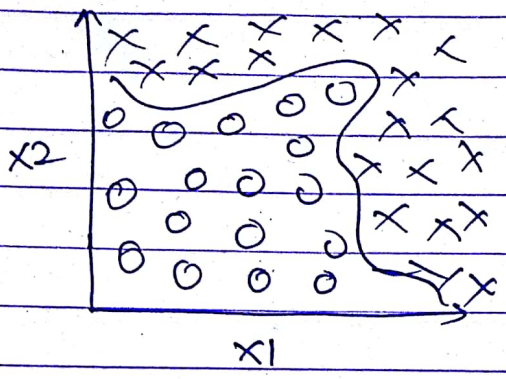
Unit - II

Supervised learning:

Neural Networks

Considering a Non linear classification ex.



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

considering all quadratic terms

$x_1^2, x_1x_2, x_1x_3 \dots x_1x_{100}$ $\left.\vphantom{x}\right\} \approx \frac{n^2}{2}$

we get such 5000 features

which is huge set of features and may over fit the data it is not a generalized equation this increases implementing complexity

∴ Including only subset of these features

ie $x_1^2, x_2^2, x_3^2, x_4^2 \dots x_{100}^2$

No. of features are reduced ie 100

these features are very less and may not fit the data

considering 3rd order polynomial feature

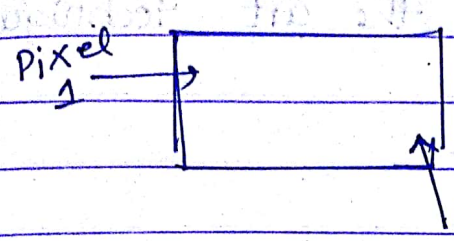
$x_1x_2x_3, x_1^2x_2, x_1x_3x_4, \dots$

the features may range to 170,000

the features dramatically increases and additional features may cause problem and there may be issue of overfitting the hypothesis

machine reducing n reduces the features but most of the ML examples have large value of n

ex:-



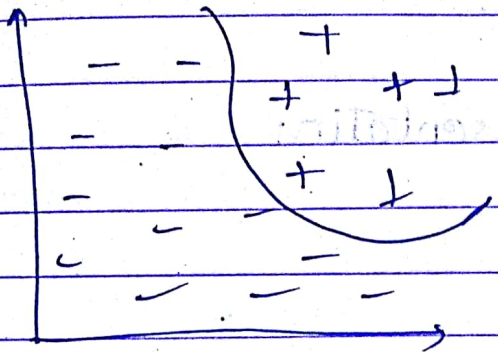


image pixels 50 × 50 — 2500 pixels

n = 2500 — gray scale

7500 if RGB

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{" 2 "} \\ \text{" 3 "} \\ \vdots \\ \text{" 2500 "} \end{bmatrix}$$

Quadratic features $(x_i x_j)$ ≈ 3 million features

Neural Networks

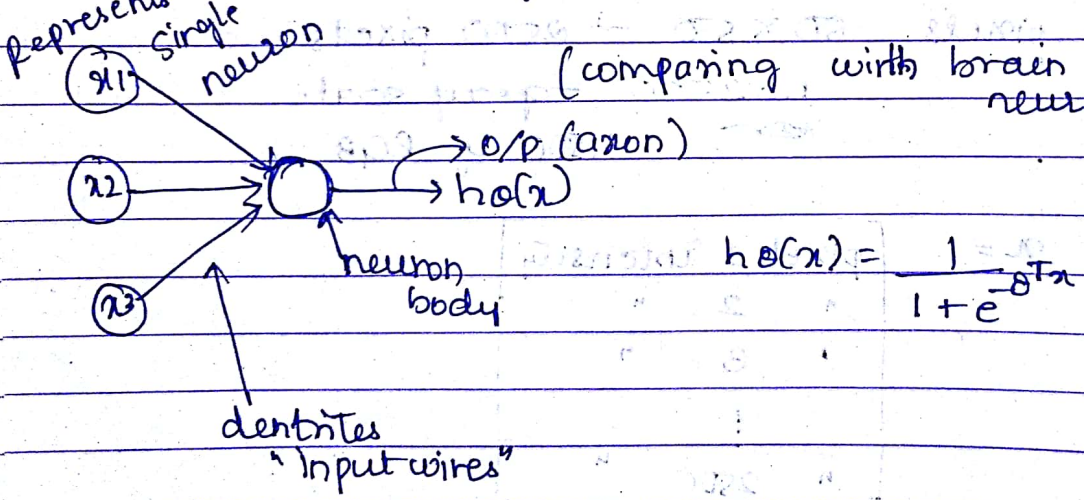
→ Algorithms that try to mimic the brain
→ widely used in 80's and early 90's
→ diminished in late 90's
→ Recent resurgence: state of the art technique
- for many applications

Neural rewiring exp (ppt)

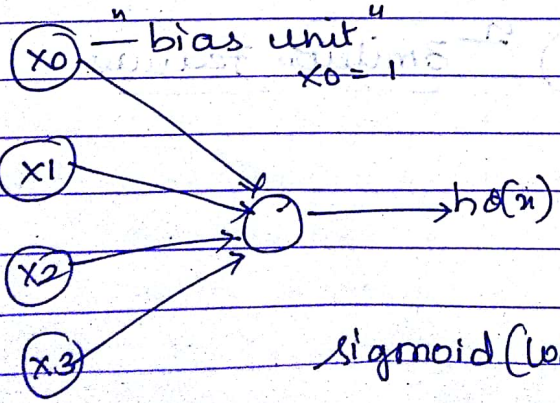
Nueral Networks: Model Representation

* Neuron in brain (ppt)

Neuron model: logistic unit



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

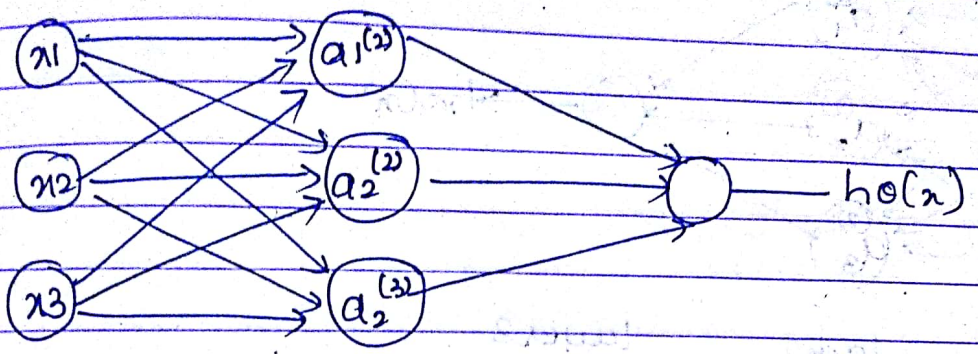


Sigmoid (logistic) activation function

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

Neural N/w

different neurons strung together



layer 1 → "input layer"

layer 2 → "hidden layer"

layer → "output layer"

cannot be observed

can be more than one

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

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

Mathematical definition

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

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

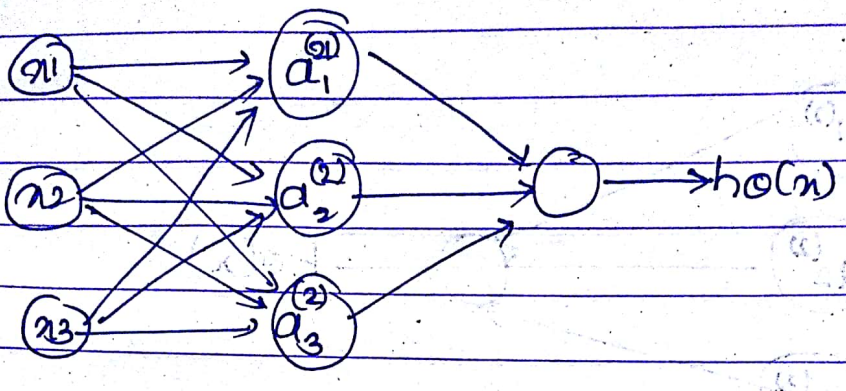
$$a_3^{(2)} = g(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3)$$

$$h\theta(x) = a_1^{(3)} = g(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)})$$

if n/w has s_j units in layer j,

s_{j+1} units layer j+1 then $\theta^{(j)}$ will be of dimension $(s_{j+1}) \times (s_j + 1)$

Forward propagation: Vectorized implementation



layer 1 layer 2 layer 3

$a^{(1)} = x$

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

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

$a_2^{(2)} = g(\underbrace{\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3}_{z_2^{(2)}})$

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

$a_3^{(2)} = g(\underbrace{\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3}_{z_3^{(2)}})$

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

$$h_\Theta(x) = g\underbrace{\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12} a_2^{(2)} + \Theta_{13} a_3^{(2)}\right)}_{z_1^{(3)}}$$

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

layer 2

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

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

~~$h_\Theta(x) =$~~

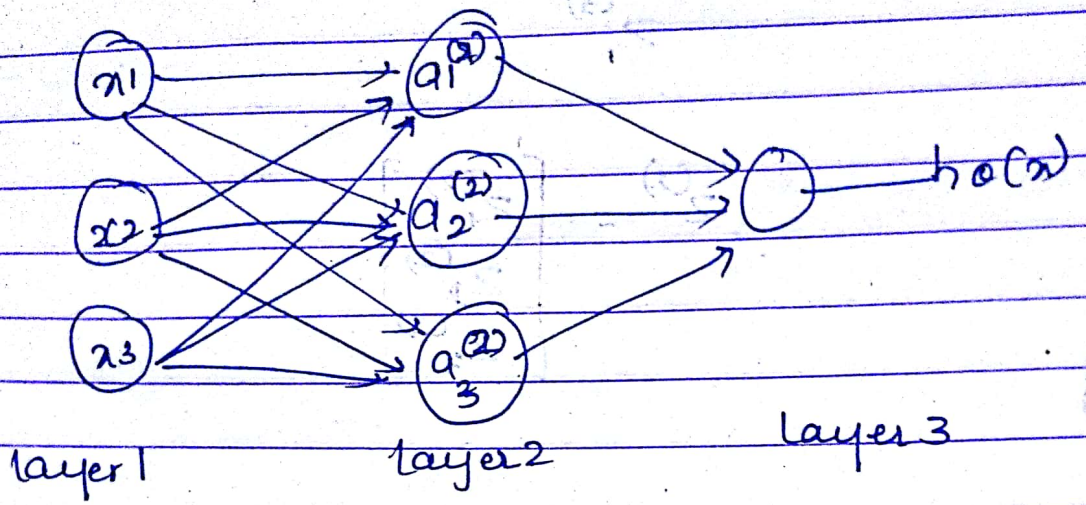
layer 3

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

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

Forward propogation because it is started of by activating the input unit then the hidden layer and the o/p layer

Neural Networks learning its own features



considering only layer 2 and 3.

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

→ logistic Regression

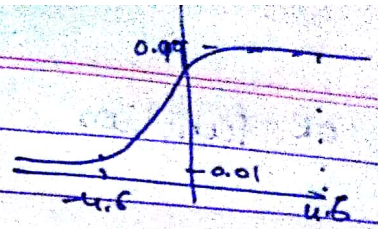
$x_1, x_2, x_3.$

$a_1^{(2)} \quad a_2^{(2)} \quad a_3^{(2)}.$

Instead of taking quadratic equation or any other polynomial and restricting the learning of features

The Θ_j $\Theta^{(1)}$ gives the flexibility of training the algorithm by itself

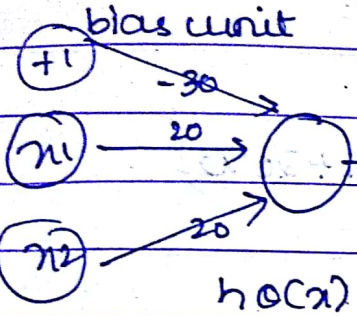
Example.



→ AND

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

$y = x_1$ AND x_2.



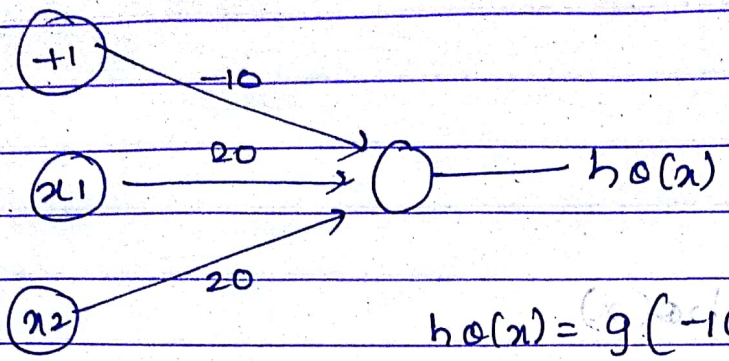
$$h_\Theta(x) = g\left(\overset{(1)}{\Theta}_{10} x_0 + \overset{(1)}{\Theta}_{11} x_1 + \overset{(1)}{\Theta}_{12} x_2\right)$$

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

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	$g(10) \approx 1$

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

OR function

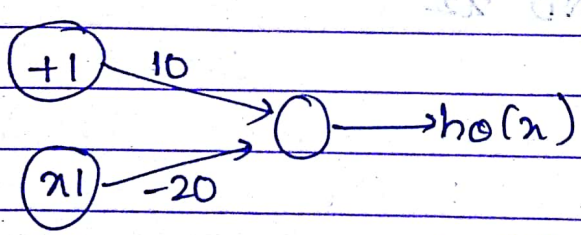


$h_\theta(x) = g(-10 + 20x_1 + 20x_2)$

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	$g(30) \approx 1$

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

NOT function



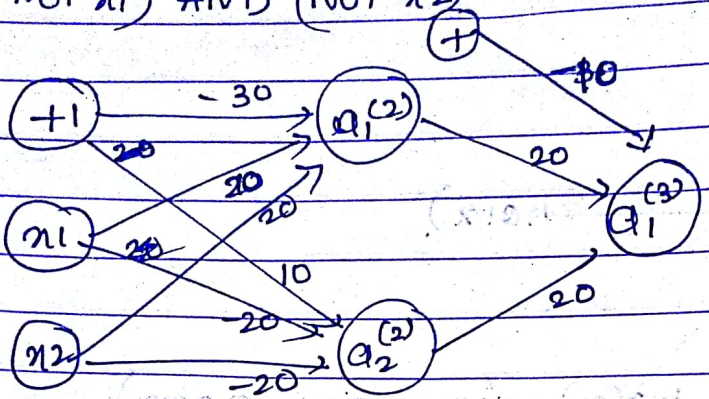
$h_\theta(x) = g(10 - 20x)$

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

$h_\theta(x) \approx$ NOT x_1

x_1 XNOR x_2

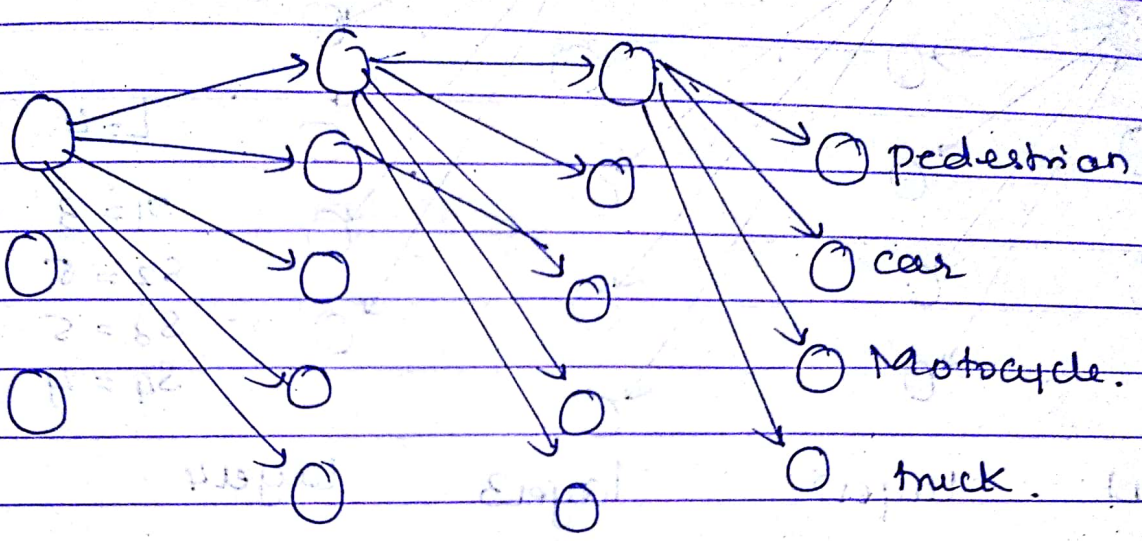
(NOT x_1) AND (NOT x_2)



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

Multi class classification (one-vs-all)

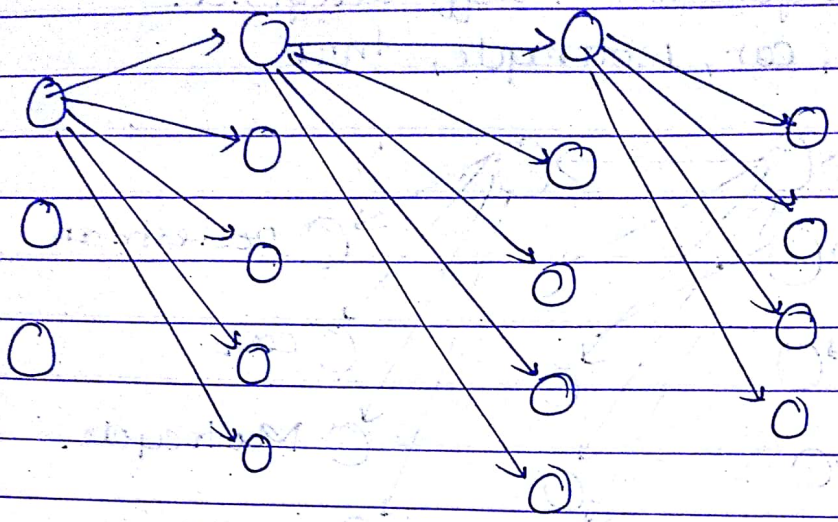
Classifying the images for 4 diff categories.
pedestrian, car, motorcycle, truck.



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

Peder Car Motor cycle truck.

Cost Function



$L = 4$

$s_1 = 3$

$s_2 = 5$

$s_3 = 5$

$s_4 = 4$

layer1 layer2 layer3 layer4

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

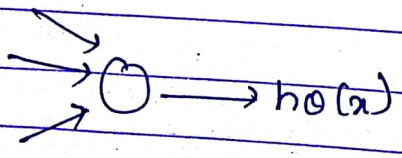
L = total No. of layers

s_l = no. of units in layer l

Binary Classification

$y = 0$ or 1

1 output unit



$s_L = s_1 = 1$

K = also denotes no. of units in layer

Multi class classification (k classes)

if $K \geq 3$ we use one-vs-all method.

Cost function

generalized form of logistic regression.

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

Neural N/w :-

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

Need to compute

$J(\Theta)$

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

Gradient computation

Back propogation algorithm

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

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

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

$\hookrightarrow (h_\Theta(x))_j \quad \delta^{(4)} = a^{(4)} - y$

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

$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* \underline{g'(z^{(2)})}$

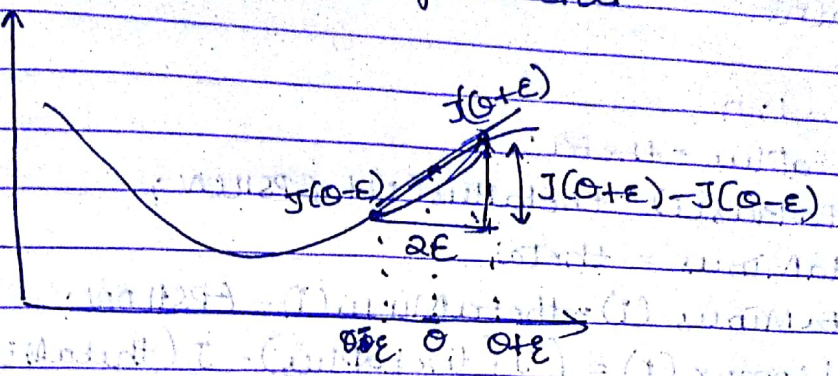
$\hookrightarrow a^{(2)} .* (1 - a^{(2)})$

The name back propogation because the algorithms starts compiling δ from output layer through input layer.

Ignoring λ i.e $\lambda = 0$

$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$

Numerical estimations of gradients



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

$$\varepsilon = 10^{-4}$$

ε value is very small so that the derivative term is approximately equal to the actual slope

Implement:

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

Parameter vector θ.

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

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

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

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

implement

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;

check gradApprox ≈ DVec
                  ↑
            Back propogation

if gradApprox is approx equal to DVec then the algorithm implemented is correct.

Implementation Note
- Implement backprop to compute DVec
- Implement numerical gradient check to compute gradApprox
- Make sure they give similar values
- Turn off gradient checking. Using backprop code for learning

↓

Important

BC sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent (for in the your code will be very slow

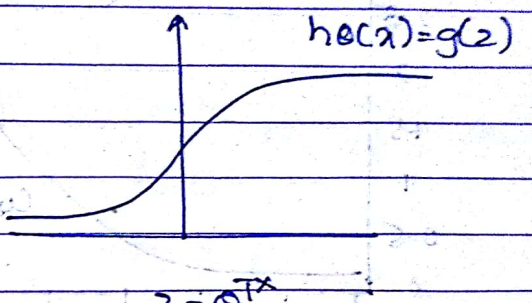
Autonomous vehicle (ex)

Suppor Vector Machines [SVM]

Objective optimization

In logistic regression

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



If $y=1$, then $h_\theta(x) \approx 1$ $\theta^T x >> 0$

If $y=0$ then $h_\theta(x) \approx 0$ $\theta^T x << 0$

Cost function

alternative view (x,y)

$$J(\theta) = -\left(y \log h_\theta(x) + (1-y)\log\left(1-h_\theta(x)\right)\right)$$

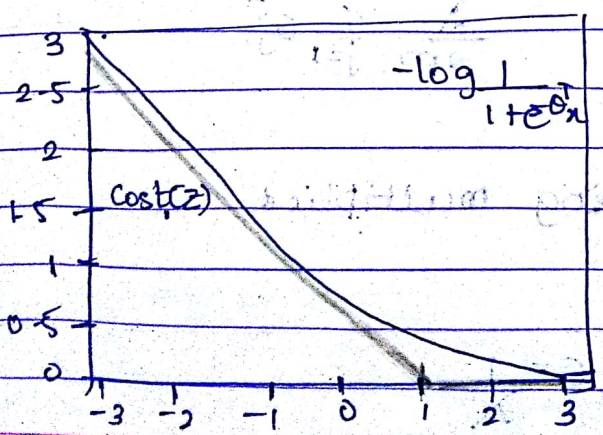
↑ extra term contributed.

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

Considering to scenarios

i) $y=1$

& $(\theta^T x >> 0)$

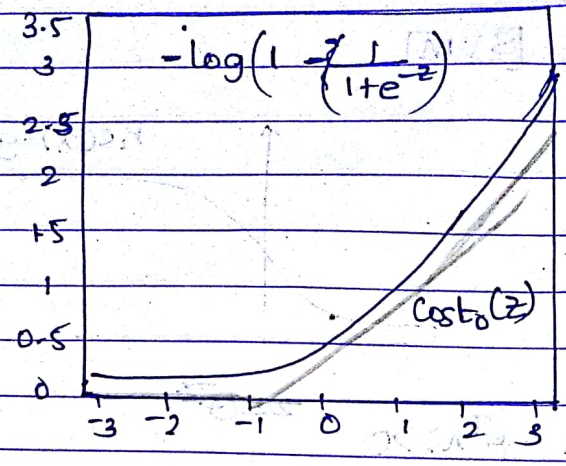


— cost function of SVM

— " " of Logistic Regression

ii) If $y=0$

$\theta^T x << 0$



Logistic regression

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

SVM:

$$\min_{\theta} \frac{1}{m}\sum_{i=1}^{m} y^{(i)} Cost_1(\theta^T x^{(i)}) + \left(1-y^{(i)}\right) Cost_0\theta^T x^{(i)} + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

m - constant value being multiplied

$A + \lambda B$

$CA + B$ i.e. $C = \frac{1}{\lambda}$

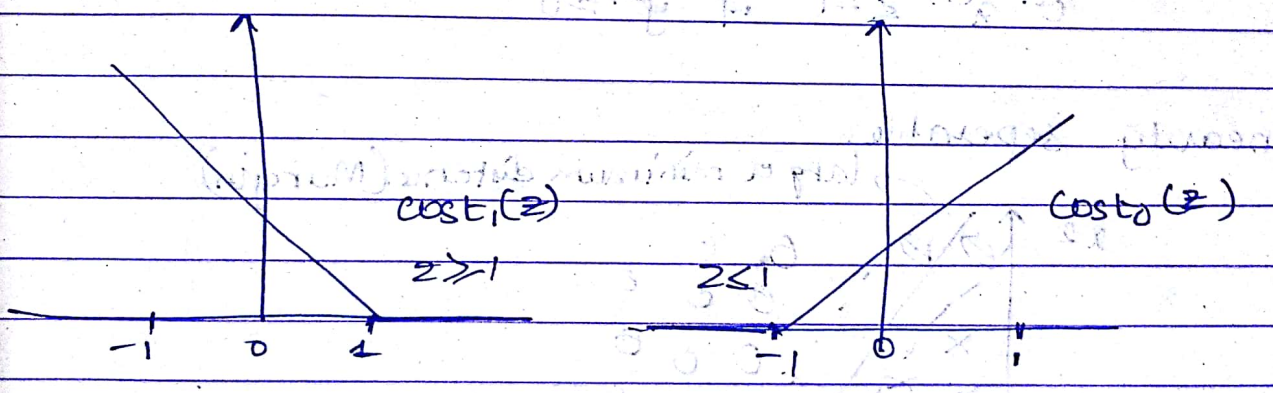
∴ Removing m and adding C

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

hypothesis

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

Large Margin Intution



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

if $y=0$ $(\theta^T x \leq 1)$ not just < 0

~~SVM decision Boundary~~

~~consider very large C~~

~~C = 100,000~~

~~not convex~~

SVM Decision Boundary

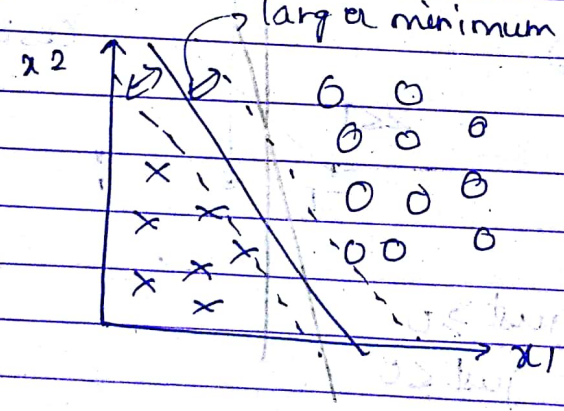
$$\min_\theta C \underbrace{\sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)})}_{0} + \frac{1}{2}\sum_{i=1}^{n} \theta_j^2$$

$$\min_\theta C \times 0 + \frac{1}{2}\sum_{i=1}^{n} \theta_j^2$$

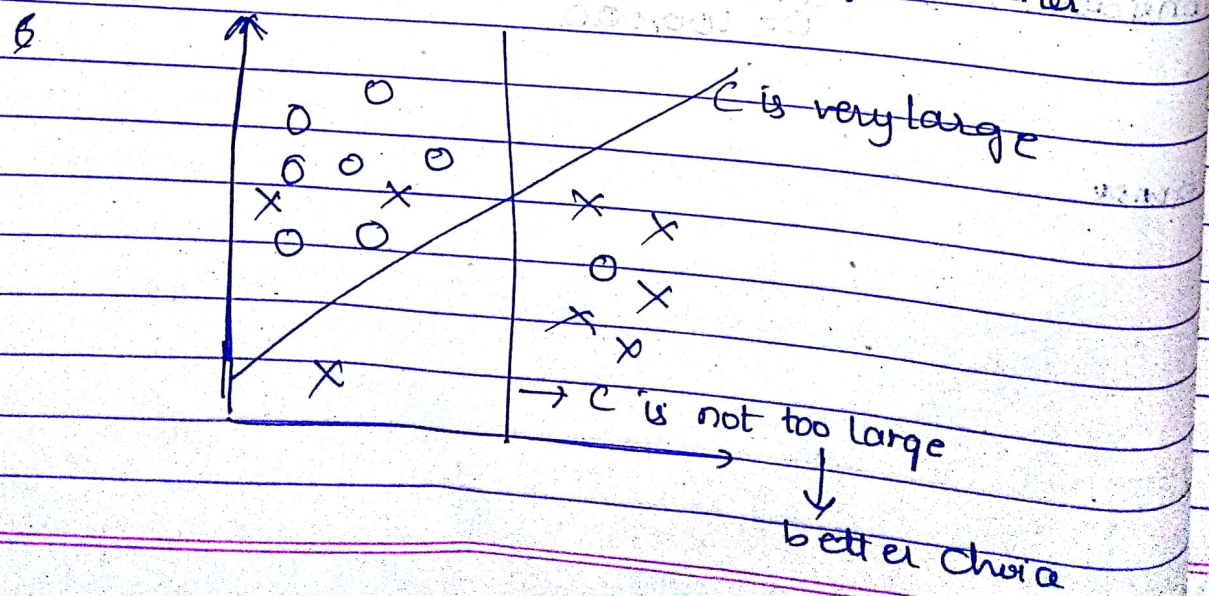
s.t. $\theta^T x^{(i)} \geq 1$ if $y^{(i)} = 1$

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

linearly seperable.

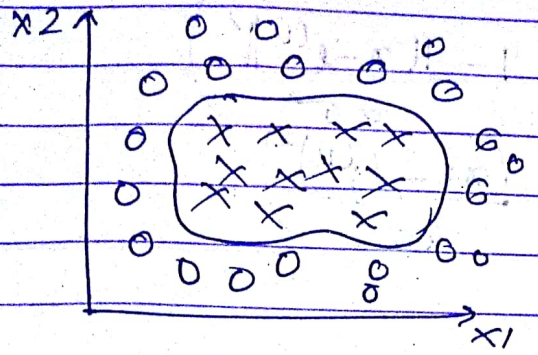


large margin classifier in presence of outliner



$\|w\|$
↳ length of vector w

Kernels I

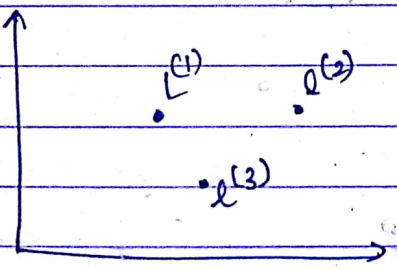


predict $y=1$ if
$\theta_0 + \theta_1 x + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$

$h_\theta(x) = \begin{cases} 1 & \text{if quadratic} \\ 0 & \text{otherwise.} \end{cases}$

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

Given x, compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$



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

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

$f_3 = \text{similarity}(x, l^{(3)}) = \exp\left(-\frac{\|x - l^{(3)}\|^2}{2\sigma^2}\right)$

↳ kernel
(Gaussian kernel)

Kernels and similarity

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

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

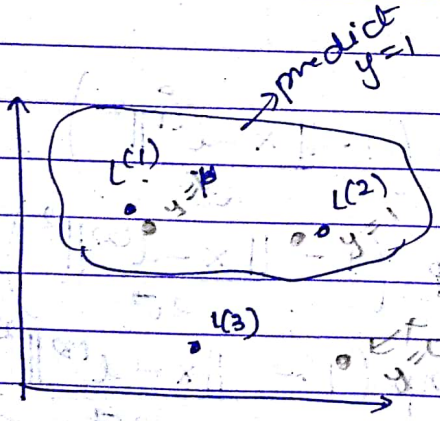
If $x \approx l^{(1)}$:

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

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

$$f = \exp\left(-\frac{(\text{large no})^2}{2\sigma^2}\right) \approx 0$$

Ex:



predict 1 when

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

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

$f_1 \approx 1 \quad f_2 \approx 0 \quad f_3 \approx 0$

$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 = -0.5 + 1 = 0.5 \geq 0$

predict $y = 1$

$f_1 \approx f_2 \approx f_3 \approx 0$

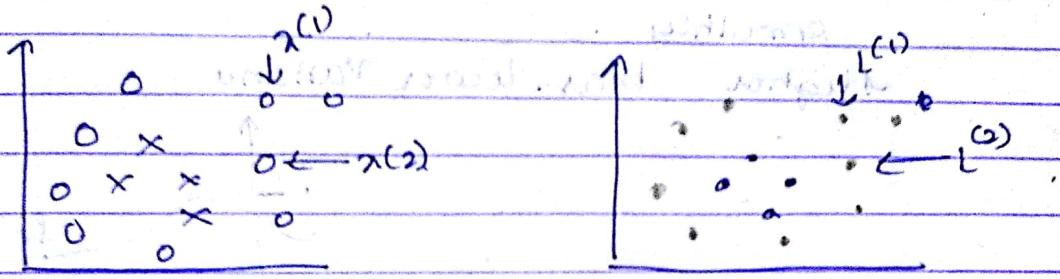
$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 = -0.5 \leq 0$

predict $y=0$

SVM with kernels ||

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

Choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)} \dots l^{(m)} = x^{(m)}$



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

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

$\vdots$

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

$f_1^{(i)} = \text{sim}(x^{(i)}, l^{(i)})$

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

$\vdots$

$f_m^{(i)} = \text{sim}(x^{(i)}, l^{(m)})$

$\rightarrow f_i^{(i)} = \text{sim}(x^{(i)}, l^{(i)}) = e^{0} = 1$

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$$

* half

SVM parameters

$C(=\frac{1}{\lambda})$ (overfitting)

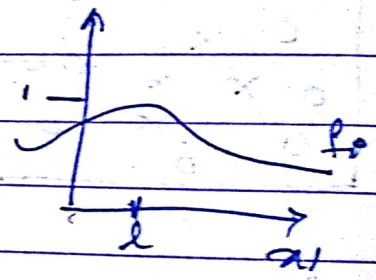
high large C : lower bias, high variance (small λ)

high small C : higher bias, low variance (large λ)

(→ under fitting)

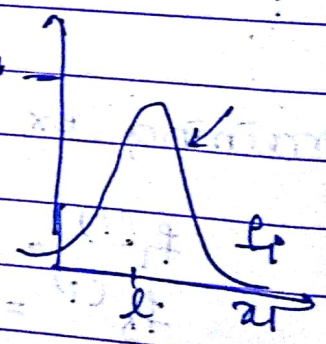
σ^2 : large σ^2 : features f_i vary more smoothly

higher bias. lower variance



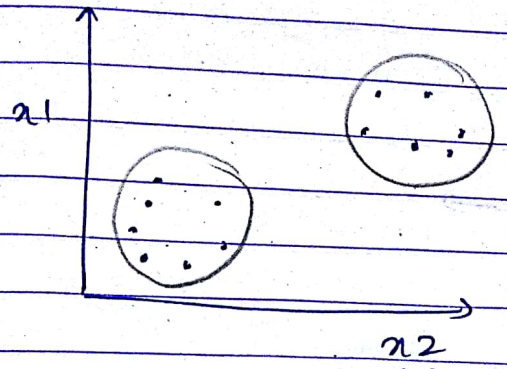
small σ^2 : features f_i vary less smoothly

lower bias high variance



Unsupervised learning

Clustering

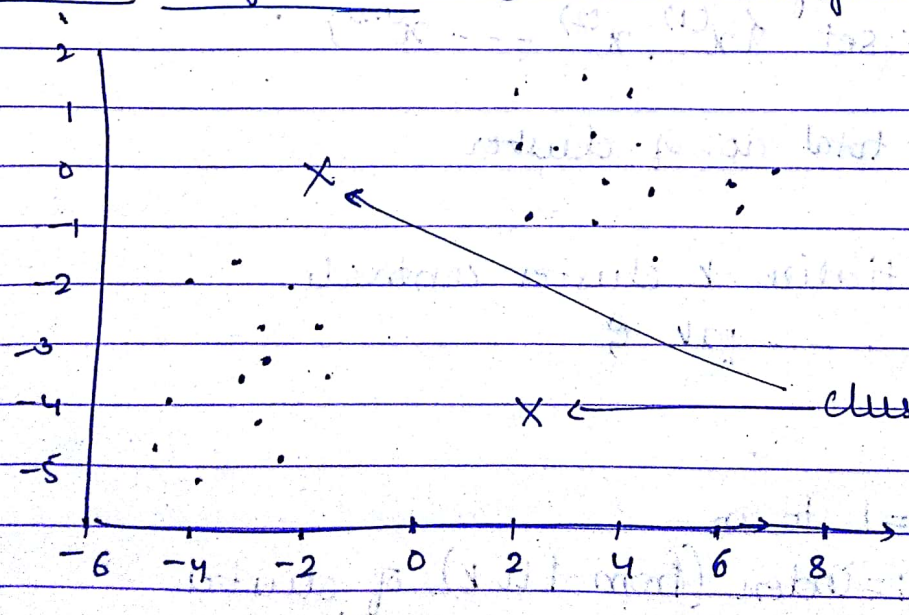


Clustering algorithm.

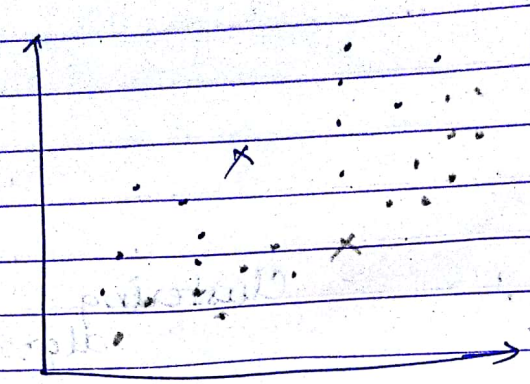
Training set : $\{x^{(1)}, x^{(2)}, x^{(3)} \ldots x^{(m)}\}$

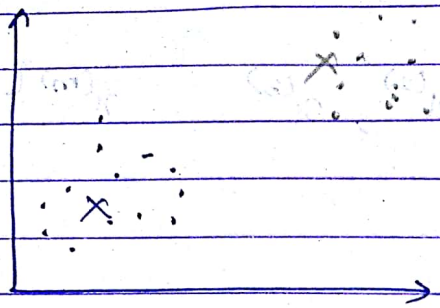
Applications
1) Market segmentation
2) Social N/w analysis
3) Organize computing clusters
4) Astronomical data analysis

K-means Algorithm (Iterative algorithm)



1) Cluster assignment set
2) Move centroid set





K-means algorithm

Input
- K (number of clusters)
Training set $\{x^{(1)}, x^{(2)} \dots x^{(m)}\}$

K - total no. of cluster

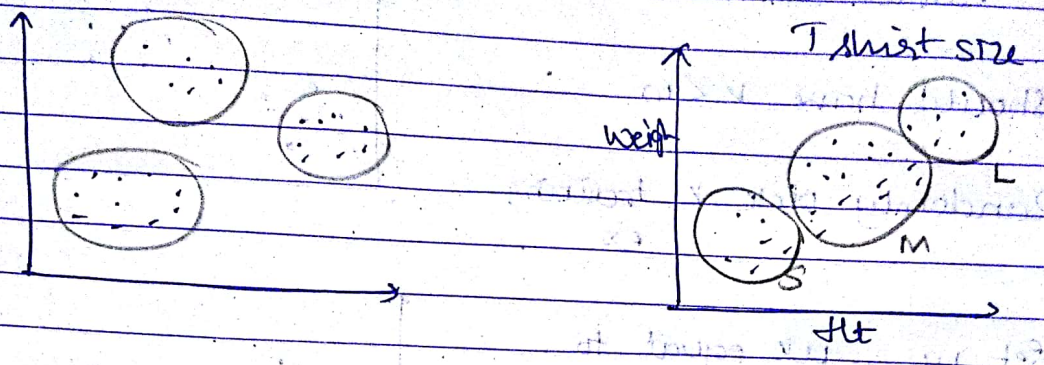
Randomly initialise K cluster centroids
$\mu_1, \mu_2 \dots \mu_K$

Repeat {

cluster assignment set { for i = 1 to m
$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$
↳ minimizing J(...) wrt $c^{(1)}, c^{(2)}, \dots c^{(m)}$

move centroid { for K = 1 to K
μ_K := avg (mean) of points assigned to cluster K
↳ minimize J(...) wrt $\mu_1 \dots \mu_K$

K-means for non separated clusters.



S, M, L.

Optimization objective (for cost function)

$c^{(i)}$ = index of cluster

μ_k = cluster centroid k

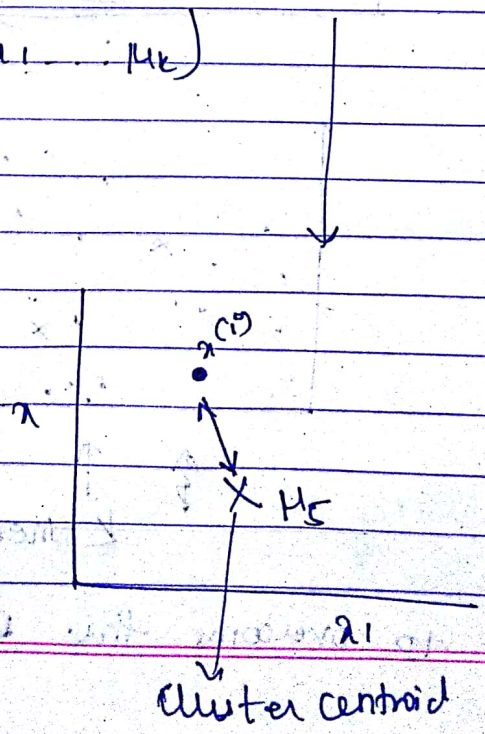
$\mu_{c^{(i)}}$ = cluster centroid of cluster to which ex $x^{(i)}$ has been assigned.

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

Optimization objective:

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

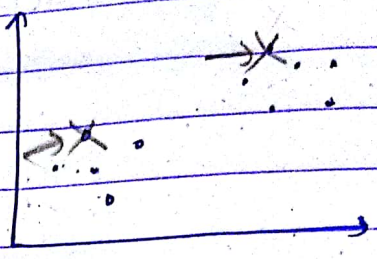
$$\min_{\substack{c^{(1)}, \ldots, c^{(m)} \\ \mu_1 \ldots \mu_k}} J(c^{(1)}, \ldots c^{(m)}, \mu_1 \ldots \mu_k)$$



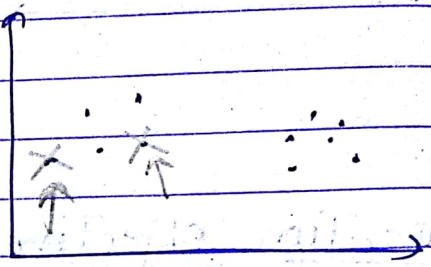
** Random initialization

Should have $K < m$

Randomly pick K training ex



Set $\mu_1 \ldots \mu_K$ equal to these K examples

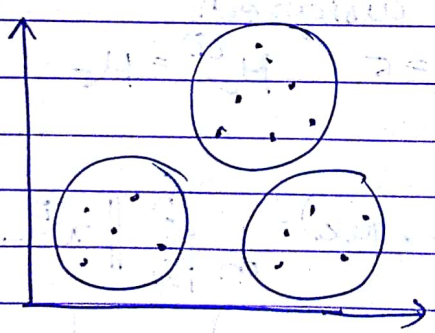


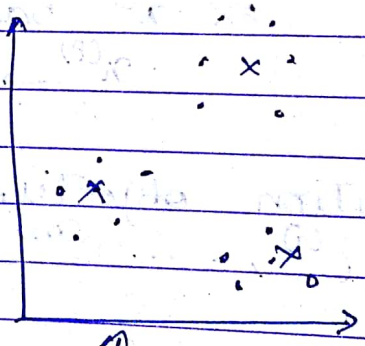
$K = 2$

$\mu_1 = x^{(i)}$

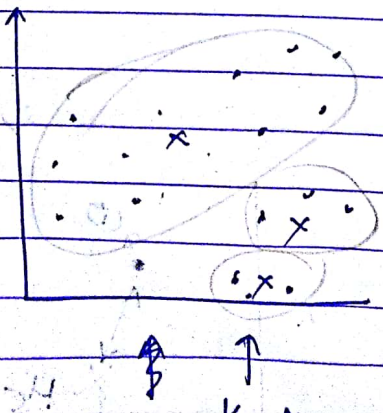
$\mu_2 = x^{(j)}$

Local optima





Best possible



K means cluster is stuck @ a local minima

to overcome this initialize K many times

Random initialization

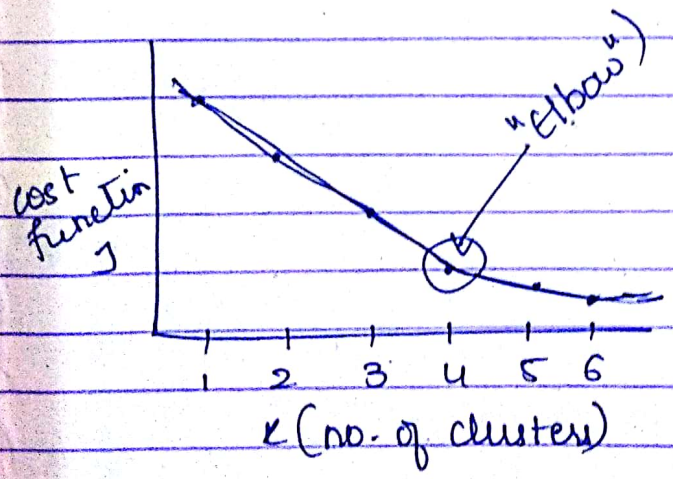
```
For i=1 to 100
{
    Randomly initialize K-means.
    Run K-means.
    Get c^(1) ... c^(m)
        μ1 ... μK
    Compute cost function (distortion)
        J(c^(1) ... c^(m), μ^(1) ... μ^(K))
}
```

Pick clustering that gives lowest cost $J(c^{(1)} \ldots c^{(m)}, \mu^{(1)} \ldots \mu^{(m)})$

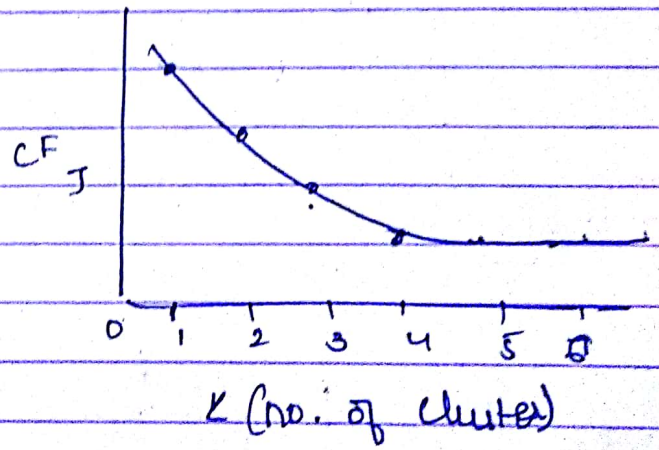
Choosing the number of clusters

→ Manually (visualization)

Elbow Method:



using 4 cluster may be right distortion goes down @ this point



theres no clear elbow there no clear answer

* Metric purpose for running k-means

Eg: