-5}$

Alright No need for feature Scaling in Normal method.

* m training Examples, n features

Gradient Descent	Normal Equation
⇒ Need to choose α	No need to chose α
⇒ Needs many iterations	Don't need to iterate
⇒ Works well even n is large like n can be in millions too	Need to compute $(X^T X)^{-1}$ [n x n] $O(n^3)$ Slow if n is very large like $n = 10^2, 10^3, 10^4$ But 10^6 Not good. Favours GD for 10^6 or above

* We know ; $\theta = (X^T X)^{-1} X^T y$

what if $X^T X$ is non-invertible ? (singular / degenerate)

i) Redundant features (linearly dependent)

x_1 = size in feets^2 } $1m = 3.28$ feet ∴ $X = [\alpha \ (R)\alpha]$

x_2 = size in m^2 } ∴ $X^T X \to [0]$; $x_1 = (3.28)^2 x_2$

∴ Delete any one ✓ ✗

ii) Too many features (e.g $m \le n$) ; $m = 10$, $n = 100$ $\theta \in \mathbb{R}^{101}$

Delete some features or use regularization. → later we will study.

* Octave Operations :

⇒	1 == 2	1 ~= 2 (not Equal to)	1 && 0 (and operat^n)	1 || 0 (or operat^n)
	ans = 0 ← false	ans = 1 ← True	ans = 0	ans = 1

⇒ xor(1,0) ans = 1

a = pi ; a

a = 3.1416

disp(a) 3.1416

disp(sprintf('2 decimal : %0.2f', a)) 2 decimal : 3.14

fs1('>> '); clear (left side)

Page No. 17
Date / /

⇒ format long
a
a = 3.141592653589793

format short
a
a = 3.1416

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

⇒ v = [1 2 3]
v = 1 2 3

z = [1; 2; 3]
z = 1
2
3

v = 1 : 0.2 : 2
v = 1.0 1.2 1.4 1.6 1.8 2.0

⇒ v = 1 : 6
v = 1 2 3 4 5 6

ones(2,3)
ans = 1 1 1
1 1 1

c = 2*ones(2,3)
c = 2 2 2
2 2 2

w = zeros(1,3)
w = 0 0 0

⇒ rand(3,3)
ans = 0.8 0.3 0.7
0.1 0.2 0.3
0.9 0.8 0.4

randn(1,3)
w = -1.4 -1.2 -0.6

w = 6 + sqrt(10)*(randn(1,1000))
hist(w)
(Draws a histogram)
his(w,50)

⇒ eye(4)
Diagonal matrix
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

help eye
Brings up documentat^n for both the command help help

help rand

A = [1 2; 3 4; 5 6];
size(A)
ans 3 2
sz = size(A);
size(sz)
ans 1 2
ls (gives list of directories on my desktop)

⇒ v = [1 2 3 4]' ; 1×4
length(v)
ans = 4
(longest dimens^n)

Pwd (location/path '/Matlal Drive')

cd (to change directory)

⇒ load features.dat
load('features.dat')
to load any file

who
Variable in the current workspace

size(features)
ans = 47 2
matrix m×n of real data

whos - gives details of all the variable

⇒ clears features removes the loaded data

v = price(1:3)
5 looks 1st 3 elements of some loaded data price

save hello.mat v;
Save v in hello.mat folder on desktop

clear
It clears everything

⇒ Save hello.txt v -ascii | A = [1 2; 3 4; 5 6]; | A(:,2)
Save data as text | A(2,:) → means every element along that row/col.
ans = 3 4

⇒ A([1 3],:) | A(:,2) = [10; 11; 12] | A = [A, [100; 101; 102]];
ans = 1 2 | A = 1 10 | A = 1 10 100
5 6 | 3 11 | 3 11 101
5 12 | 5 12 102
get from 1st & 3rd row | assigning new value to the matrix | append another column vector to right

⇒ A(:) | A = [1;2;3] ans 1 4 | C = [A; B] | size(C)
puts all elements of A into a single vector | B = [4;5;6] 2 5 | ans = 2 on top of each other | ans = 6 1
C = [A B] 3 6
concatenates A & B

⇒ A = [1 2; 3 4; 5 6] | A*C multiply A & C | A.*B multiply individually with same i,j values of matrix | A.^2
B = [11 12; 13 14; 15 16] | ans 11 24 / 39 56 / 75 96 | ans 1 4 / 9 16 / 25 36
C = [1 1; 2 2] | Squaring the matrix

⇒ V = [1; 2; 3]; | log(V) | exp(V) | abs(V) (modulus) | -V
1./V Reciprocal | ans 0.00 | ans 2.71 | ans 1 | ans -1
ans 1.00 | 0.693 | 7.38 | 2 | -2
0.50 | 1.098 | 20.08 | 3 | -3
0.33

⇒ V + ones(length(V),1) | V+1 | A' Transpose (A') A matrix
ans 2 / 3 / 4 (& add +1 in all the numbers) | ans 2 / 3 / 4 | ans 1 3 5 / 2 4 6 | ans 1 4 / 2 5 / 3 6

⇒ a = [1 15 2]; | [val, ind] = max(a) (value, index) | a < 3 | find(a<3) | magic(3)
val = max(a), val = 15 | ans 1 0 1 (T/F) | ans 1 3 | 8 1 6 / 3 5 7 / 4 9 2
ans val = 15, ind = 2 | magic square

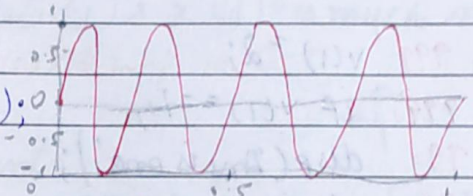
max numbers zeros cols

Page No. 19
Date / /

⇒ magic(A); | a=[1 15 2]; | max(A,[],1) | sum(A,1)
[r,c] = find(A>=7) | sum(a) 18 | ans 8 9 7 | 15 15 15
ans r= 1 3 2 | c= 1 2 3 | ie (1,1) (3,2) (2,3) | prod(a) 30 | max(A,[],2) | sum(A,2) 15 15 15
floor(a) 1 15 2 | ans 8 9 7 col wise

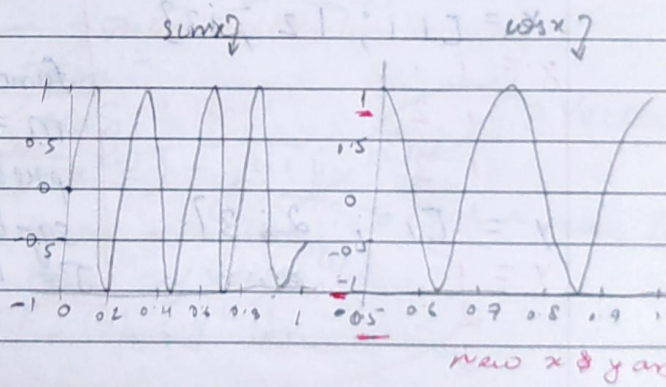
⇒ eye(3); | sum(sum(A*eye(3))); | flipud(eye(3)) | pinv(A) inverse
A*eye(3) ans = 15 | 0 0 1 flip | 0.147 -0.14 0.06
8 0 0 | 0 1 0 it | -0.06 0.02 0.10
0 5 0 | 1 0 0 | -0.01 0.18 -0.10
0 0 2

⇒ x=[0:0.01:1];
y1=sin(2*pi*4*x);
plot(x,y1)



⇒ y2=cos(2*pi*4*x);
hold on; to overlap new plot
plot(x,y2,'r');
xlabel('time')
ylabel('value')
legend('sin','cos')
title('my plot')

splits the sin & cos graph ⇒ figure(1); plot(x,y1);
figure(2); plot(x,y2);
sin & cos diff graph ⇒ subplot(1,2,1);
plot(x,y1);
subplot(1,2,2);
plot(t,y2);
change axis ⇒ axis([0.5 1 -1 1])
x-axis y-axis



clf; clear figure

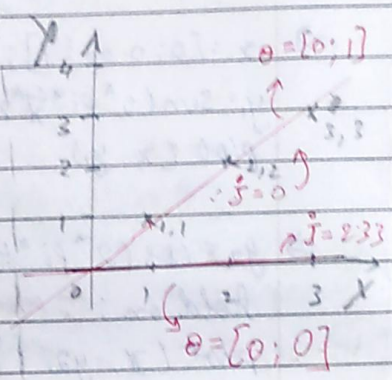
⇒ A = magic(5); * See the figure in Notes
imagesc(A)
imagesc(A), colorbar, colormap gray;

⇒ for i = 1:3,
v(i) = 2^i;
end;
* v = 2 4 8

indices = 1:3;
for i = indices,
disp(i);
end;
indices = 1 2 3

V = 2 4 8 16 32 64
i = 1;
while i <= 4
v(i) = 99
i = i+1;
end;
99 99 99 99 32 64

⇒ i=1;
while True,
v(i)=999;
i=i+1;
if i == 3
break;
end;
end;
Ans 999 v(1) = 2;
999 if v(1) == 1,
999 disp('Ans is one');
99 elseif v(1) == 2,
99 disp('Ans is two');
32 else
64 disp('No ans');
end; The Ans is two



* Now we want to predict cost f^n using Octave :

X = [1 1; 1 2; 1 3]
X = 1 1
1 2
1 3

y = [1; 2; 3]
y = 1
2
3

theta = [0; 1];
j = costFunction(X, y, theta)
j = 0

theta = [0; 0];
j = costFunction(X, y, theta)
j = 2.33
% i.e. $\frac{1^2+2^2+3^2}{2\times 3} = 2.33$
m

make a notepad - m

```
function J = costFunction(X, y, theta)
m = size(X, 1);
predictions = X * theta;
sqrErrors = (predictions - y) .^ 2;
J = 1/(2*m) * sum(sqrErrors);
```

As indexing starts from 1 in matlab

★ Vectorization :

$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j = \theta^T x$

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$ (theta 1, theta 2, theta 3) $X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$

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

Vectorized Implementation
prediction = theta' * x;
Both will give same output But vectorized imp. is much more efficient.

★ vectorized imp. in C++

⇒ Gradient Descent :

$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$ (for all j) $j = 0, 1, 2 \ldots$

$\delta =$

$n = 2$

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

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

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

simultaneous updates

vector substraction

Vectorized Implementation :

$\theta := \theta - \alpha \delta$ ($\mathbb{R}^{n+1} \to \mathbb{R}^{n+1}$)

where $\delta = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$ ($\mathbb{R}^{n+1}$)

Vector

Real Number $\mathbb{R}$

$\delta = \begin{bmatrix} \delta_0 \\ \delta_1 \\ \delta_2 \end{bmatrix}$ $\delta_0 = \frac{1}{m} \sum h_\theta(x^{(i)} - y^{(i)}) x_0^{(i)}$

$\delta_1 = \frac{1}{m} \sum h_\theta(x^{(i)} - y^{(i)}) x_1^{(i)}$

$\vdots$

what does summation says

$x_i^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \end{bmatrix}$

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

$= (h_\theta(x^{(1)}) - y^{(1)}) x^{(1)} + (h_\theta(x^{(2)}) - y^{(2)}) x^{2} \ldots$

Number · vector

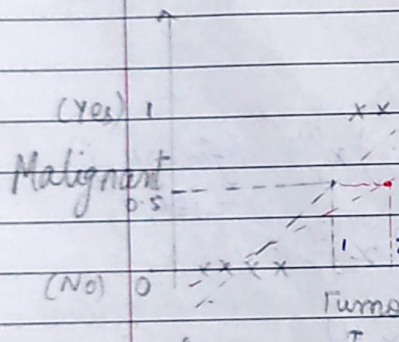
like $u(j) = 2v(j) + 5w(j) \Rightarrow u = 2v + 5w$ (for all j)

Week - 3

* Logistic Regression Classification :

Ex Email : Spam / Not spam ?
Online Transactions : Fraudulent (Yes / No)
Tumor : Malignant / Benign ?

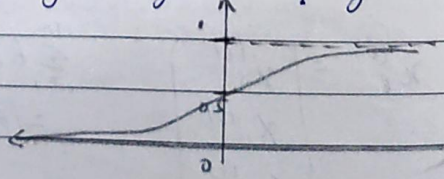
$y \in \{0, 1\}$ 0 : "Negative class" (e.g, benign tumor)
1 : "Positive class" (e.g, malignant tumor)
$y \in \{0, 1, 2, 3\}$ → Multiclassificat^n problem



Threshold classifier output $h_\theta(x)$ at 0.5
If $h_\theta(x) \geq 0.5$, predict $y = 1$
If $h_\theta(x) \leq 0.5$ predict $y = 0$
what if there is a new training set?
So now our line will look like ig
Line 1 was accurate but we are getting line 2. ∴ Linear regression fails in a classification problem

⇒ Classification : $y = 0$ or 1
$h_\theta(x)$ can be > 1 or < 0
We will make a model
Logistic Regression : $0 \leq h_\theta(x) \leq 1$
↳ classificat^n algorithm
In linear regression we had $h_\theta(x) = \theta^T x$
In logistic regression $h_\theta(x) = g(\theta^T x)$
$g(z) = \frac{1}{1 + e^{-z}}$ } sigmoid function / Logistic funct^n

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



Parameters θ

⇒ Interpretation of hypothesis output

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

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

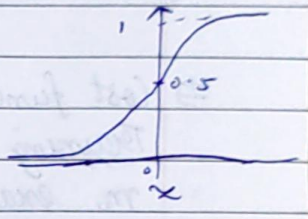
$h_\theta(x) = 0.7$, $y=1$

∴ Tell patient that 70% chance of tumor being malignant.

$h_\theta(x) = P(y=1 \mid x; \theta)$ → "prob. that $y=1$, given x, parameterized by θ"

$y = 0$ or 1

$\therefore P(y=0 \mid x; \theta) = 1 - P(y=1 \mid x; \theta)$



⇒ Logistic Regression :

$h_\theta(x) = g(\theta^T x) = P(y=1 \mid x; \theta)$

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

Predict $y=1$ if $h_\theta(x) \geq 0.5$ $\theta^T x \geq 0$

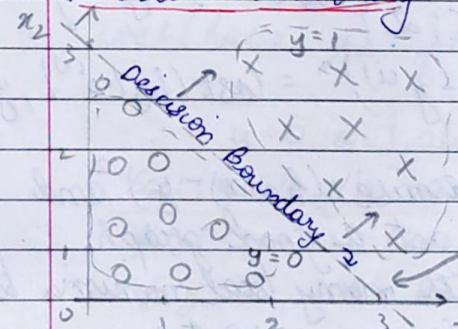
$y=0$ if $h_\theta(x) \leq 0.5$ $\theta^T x < 0$

$g(z) \geq 0.5$ when $z \geq 0$

$g(z) < 0.5$; $\theta^T x < 0$

$\therefore h_\theta(x) = g(\theta^T x) \geq 0.5$ whenever $\underbrace{\theta^T x}_{z} \geq 0$

⇒ Descision Boundary :



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

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

$-3 \quad 1 \quad 1$

Predict "$y=1$" if $-3 + x_1 + x_2 \geq 0$ ($\theta^T x$)

$x_1 + x_2 \geq 3$

$x_1 + x_2 < 3$; $y = 0$

Descisⁿ Bound : is only a property of Hypothesis not of dataset

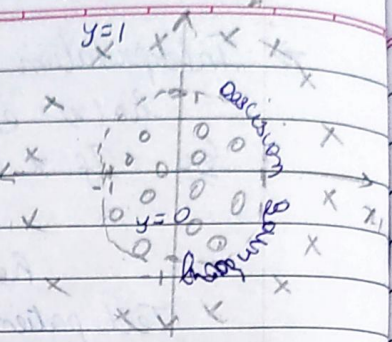
⇒ Non-linear Decision boundaries :

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

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

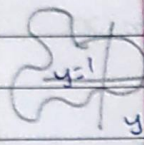
$\therefore y = 1$ if $-1 + x_1^2 + x_2^2 \geq 0$

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



We can often get more complex higher order features

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



⇒ Cost function for Logistic Regression :

Training set : $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \cdots (x^{(m)}, y^{(m)})\}$

m examples $x \in \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$ $x_0 = 1$; $y \in \{0, 1\}$

$\hookrightarrow \mathbb{R}^{n+1}$

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

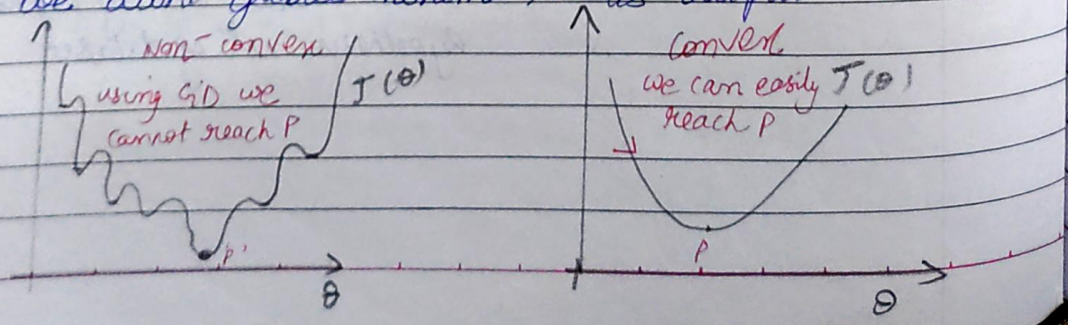
∴ How to choose parameter θ ?

Cost funct^n

linear regression $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$

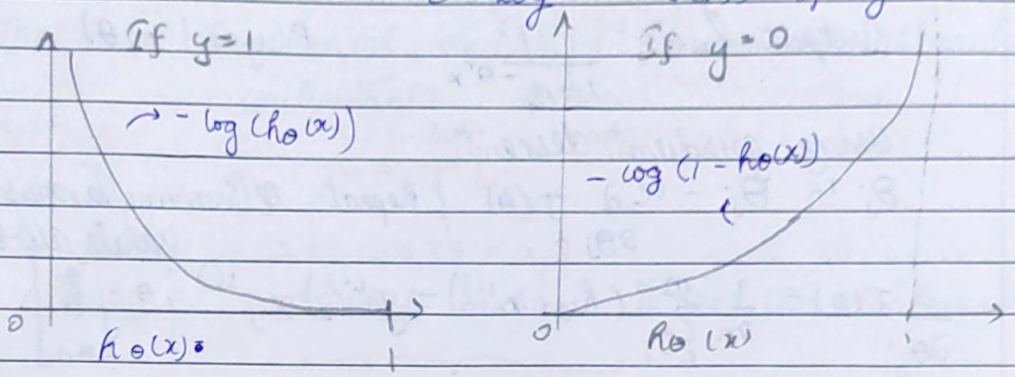
Now take $\frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2 = \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$

We dont use square Error because $(h_\theta(x) - y)^2$ and $h_\theta(x) = 1/1+e^{-z}$ we wont get a good graph we will have a graph with many local minima but we want global minima for as output.



⇒ Logistic Regression cost function :

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



cost = 0 ; if $y = 1$, $h_\theta(x) = 1$
But as $h_\theta(x) \to 0$; cost $\to \infty$
If $h_\theta(x) = 0$ but $y = 1$
we'll penalize learning algo
by a very large cost.

⇒ Simplified cost function and gradient descent :

Logistic Regression cost function

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

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

(Compressed way of writing cost fn)

$y = 0$ or 1 always

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

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

if $y = 0$: $\text{cost}(h_\theta(x), y) = -\log(1 - h_\theta(x))$

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

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

⇒ To fit parameter θ:

$\min_\theta J(\theta)$

To make a prediction given new x:

Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ $\quad P(y=1 \mid x;\theta)$

∴ Using gradient descent

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ } Repeat & (Simultaneously update all θ_j)

$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$; $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$

Algo looks identical to linear regression

★ Advanced optimization for Logistic Regression :

We have other algorithms too except Gradient Descent.

⇒ Conjugate gradient } Advantages : No need to manually pick α

⇒ BFGS often faster than GD.

⇒ L-BFGS } Disadvantage : More complex

Ex: $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$ $\quad J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$

$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$; $\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$

We can clearly see $J(\theta)$ is min when $\theta_1 \& \theta_2 = 5$

We can solve this on Octave too :

```
function [jVal, gradient] = costFunction(theta)
  jVal = (theta(1)-5)^2 + (theta(2)-5)^2;
  gradient = zeros(2,1);   (grad = [0 0])
  gradient(1) = 2*(theta(1)-5);   ← diff wrt θ1
  gradient(2) = 2*(theta(2)-5);   ←
```

setting my parameters

⇒ Use this advance code :

```
options = optimset('GradObj', 'on', 'MaxIter', '100');   % 100 iterations
initialTheta = zeros(2,1);   % initializing θ; i.e. initial guess
[optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, options);
```

fminunc: unconstraint; @costFunction: pointer to the cost function; initialTheta: initial θ; options: options which we had set

θ ∈ ℝ^d, d ≥ 2

optTheta = 5.00, 5.00

functionVal = 1.577e-030 ≈ 0

exitFlag = 1

% help fminunc % read this documentary

⇒ Basic framework for algorithm

theta = [θ0, θ1, ..., θn] → theta(1), theta(2), ..., theta(n+1)

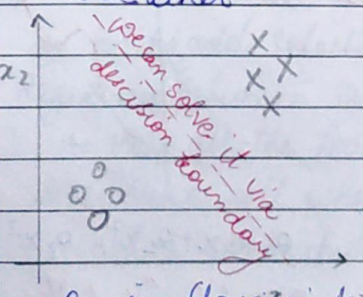
```
function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(θ)];
    gradient(1) = [code to compute ∂/∂θ0 J(θ)];
    gradient(2) = [code to compute ∂/∂θ1 J(θ)];
    :
    gradient(n+1) = [code to compute ∂/∂θn J(θ)];
```

⇒ Multiclass classification one vs all :

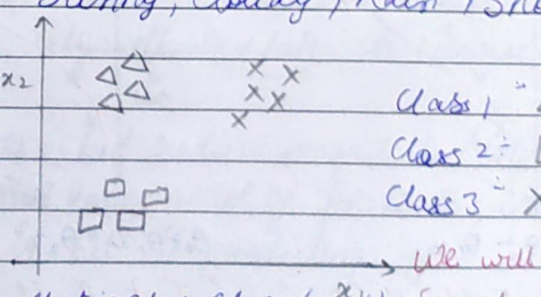
Eg Email foldering / tagging :- Work (y=1), Friends (y=2), Family (y=3), Hobby (y=4)

Medical diagram :- Not ill (y=1), Cold (y=2), Flu (y=3)

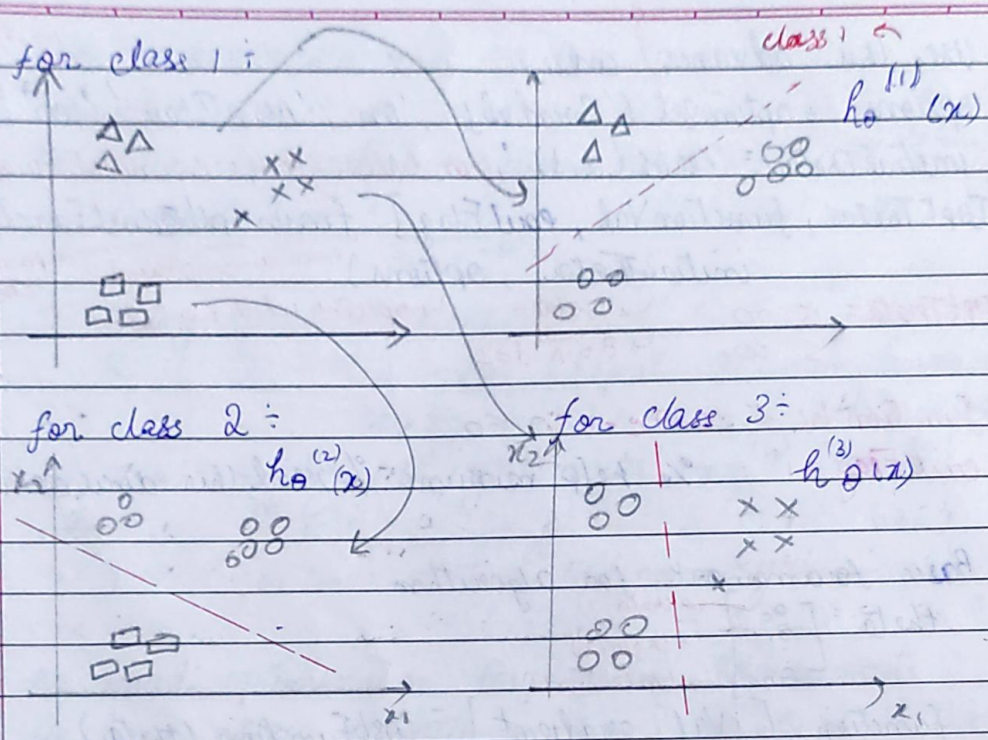
Weather :- Sunny (y=1), Cloudy (y=2), Rain (y=3), Snow (y=4)



Binary classification



Multi-class classification for class 1; rest two we will make them same; & treat as binary classification



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

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

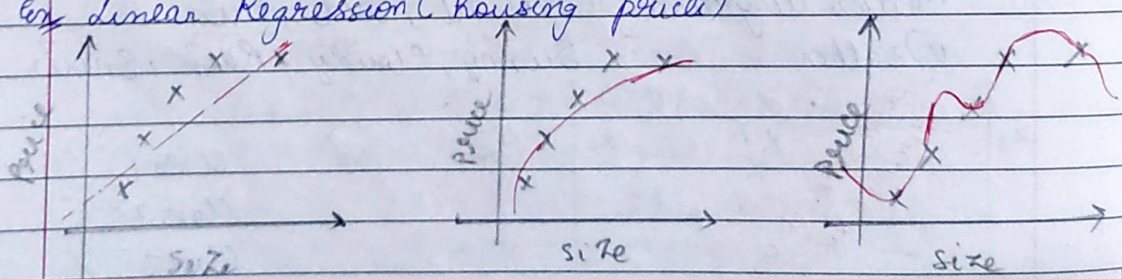
On a new input x, to make a prediction, pick the class i that maximizes

$\max_i h_\theta^{(i)}(x)$

★ **Regularization :**

The problem of overfitting :

Ex: Linear Regression (Housing prices)

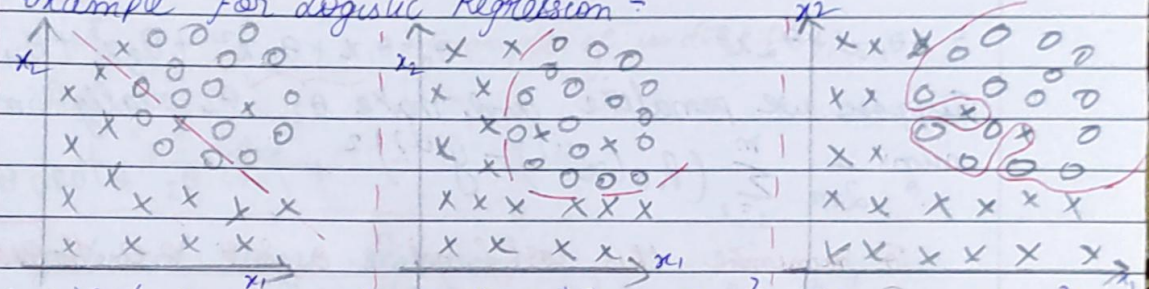


$\theta_0 + \theta_1 x$ | $\theta_0 + \theta_1 x + \theta_2 x^2$ | $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Underfit "High Bias" | "Just Right" | Overfit "High Variance"

⇒ Overfitting : If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples)

⇒ Example for logistic Regression :

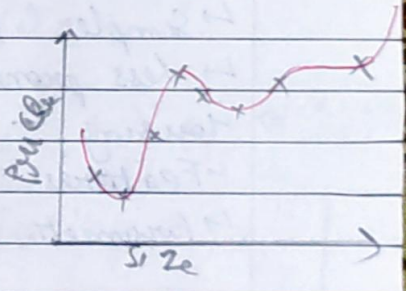


$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$ | $g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$ | $g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 \ldots)$

g = sigmoid funct^n | Underfit | Overfit

⇒ Addressing Overfitting :

x_1 = size of house
x_2 = No. of Bedrooms
x_3 = No. of floors
x_4 = age of house
x_5 = kitchen size
x_{100} = ...



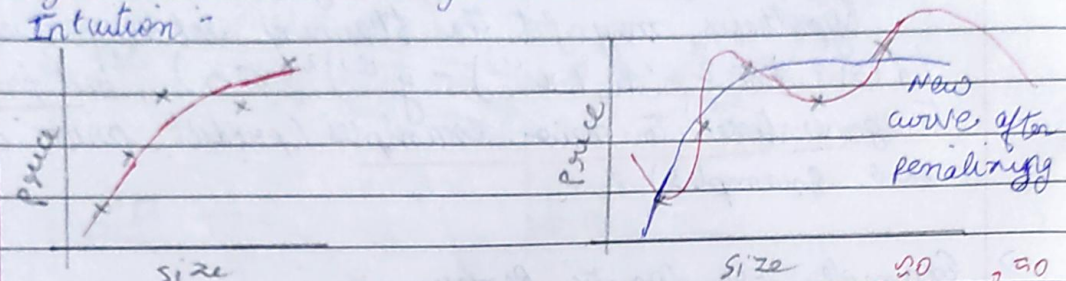
⇒ Options to resolve this overfitting :

i) Reduce number of features :
   ↳ Manually select which features to keep
   ↳ Model selection algorithm (later in course)

ii) Regularization :
   ↳ Keep all the features, but reduce magnitude/value of parameters θ_j
   ↳ Works well when we have a lot of features, each of which contributes a bit to predicting y.

⇒ Regularization - Cost function :

Intution :



$\theta + \theta_1 x + \theta_2 x^2$ $\qquad$ $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$ ($\theta_3 \approx 0$, $\theta_4 \approx 0$)

Suppose we penalize and make θ_3, θ_4 really small

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

To minimize this cost funct^n θ_3 & $\theta_4 \approx 0$ to reduce $J(\theta)$

So we are reducing θ_3 & θ_4 close to zero

⇒ Regularization :

↳ Small values for parameters $\theta_0, \theta_1 \cdots \theta_n$

↳ "Simpler" hypothesis $\quad [\theta_3 \& \theta_4 \approx 0]$

↳ less prone to overfitting

⇒ Housing :

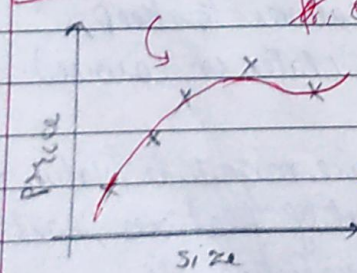
↳ Features : $x_1, x_2 \cdots x_{100}$

↳ Parameters : $\theta_0, \theta_1, \theta_2 \cdots \theta_{100}$ $\quad$ regularizat^n parameter

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

we don't take θ_0 into consideration

~~θ_0~~, $\theta_1, \theta_2 \cdots \theta_{100}$; θ_0 is small in practice



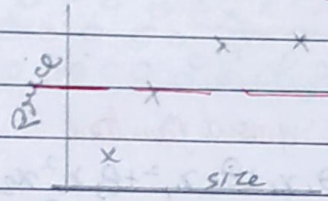
After using new cost function this graph will not be a complete quadratic function ; but it will be smother to the given dataset

⇒ What if λ is set to an extremely large value (say $\lambda = 10^{10}$)?

If this happens then we have to eliminate θ_j i.e. $\theta_1, \theta_2, \theta_3, \theta_4 \approx 0$

So remaining hypothesis will be $h_\theta(x) = \theta_0$

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



⇒ Regularized linear regression :

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

$\min_\theta J(\theta)$

Previously we were using Gradient descent

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

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\, x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \quad j = 1, 2, \ldots$$

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

$1 - \alpha\frac{\lambda}{m} < 1$ (may be 0.99) $\quad \theta_j \times 0.99$

⇒ Normal Equation

$$X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}_{m \times (n+1)} \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m$$

$\min_\theta J(\theta)$; $\frac{\partial}{\partial \theta_j} J(\theta) \overset{\text{set}}{\Rightarrow} 0$

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}_{(n+1)\times(n+1)}\right)^{-1} X^T y$$

Eg $n = 2$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

⇒ Non-invertibility :

Suppose $m \leq n$,

#Examples #features ; $\theta = (X^T X)^{-1} X^T y$

non-invertible / singular

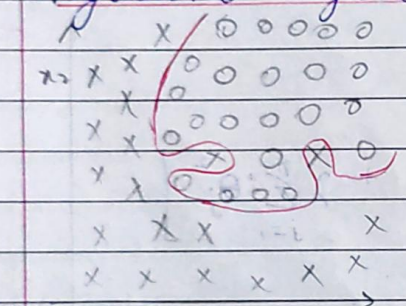
Use pinv function

If $\lambda > 0$

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

Invertible

⇒ Regularized logistic regression :- Sigmoid function



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

Cost function :-

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

New cost function

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

$\theta_1, \theta_2 \cdots \theta_n$

⇒ Gradient Descent :-

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \rightarrow j = 0, 1, 2 \cdots n$$

Not for $j = 0$

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

⇒ Advance optimization :- → fminunc (@costfunction)

function [jVal, gradient] = costFunction (theta)

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$ → theta(1), theta(2), theta(n+1)

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

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

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

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

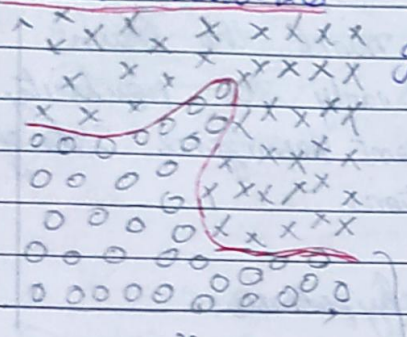
gradient(2) = [code to compute $\frac{\partial}{\partial \theta_1} J(\theta)$];

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

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

Week - 4

★ Neural Networks :



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 \dots)$

when x_i is large

x_1 = size

x_2 = #bedrooms

x_{100} = ...

$n = 100$

∴ There can be around 5000 features (quadratic combinations of x_i i.e $x_1^2, x_1 x_2, x_1 x_4, x^2, x_2 x_3 \dots$) $\approx O(\frac{n^2}{2})$

But if we use features consisting of $x_1^2, x_2^2, x_3^2, x_4^2 \dots x_{100}^2$ we will have 100 feature so can get shapes like ellipse But we can never get this → plot

⇒ If we use cubic feature $x_1 x_2 x_3$, $x_1^2 x_2$, $x_{10} x_{11} x_{77}$ ...

$O(n^3)$ ↝ 170,000 possible features

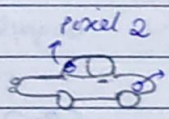
⇒ Computer Vision :

→ This is a car what humans see

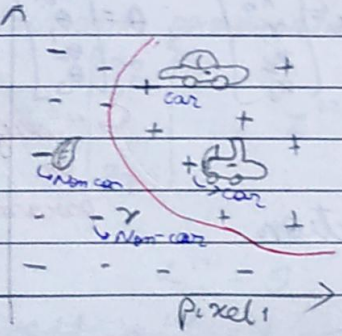
1	5	9	4	8
7	6	2	1	3
42	5	64	9	69

→ But the camera sees the specified that these are cars and these are not cars & then we give our testing set.

we give our training set where it is



⇒ → learning algorithm



50 × 50 pixel image → 2500 pixels

$n = 2500$ (7500 if RGB)

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

Quadratic features $(x_i \times x_j) \approx 3$ million features

* Neurons and brains :

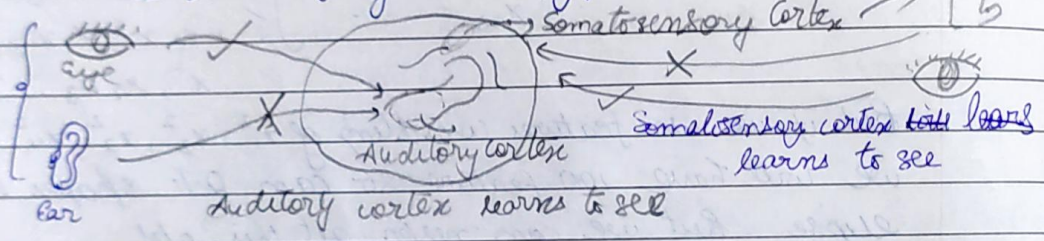
Neural Networks :

Origin → Algorithm that tryy to mimic the brain.

Very widely used in 80's and early 90's ; popularity diminished in late 90's. Recent resurgence : State of the art technique for many applications.

⇒ The "one learning algorithm" hypothesis

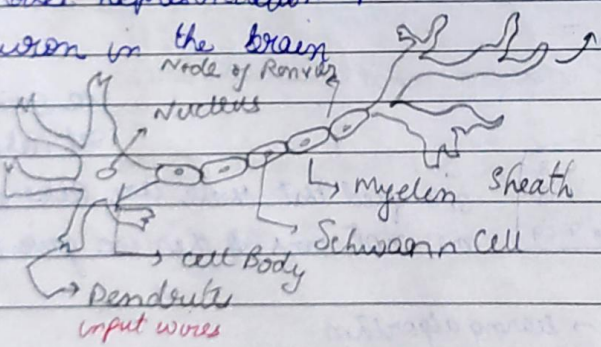
Neural Rewiring



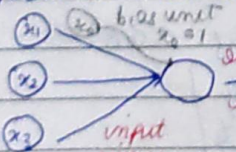
** See notes for Awesome Examples of Neural Networks

⇒ Model Representation I :

Neuron in the brain



Neuron model : Logistic unit



$h_\theta(x)$

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

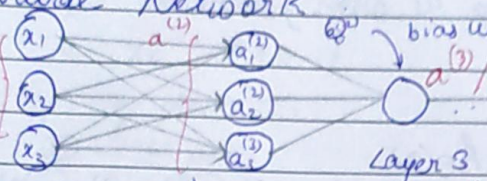
$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$; $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$

↳ "weights" or parameters

Sigmoid (logistic) activation function

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

⇒ Neural Network :



For graphical method

we can add $x_0 / a_0^{(1)}$ not that important

Layer 1 Input layer 3 units

Layer 2 Hidden layer 3 hidden units

Layer 3 output layer

$\rightarrow \Theta^{(1)} \in \mathbb{R}^{3\times 4}$ matrix

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

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

hidden layer values

$a_1^{(2)} = g(z_1^{(2)})$ $a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2^2 + \Theta_{13}^{(1)} x_3) \rightarrow z_1^{(2)}$

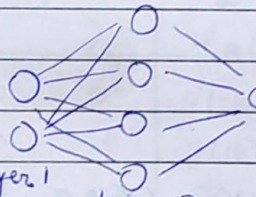
$a_2^{(2)} = g(z_2^{(2)})$ $a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_3 + \Theta_{23}^{(1)} x_4) \rightarrow z_2^{(2)}$

$a_3^{(2)} = g(z_3^{(2)})$ $a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_2 + \Theta_{32}^{(1)} x_3 + \Theta_{33}^{(1)} x_4) \rightarrow z_3^{(2)}$

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

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

Eg



$\therefore$ find dimens^n of $\Theta^{(1)}$

$s_j = 2$

$s_{j+1} = 4$

$\therefore$ Dimens^n $= (s_{j+1}) \times (s_j + 1)$ (+1 for bias node a_0)

$\Theta^{(1)} = 4 \times 3$ dimens^n

⇒ Forward propogation : Vectorized Implementation

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

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

3 dim Vector $\leftarrow a^{(2)} = g(z^{(2)}) \rightarrow \mathbb{R}^3$ dim. Vect

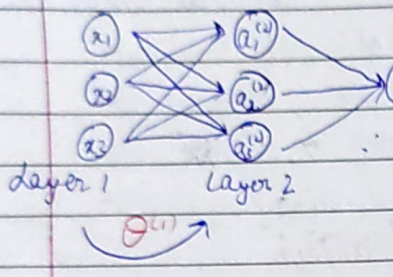
as $z^{(2)}$ is a product matrix of Θ & x.

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

Add $a_0^{(2)} = 1$ $\frown$ $\therefore a^{(2)}$ is a 4 dim feature vector

$z^{(3)} = \Theta^{(2)} a^{(2)}$, $h_\theta(x) = a^{(3)} = g(z^{(3)})$

⇒ Neural Network learning its own features :-



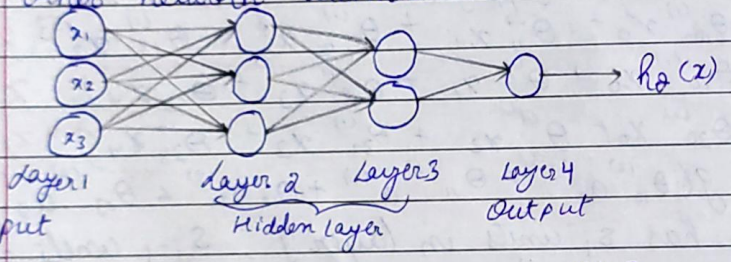
$\therefore h_\Theta(x) = g(\Theta_0^{(2)} \cdot a_0^{(2)} + \Theta_1^{(2)} \cdot a_1^{(2)} + \Theta_2^{(2)} a_2^{(2)} + \Theta_3^{(1)} \cdot a_3^{(1)})$

i.e $h_\Theta(x)$ depends on $a_1^{(2)}, a_2^{(2)}$ & $a_3^{(2)}$

And these parameters depends on x_1, x_2, x_3

We say that it is learning through its own features

⇒ Other network architectures :-



∴ How would you compute $a^{(2)}$?

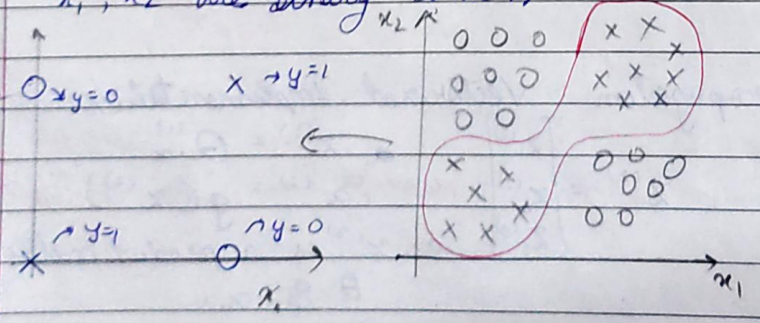
$\therefore a^{(2)} = g(z^{(2)})$ & $z^{(2)} = \Theta^{(1)} a^{(1)}$

⇒ Neural Network : Representations :-

Examples and Intuition I

Non-linear classification example : XOR / XNOR

x_1, x_2 are binary (0 or 1).



$y = x_1$ XOR x_2

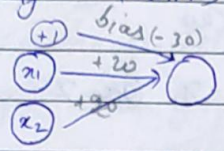
↳ x_1 XNOR x_2

i.e NOT (x_1 XOR x_2)

⇒ Simple example : AND

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

$y = x_1$ AND x_2

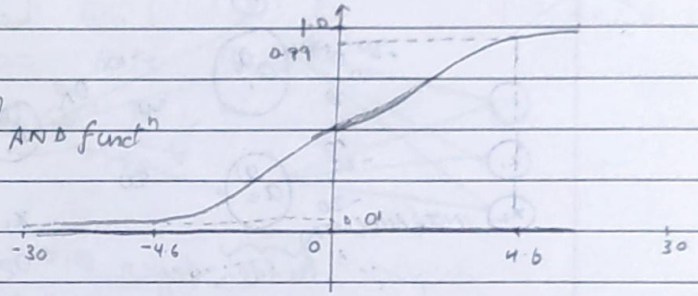


$\therefore h_\theta(x) = g(-30 + 20x_1 + 20x_2)$

$\theta_{10}^{(1)}$ $\theta_{11}^{(1)}$ $\theta_{12}^{(1)}$

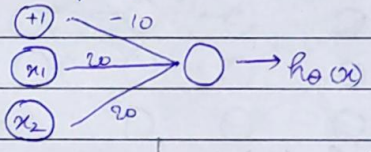
x_1	x_2	$h_\theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

AND funct^n



$h_\theta(x) \approx x_1$ AND x_2

⇒ Example : OR



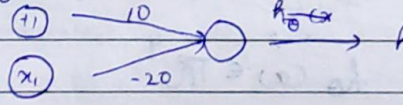
$h_\theta(x) = g(-10 + 20x_1 + 20x_2)$

x_1	x_2	$h_\theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(30) \approx 1$

OR funct^n

$h_\theta(x) \approx x_1$ OR x_2

⇒ Negation : NOT



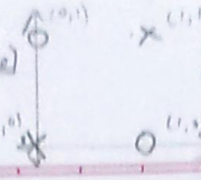
x_1	$h_\theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

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

(NOT x_1) AND (NOT x_2) = 1

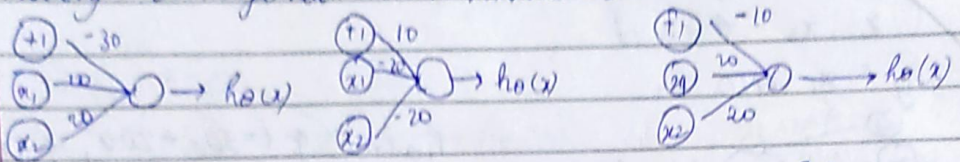
If and only if $x_1 = x_2 = 0$

- We can create complicated $h_\theta(x)$ [Non linear hypothesis] via Neural networks

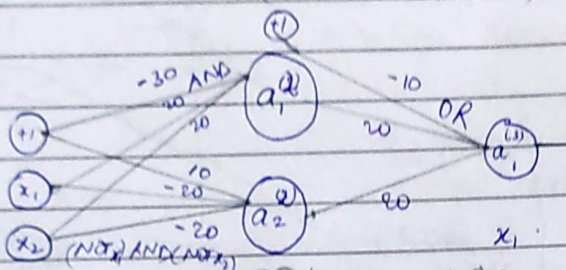


Page No. 46
Date / /

⇒ Putting it together: x_1 XNOR x_2

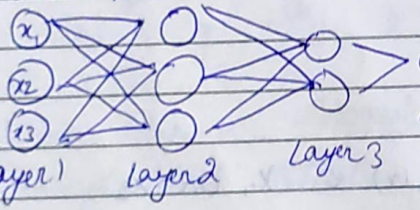


x_1 AND x_2 (NOT x_1) AND (NOT x_2) x_1 OR x_2

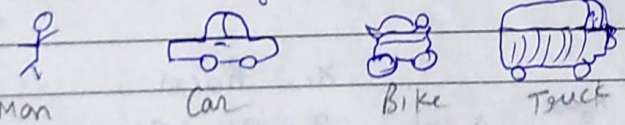


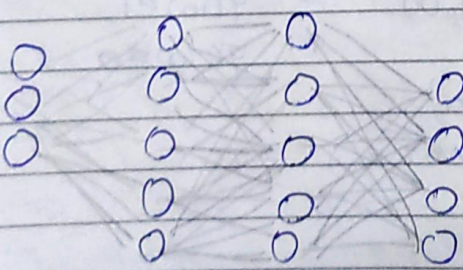
x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\theta(x)$	
(0,0) 0	0	0	1	→ 1	are made XNOR gate using diff gates.
(0,1) 0	1	0	0	→ 0	
(1,0) 1	0	0	0	→ 0	
(1,1) 1	1	1	0	1	

⇒ Neural Network Intuition



⇒ Multiclass Classification : Multiple output units one Vs all





Output will be a vector of 4 numbers

Output for man will be $h_\theta(x) \in \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\theta(x \in \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$

$h_\theta(x) \in \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$ Bike — Man — Car

⇒ Training set : $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \cdots (x^{(m)}, y^{(m)})$

$y^{(i)}$ one of $\left\{\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}\right\}$

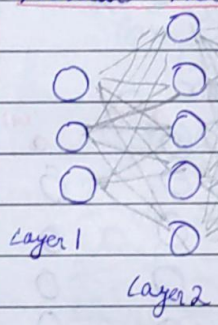
Previously $y \in \{1, 2, 3\}$

man car Bike Truck

$(x^{(i)}, y^{(i)})$ $\quad h_\theta(x^{(i)}) \in y^{(i)}$ (IR)

Week - 5

* Neural Networks - Cost function :



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

↳ L = total no. of layers in network

$L = 4$

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

$s_1 = 3$; $s_2 = 5$; $s_4 = s_L = 4$

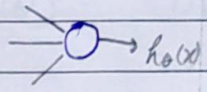
Binary classification

$y = 0$ or 1

1 output unit

$h_\theta(x) \in IR$

$s_L = 1$; $K = 1$



multi Class - Classificat^n (K classes)

$y \in IR^K$ Eg $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

Man Car Bike Truck

K output units

$h_\theta(x) \in IR^K$

$s_2 = K \quad (K \geq 3)$

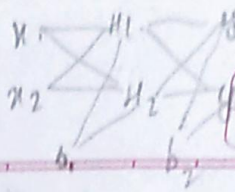
⇒ cost function :

Logistic Regression :

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

$H_1 = x_1 w_1 + x_2 w_2 + b_1$

$\xi = \frac{1}{1+e^{-z}} \Rightarrow x_1 = \frac{1}{1+e^{-z}}$



Page No. 40
Date / /

Neural Networks :

$R_\theta(x) \in \mathbb{R}^K$; $(h_\theta(x))_i$ = i^{th} Output

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\theta(x^{(i)})\right)_k + (1 - y_k^{(i)}) \log\left(1 - h_\theta(x^{(i)})\right)_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}} \left(\theta_{ji}^{(l)}\right)^2$$

Sum of all y_k's

$\theta_{ij}^{(l)} \in \mathbb{R}$

⇒ Backpropogation algorithm :

So we need to compute $J(\theta)$ & $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$

Ex If we had only 1 training example :

(x, y)

∴ forward propogation

$a^{(1)} = x$

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

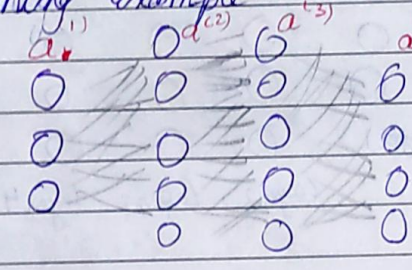
$a^{(2)} = g(z^{(2)})$ [add $a_0^{(2)}$]

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

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

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

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



⇒ Compute gradient using Backpropogation algorithm :

Intuition : $\delta_j^{(l)}$ = "error" of node j in layer l.

For each output unit (layer $L = 4$)

$\delta_j^{(4)} = a_j^{(4)} - y_j$ or $\delta^{(4)} = a^{(4)} - y$

↳ $(h_\theta(x))_j$

$\delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$ → $a^{(3)} .* (1 - a^{(3)})$

$\delta^{(2)} = (\theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$ → $a^{(2)} .* (1 - a^{(2)})$

No $\delta^{(1)}$ as these are features so no errors in it

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} \cdot \delta_i^{(l+1)} \quad \text{[ignoring } \lambda \text{; if } \lambda = 0\text{]}$$

⇒ Back propagation algorithm :

Training set $\{(x^{(1)}, y^{(1)}), \cdots (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{l} = 0$ {for all l, i, j}

used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\theta)$

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3 \cdots L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}$ $\delta^{(L-2)} \cdots \delta^{(2)}$ ~~$\delta^{(1)}$~~

$$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

Vectorized implementation

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

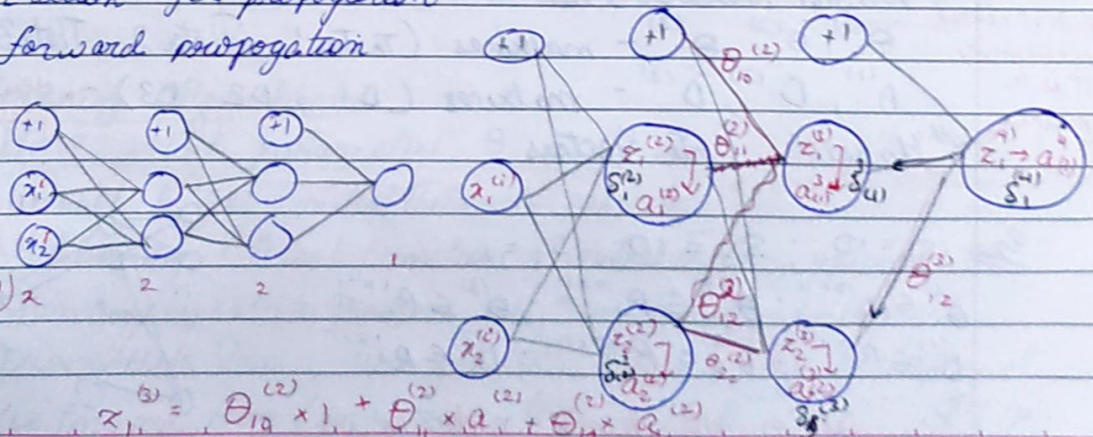
$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \quad \text{if } j \neq 0$$

$$D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{if } j = 0$$

$$\therefore \frac{\partial}{\partial \Theta_{ij}^{l}} J(\theta) = D_{ij}^{(l)}$$

⇒ Intuition for propogation :

forward propogation



$$z_1^{(3)} = \Theta_{10}^{(2)} \times 1 + \Theta_{11}^{(2)} \times a_1^{(2)} + \Theta_{12}^{(2)} \times a_1^{(2)}$$

Now for single example $x^{(i)}, y^{(i)}$, the case of 1 output unit and ignoring regularization ($\lambda = 0$)

$$\text{cost}(i) = y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log h_\theta(x^{(i)})$$

This cost(i) $\approx (h_\theta(x^{(i)}) - y^{(i)})^2$

i.e how well is the network doing on example i?

for Backward propogation

$\delta_j^{(l)}$ = "Error" of cost for $a_j^{(l)}$ (unit j in layer l).

Formally $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(i)$ (for $j \geq 0$), where

$$\text{cost}(i) = y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log h_\theta(x^{(i)})$$

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

product - value predicted

$$\delta_2^{(2)} = \Theta_{12}^{(2)} \cdot \delta_1^{(3)} + \Theta_{22}^{(2)} \cdot \delta_2^{(3)}$$

$$\delta_2^{(3)} = \Theta_{12}^{(3)} \cdot \delta_1^{(4)}$$

* Backpropogation implementation - Unrolling parameters: Advanced Optimization:

function [jVal, gradient] = costFunction (theta)

↳ $\mathbb{R}^{n+1}$ ↳ $\mathbb{R}^{n+1}$ (vectors)

In logistic regression theta was a vector but if neural networks it is a matrix

optTheta = fminunc (@costFunction, initialTheta, options)

Neural Networks ($L = 4$):

$\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ - matrices (Theta1, Theta2, Theta3) → as L=4

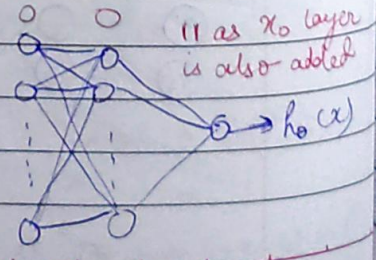
$D^{(1)}, D^{(2)}, D^{(3)}$ - matrices (D1, D2, D3) → New grad term

"Unroll" into vectors

Ex $s_1 = 10$; $s_2 = 10$; $s_3 = 1$

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

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



Let convert it into matrices in Octave

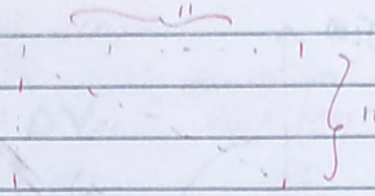
$\theta^{(1)}$ $\theta^{(2)}$ $\theta^{(3)}$

thetaVec = [Theta1(:); Theta2(:); Theta3(:)];
DVec = [D1(:); D2(:); D3(:)];

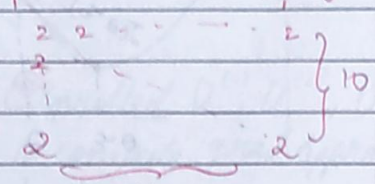
to give values & reshape θ...

Theta1 = reshape(thetaVec(1:110), 10, 11);
Theta2 = reshape(thetaVec(111:220), 10, 11);
Theta3 = reshape(thetaVec(221:231), 1, 11);

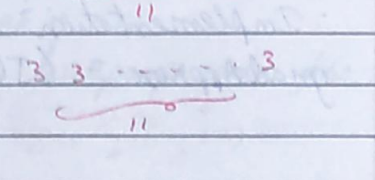
Let's code in octave
theta1 = ones(10, 11) → Theta1



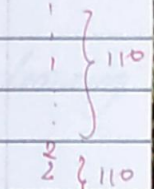
theta2 = 2*ones(10, 11) Theta2



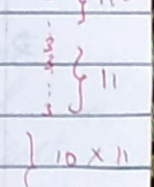
theta3 = 3*ones(1, 11) Theta3



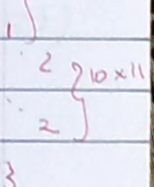
* % To unroll & make thetaVec
thetaVec = [Theta1(:); Theta2(:); Theta3(:)] thetaVec =



% Let's extract or reshape theta vect to get our theta1
Theta1 = reshape(thetaVec(1:110), 10, 11) Theta1 =



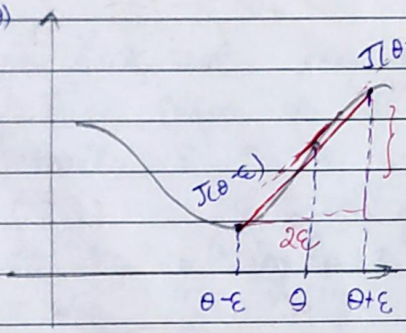
Theta2 = reshape(thetaVec(111:220), 10, 11)
Theta2 =



Theta3 = reshape(thetaVec(221:231), 1, 11) Theta3 = 3 3 ... 3
(11)

⇒ Learning Algorithm:
Have initial parameters $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$ → we will pass $\theta^{(1)}$ $\theta^{(2)}$ $\theta^{(3)}$ into long matrix initialTheta
Unroll to get initialTheta to pass to
fminunc(@costFunction, initialTheta, options)
function [jVal, gradientVec] = costFunction(thetaVec)
we will extract $\theta^{(1)}$ $\theta^{(2)}$ $\theta^{(3)}$ use reqd
From thetaVec we get $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$ by reshaping it
Use forward prop / back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ & $J(\theta)$
Unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get gradientVec → then we will feed it

⇒ Gradient checking : What was the need of gradient checking because when we use foward / back propogation like for δ calculation there are chances of bug or errors. So we use grad checking Numerical estimation of gradents



$\therefore \frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$ $\varepsilon \approx 10^{-4}$

This is a double sided approximat^n

We used to study first principle that was not that apron

∴ Implementation :

```
gradApprox = (J(theta + EPSILON) - J(theta - EPSILON)) /
                                    (2 * EPSILON)
```

⇒ Parameter Vector θ :

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

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

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

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

⋮

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

Let's implement it in octave :

for i = 1:n,
  thetaPlus = theta;
  thetaPlus(i) = thetaPlus(i) + EPSILON;
  thetaMinus = theta;
  thetaMinus(i) = thetaMinus(i) - EPSILON;
  gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2*EPSILON);
end;

$\theta_1, \theta_2, \theta_i + \varepsilon, \theta_n$

$\frac{\partial}{\partial \theta_i} J(\theta)$

Now check that gradApprox ≈ DVec (this vec got from Back propogation)

⇒ Implementation Note:
i) Implement backprop to compute DVec (unrolled $D^{(1)}, D^{(2)}, D^{(3)}$).
ii) Implement numerical gradient check to compute gradApprox.
iii) Make sure they give similar values
iv) Turn off gradient checking. Using backprop code for learning

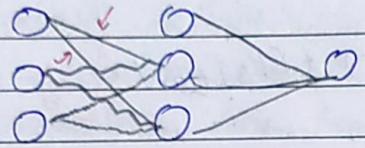
→ DVec
$\delta^{(4)}, \delta^{(3)}, \delta^{(2)}$

⇒ Important:
i) Be sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent (or in the inner loop of costFunction(...)) your code will be very slow.

⇒ Random initialization:
Initial value of θ
For gradient descent and advanced optimization method, need initial value for θ.
optTheta = fminunc(@costFunction, initialTheta, options)
So can we consider gradient Descent
Set initialTheta = zeros(n,1)?
✗ Dont do that in neural networks
this works fine for regressions but not here

If we initialize θ as zero

$\theta_{ij}^{(l)} = 0$ for all i, j, l



$a_1^{(2)} = a_2^{(2)}$
Also $\delta_1^{(2)} = \delta_2^{(2)}$

$\therefore \frac{\partial}{\partial \theta_{01}^{(1)}} J(\theta) = \frac{\partial}{\partial \theta_{02}^{(1)}} J(\theta)$ $\quad \therefore \theta_{01}^{(1)} = \theta_{02}^{(1)}$ Non Zero maybe But same some constant C

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

$a_1^{(2)} = a_2^{(2)}$ i.e all hidden units are same

Re-initialize, re-run

$\therefore$ We use random initialization : Symmetry breaking

Initialize each $\theta_{ij}^{(l)}$ to a random value in $[-\varepsilon, \varepsilon]$

(i.e $-\varepsilon \leq \theta_{ij}^{(l)} \leq \varepsilon$) b/w $-\varepsilon$ to ε

Eg:

random 10x11 matrix (b/w 0 & 1) $\quad [-\varepsilon, \varepsilon]$

```
Theta1 = rand(10,11) *(2*INIT_EPSILON) - INIT_EPSILON;
Theta2 = rand(1,11) *(2*INIT_EPSILON) - INIT_EPSILON;
```

⇒ Putting it together :

Pick a neural architecture (connectivity pattern between neurons)

3 5 4 — Hidden layer

3 5 5 4 — Hidden layers

3 5 5 5 4 — Hidden layer

No. of input units : Dimensions of features $x^{(i)}$

No. of output units : Number of classes

$y \in \{1, 2, 3 \cdots 10\}$, $y = \begin{bmatrix} \vdots \\ \cdot \end{bmatrix}$ or $\begin{bmatrix} \cdot \\ \vdots \end{bmatrix}$

Reasonable default : 1 hidden layer, or if > 1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

=> Let's train a neural network.

i) Randomly initialize weights

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

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

iv) Implement Back prop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$.

for $i = 1:m$, { $(x^{(1)}, y^{(1)})$ $(x^{(2)}, y^{(2)})$ . . . $(x^{(m)}, y^{(m)})$

perform forward propagation and back propagation using example $(x^{(i)}, y^{(i)})$

(get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, 3 \cdots L$).

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

. . . }

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

v) Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backprop vs. using numerical estimate of gradient of $J(\Theta)$. Then disable gradient checking code.

vi) Use gradient descent or advance optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ. ↳ $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Non convex $J(\Theta)$ not a big problem

* Neural networks on graph
* Autonomous Driving

Week - 6

* Advice for applying Machine learning

Debugging a learning algorithm :
Suppose you have implemented regularized linear regression to predict housing prices.

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

However when your test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should you try next?

↳ Get more training examples
↳ Try smaller sets of features.
↳ Try getting additional features
↳ Try adding polynomial features (x_1^2, x_2^2, x_1x_2 etc)
↳ Try decreasing λ
↳ Try increasing λ

⇒ Machine learning Diagnostics :-
Diagnostics : 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.

⇒ Evaluation of hypothesis :-
We know about overfitting of data that is one issue as the overfit data completely plots the graph but fails to predict test data i.e not in training set.

⇒ Evaluating your hypothesis :

Dataset

Size	Price		
2104	400	Training set (70%)	$(x^{(1)}, y^{(1)})$
1600	330		$(x^{(2)}, y^{(2)})$
2400	369		⋮
1416	232		
3000	540		
1985	300		
1534	315		$(x^{(m)}, y^{(m)})$
1427	199	Test set (30%)	$(x^{(1)}_{test}, y^{(1)}_{test})$ m_{test} = no. of test examples
1380	211		$(x^{(2)}_{test}, y^{(2)}_{test})$
1490	243		$(x^{(m_{test})}_{test}, y^{(m_{test})}_{test})$

⇒ Training/testing procedure for linear regression :

↳ Learn parameter θ from training data (minimize training error $J(\theta)$) ↳ 70%.

↳ Compute test set error : $J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x^{(i)}_{test}) - y^{(i)}_{test})^2$ ↳ 30%.

for logistic regression :

⇒ $J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} y^{(i)}_{test} \log h_\theta(x^{(i)}_{test}) + (1 - y^{(i)}_{test}) \log h_\theta(x^{(i)}_{test})$

Misclassificat^n error (0/1 misclassification error)

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

Test error $= \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\theta(x^{(i)}_{test}), y^{(i)}_{test})$

we use $J_{cv}(\theta)$ here and then put it for J_{test}.

⇒ Model Selection : d = degree of polynomial → $J_{test}(\theta^{(d)})$

$d=1$ $h_\theta(x) = \theta_0 + \theta_1 x$ → $\theta^{(1)}$ parameters i get by fitting this model

$d=2$ $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$ → $\theta^{(2)}$ → $J_{test}(\theta^{(2)})$

⋮ ⋮

$d=10$ $h_\theta(x) = \theta_0 + \theta_1 x - - - - - \theta_{10} x^{10}$ → $\theta^{(10)}$ → $J_{test}(\theta^{(10)})$

if any $J_{test}(\theta^{(i)})$ is min we optimistically estimate of generalisat^n error

⇒ Evaluating your hypothesis :

	Size	price	
	2104	400	Training set $(x^{(i)}, y^{(i)})$
	1600	350	
60%	2400	369	
	1416	232	
	3000	540	
	1985	300	$(x^{(m)}, y^{(m)})$
	1534	315	Cross validat^n set (CV) $(x_{cv}^{(i)}, y_{cv}^{(i)})$
20%	1427	199	$(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$
	1380	212	Test Set
20%	1494	243	$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

m_{cv} = no. of CV examples $(x_{cv}^{(i)}, y_{cv}^{(i)})$

⇒ Train / validat^n / test error :

Training Error :

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

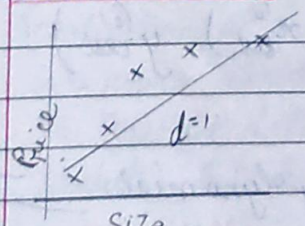
Cross Validation Error :

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

Test Error

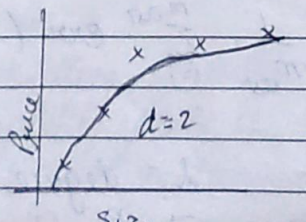
$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^{(i)} - y_{test}^{(i)})^2$$

⇒ Bias / Variance



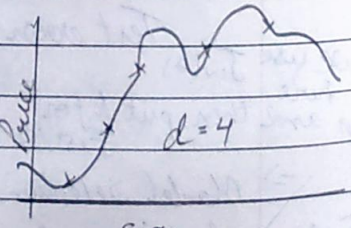
$\theta_0 + \theta_1 x$

High Bias (underfit)



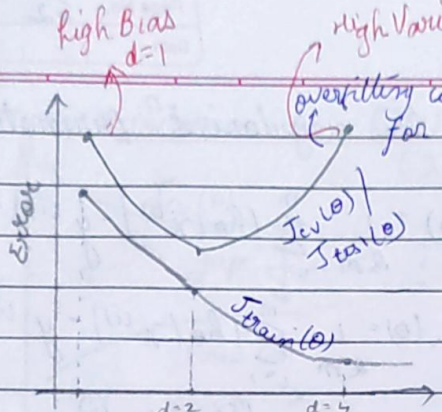
$\theta_0 + \theta_1 x + \theta_2 x^2$

"Just Right"



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

High Variance (overfit)



Training Error :

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

CV error :

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Bias (underfit) : $J_{train}(\theta)$ will be high
$J_{cv}(\theta) \approx J_{train}(\theta)$

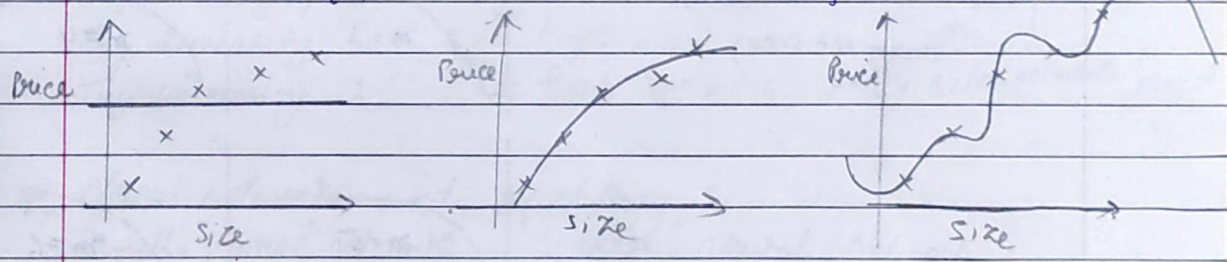
Variance (overfit) : $J_{train}(\theta)$ will be low
$J_{cv}(\theta) \gg J_{train}(\theta)$

we are talking about Errors.

⇒ Impact of Regularization on variance & Bias :

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

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



large $\lambda \approx 1000$
High Bias (underfit)
$\theta_1 \approx 0, \theta_2 \approx 0 \cdots$
$h_\theta(x) \approx \theta_0$

Intermediate λ
"Just right"

Small $\lambda \approx 0$
High Variance (overfit)
∴ It will follow 4 degree polynomial

∴ How to choose the regularization parameter λ ?

we can try diff values of λ :

Say $\lambda = 0$
$\lambda = 0.01$
$\lambda = 0.02$
$\lambda = 10$

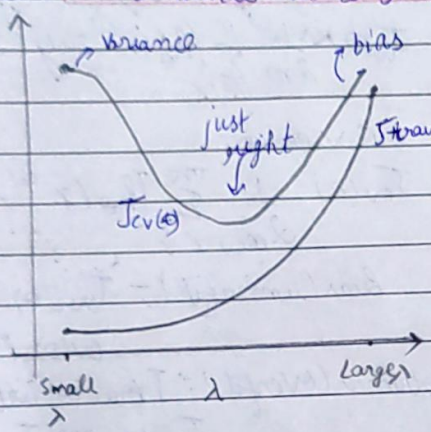
$\rightarrow \min_\theta J(\theta)$

$\rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$
$\rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$
$\rightarrow \theta^{(3)}$
$\rightarrow (\theta^{(12)})\ J_{cv}(\theta^{(12)})$

$\rightarrow J_{cv}(\theta^{(5)})$

Pick (say) $\theta^{(5)}$. Test Error = $J_{test}(\theta^{(5)})$

⇒ Bias / Variance as a funct^n of the regularizat^n parameter λ :



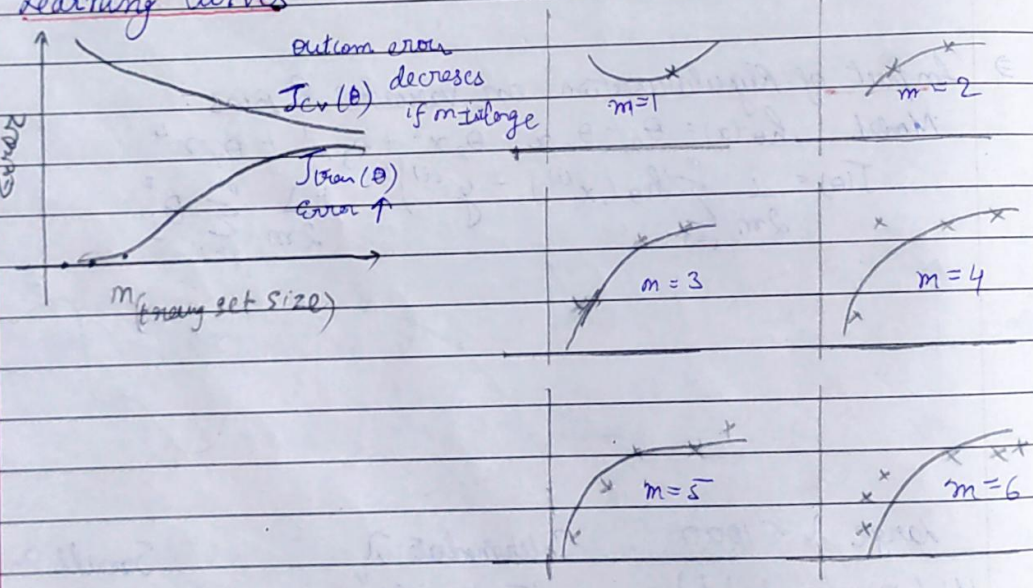
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

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

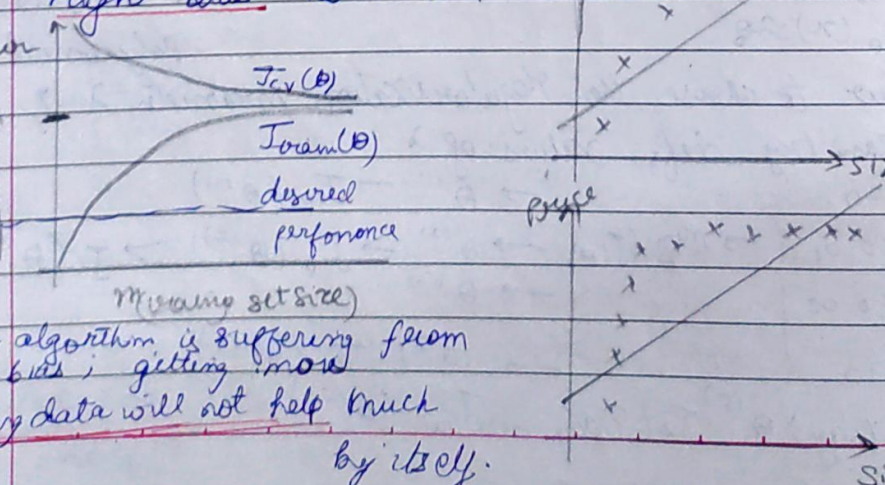
$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}} h_\theta\left(x_{cv}^{(i)}\right) - y_{cv}^{(i)}\Big)^2$$

⇒ Learning Curves :-

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

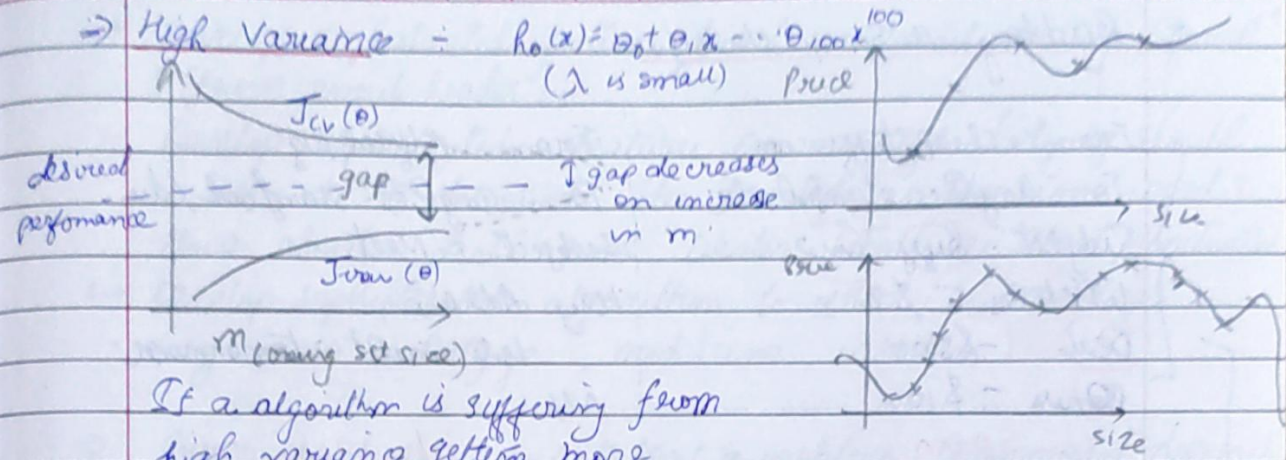


⇒ High Bias : $h_\theta(x) = \theta_0 + \theta_1 x$



If a algorithm is suffering from high bias, getting more training data will not help much by itself.

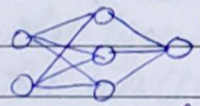
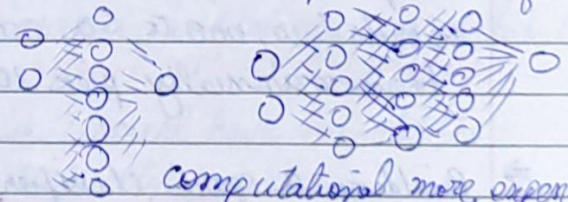
⇒ High Variance : $h_\theta(x) = \theta_0 + \theta_1 x + \cdots \theta_{100} x^{100}$ (λ is small)



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

⇒ Debugging a learning algorithm :
Get more training Examples → fixes high variance
Try smaller sets of features → fixes high variance
Try getting additional features → fixes high bias
Try adding polynomial features ($x_1^2, x_2^2, x_1 x_2$ etc) → fixes high bias
Try decreasing λ → fixes high bias
Try increasing λ → fixes high variance for a better accurate model

⇒ Neural networks and overfitting :

"Small" neural network (fewer parameters; more prone to underfitting)	"Large" Neural Network (more parameters; more prone to overfitting)
	
computational cheaper	computational more expensive. Use regularization (λ) to address overfitting.

* Building a Spam classifier:

From: cheapstuffs.com	From: Alfred Ng
To: ang@cs.stanford.edu	To: ang@cs.stanford.edu
Subject: Buy now	Subject: Meet
Watches - $2	Hey Andrew,
Deal - $500	Let's meet tomorrow.
Bonus - $100	Alf
Spam	Non-Spam

⇒ Build a 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, andrew, now ....

Now $x = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ \vdots \end{bmatrix}$ andrew, buy, deal, discount, now (Alphabetical order)

$x \in \mathbb{R}^{100}$

$x_j = 1$ if word j appears in email; 0 otherwise

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

⇒ Building a spam classifier

How to spend your time to make it have low error?

↳ collect lots of data

↳ Eg. "honeypot" project

↳ Develop sophisticated features based on email routing information (from email header).

↳ Develop sophisticated features based on email routing informat" (from email header).
↳ Develop sophisticated features for message body eg should "discount" & "discounts" be treated as the same word? How about "deal" and "Dealer"? Features about punctuation.
↳ Develop sophisticated algorithm to detect misspellings (e.g. ~~mor~~ mOrtgage, med!cine, w4tches)

⇒ Error Analysis: How to start a problem; 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 example it is making errors on.

⇒ Error Analysis:
m_{cv} = 500 examples in cross validation set.
Algorithm misclassifies 100 emails.
Manually examine the 100 errors, and categorize them based on:
i) What type of email it is → pharma, replica, steal passwords ...
ii) What cues (features) you think would have helped the algorithm classify them correctly

Pharma: 12
Replica / fake: 4
Steal password: 53
Other: 31

↳ Deliberate misspellings: 5 (mOrgage, med!cine, etc)
↳ Unusual email routing: 16
↳ Unusual (spamming) punctuat"(say!!) 32

⇒ The importance of numerical evaluation :
↳ Should discount / discounts/ discounted / discounting be treated as the same word?
↳ Can use 'stemming' software (Eg: "Porter stemmer")
used in NLP (Natural language processing) software "stemming"
can be sometimes difficult to differentiate
universe / university same starting initials
error analysis may not be helpful for deciding if this is likely to improve performance. Only solution is to try it and see if it works.
Need numerical evaluation (e.g cross validation error) of algorithm's performance with and without stemming.
Without stemming 5% error
With stemming 3% error
Distinguish upper v/s lower case (Mom / mom) 3.2% error

⇒ Error metrices of skewed classes :
Cancer classification example :
Train logistic regression model $h_\theta(x)$. { y=1 if cancer, y=0 otherwise
Find that you got 1% error on test set
(99% correct diagnoses)
Only 0.50% of patients have cancer → skewed classes
Now 1% error is huge
function y = predictCancer(x)
y=0 % ignore x → This will give only 0.5% error.
return

⇒ Precision / Recall :
y=1 in presence of rare class that we want to detect

	Actual Class: 1	Actual Class: 0
Predicted class: 1	True positive	False positive
Predicted class: 0	False Negative	True Negative

"Precision" :- [of all patients where we predicted y=1, what fraction actually has cancer.

True positives / #predicted pos = True positives / True pos + False pos

⇒ Recall (Of all patients that actually have cancer, what fraction did we correctly detect as having cancer?)

$$\text{Recall} = \frac{\text{True positives}}{\#\text{actual positives}} = \frac{\text{True positives}}{\text{True pos} + \text{False negatives}}$$

if our model predicts $y=0$ always $\therefore$ recall $= 0$

Model with high precision & Recall is a good model

⇒ Trading of precision and recall :

logistic regression : $0 \le h_\theta(x) \le 1$

Predict 1 if $h_\theta(x) \ge 0.5$ 0.7 } High precision, lower recall

Predict 0 if $h_\theta(x) < 0.5$ 0.7 }

i) Suppose we want to predict $y=1$ (cancer) only if very confident.

ii) Suppose we want to avoid missing too many cases of cancer (avoid false negatives).

∴ we can keep the threshold low

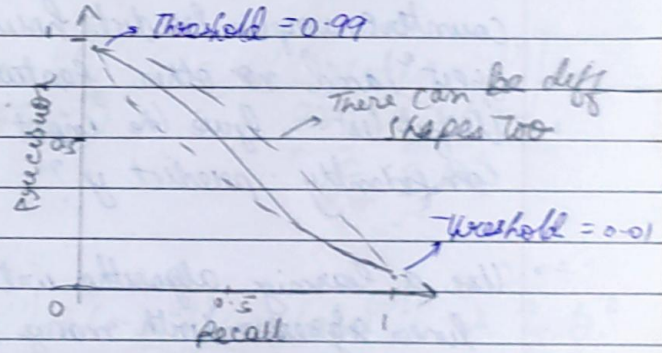
1 $h_\theta(x) \ge 0.5$ 0.3 } higher recall, lower precision

0 $h_\theta(x) < 0.5$ 0.3 }

More generally predict 1 if $h_\theta(x) \ge$ threshold.

$$\text{Precision} = \frac{\text{true positives}}{\text{No. of predicted positives}}$$

$$\text{Recall} = \frac{\text{True positives}}{\text{No. of actual positives}}$$



⇒ F Score (F Score) :

How to compare precision / recall numbers?

	Precision (P)	Recall (R)	Average	F Score
Algorithm 1	0.5	0.4	0.45	0.444
Algorithm 2	0.7	0.1	0.4	0.175
Algorithm 3	0.02	Predict $y=1$ all the time 1.0	0.51	0.0392

$$\text{F Score} = \frac{2PR}{P+R}$$

if $P=0$ or $R=0$ $\rightarrow$ F-Score $=0$

$P=1$ or $R=1$ $\rightarrow$ F-Score $=1$

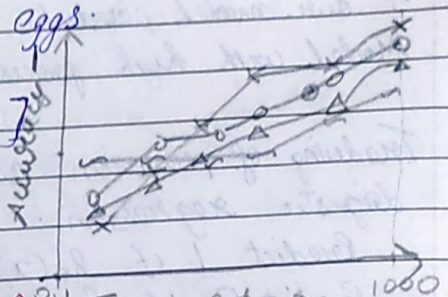
⇒ Data for machine learning :-

Design a high accuracy learning system

Eg: Classify between confusable words.
{to, two, too}, {then, than}

For breakfast I ate ~~two~~ eggs.

Algorithms :-
↳ Perception ○ [Logistic Regression]
↳ Winnow ×
↳ Memory-based △
↳ Naive Bayes ▪▪▪



"It's not who has the best algorithm that wins. It's who has the most data wins".

⇒ Large data rationale :-

↳ Assume feature $x \in R^{n+1}$ has sufficient information to predict y accurately.

↳ Eg: For breakfast I ate ~~two~~ eggs. ~~to~~ ~~tw~~

Counterexample :- Predict housing price from only size ($feet^2$) and no other features.

↳ Useful test :- Given the input x, can a human expert confidently predict y?

⇒ Use a learning algorithm with many parameters (eg. logistic/linear regression with many features, neural network with many hidden units). Low bias Algorithm

(Training error) $J_{train}(\theta)$ will be small

Use a very large training set (unlikely to overfit) Low Variance

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

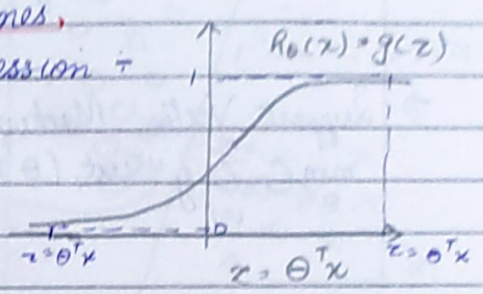
$\therefore J_{test}(\theta)$ will be small

Week - 7

Support Vector Machines.

⇒ Alternate view of logistic regression :

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



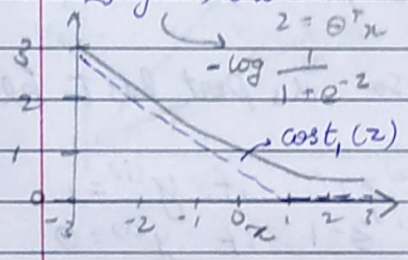
if $y = 1$; we want $h_\theta(x) \approx 1$
& $\theta^T x >> 0$

if $y = 0$; we want $h_\theta(x) \approx 0$
& $\theta^T x << 0$

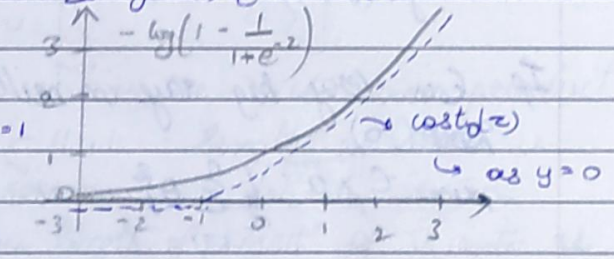
⇒ Cost of example : $-(y \log h_\theta(x) + (1-y) \log(1 - h_\theta(x)))$

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

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



& $y = 0$; (want $\theta^T x << 0$)



Logistic Regression :

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \underbrace{(-\log h_\theta(x^{(i)}))}_{\text{cost}_1(\theta^T x^{(i)})} + (1-y^{(i)}) \underbrace{(-\log(1-h_\theta(x^{(i)})))}_{\text{cost}_0(\theta^T x^{(i)})}\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

Support Vector Machine :

$$\min_\theta \cancel{\frac{1}{m}} \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) + \frac{\lambda}{2m}\sum_{i=0}^{n} \theta_j^2$$

constants can be removed ∧

Ex $\min_u K[(u-5)^2 + 1] \to u = 5$ ∴ No use of K.

$A + \lambda B \to CA + B$; Different Methods of parameterization

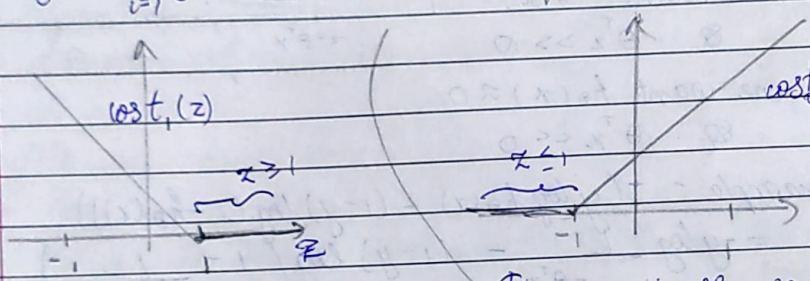
SVM hypothesis : $\min_\theta C \sum_{i=1}^{m}\left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n} \theta_j^2$

C A B

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

⇒ Support Vector Machine :

$$\min_\theta C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$



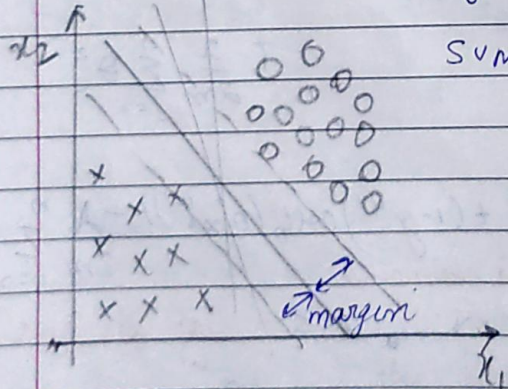
If $y=1$, $\theta^T x \geq 1$ (not just ≥ 0)

If $y=0$, we want $\theta^T x \leq -1$ (not just < 0)

If C is very big say in millions so this part has to be Zero(0)

$$\min C \times 0 + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2 \quad \text{s.t.} \quad \theta^T x^{(i)} \geq 1 \text{ if } y^{(i)} = 1$$
$$\theta^T x^{(i)} \leq -1 \text{ if } y^{(i)} = 0$$

⇒ SVM Decision Boundary : Linear Separable Case :



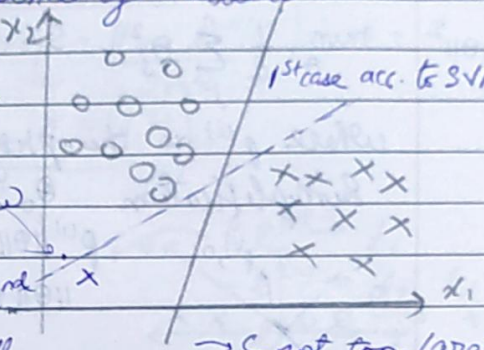
SVM is a large margin classifier.

Page No. 61
Date

⇒ Large margin classifier in presence of outliers :

Case 1st when we have clustered o & x

Case 2nd when there is an (x) so our new SVM will be 2nd line if C is very large.



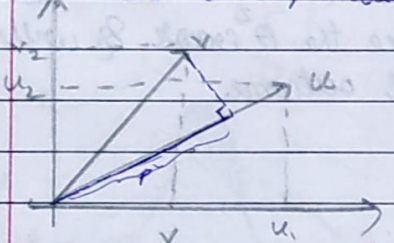
$C \to \frac{1}{\lambda}$ i.e λ is very small → C not too large

If C is not large we can ignore these small outliers.

⇒ Maths behind SVM (optional but not for me lol) :

Vector inner product :

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad ; \quad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \quad ; \quad u^T = [u_1 \; u_2]$$



$u^T v$ is the vector inner product

$\|u\|$ = length of vector u = $\sqrt{u_1^2 + u_2^2} \in \mathbb{R}$

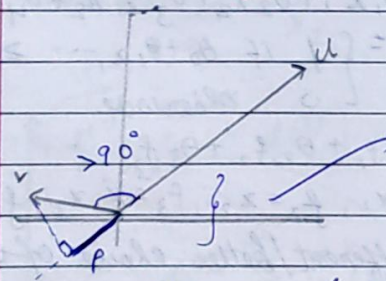
norms

Signed ← p is length of projection of v onto u

it can be (+) or (−)

$$u^T v = p \cdot \|u\| = v^T u = u_1 v_1 + u_2 v_2$$

Vector Inner Product



$u^T v = p \cdot \|u\|$ But here $p < 0$

⇒ SVM decision boundary :

$\min_\theta \frac{1}{2} \sum_{j=1}^{n} \theta_j^2 = \frac{1}{2}(\theta_1^2 + \theta_2^2)$

s.t. $\theta^T x^{(i)} \geq 1$ if $y^{(i)} = 1$

$\theta^T x^{(i)} \leq -1$ if $y^{(i)} = 0$

for simplification $\theta_0 = 0$ & $n = 2$

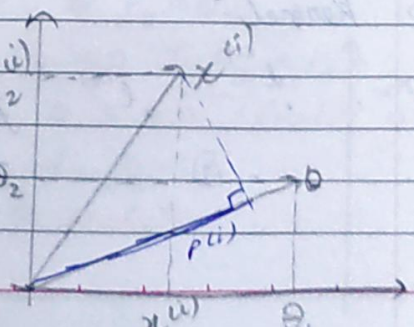
$$\therefore \frac{1}{2}(\theta_1^2 + \theta_2^2) = \frac{1}{2}\left(\sqrt{\theta_1^2 + \theta_2^2}\right)^2 = \|\theta\|$$

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

$\theta^T x^{(i)} = ? = p^{(i)} \cdot \|\theta\|$

$= \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}$

$u^T v$



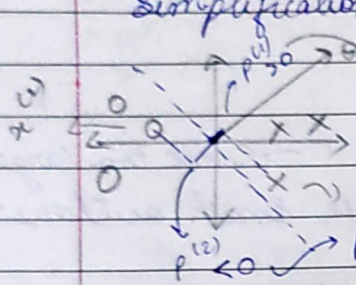
⇒ SVM Decision Boundary :

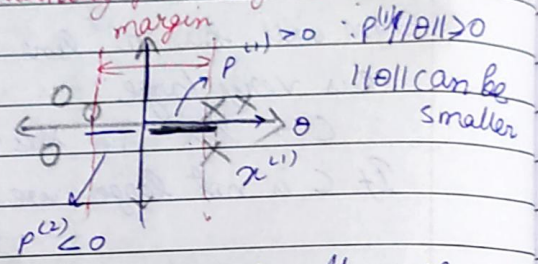
$\frac{1}{2}\|\theta\|^2 = \min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$ S.T. $p^{(i)}\cdot\|\theta\| \geq 1$ if $y^{(i)}=1$

$p^{(i)}\cdot\|\theta\| \leq -1$ if $y^{(i)}=0$

where $p^{(i)}$ is the projection of $x^{(i)}$ onto the vector θ

Simplification $\theta_0 = 0$ decision boundary pass through origin

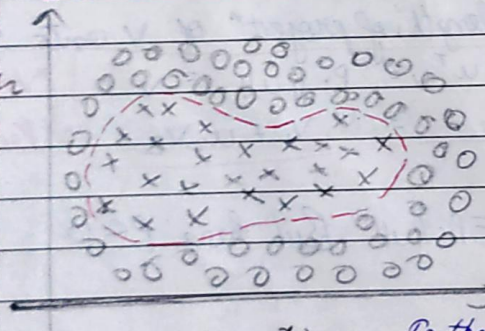




This is the hypothesis of SVM due to which θ will be small which will reduce the θ^2 error & will give best possible outcome.

⇒ Kernels - 1 :

Non-linear Decision Boundary :



predict $y=1$ if

$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 \cdots \geq 0$

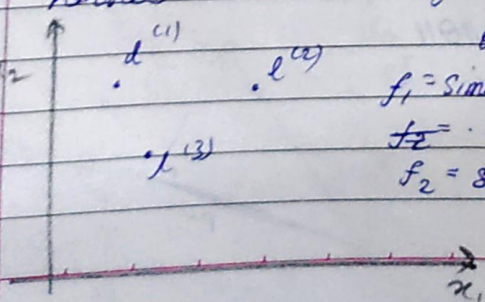
$h_\theta(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 \cdots \geq 0 \\ 0 & \text{otherwise} \end{cases}$

$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \cdots$

$f_1 = x_1,\ f_2 = x_2,\ f_3 = x_1 x_2,\ f_4 = x_1^2 \cdots$

Is there a different/better choice of the feature $f_1, f_2, f_3 \cdots$?

⇒ Kernel :



Given x, compute new features depending on proximity to landmarks.

$f_1 = \text{Similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$

$f_2 = \text{Similarity}(x, l^{(2)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right)$

↳ Kernel

Gaussian kernel $K(x, l^{(i)})$

⇒ **Kernel & Similarity :**

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$$= \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

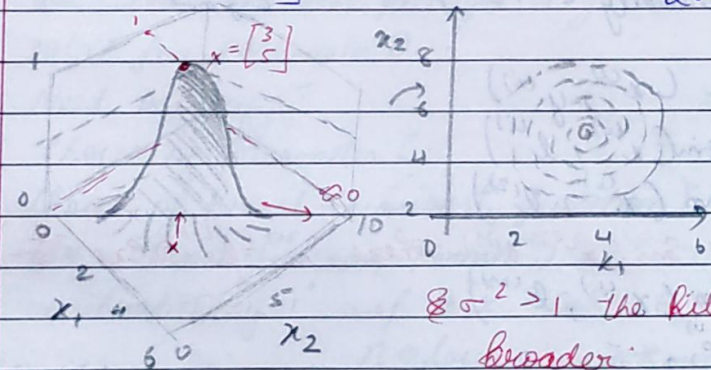
$$f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$$

If x is far from $l^{(1)}$:

$$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0$$

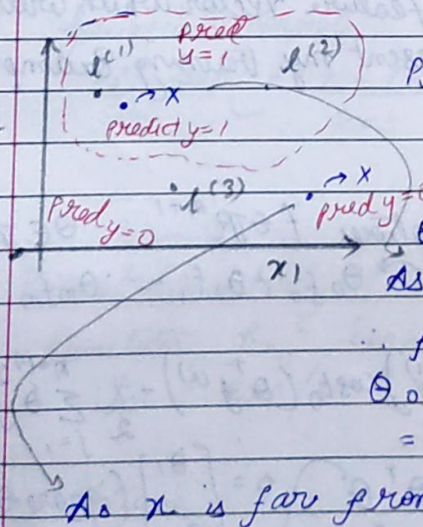
$l^{(1)} \to f_1$
$l^{(2)} \to f_2$
$l^{(3)} \to f_3$

Ex $l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$; $f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$ & $\sigma^2 = 1$



If $\sigma^2 = 0$ the width of the bump will be narrower & countour plots will be close to each other.

& $\sigma^2 > 1$, the hill becomes broader and broader.



Predict "1" when :

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

$\theta_0 = -0.5$, $\theta_1 = 1$, $\theta_2 = 1$, $\theta_3 = 0$

As x is close to $l^{(1)}$

$\therefore f_1 \approx 1$; $f_2 \approx 0$ & $f_3 \approx 0$

$\theta_0 + \theta_1 \times 1 + \theta_2 \times 0 + \theta_3 \times 0 =$

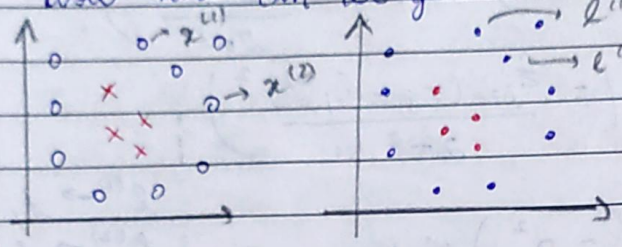
$= -0.5 + 1 = 0.5 \geq 0$ · Predict $y = 1$

As x is far from all · $f_1, f_2, f_3 \approx 0$

$\therefore \theta_0 + \theta_1 f_1 \cdots = \theta_0 = -0.5 < 0$ predict $y = 0$

⇒ Kernels 2 :

Now how can we get those $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?



SVM with Kernels :

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

choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$

Given example x:

$f_1 = \text{similarity}(x, l^{(1)})$

$f_2 = \text{similarity}(x, l^{(2)})$

$\ldots$

$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$ $f_0 = 1$ like $x_0 = 1$

for training ex $(x^{(i)}, y^{(i)})$

$x^{(i)} \rightarrow \begin{cases} f_1^{(i)} = \text{sim}(x^{(i)}, l^{(1)}) \\ f_2^{(i)} = \text{sim}(x^{(i)}, l^{(2)}) \\ \vdots \\ f_i^{(i)} = \text{sim}(x^{(i)}, l^{(i)}) \text{ where } x^{(i)} = l^{(i)} \Rightarrow \exp\left(\frac{-0}{2\sigma^2}\right) = 1 \\ f_m^{(i)} = \text{sim}(x^{(i)}, l^{(m)}) \end{cases}$

$\therefore f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$ $f_0^{(i)} = 1$ ← New feature vector which will represent my training example.

⇒ Hypothesis : Given x, compute features $f \in \mathbb{R}^{m+1}$ $\quad \theta \in \mathbb{R}^{m+1}$

Predict "$y = 1$" if $\theta^T f \geq 0$ $\rightsquigarrow \theta_0 f_0 + \theta_1 f_1 + \cdots \theta_m f_m$

Training :

$$\min_{\theta} C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{n=m} \theta_j^2$$

we can also write $\sum_j \theta_j^2 = \theta^T \theta$, or $\theta^T M \theta$ $\quad \theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$ (ignore θ_0) $\rightarrow \|\theta\|^2$

⇒ SVM parameters :

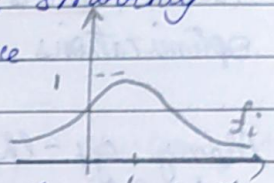
$C (= 1/\lambda)$ Large C : Lower Bias, High Variance (small λ) — overfit

Small C : Higher Bias, low Variance (large λ) — underfit

σ^2 Large σ^2 : Features f_i vary more smoothly

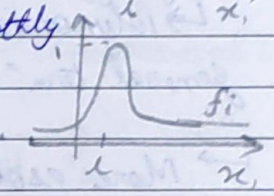
Higher bias, lower variance

$$\exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$$



Small σ^2 : Features f_i vary less smoothly

Lower bias, higher Variance



⇒ Using an SVM :

Use SVM softwares package (e.g liblinear, libsvm, ···) to solve for parameter θ.

Need to specify :

Choice of parameter C.

Choice of kernel (similarity function).

Ex No Kernel ("linear kernel")

Predict "y = 1" if $\theta^T x \geq 0$ $\quad \theta_0 + \theta_1 x_1 \cdots \theta_n x_n \geq 0$

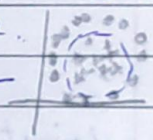
$n \to$ large ; $m \to$ small $\quad x \in \mathbb{R}^{n+1}$

Gaussian Kernel :

$f_i = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$ where $l^{(i)} = x^{(i)}$. $\quad x \in \mathbb{R}^n$ & n small

Need to choose σ^2. $\quad m \to$ large

use gaussian kernel



⇒ Kernel (similarity) functions :

function $f_{f_1} = \text{kernel}(x_1, x_2)$

$$f = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$$

$x^{(i)} \to l^{(i)}$

$x \to f_1, f_2, \ldots, f_m$

return

Note : Do perform feature scalling before using the Gaussian kernel

$\|x - l\|^2$ ∵ $v = x - l$ & $\|v\| = v_1^2 + v_2^2 \cdots v_n^2 = (x_1 - l_1)^2 + (x_2 - l_2)^2 \cdots (x_n - l_n)^2$

↳ 1000 sqfeet^2 → 1-5 bedroom ∴ $(x_1 - l_1)^2$ ⇒ huge ∴ use feature scalling

⇒ Other choices of Kernel :

Note : Not all similarity functions similarity (x, l) make valid kernels. (Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge)

⇒ Many off-the-shelf kernels available

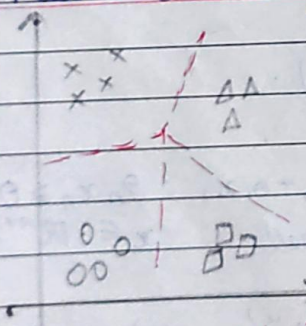
↳ Polynomial Kernel : $k(x, l) = (x^T l)^2, (x^T l)^3, (x^T l+1)^3, (x^T l+5)^4 \ldots$

General term $= (x^T l + \text{const})^{\text{degree}}$

↳ More esoteric : String Kernel, chi-square Kernel, Histogram intersection Kernel ----

similarity (x, l) if x is string we might use string kernel

⇒ Multi Class classification :



$y \in \{1, 2, 3 \ldots k\}$?

Many SVM packages already have built-in multi class classification functionality

Otherwise, use one-vs-all method

[Train K SVM's one to distinguish $y = i$ from the rest, for $i = 1, 2 \ldots k$) get $\theta^{(1)}, \theta^{(2)} \ldots \theta^{(k)}$

$y=1 \rightarrow \theta^{(1)}$, $y=2 \rightarrow \theta^{(2)}$, $y=k \rightarrow \theta^{(k)}$

Pick class i with largest $(\theta^{(i)})^T x$

⇒ Logistic Regression V/s SVM's :

n = number of feature ($x \in \mathbb{R}^{n+1}$), m = number of training examples

↳ If n is large (relative to m) (Eg $n \geq m$, $n=10,000$, $m=1000$)
Use logistic regression or SVM without a kernel ("linear kernel")

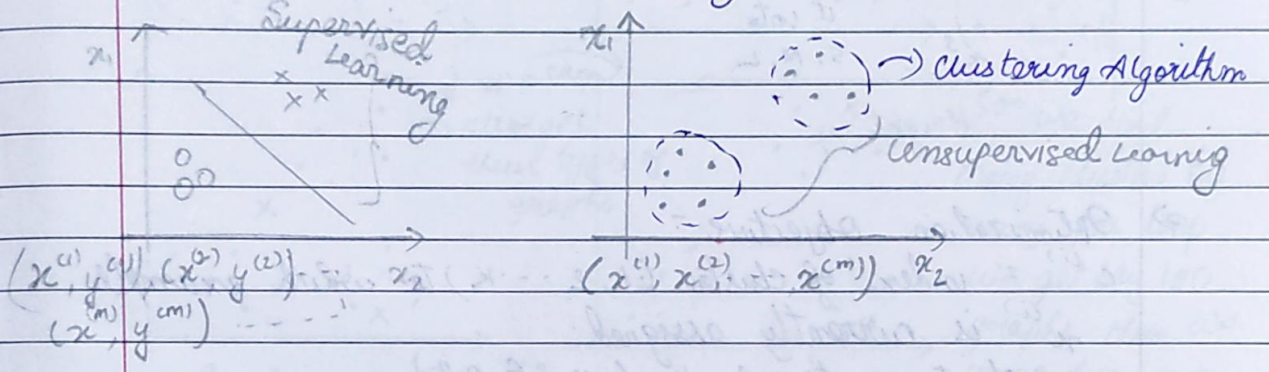
↳ If n is small, m is intermediate (Ex $n = 1-1000$, $m = 10-10000$)
Use SVM with gaussian kernel

↳ If n is small, m is large (Ex $n = 1-1000$, $m = 50,000+$)
Create/add more features; then use logistic regression or SVM without a kernel.

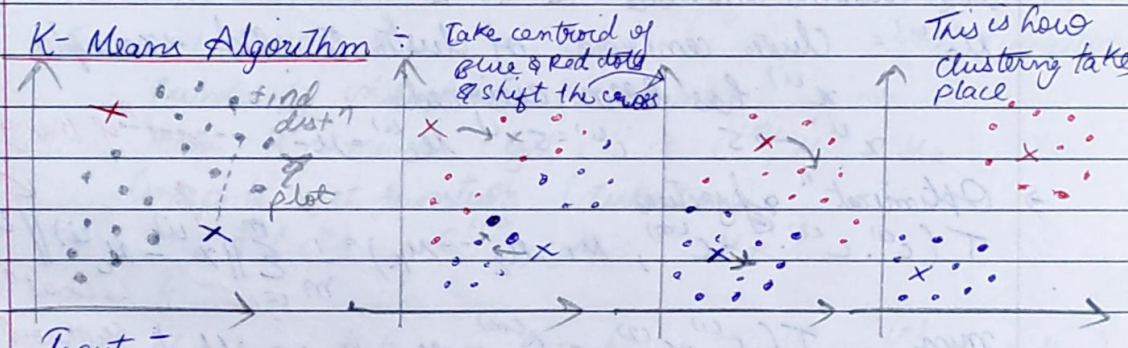
↳ Neural network likely to work well for most of those settings but may be slower to train.

Week-8

Unsupervised Learning



⇒ **K-Means Algorithm** :



Input :
↳ K (number of clusters)
↳ Training set $\{x^{(1)}, x^{(2)}, x^{(3)} \cdots x^{(m)}\}$
$x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

Randomly initialize K cluster centroids $\mu_1, \mu_2, \cdots \mu_K \in \mathbb{R}^n$
Repeat {

cluster assignment closer to red or blue

{ for $i = 1$ to m
$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$ $\min_k \| x^{(i)} - \mu_k \|^2$ → $c^{(i)}$

minimize $J(\cdots)$ wrt $c_1, c_2 \cdots c_{(m)}$ & ($\mu_1 \cdots \mu_K$ fixed)

move centroid

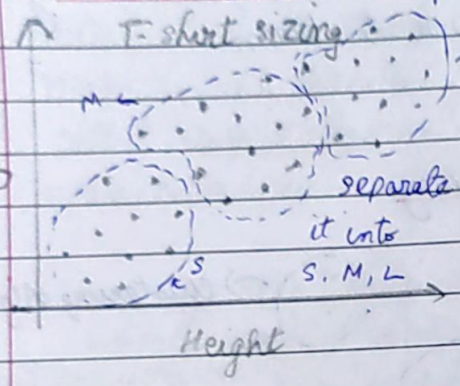
{ for $k = 1$ to K
μ_k := average (mean) of points assigned to cluster K
$\mu_2 = \frac{x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}}{4}$
$c^{(1)} = 2;\ c^{(5)} = 2,\ c^{(6)} = 2,\ c^{(10)} = 2$

minimize $J(\cdots)$ wrt $\mu_1, \mu_2 \cdots \mu_K$

moves centroid step
}

⇒ K-Means for non-separated clusters:

T-shirt sizing



⇒ Optimization objective:

$c^{(i)}$ = index of cluster (1, 2 ··· K) to which example $x^{(i)}$ is currently assigned.

μ_k = cluster centroid k ($\mu_k \in R^n$)

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned

$x^{(i)} \rightarrow 5 \quad \therefore c^{(i)} = 5 \quad \mu_{c^{(i)}} = \mu_5$

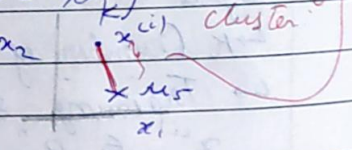
⇒ Optimizat^n objective:

$$J(c^{(1)}, c^{(2)} \cdots c^{(m)}, \mu_1, \mu_2 \cdots \mu_K) = \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - \mu_{c^{(i)}} \|^2$$

↳ locat^n of training ex

↳ locat^n of centroid of cluster

$$\min_{c^{(1)} \cdots c^{(m)}, \mu_1 \cdots \mu_K} J(c^{(1)}, c^{(2)} \cdots c^{(m)}, \mu_1, \mu_2 \cdots \mu_K)$$

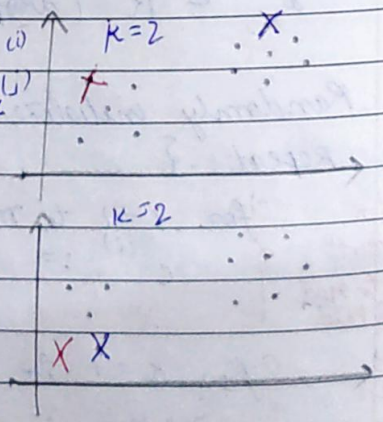


⇒ Random initialization: $\mu_1 = x^{(i)}$, $\mu_2 = x^{(j)}$

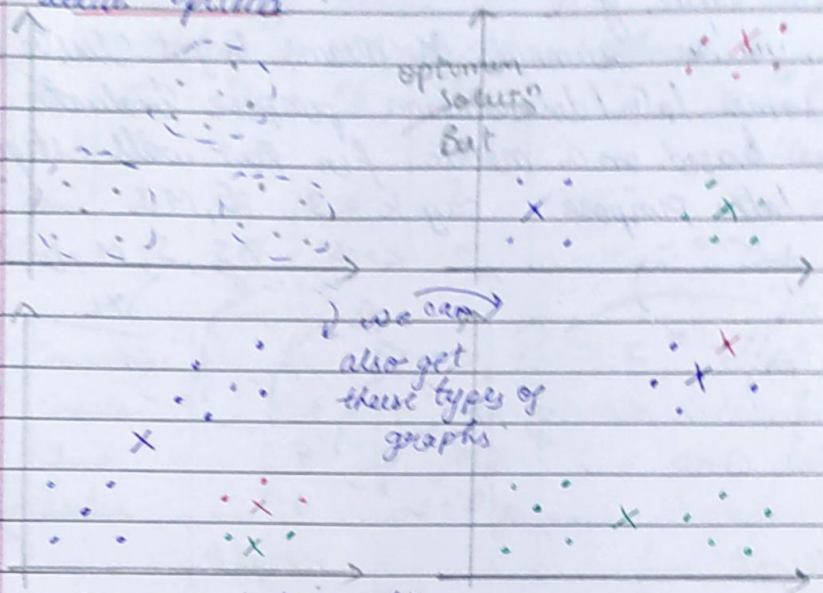
↳ Should have $K < m$

↳ Randomly pick K training examples.

↳ Set $\mu_1 \cdots \mu_K$ equal of these K examples



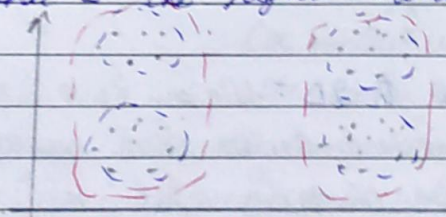
e) Local optima :



So we find many clusters by using for loop we get say 100 graphs then we compute lowest cost function

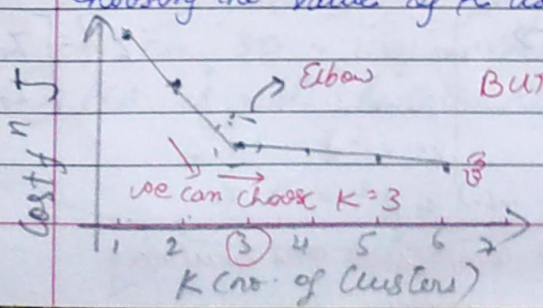
Random initialization
For $i = 1$ to 100 {
Randomly initialize K-means
Run K-means. Get $c^{(1)} \cdots c^{(m)}, \mu_1 \cdots \mu_K$.
Compute cost function (distortion)
$J(c^{(1)} \cdots c^{(m)}, \mu_1 \cdots \mu_K)$
}
Pick clustering that gave lowest cost $J(c^{(1)} \cdots c^{(m)}, \mu_1 \cdots \mu_K)$

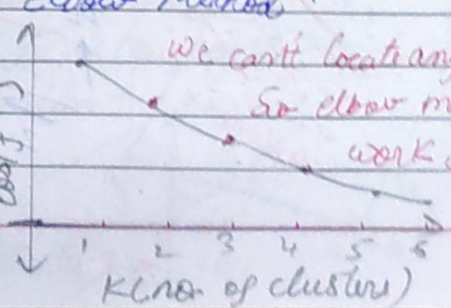
⇒ Clustering : Choosing the number of clusters :
What is the right value of K ?



for some there are 4 clusters
for some there are 2 clusters
3, 5, 6 clusters
So how to choose that ?

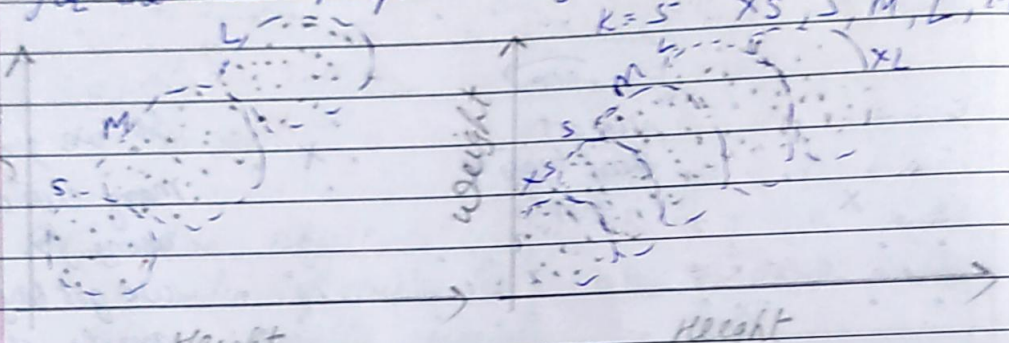
⇒ Choosing the value of K using Elbow Method :



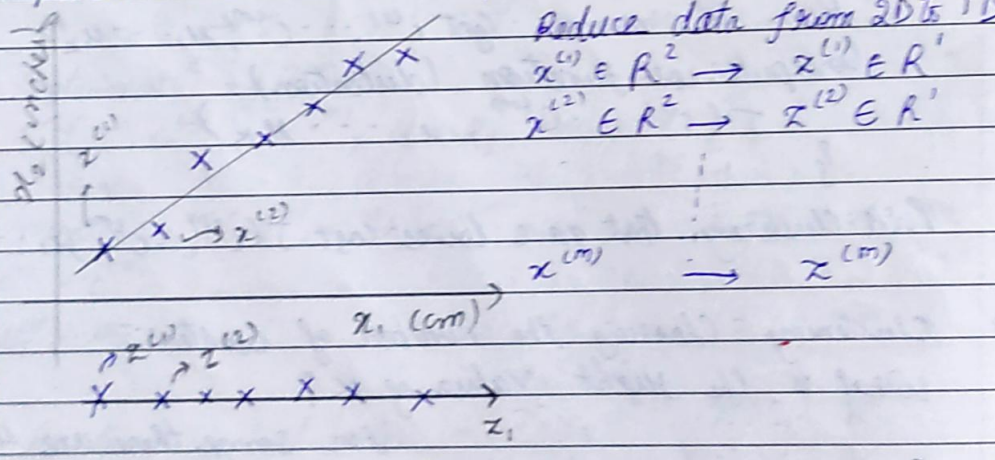


Observe we can't locate any elbow
So elbow method won't work well here

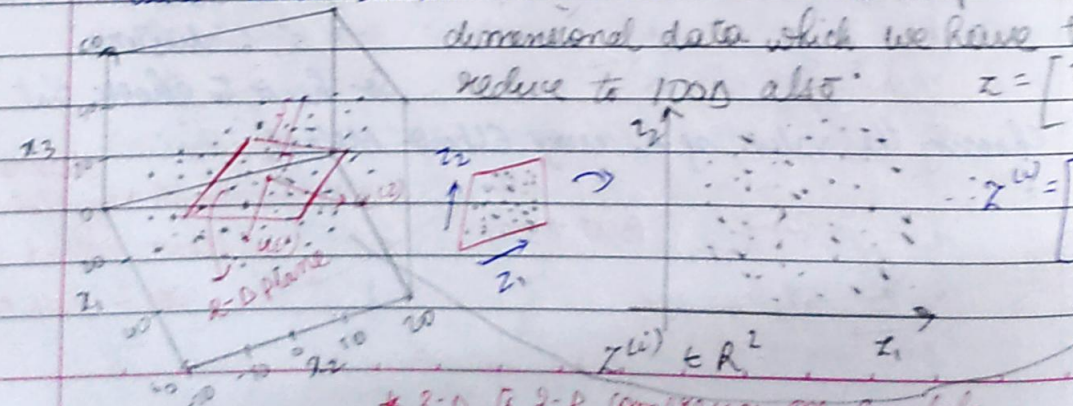
⇒ Choosing the value of K :
Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-Means based on a metric for how well it performs for that later purpose. say k = 3 S, M, L



* Unsupervised learning - Dimensionality Reduction :
Motivation 1 : Data Compression.



⇒ Reduce data from 3D to 2D : We can have upto 1000 dimensional data which we have to reduce to 1000 also.



* 3-D to 2-D compressor and Emulator

⇒ Dimensionality Reduction :

Motivation 2 : Data Visualization

Country	x_1	x_2	x_3	x_4	...
Canada	8	3	4	7	...
China	3	2	5	8	...
India	4	6	9	10	...
⋮					

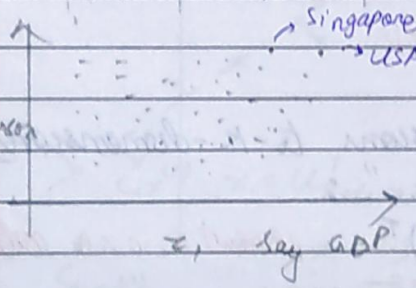
$x^{(i)} \in \mathbb{R}^{50}$

Instead of 50 features we can represent it two maybe say like this

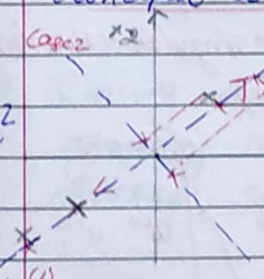
→ $z^{(i)} \in \mathbb{R}^2$

We Reduced data from 50D to 2D

Country	z_1	z_2
Canada	1.6	1.2
China	1.7	0.3
India	1.6	0.2
⋮		



⇒ Principal Component Analysis Problem formulation : (PCA)



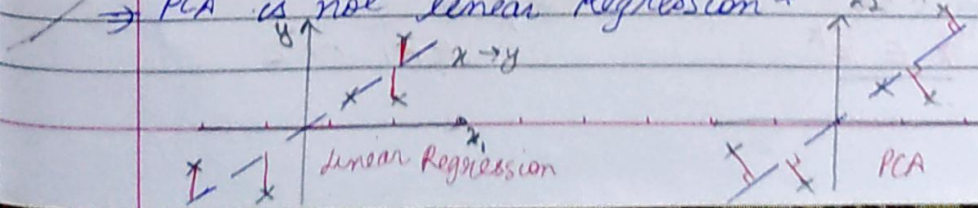
Acc to PCA we will select case 1 as the square distance from the line is much more smaller than case 2 wala line

↳ Reduce from 2D to 1D : Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the project error

↳ Reduce from n-dimension to k dimension : Find k vector $u^{(1)}, u^{(2)} \dots u^{(k)}$ onto which to project the data, so as to minimize the projection error

→ For 3D, we convert it to 2D so $k = 2$.

⇒ PCA is not linear Regression :



⇒ PCA Algorithm :-

Before using Algorithm we must always do Data preprocessing

↳ Data preprocessing :-

Training set : $x^{(1)}, x^{(2)} \ldots x^{(m)}$

Preprocessing (feature scaling / mean normalization)

$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$ } Mean of each feature

Replace each $x_j^{(i)}$ with $x_j - \mu_j$

If different features on different scales (eg x_1 = size, x_2 = bedrooms), scale features to have comparable range of values. We do feature scaling

$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$

⇒ Algorithm :-

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix" :

$\Sigma = \frac{1}{m} \sum_{i=1}^{n} (x^{(i)})(x^{(i)})^T$ ... product is $n \times n$ matrix

(Σ represents matrix, greek alphabet sigma; $x^{(i)}$ is $n \times 1$, $(x^{(i)})^T$ is $1 \times n$, result is $n \times n$)

Compute "eigen vectors" of matrix Σ :

[U, S, V] = svd(Sigma); (svd → singular value decomposition)

We can also use eig(Sigma)

$U = \begin{bmatrix} | & | & | & & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \cdots & u^{(n)} \\ | & | & | & & | \end{bmatrix}$ $U \in \mathbb{R}^{n \times n}$

$u^{(1)}, \ldots u^{(k)}$

$x \in \mathbb{R}^n \rightarrow x \in \mathbb{R}^k$

$\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix}$ $n \times k$

$\therefore z^{(i)} = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix}^T x^{(i)} = \begin{bmatrix} - & u^{(1)} & - \\ & \vdots & \\ - & u^{(k)} & - \end{bmatrix} x^{(i)}$ Reduce

$n \times k$; $k \times n$; $n \times 1$

$x \in \mathbb{R}^k$; $k \times 1 \rightarrow z$

⇒ PCA Algorithm Summary:
After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

$$\text{Sigma} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T$$ → In octave

Sigma = (1/m) * X' * X;

[U, S, V] = svd(Sigma);

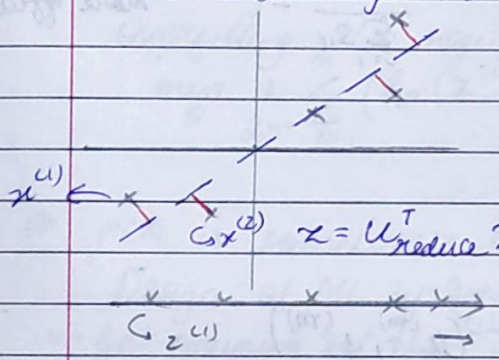
Ureduce = U(:, 1:K); } grab first K columns

$z = U_{reduce}' * x$;

$x \in \mathbb{R}^n$ & ~~$x_0 = 1$~~

$$X = \begin{bmatrix} - & - & x^{(1)T} & - & - \\ & & \vdots & & \\ - & - & x^{(m)T} & - & - \end{bmatrix}$$

⇒ Reconstruction from Compressed Representation:



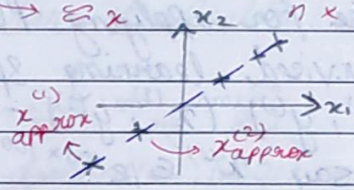
How can we go back from $z^{(i)} \to x^{(i)}$

$z \in \mathbb{R} \to x \in \mathbb{R}^2$

$x_{approx} = U_{reduce} \cdot z$

$\hookrightarrow \mathbb{R}^n$ $\quad n \times K \quad K \times 1$

$\hookrightarrow \in x_1$ $\quad n \times 1$



⇒ Choosing number of principal components (K):
choosing K (no. of principal components)
Average squared projection error: $\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{approx}^{(i)}\|^2$

Total variation in the data: $\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2$

Typically choose K to be smallest value so that

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01 \;/\; 0.05 \;(1\%/5\%)$$

99% of variance is retained / 95%

⇒ Choosing k (number of principal components)

Algorithm:

Try PCA with $k = 1, 2, 3 \ldots$ Say $k = 17$

Compute $U_{reduce}, z^{(1)}, z^{(2)} \ldots z^{(m)}$ we get 99% of variance retained

$x_{approx}^{(1)} \ldots x_{approx}^{(m)}$

Check if

$$\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} \leq 0.01 \; ?$$

First pick the smallest value of k for which $\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1} S_{ii}} \geq 0.99$ Then increase k [99% of variance retained]

$[U, S, V] = \text{Svd}(\text{Sigma})$

$S = \begin{bmatrix} S_{11} & 0 & \cdots & \\ 0 & S_{22} & 0 & \\ 0 & 0 & S_{33} & 0 \\ & & 0 & S_{nn} \end{bmatrix}$ for $k = 3$

for given k

$$\rightarrow 1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1} S_{ii}} \leq 0.01$$

or $\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1} S_{ii}} \geq 0.99$

we can increase k it is much more efficient

⇒ Advice for applying PCA:

Supervised learning speedup

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

say $x^{(i)} \in \mathbb{R}^{100 \times 100}$ like picture pixel grid

Exact inputs:

Unlabeled dataset: $x^{(1)}, x^{(2)} \ldots x^{(m)} \in \mathbb{R}^{10000}$

$z^{(1)}, z^{(2)} \ldots z^{(m)} \in \mathbb{R}^{1000}$ (say)

PCA: $x \downarrow \text{PCA} \downarrow z$

∴ New training set:

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

Now we can use regression, neural networks... whatever we want.

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

Note: Mapping $x^{(i)} \rightarrow z^{(i)}$ should be defined by running PCA only on training set. This mapping can be applied as well to the examples $x_{cv}^{(i)}$ and $x_{test}^{(i)}$ in the cross validation and test sets.

⇒ Applicat^{n} of PCA :

↳ compression : i) Reduce memory/disk needed to store data
ii) Speed up learning algorithm
Choose k by % of variance retain

↳ Visualization : $k = 2$ or 3 as we can only visualize that

⇒ Bad use of PCA : To prevent overfitting : Bad!
Use $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features to $k < n$ (1000 → 100)
Thus fewer features, less likely to overfit.

↳ This might work ok, but isn't a good way to address overfitting. Use regularization instead.

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

Regularization is better than PCA for overfitting

⇒ PCA is sometimes used where it shouldn't be :
Design of ML system :

↳ Get training set $\{(x^{(1)}, y^{(1)}), \cdots (x^{(m)}, y^{(m)})\}$
↳ Run PCA to reduce $x^{(i)}$ in dimension to get $z^{(i)}$
↳ Train logistic regression on $\{(z^{(1)}, y^{(1)}), \cdots (z^{(m)}, y^{(m)})\}$
↳ Test on test set : Map $x_{test}^{(i)}$ to $z_{test}^{(i)}$. Run $h_\theta(z)$ on $\{(z_{test}^{(1)}, y_{test}^{(1)}) \cdots (z_{test}^{(m)}, y_{test}^{(m)})\}$

BUT, how about doing the whole thing without PCA?
↳ Before implementing PCA, first try running whatever you want to do with original/raw data $x^{(i)}$. Only if that doesn't do what you want, then implement PCA and consider using $z^{(i)}$.

(don't apply PCA unnecessarily)

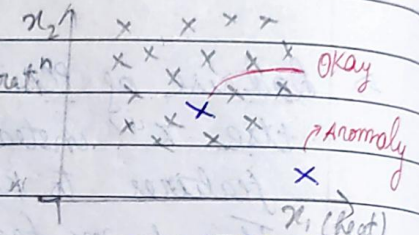
Page No. 76
Date 1 4 21

Week - 9

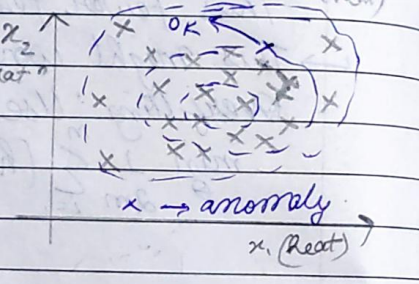
Anomaly Detection

⇒ Anomaly detection example: For aircraft engine features
x_1 = heat generated
x_2 = vibration intensity
...

Dataset: $\{x^{(1)}, x^{(2)}, \cdots x^{(m)}\}$
New engine: x_{test}



⇒ Density estimation:
Dataset: $\{x^{(1)}, x^{(2)}, \cdots x^{(m)}\}$
Is x_{test} anomalous?
Model $p(x)$
$p(x_{test}) < \varepsilon$ → flag anomaly
$p(x_{test}) \geq \varepsilon$ → OK



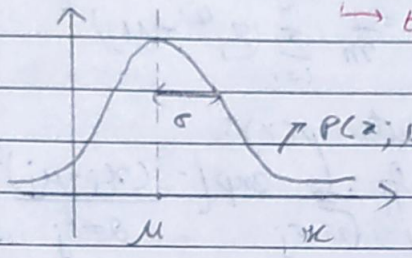
⇒ Anomaly detection example:
i) Fraud detect^n:
↳ $x^{(i)}$ = features of user i's activities
↳ Model $p(x)$ from data.
↳ Identify unusual users by checking which have $p(x) < \varepsilon$
ii) Manufacturing
iii) Monitoring computers in a data center:
$x^{(i)}$ = features of machine i
x_1 = memory use, x_2 = number of disk accesses/sec
x_3 = CPU load, x_4 = CPU load / network traffic
...
$p(x) < \varepsilon$

⇒ Gaussian Distribution :- (Normal)

Say $x \in R$, if x is a distributed Gaussian with mean μ & variance σ^2.

$x \sim \mathcal{N}(\mu, \sigma^2)$ → Script N stands for Normal = Gaussian

↳ tilda → "distributed as"



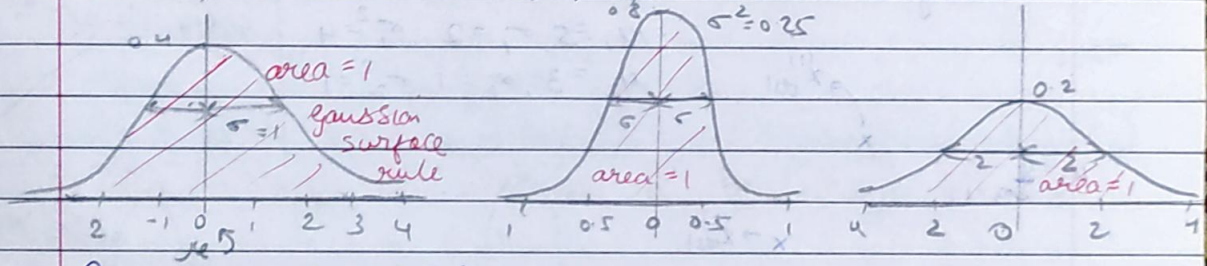
~~P(x)~~ $p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$

σ → standard deviation

σ^2 → Variance

⇒ Gaussian Distribution Example :-

$\mu = 0, \sigma = 1$ $\quad \mu = 0, \sigma = 0.5$ $\quad \mu = 0, \sigma = 2$

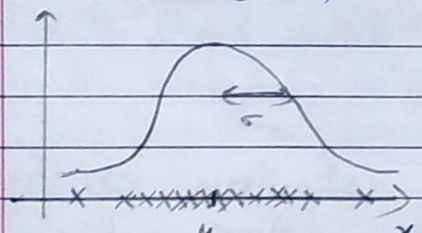


⇒ Parameter estimation :-

Dataset : $\{x^{(1)}, x^{(2)} \cdots x^{(m)}\}$ $\quad x^{(i)} \in R$

$x^{(i)} \sim \mathcal{N}(\mu, \sigma^2)$

$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$ } mean of numbers



$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^2$

In statistics we use $(m-1)$ but in ML we ignore this factor as m is very large.

⇒ Algorithm :-

Density Estimation :- $x \in R^n$ $\quad x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$

Training set : $\{x^{(1)}, \{x^{(2)}, \cdots x^{(m)}\}$ $\quad x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$

$p(x) = p(x_1; \mu_1, \sigma_1^2)\, p(x_2; \mu_2, \sigma_2^2)\, p(x_3; \mu_3, \sigma_3^2) \cdots p(x_n; \mu_n, \sigma_n^2)$

$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$

↑ product symbol

⇒ Anomaly detection Algorithm :

i) Choose features x_i that you think might be indicative of anomalous examples. $\{x^{(1)}, \ldots, x^{(m)}\}$

ii) Fit parameters $\mu_1, \ldots, \mu_n, \sigma_1^2, \ldots, \sigma_n^2$

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \quad \& \quad \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

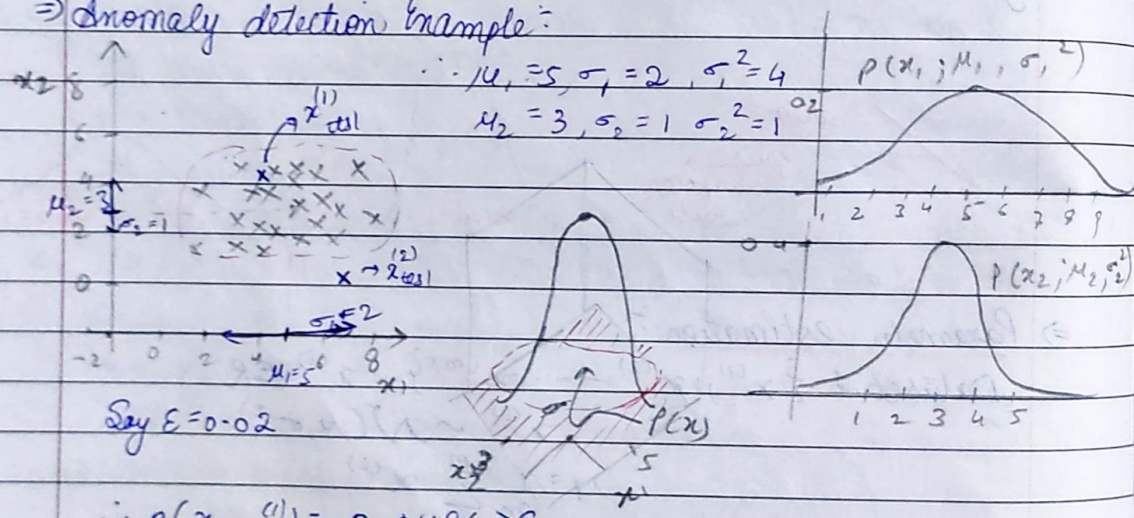
$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_n \end{bmatrix}$

iii) Given new example x, compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if $p(x) < \varepsilon$

⇒ Anomaly detection Example :



Say $\varepsilon = 0.02$

$\therefore p(x^{(1)}_{test}) = 0.0426 \geq \varepsilon$

$p(x^{(2)}_{test}) = 0.0021 \leq \varepsilon$

⇒ Developing and Evaluating an Anomaly Detection System :

↳ Importance of real number evaluation

When develops a learning algorithm (choosing features etc) making decisions is much easier if we have a way of evaluating our learning algorithm.

↳ Assume we have some labeled data, of anomalous and non-anomalous examples ($y = 0$ if normal, $y = 1$ if anomalous).

↳ Training set : $x^{(1)}, x^{(2)} \dots x^{(m)}$ (assume normal examples/ not anomalous)

↳ cross validation set : $(x_{cv}^{(1)}, y_{cv}^{(1)}) \dots (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

Test set : $(x_{test}^{(1)}, y_{test}^{(1)}) \dots (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

$y=1$

⇒ Aircraft engines motivating Examples :

10000 good (normal) engines

20 flawed engines (anomalous) 2-50

↳ Training set : 6000 good engines ($y=0$) $p(x) = p(x;\mu_1,\sigma_1^2) \dots p(x;\mu_n,\sigma_n^2)$

CV : 2000 good engines ($y=0$), 10 anomalous ($y=1$)

Test : 2000 good engines ($y=0$), 10 anomalous ($y=1$)

Alternative : ✗

Training set : 6000 good engines

CV & Test : 4000 good engines ($y=0$), 10 anomalous ($y=1$)

Some put same 4K data in cv & Test which is not good

⇒ Algorithm Evaluation :

↳ Fit model $p(x)$ on training set $\{x^{(1)} \dots x^{(m)}\}$

↳ On a cross validation / test example x, predict

$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \varepsilon \text{ (normal)} \end{cases}$$

$\{x_{test}^{(i)}, y_{test}^{(i)}\}$

predicting $y=0$ is a very high accuracy always

↳ Possible evaluation metrics :

True positive, false positive, false negative, true negative

Precision / Recall

F_1-Score

Can also use cross-validation set to choose parameter ε

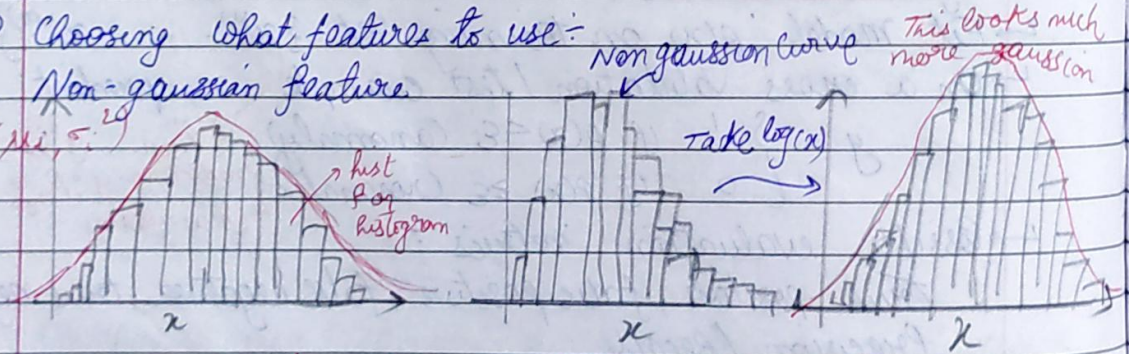
Select diff value of ε to get maximum F_1 score

⇒ Anomaly detection V/s Supervised learning :-

Anomaly detection	Supervised learning
→ Very small number of positive examples ($y=1$). (0-20 is common)	→ Large number of positive & negative examples
→ Large number of negative ($y=0$) Examples. P(x) like aircraft	
→ Many different "types" of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.	→ Enough positive examples for algorithm to get a sense of what positive examples are like, future positive examples likely to be similar to ones in training set.
↳ Fraud detection	↳ Email spam classification
↳ Manufacturing (eg- aircraft engines)	↳ Cancer classification
↳ Monitoring machines in a data center	↳ Weather predict^{n} (Summer, rain, winter)

⇒ Choosing what features to use :-

Non-gaussian feature

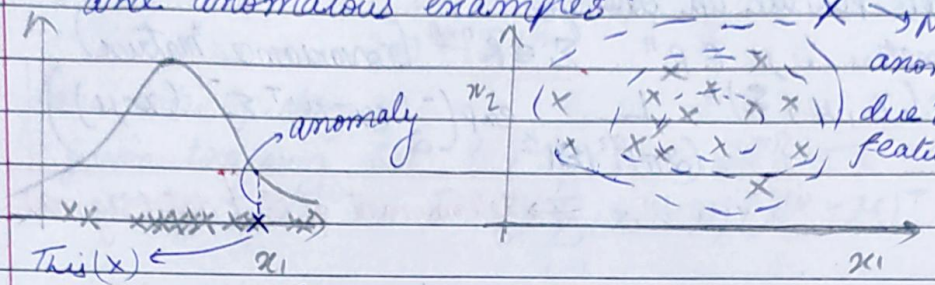


$x_1 \to \log(x_1)$ / $x_2 \to \log(x_2+1)$ / $\log(x_i+c)$ ··· can be used like $x_3 \to \sqrt{x_3}$ / $x_3^{1/2}$ / $x_4 \to x_4^{1/3}$ To look data gaussian

⇒ Error analysis for anomaly detection :-

↳ Want $p(x)$ large for normal example x.
$p(x)$ small for anomalous example x.

↳ Most common problem :-
$p(x)$ is comparable (say, both large) for normal and anomalous examples :-

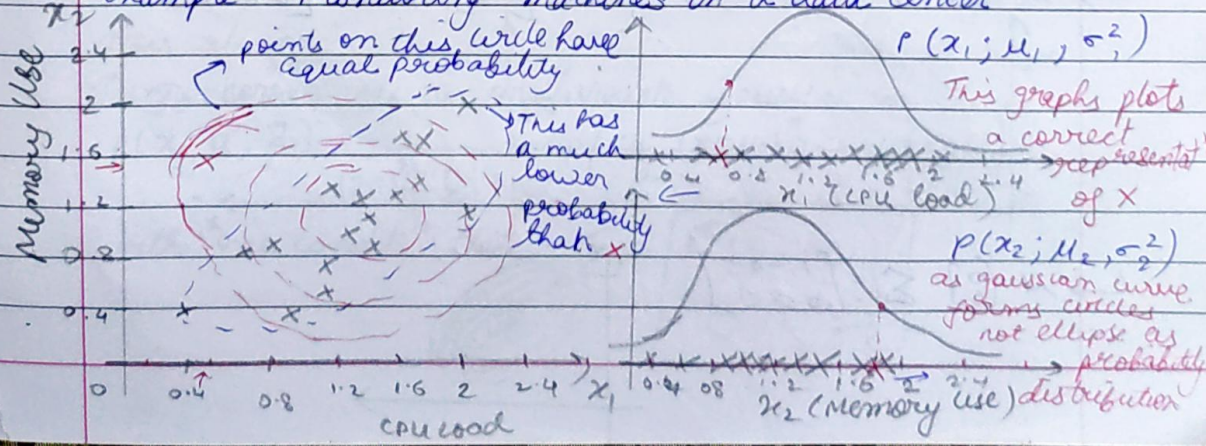


Show be normal but we find it anomaly So we can add new features to it say (x_2) & make a new plot & then make that plot in use & show that the a new example is shown anomalous

⇒ Monitoring computers in a data center :-
Choose features that might take on usually large or small values in the event of an anomaly.
x_1 = memory use of computer.
x_2 = number of disc accesses/sec
x_3 = CPU load
x_4 = network traffic
} If it gives a bigger value or being dependent on each other

New x_5 = $\frac{\text{CPU load}}{\text{network traffic}}$ / x_6 = $\frac{(\text{CPU load})^2}{\text{network traffic}}$

⇒ Multivariate Gaussian Distribution :-
Example : Monitoring machines in a data center



⇒ Multivariate Gaussian (Normal) distribution :

$x \in R^n$ Don't model $p(x_1)$, $p(x_2)$ ... etc. separately. Model $p(x)$ all in one go.

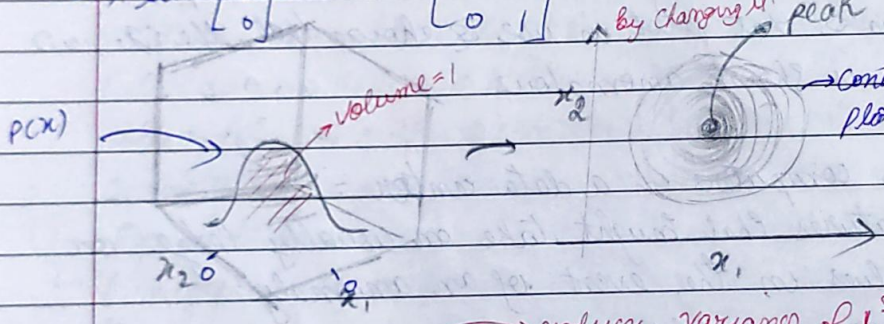
Parameters : $\mu \in R^n$, $\Sigma \in R^{n \times n}$ [covariance matrix]

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

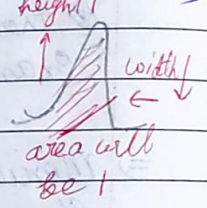
↳ determinant of Σ / det (sigma)

⇒ Multivariate Gaussian (Normal) Examples :

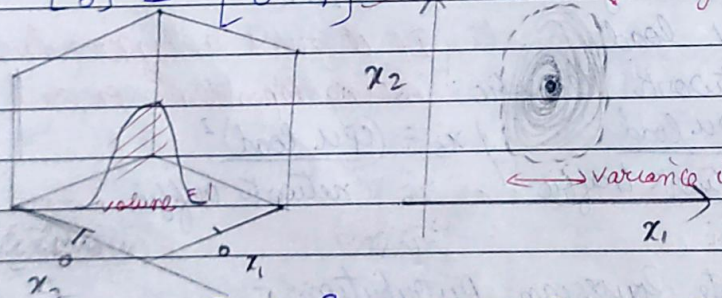
↳ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ Centre peak can be altered by changing μ



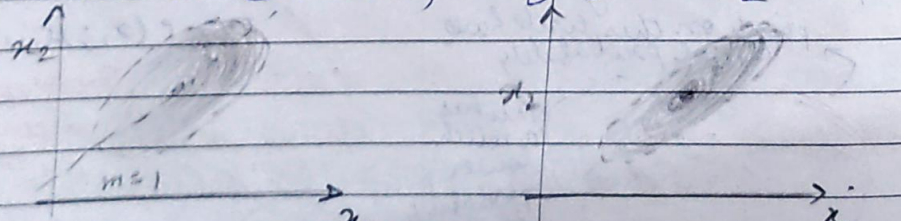
for $\Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$



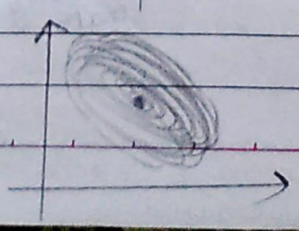
↳ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 1 \end{bmatrix}$ → reduces variance of 1st feature x_1 → variance of 2nd feature x_2



↳ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$; $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$



↳ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$



⇒ Anomaly Detection using the Multivariate gaussian Distribution:

Parameters μ, Σ ; $\mu \in R^n$ & $\Sigma \in R^{n \times n}$

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

Parameter fitting :

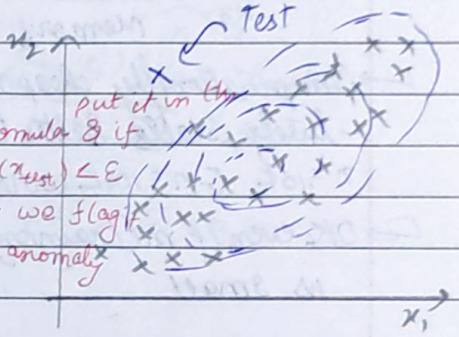
Given training set $\{x^{(1)}, x^{(2)} \cdots x^{(m)}\} \rightarrow x \in R^n$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \quad ; \quad \Sigma = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

i) Fit model $p(x)$ by setting

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$



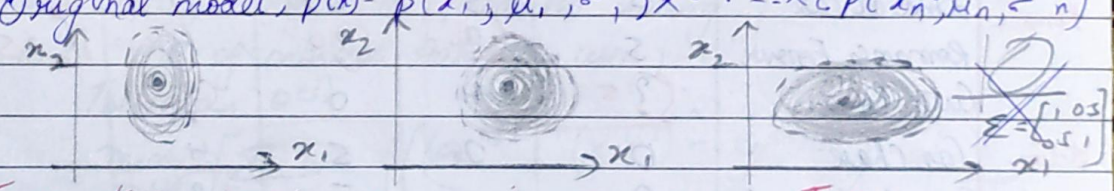
ii) Given a new example x, compute

$$p(x) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

Flag an anomaly if $p(x) < \varepsilon$.

⇒ Relationship to Original model :

Original model; $p(x) = p(x_1; \mu_1, \sigma_1^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)$



The countours are along same axis as x_1 & x_2. These are axis alligned.

∴ It corresponds to multivariate Gaussian.

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

with one condit^n that $\Sigma = \begin{bmatrix} \sigma_1^2 & & 0 \\ & \sigma_2^2 & \\ 0 & & \sigma_n^2 \end{bmatrix}$

Original Model

$p(x_i; \mu_i, \sigma_i^2) \times p(x_n; \mu_n; \sigma_n^2) \hookrightarrow p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$

↳ Manually create features to capture anomalies where x_1, x_2 take unusual combinat^n of value

Ex $x_3 = \frac{x_1}{x_2}$ = CPU load / memory

↳ Computationally cheaper (alternative) - Likely, scales better to larger n)
n = 10K, 100K works fine

↳ OK even if m (training set size) is small

Multivariate Gaussian

↳ Automatically captures correlations between features

$\Sigma \in \mathbb{R}^{n \times n}$ → for big n Σ^{-1} expensive

↳ computationally more expensive

Σ singular ← $x_1 = x_2$ / $x_3 = x_4 + x_5$

$\Sigma \sim \frac{n^2}{2}$ parameters

↳ Must have $m > n$ or else Σ is non-invertible or $m \geq 10n$

★ Recommender System:

Problem formulation:

Example: Predict^n movie ratings

User rates movies using zero ~~one~~ - five stars

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	
Romantic: Love at last	5	5	0	0	$n_u = 4$
Romance Forever	5	? 4.5	? 0	0	$n_m = 5$
Cute love	? 5	4	0	? 0	
Action: Car Chase	0	0	5	4	
Swords v/s Karate	0	0	5	? 4	

n_u = no. of users

n_m = no. of movies

$r(i,j) = 1$ if user j has rated movie i

$y^{(i,j)}$ = rating given by user j to movie i (defined only if $r(i,j) = 1$)

0 ... 5

We have to predict the ?

⇒ Content based Recommendation System :

Same table as left

	Movie	Alice(1) $\theta^{(1)}$	Bob(2) $\theta^{(2)}$	Carol(3) $\theta^{(3)}$	Dave(4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
$x^{(1)}$	1 Love at last	5	5	0	0	0.9	0
$x^{(2)}$	2 Romance forever	5	?	?	0	1.0	0.01
$x^{(3)}$	3 Cute love	4.95	4	0	?	0.99	0
$x^{(4)}$	4 Car chase	0	0	5	4	0.1	1.0
$x^{(5)}$	5 Swords vs karate	0	0	5	?	0	0.9

$x_0 = 1$, $x_1 = \begin{bmatrix} 0.9 \\ 0 \end{bmatrix}$

$\theta^{(1)} \in \mathbb{R}^{n+1}$, $n = 2$

For each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^3$. Predict user j as rating movie i with $(\theta^{(j)})^T x^{(i)}$ stars.

$x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix}$ $\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$ (Scay)

$(\theta^{(1)})^T x^{(3)} = 5 \times 0.99 = 4.95$

⇒ Problem formulation :

$r(i,j) = 1$ if user j has rated movies i (0 otherwise)

$y^{(i,j)}$ = rating by user j on movie i (if defined)

$\theta^{(j)}$ = parameter vector for user j

$x^{(i)}$ = feature vector for movie i

For user j, movie i, predicted rating $(\theta^{(j)})^T (x^{(i)})$

$m^{(j)}$ = no of movies rated by user j

To learn $\theta^{(j)}$:

$$\min_{\theta^{(j)}} \left[\sum_{i:r(i,j)=1} \left((\theta^{(j)})^T (x^{(i)}) - y^{(i,j)} \right)^2 \right] \frac{1}{2m^{(j)}} + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

~~$\theta^{(j)} \in \mathbb{R}^{n+1}$~~ To learn $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$:

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

$$J(\theta^{(1)}, \dots, \theta^{(n_u)})$$

⇒ Gradient descent update :

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right) x_k^{(i)} \quad (\text{for } k=0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right) x_k^{(i)} + \lambda \theta_k^{(j)} \right) \quad (\text{for } k \neq 0)$$

$\frac{\partial}{\partial \theta_k^{(j)}} J(\theta^{(1)}, \dots, \theta^{(n_u)})$

⇒ Collaborative Filtering :

Problem motivation : What if we dont know x_1 & x_2 ? $x_0 = 1$

	movies	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 romance	x_2 action
$x^{(1)}$	Love at last	5	5	0	0	?1.0	?0.0
$x^{(2)}$	Romance forever	5	?	?	0	?	?
$x^{(3)}$	Cute love	?	4	0	?	?	?
$x^{(4)}$	Car chase	0	0	5	4	?	?
$x^{(5)}$	Sword vs karate	0	0	5	?	?	?

$\theta_1 = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$ $\theta_2 = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$ $\theta_3 = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$ $\theta_4 = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$ → x_0 → romance x_1 → Action x_2

So that $(\theta^{(1)})^T \cdot x^{(1)} \approx 5$ & $(\theta^{(3)})^T x^{(1)} \approx 0$ $\quad x^{(1)} = \begin{bmatrix} 1 \\ 1.0 \\ 0.0 \end{bmatrix}$

$(\theta^{(2)})^T \cdot x^{(1)} \approx 5$ $\quad (\theta^{(4)})^T x^{(1)} \approx 0$

⇒ Optimization Algorithm :

Given $\theta^{(1)}, \dots, \theta^{(n_u)}$ to learn $x^{(i)}$:

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2$$

Given $\theta^{(1)}, \dots, \theta^{(n_u)}$ to learn $x^{(1)}, \dots, x^{(n_m)}$:

$$\min_{x^{(1)}, \dots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2$$

⇒ According to Collaborative filtering :
Given $x^{(1)} \dots x^{(n_m)}$ (and movie ratings),
we can estimate $\theta^{(1)} \dots \theta^{(n_u)}$
Given $\theta^{(1)} \dots \theta^{(n_u)}$,
We can estimate $x^{(1)} \dots x^{(n_m)}$
∴ Guess $\theta \rightarrow x \rightarrow \theta \rightarrow x \dots$ $\theta \in R^n$
$x_0 = 1$ $x \in R^n$ as x_0 is cancelled

⇒ Colaborative Filtering optimization objective :
↳ Given $x^{(1)} \dots x^{(n_m)}$ estimate $\theta^{(1)} \dots \theta^{(n_u)}$:

$$\min_{\theta^{(1)} \dots \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

↳ Given $\theta^{(1)} \dots \theta^{(n_u)}$ estimate $x^{(1)} \dots x^{(n_m)}$:

$$\min_{x^{(1)} \dots x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

↳ Minimizing $x^{(1)} \dots x^{(n_m)}$ and $\theta^{(1)} \dots \theta^{(n_u)}$ simultaneously :

$$J(x^{(1)} \dots x^{(n_m)}, \theta^{(1)} \dots \theta^{(n_u)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{1}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

$$\min_{\substack{x^{(1)} \dots x^{(n_m)} \\ \theta^{(1)} \dots \theta^{(n_u)}}} J(x^{(1)} \dots x^{(n_m)}, \theta^{(1)} \dots \theta^{(n_u)})$$

⇒ Collaborative Filtering Algorithm :
i) Initialize $x^{(1)} \dots x^{(n_m)}, \theta^{(1)} \dots \theta^{(n_u)}$ to small random values.
ii) Minimize $J(x^{(1)} \dots x^{(n_m)}, \theta^{(1)} \dots \theta^{(n_u)})$ using gradient descent (or an advanced optimizat^n algorithm). Eg for every $j = 1 \dots n_u$, $i = 1 \dots n_m$,

$$x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

$\frac{\partial}{\partial x_k^{(i)}} J(\dots)$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

$\frac{\partial}{\partial \theta_k^{(j)}} J(\dots)$

There is no x_0 & θ_0.

For a user with parameters θ & a movie with (learned) features x: predict a star rating of $\theta^T x$.
$(\theta^{(j)})^T (x^{(i)})$

⇒ Vectorization : Low Rank Matrix Factorization
What can we do with this algorithm & given one product can u find other products related to it.
Collaborative filtering :

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix} \quad \text{Predicted Ratings} = \begin{bmatrix} (\theta^{(1)})^T x^{(1)} & \cdots & (\theta^{(n_u)})^T x^{(1)} \\ \vdots & & \vdots \\ (\theta^{(1)})^T x^{(n_m)} & \cdots & (\theta^{(n_u)})^T x^{(n_m)} \end{bmatrix} = (i,j) \to (\theta^{(j)})^T (x^{(i)})$$

$r(i,j)$

$$X = \begin{bmatrix} \cdots & (x^{(1)})^T & \cdots \\ & \vdots & \\ \cdots & (x^{(n_m)})^T & \cdots \end{bmatrix} \quad \Theta = \begin{bmatrix} \cdots & (\theta^{(1)})^T & \cdots \\ & \vdots & \\ \cdots & (\theta^{(n_u)})^T & \cdots \end{bmatrix}$$

∴ we can compute
predicted rating $= X \cdot \Theta^T$
low rank matrix factorization.

⇒ Finding related movies:
For each product i, we learn a feature vector $x^{(i)} \in \mathbb{R}^n$
x_1 = romance ; x_2 = action ; x_3 = comedy ; x_4 = ...
↳ How to find movies j related to movie i?
Small $\|x^{(i)} - x^{(j)}\|$ ∴ movies i & j are similar
↳ 5 most similar movies to movie i:
Find the 5 movies j with the smallest $\|x^{(i)} - x^{(j)}\|$

⇒ Implementational Detail: Mean Normalization

User who have not rated any movies :

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)	Eve(5)
Love at last	5	5	0	0	? ∘ 2.5
Romance forever	5	?	?	0	? ∘ 2.5
Cute Love	?	4	0	?	? ∘ 2
Car Chase	0	0	5	4	? ∘ 2.25
Swords v/s Karate	0	0	5	?	? ∘ 1.25

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

$$\min_{x^{(1)}\dots x^{(n_m)},\ \theta^{(1)}\dots\theta^{(n_u)}} \frac{1}{2}\sum_{(i,j):r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}\left(x_k^{(i)}\right)^2 + \left[\frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}\left(\theta_k^{(j)}\right)^2\right]$$

No role for eve as he has not rated any movie

for $n=2$; $\theta^{(5)} \in R^2$ for Eve (5)

$$\frac{\lambda}{2}\left[\left(\theta_1^{(5)}\right)^2 + \left(\theta_2^{(5)}\right)^2\right]$$

if we minimize it we will get $\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

∴ We must do $(\theta^{(5)})^T x^{(i)} = 0$

~~So~~ we will rate is 0 everywhere ✗ Not good

⇒ Mean normalization :

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}, \quad \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}$$

$Y - \mu \rightarrow Y =$

$$\begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

Now we will use this Y as and learn it $\theta^{(j)}$ & $x^{(i)}$

For user j on movie i predict

$(\theta^{(j)})^T (x^{(i)}) + \mu_i$

User 5 (Eve): $\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\therefore (\theta^{(5)})^T (x^{(i)}) \rightarrow 0 + \mu_i$

∴ We can say that he will predict the average of all ratings

Week - 10

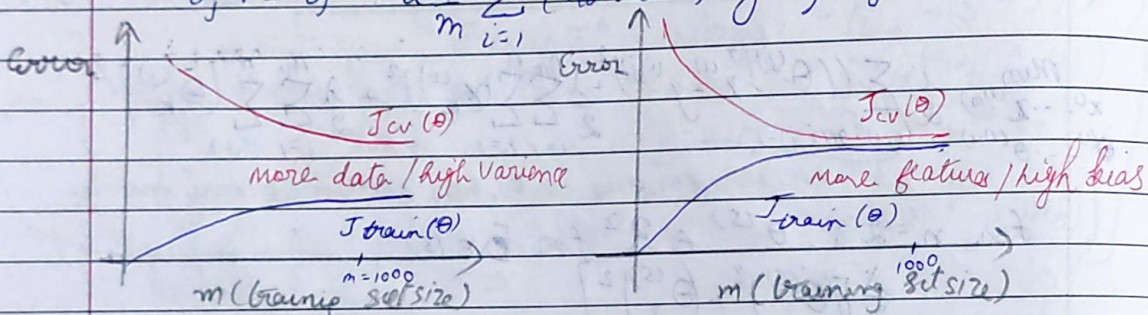
* **Gradient Descent with large Datasets :**

"It's not who has best algorithm that wins
It's who has the most data".

⇒ **Learning with large Datasets :**

$m = 100,000,000$ Census Example

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



If we have millions of data we can plot learning curves to check wether more data is required or more features are required.

⇒ **Stochastic Gradient Descent :**

this is a modification of gradient descent for big m.

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

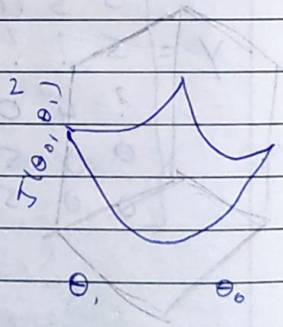
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$\frac{2}{2\theta} J_{train}(\theta)$

Repeat {

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

(for every $j = 0, \cdots n$) } If m is large this will take long
say $m = 300000000$

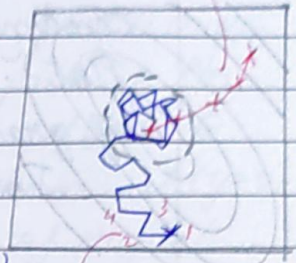


→ Batch Gradient Descent

Stochastic Gradient Descent:

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

$$J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$



i) Randomly shuffle dataset
ii) Repeat { → Repeat it say (1 - 100) for $i = 1 \cdots m$ {

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

(for $j = 0 \cdots n$) }} → $\frac{\partial}{\partial \theta_j} cost(\theta, (x^{(i)}, y^{(i)}))$

$(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$ ⋯→ ⋯ Speeds up

Instead of summing up in batch GD here we are iterating & improving with each step. Not accurate but gives a value surrounding it.

⇒ **Mini-batch gradient descent:**

Batch GD: use all m examples in each iteration
Stochastic GD: use 1 example in each iteration
Mini-batch GD: Use b examples in each iteration.

b = mini-batch size.
b can be 10 - 100

$m = 1000$; Get $b = 10$ examples $(x^{(i)}, y^{(i)}) \cdots (x^{(i+9)}, y^{(i+9)})$

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) \, x_j^{(k)}$$

$i := i + 10$

⇒ **Mini-batch gradient descent:**

Say $b = 10$; $m = 1000$.

Repeat {

for $i = 1, 11, 21, 31, \cdots 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) \, x_j^{(k)}$$

(for every $j = 0 \cdots n$) } }

⇒ Stochastic Gradient Descent convergence :
Checking for convergence
Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient Descent.

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

$m = 300,000,000$

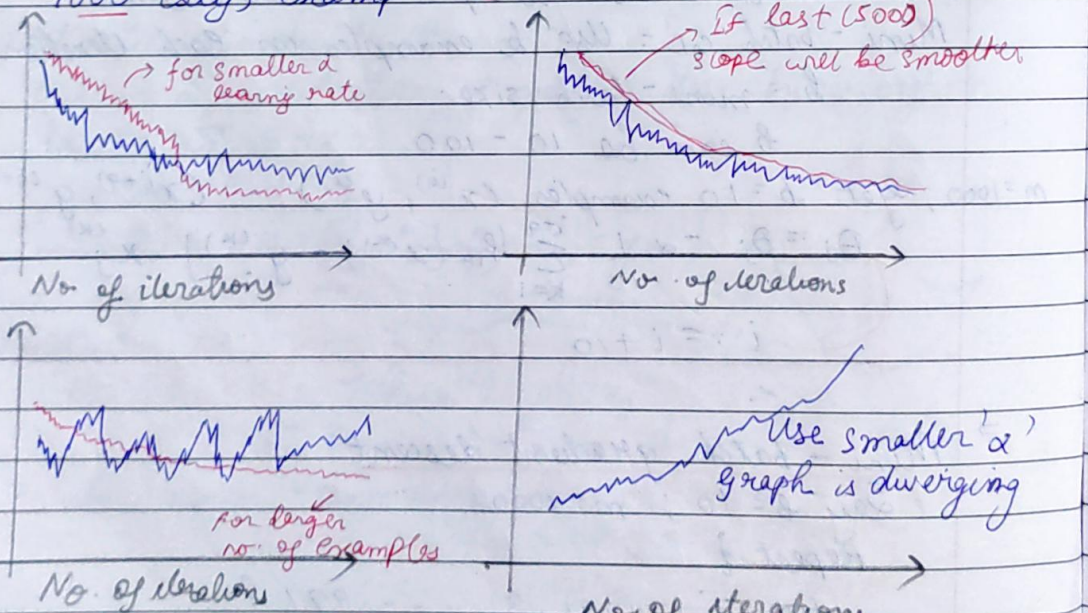
⇒ Stochastic GD :

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

↳ During learning compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$.

↳ Every 1000 iteration (say) plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by algo.

⇒ Checking for convergence :
Plot $cost(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 (say) example.



Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge [Eg. $\alpha = \frac{const\ 1}{iterat^n\ number + const\ 2}$]
as iterat^n number ↑ α ↓ so it gets close to global minima

⇒ Online Learning :

↳ Shipping service website where user comes, specifies origin and destination, you offer to ship their package for some asking price and users sometimes choose to use your shipping service ($y=1$), sometimes not ($y=0$).

↳ Features x capture properties of user of origin/destination and asking price. We want to learn $p(y=1 \mid x;\theta)$ to optimize price.

(price) ← we can use logistic regression

Repeat forever {
  Get (x,y) corresponding to user.
  Update θ using (x,y) — we use this update it
  $\theta_j := \theta_j - \alpha (h_\theta(x) - y) x_j \quad (j = 0 \cdots n)$
}

↳ Can adapt to changing user preferences or else update wont be good.

⇒ Online learning Example :

Product search (learning to search)

User searches for "Android phone 1080p camera".

Have 100 phones in store. We will return 10 results

↳ $y=1$ if user clicks on link, $y=0$ otherwise

Learn $p(y=1 \mid x;\theta)$ } predicted click through rate (CTR)

↳ Use to show user the 10 phones they're most likely to click on.

Other examples : Choosing special offers to show user, customized selection of news articles; product recommendation ...

⇒ Map Reduce and Data Parallelism :

Batch gradient descent: $m = 400$ this can be applied to 4×10^9

$$\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Machine 1 : Use $(x^{(1)}, y^{(1)}) \cdots (x^{(400)}, y^{(400)})$

$$temp_j^{(1)} = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Machine 2 : Use $(x^{(101)}, y^{(101)}) \cdots (x^{(200)}, y^{(200)})$

$$temp_j^{(2)} = \sum_{i=101}^{200} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Say $m = 400$ $(x^{(m)}, y^{(m)})$

Same for Machine 3 & 4.

$temp_j^{(1)}, temp_j^{(2)}, temp_j^{(3)}, temp_j^{(4)}$ — combine

$$\theta_j := \theta_j - \alpha \frac{1}{400} (temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)})$$

i.e sum of all the parts

for $(j = 0, \cdots n)$

⇒ Map - reduce :

Training set in Equal parts → Computer 1, Computer 2, Computer 3, Computer 4 → Combine results $\sum_{i=1}^{400} (\cdots)$

⇒ Map - reduce and summation over the training set :

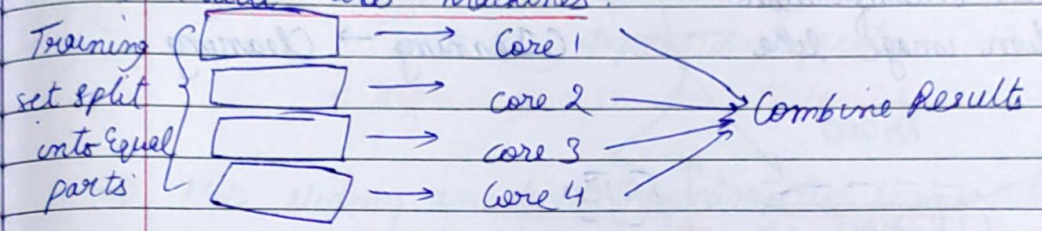
Many learning algorithms can be expressed as computing sums of functions over the training set.

Ex for advanced optimization with logistic regression need :

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

↳ $\frac{\partial}{\partial \theta_j} J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

⇒ Multi-core Machines:



Week - 11

★ Photo OCR : Photo optical character recognition.

⇒ Photo OCR Pipeline :-

i) Text detection
ii) Character Segmentation → PUB I - MALL
iii) Character Classification → P → P U → u B → B
iv) Correction image like → C1eaning → Cleaning

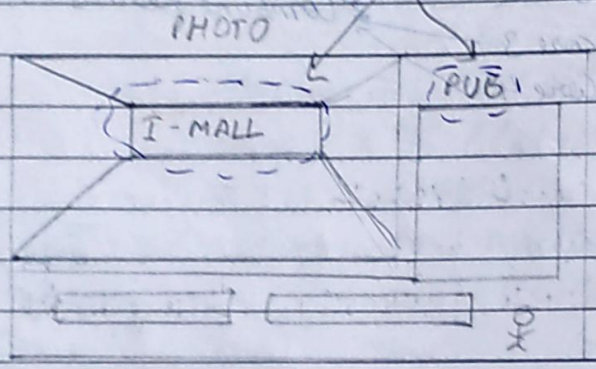
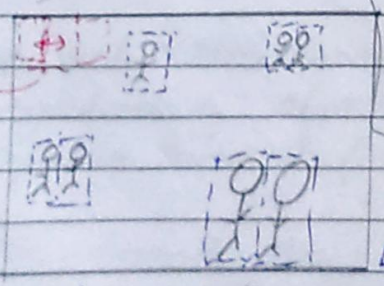


Photo OCR Pipeline
- Image
- Text detection 1-5 engineers
- Character Segmentat^{n} 1-5 engineers
- Character Recognition 1-5 engineers

⇒ Sliding Windows :-



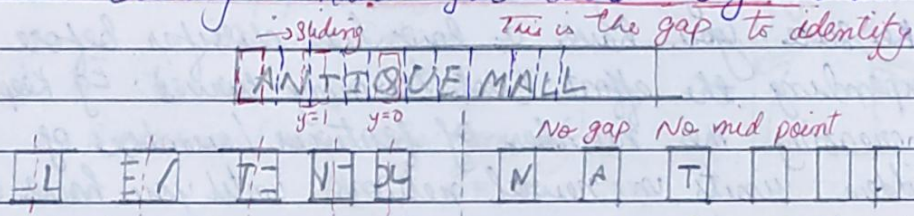
detecting Pedestrians.

First we give computer a lot of images of pedestrians & non-pedestrians. Then we make a small window and move it around the photo & try to detect pedestrians
↳ We can select different size of windows too & it identifies pedestrians.

↳ We can also perform it for text detect" like give train your computer with lots of text & non-text as training set. Then it identifies the text from the image.

* This is how it is identified
Using Expansion operator
* After Expansion easy to identify

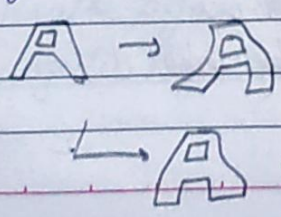
⇒ 1-D sliding window for character segmentation :



Positive Examples (y=1) Negative Examples (y=0)
Train it and then test it on Antique Mall using Sliding window.

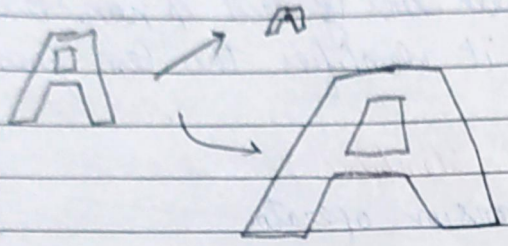
⇒ Getting lots of Data and Artificial Data :
↳ Character Recognition :
* Letter Recognition using different fonts too
↳ By synthesizing by introducing distortion.
* Using distortion to create more training set
↳ Synthesizing data by introduct" distort" : Speech Recognition
* Speed addit" to create more training set

↳ Distortions introduced should be representation of the type of noise / distortions in the test set :



These two can be a good training set, for audio we can add background noise, beeps ...

↳ Usually does not help to add purely random/ meaningless noise to your data.



x_i = intensity (brightness) of pixel i
$x_i \leftarrow x_i$ + random noise
By increasing size, intensity won't count as good training set

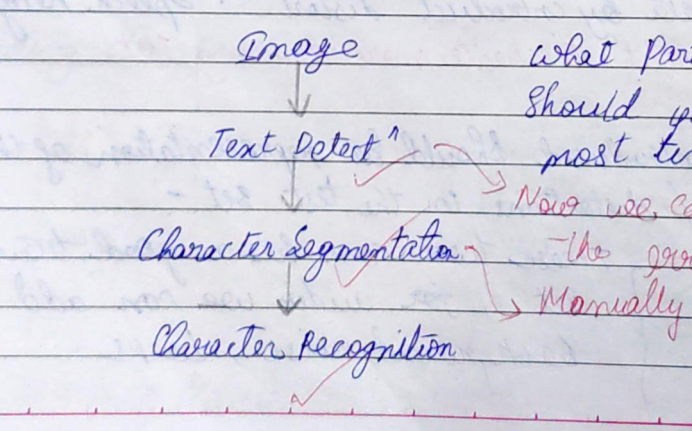
⇒ Discussion on getting more data:

i) Make sure you have a low bias classifier before expending the effort. (Plot learning curves). Eg. Keep increasing the number of features / numbers of hidden units in neural network until you have a low bias classifier.

ii) "How much work would it be to get 10x as much data as we currently have?"

↳ Artificial data synthesis → hours? say m = 1000
↳ collect / label it yourself → 10 sec / example
↳ "Crowd source" (Eg. Amazon Mechanical Turk) m = 10,000 set few days may be

⇒ Ceiling Analysis: What part of the pipeline to work on next?
Estimating the errors due to each component (ceiling Analysis)



Component	Accuracy	(for overall system after correcting the pipelines)
Overall System	72%	
Text detection	89%	↓ 17%
Character Segmentation	90%	↓ 1%
Character Recognition	100%	↓ 10%

⇒ Another ceiling analysis Example :
Face recognition from images (Artificial Example)

Let's make a pipeline first :

Camera Image

↓

Preprocess (remove Background)

↓

Face Detection

↓

Eyes Segmentation | Nose Segmentation | Mouth Segmentation

↓

Logistic Regression

↓

Label

Component	Accuracy	
Overall System	85%	↓ 0.1% → Not a worth thing to spend on time
Preprocess (Remove backg^n)	85.1%	↓ 5.9%
Face Detection	91%	↓ 4%
Eyes Segmentation	95%	↓ 1%
Nose Segmentat^n	96%	↓ 1%
Mouth Segmentat^n	97%	↓ 3%
Logistic Regression	100%	

★ Summary :

i) Supervised learning :- $(x^{(i)}, y^{(i)})$
↳ Linear Regression, logistic Regression, neural Networks, SVM's

ii) Unsupervised learning :- $(x^{(i)})$
↳ K-Means, PCA, Anomaly Detection.

iii) Special Applications / Special topics :-
↳ Recommender Systems, Large scale Machine learning.

iv) Advice on building a machine learning system.
↳ Bias / variance, regularization, deciding what to work on next : evaluation of learning algorithms, learning curves, error analysis, ceiling analysis.

THANK YOU SIR :- Andrew Ng