ANOOP SIR

Books-

Pattern classification – Duda Hart & stock

Neural Networks – Simon Haykin

Read- (Prerequisite)

Pro. distribution f(n), Multivariate densities (joint & conditional)

Vectors, spaces, subspaces, Eigen values & vectors

Assignments- 24
Project- 20
Mid(2) - 30
final - 20
others- 6

Tasks in AI

- sensing
- Recognition
- Reasoning
- Acting

Main focus on Recognition-

- Where am I
- what do I see around
- who is this person

- Identification of a pattern as a member of a category we already know or are familiar with (Like image → text)

• A pattern is the opposite of a chaos ; it is an entity vaguely defined, that could be given a name.

→ Intra class variability

- T Y T

Inter class variability (similarity)

- OD 8B 2Z

characters that look similar

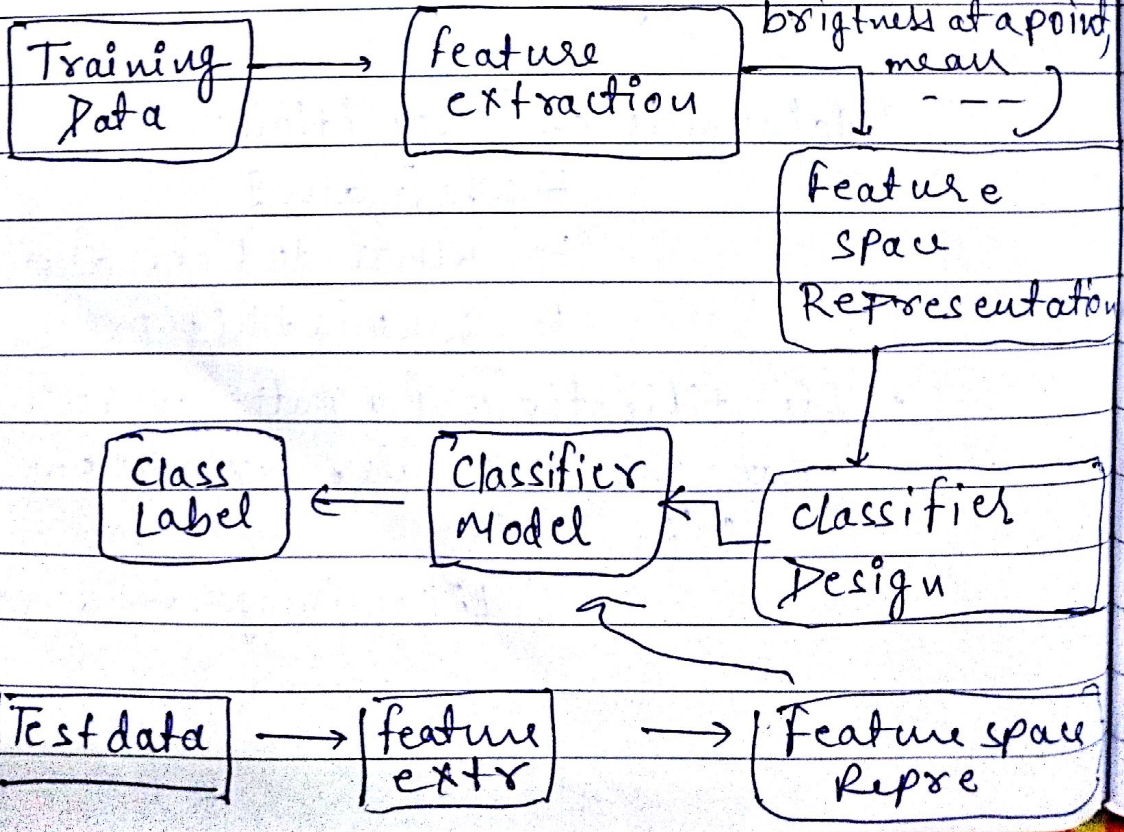
* In recog. we categorize in class.

COGNITION - formation of new classes

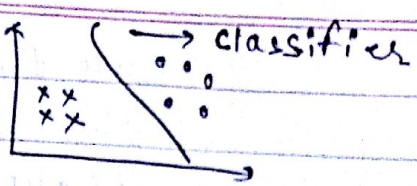
RECOGNITION - known classes

PATTERN RECOGNITION PROCESS :-

(basis's of image can be light, brightness at a point, mean ---)



Like feature space -



08-08-2014

Note:- 22nd Aug. class test

Model

signal (can be email, speech, image - -)

↓

Representation (feature space) $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ (n dimensional features)

↓

learning (Training) (figure out what seperates the classes)

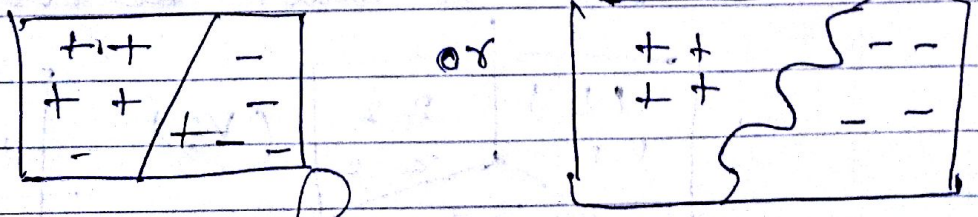
** The Process of Raw data → Representa. is the soul of your system

↓

testing (classification)

I]. LEARNING -

How the seperator Look like



or any other

We use the concept of Generalisation [the generalised curve will have few mistakes but simple representation]

→ 1. Generative Model

M(Apple) or M(orange) (You can generate a real model (apples))

Probability of fruit being apple

being orange

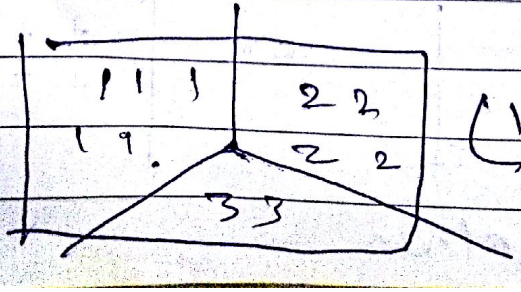
(Note here that our universe is limited to Apple + Oranges)

2. Discriminative Model

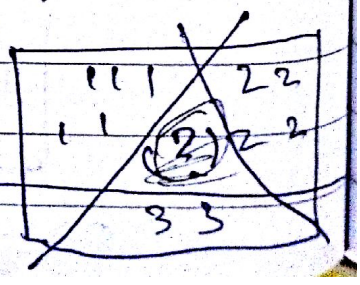
Here we are only interested in discriminating the models (what is the classifier) rather than being generative.

Note:- We can combine models also like we can use pro. of generative models to design a discriminative ones.

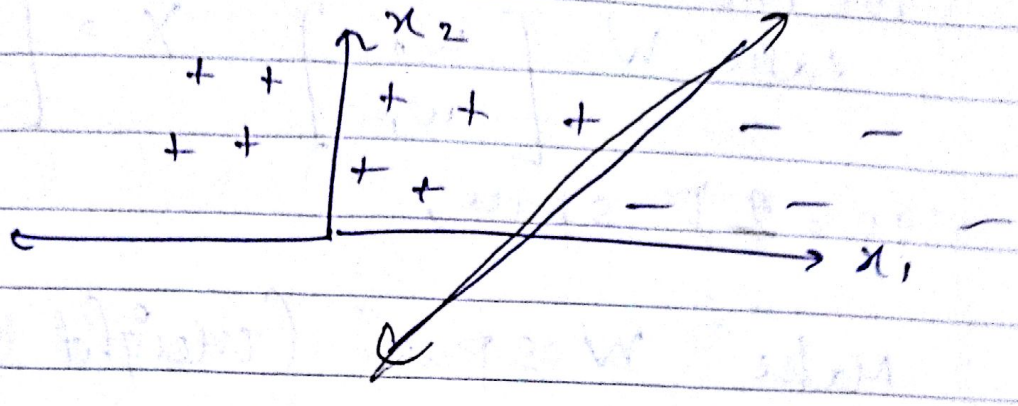
The classifier must be good



(VS)



eg- In 2D space



Line = $w_1 x_1 + w_2 x_2 + w_0 = 0$

$\therefore W = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$ and given a new sample $X = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$

(added 1 for convenience)

We need to put point in eqⁿ, for samples, compute ∴ find W such that

$W^T X > 0$ if ~~*~~ points $\in$ +ve

$W^T X < 0$ if ~~*~~ points $\in$ -ve

[find W?]

W = classifier

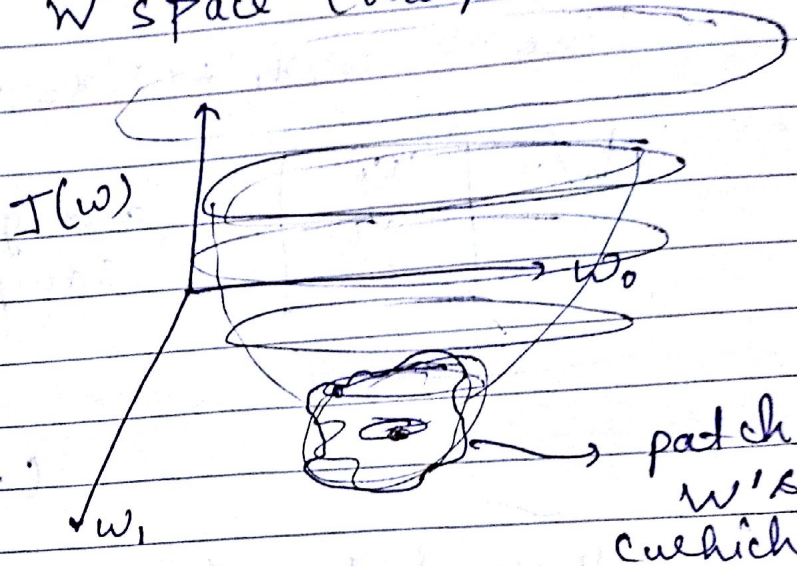
X = feature vector

for find W (best)

take $W = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$ $X = \begin{bmatrix} 1 \\ x_1 \end{bmatrix}$

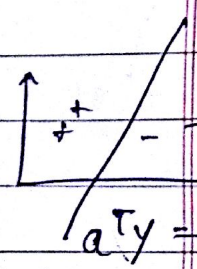
for 2D space,

Make W space (weight space)



We have a function J(w) such that when its value ↓ it comes nearer to patch

Make $a = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$ $y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$



For samples $y_0 = 1$ (if true class)

$y_0 = +1$ (if - ve class and negate all)

Now by doing this

$a^T y > 0$ for all samples (y)

Now we need to find $J(w)$ such that its min. ⟶ Patch

We randomly choose "a"
and Make set λ where the values are misclassified

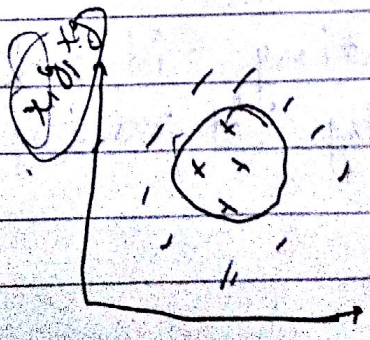
$$J(w) = \sum_{y_i \in \lambda} -a^t y_i$$ (based on that a)

If value of $J(w)$ is very high ⇒ Not a good classifier
so if $J(w) = 0$ ⇒ Yes it is

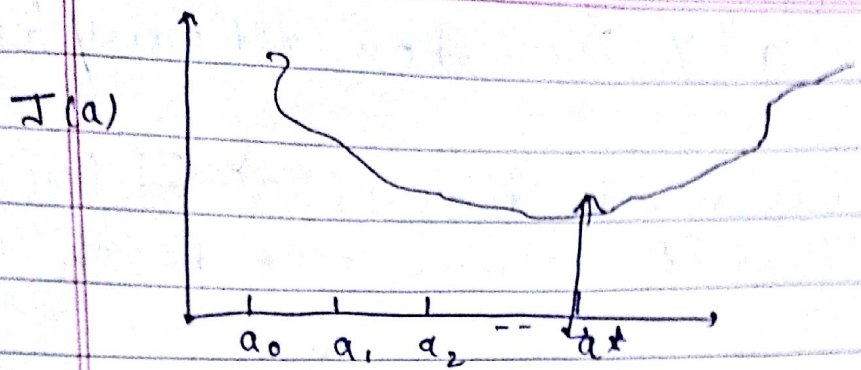
** So after randomly choosing an 'a' and computing 'J(w)' move in the direction of gradient descent to reach good 'a'

Note that 'a' and 'y's are vectors

'Vector Algebra', differentiation etc



12th-Aug-2014



we need to move in the direction of a for which slope is maximum. (its opp. dir)
[it is a vector as it gives direction]

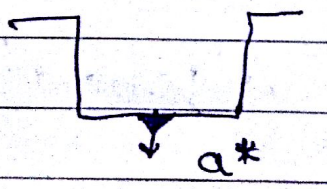
$\nabla J(a)$ is pointing towards left we need to move in descent dir.

$$a(K+1) = a(K) - \eta_K \nabla J(a)_K$$

η = learning rate
(how fast you want to learn)

Loop above eqn until $|\eta \nabla J(a)| < \theta$
$\theta \approx 0.0..1$

* $J(a)$ doesn't work always as

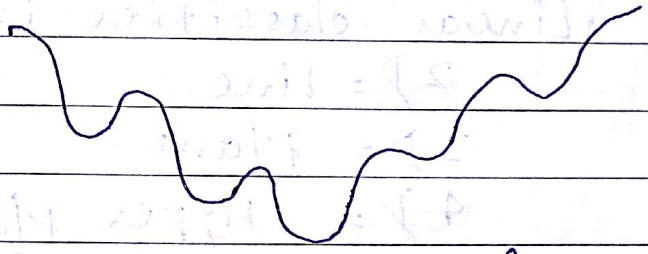


but $\nabla J(a) \to 0$

in the starting itself, however practically it rarely happens.

- if η is constant throughout we might oscillate hence η_k is used, learning is fast in the beginning and reduces as we approach the soln.

- Problem might arise



we might stuck in a local minimum.

Try out different $J(\cdot)$ functions

1. Perceptron

$$J_p(a) = \sum_{y_i \in Y_{MC}} -(a^T y)$$

$$\nabla J_p(a) = \sum_{y_i \in Y_{MC}} -y$$

Dif.. wrt 'a' [Read vector algebra]

So algorithm →

a_0 = init , $k = 0$

do $k = k + 1$

$a(k) = a_{k-1} + \sum_{y_i \in Y_{MC}} y$

until $Y_{MC} = \phi$

"Batch Perceptron Algorithm"

2. Single Sample Perceptron Algorithm:-

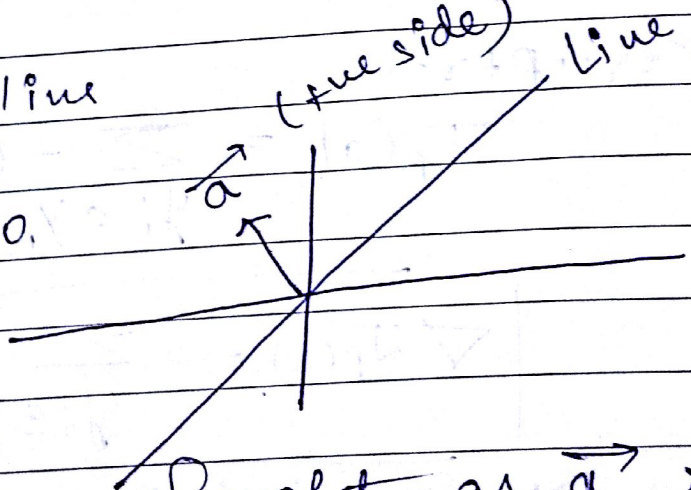
$$[a^{k+1} = a^k + y^{k+1}]$$

where y^{k+1} = any one misclassified sample [It is not power]

→ Linear classifier in
2D = Line
3D = Plane
4D = Hyper Plane
and we will use Hyper planes as a generalised plane for lower or higher dimensions also

eqn of a line

$a^T y = 0$

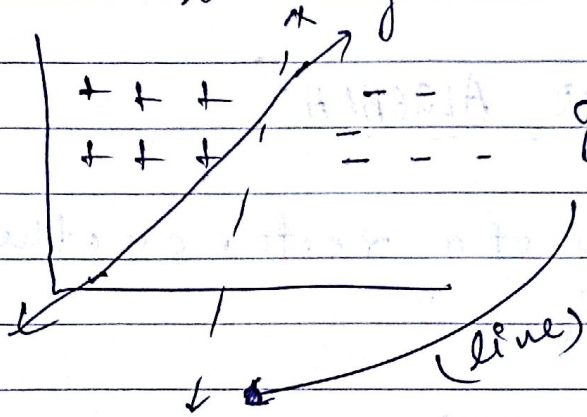


can be thought as $\vec{a}$ passing through origin and ⊥ to line

[Note that for above method we have assumed that such an a^* exist which linearly separates the samples..]

- By this method all a's are moving towards αa^* (Perfectly right)
  α = constant, may not be 1
  ie- Magnitude of difference keeps on decreasing...

⇒ Note that by this algorithm we might find a line just separating the samples



good way to approach

is to have $a^T y > \phi$ (margin) instead of $a^T y > 0$

or you can use different J's

$$J_q(a) = \sum_{y \in Y} (a^t y)^2$$

{ smooth fn }

⇒ slightly preferable.

or use $(a^t y - b)^2$ — will take care of slow landing issues of $(a^t y)^2$

so we have ⇒

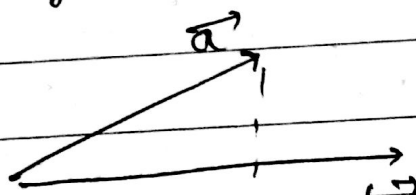
$$J_\gamma(a) = \frac{1}{2} \sum_{y \in Y_{MC}} \frac{(a^t y - b)^2}{\|y\|^2}$$

$$\nabla J_\gamma(a) = \sum_{y \in Y_{MC}} \frac{a^t y - b}{\|y\|^2} \cdot y$$

(Relaxation algorithm)

"VECTOR ALGEBRA"

1. Projection of a vector on other



$$\gamma \rightarrow \left[\frac{\vec{a} \cdot \hat{b}}{}\right]$$

2. A fn is linear if
   - $f(\alpha a) = \alpha f(a)$
   - $f(x+y) = f(x) + f(y)$

So

$$f(\alpha x_1 + \beta x_2) = \alpha f(x_1) + \beta f(x_2)$$

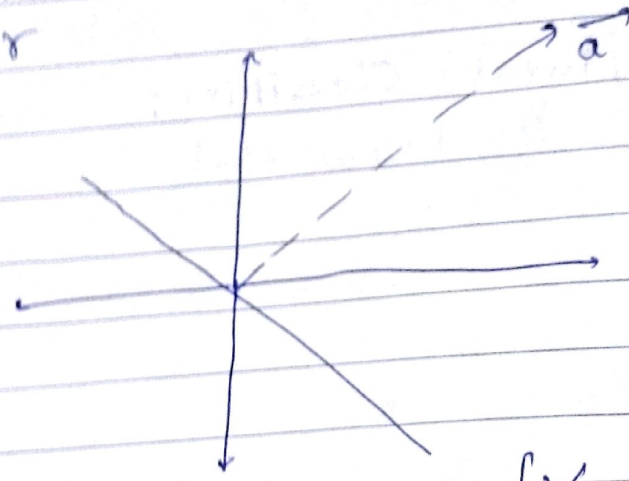
. for making the classifier, select some subset of labeled samples to make the classifier and use the rest for checking the authenticity of the classifier (its accuracy)

3. Linear Classifier:-
You know that.

19-Aug-2014

Note: $b = a^T y_i$ "b is the perpendicular distance between $\vec{a}$ and y_i"

Very IMP:→ If $\vec{a}$ is a solution vector Then $\alpha\vec{a}$ is also a solution vector



So when you are doing $\{\forall_i\ a^t y_i \geq b\}$ b can be any number you can have a solution for

$\forall_i\ a^t y_i \geq 1$, and multiply solution $\hat{\alpha}$ with 'b' to have $b\hat{\alpha}$ as another soln satisfying $\forall i\ a^t y_i \geq b$, [we will find a solution always because if $\hat{\alpha}$ is a soln then $\beta\hat{\alpha}$ is also]

But for $[\forall_i\ a^t y_i = b]$, probably we can't find a soln.

To prove the $[\forall i a^t y_i = b]$ may not give a soln write it as.

(each row is a training solution)

$$\begin{bmatrix} 1 & y_{11} & \cdots & y_{1d} \\ 1 & & & \\ 1 & & & \\ y_{n0} & & & y_{nd} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

$n \times \hat{d}$ $\hat{d} \times 1$ $n \times 1$

$$Ya = b$$

Now if we have $[\forall i a^t y_i = b_i]$, You can find the solution as $\vec{a}$ by

$$\boxed{Ya = b}$$

Y = Augment feature vector Matrix

Now if

$n > d+1$, over determinant ∞ soln

$n = d+1$, unique soln

$n < d+1$, ∞ solution, underdetermined

⇒ So basically $[Y]$ has to be a square matrix $[n = d+1]$

Rows in Y has to be linearly independent for unique soln.

$$\boxed{a = Y^{-1} b}$$

So if we want to find $[\forall_i a^t y_i = b]$ in 2D space

$\Rightarrow d = 3, \quad n = [3+1]$

ie = It will produce a limit on number of points (samples).

Check and Read Book for

$$\boxed{Ya = b}, \{n = d+1\}$$

** But in reality

$[n >> d+1]$, so we need to Look into this, so instead of having

$$Ya = b \quad ---(1)$$

Take

$$e = Ya - b \quad ---(2)$$

and minimise the error 'e'

$$\min (e = Ya - b) \; (\|e\|^2)$$

So we need to Minimise

$$\|e\|^2 = \sum_{i=1}^{n} (a^t y_i - b_i)^2$$

So

$$J_s(a) = \|e\|^2 = \sum_{i=1}^{n} (a^t y_i - b)^2$$

{ This is over all the samples }

So we need to minimise $J_s(a)$ and find a unique solution.

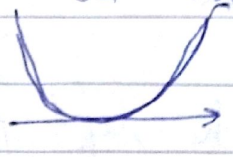
at the minimum of a function its gradient is '0'.

So $\nabla J_s(a) = 0$ { Intuitively it is a min. point ∵ Its map is 0 }

$$\nabla J_s(a) = 2 \sum_{i=1}^{n} (a^t y_i - b_i) y_i = \vec{0}$$

Here $0 = \vec{0}$ as it is (constant) $\vec{y_i}$

so it says $\vec{0}$ at minimum we have



$$2 Y^t (Ya - b) = \vec{0}$$

$$Y^t (Ya - b) = \vec{0}$$

$$Y^t Ya - Y^t b = 0$$

$$Y^T Ya = Y^T b$$

$(d \times n)\ (n \times d)$

So : →

$$\boxed{a = (Y^T Y)^{-1} Y^T b}$$

and $Y^T Y$ = square Matrix $(d \times d)$

solution for $(n > d+1)$

Date
Page

Good Thing →

when $n = \hat{d}$, $[a = Y^{-1} b]$

and $n > \hat{d}$, $[a = (Y^t Y)^{-1} Y^t b]$

Y^{+} (pseudo inverse)

So:→

$$\underset{(\hat{d}\times 1)}{a} = \underset{(\hat{d}\times n)}{Y^{+}} \cdot \underset{(n\times 1)}{b}$$

— unique soln

(least square soln)

Y^+ is called the left pseudo inverse as $[Y^t Y = I]$ and not $[YY^t \neq I]$

— [Too Good Soln ☺]

⟹

```
Y1 = [ones(size(X1,1),1) X1];
Y2 = -1 * [ones(size(X2,1),1) X2];
Y = [Y1' Y2']';
b = ones(size(Y,1),1);
a = (Y'*Y)^(-1) * Y' * 1 * b
```

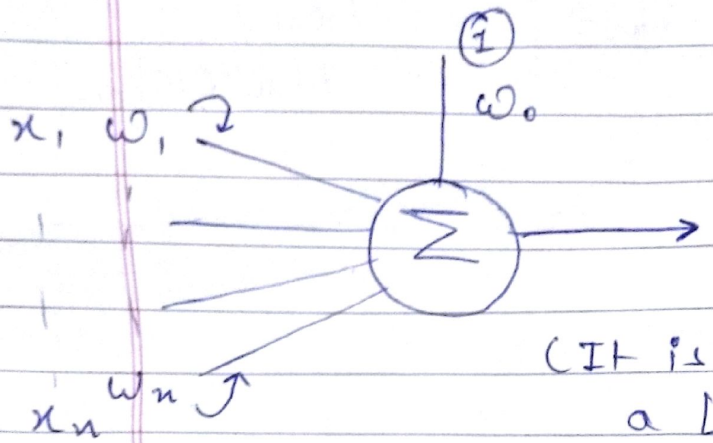
→ How come scientists know size of universe, distance to a distant galaxy, or even know that there is another galaxy (because probably light has not reached there if (send from here) — find out...

2/Sept/2014

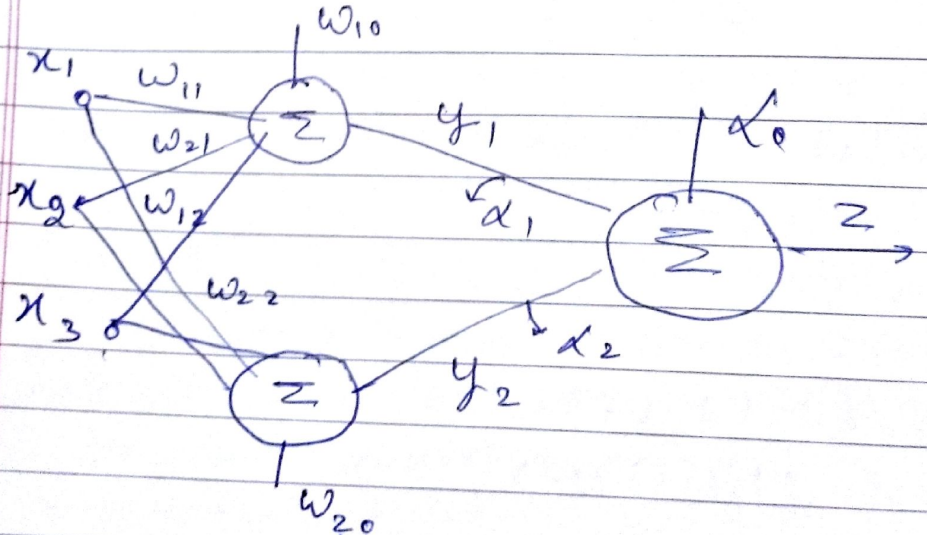
Generative Models:-

- Perceptron

$\sum w_i x_i + w_0$ → Resembles something like brain. (Neurons)



(It is a model of a Linear classifier)



$$y_1 = \sum_{i=1}^{d} w_{1i} x_i + w_{10}$$

$$y_2 = \sum_{i=1}^{d} w_{2i} x_i + w_{20}$$

$$z = \alpha_1 y_1 + \alpha_2 y_2 + \alpha_0$$

So above thing when expanded is $\sum_{i=1}^{d} w_{new_i} x_i + w_{new_0}$

(same eqn as LC but with different weights)

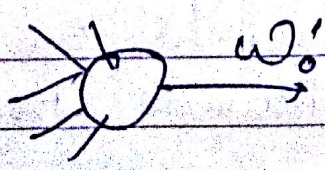
- Honey bee
  - small brain (few 1000 neurons)

  with such a small number of them, they are able to do very-very complex tasks.

  — our brain
  - billion neurons.
  - Trillion connections.

- So since it is found that every 100 neurons came down to 1, ∴ we are doing a Linear combination of Linear things ⇒ we get Linear

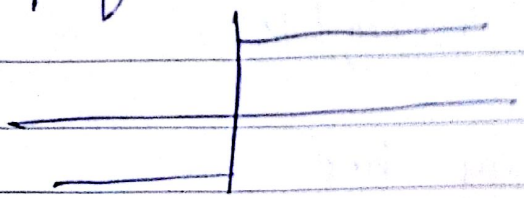
To introduce non-linearity



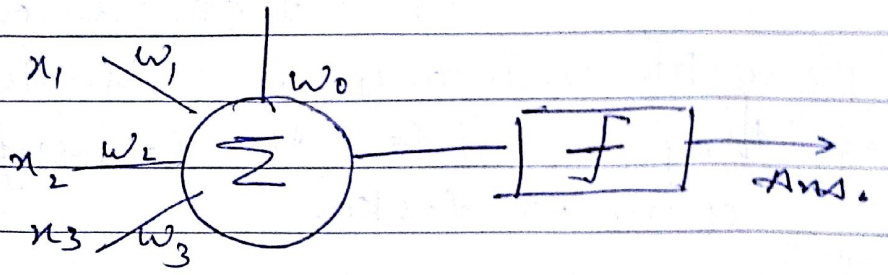
⇒ output = $y = f(w_0')$

so we can have a variety of functions at the terminal.

Simplest Non-linearity.
= step function



(Thresholding)



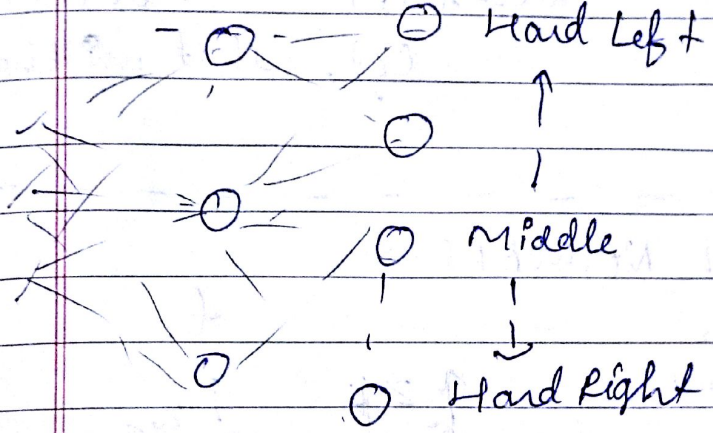
(Simplest NL Model)

This small NLinearity can solve many complex problems and There is a theory that shows that a (2 Layer or 3 Layer) Neural Networks, can model any complex function [classifies].

→ Theoritically
All problems of world can be solved using NN, but we have to train it...

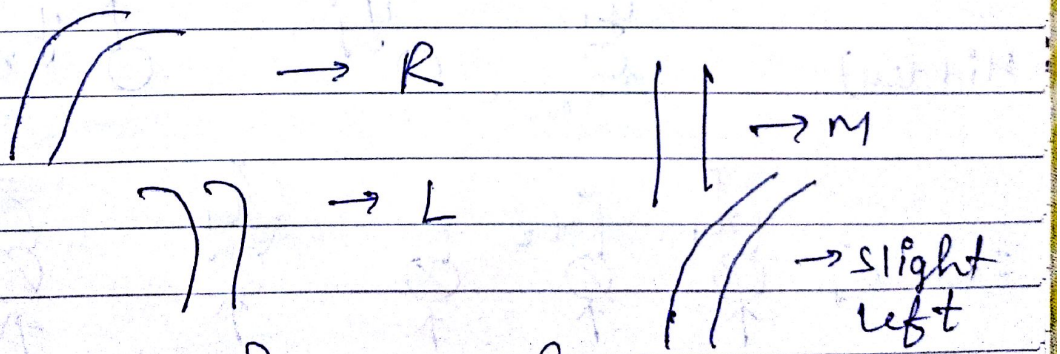
"NAVLAB"

- CMU, self driving car
  - train the car how to drive rather than creating features. and doing a lot of things.
  - Parameters → Accelerate, Brake, Turns left, Right



[Input was the Pixel values of Road]

- a Large set of Input – output

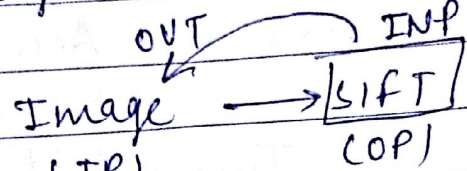


(so drive the car for sometime NN will train and within 2 hrs it will be trained to do so)

— weights will be determined by ff → eff.

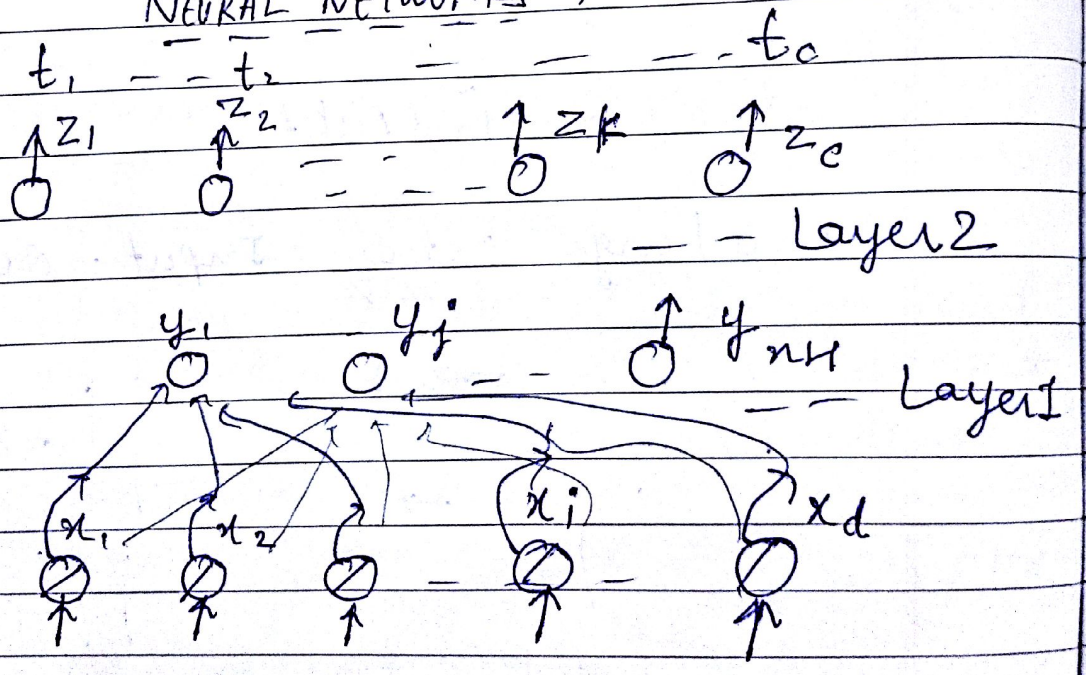
— Teaching a NN with more no. of Layers is easier, than a 2 Layer.

Checkout → App. of NN in IP, [Line filtering]
[Image Reconstruction from SIFT]



Neural {SIFT → OUTPUT}
(?), what will happen

NEURAL NETWORKS →



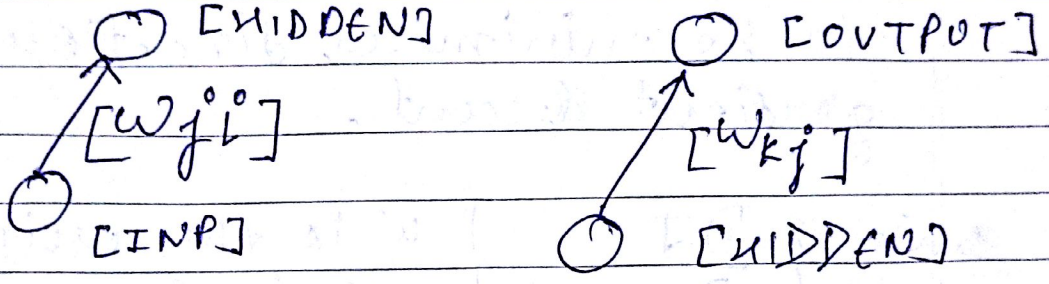
'i' ⇒ Input layer
'j' ⇒ Hidden layer
'k' ⇒ output layer

nH = number of nodes in Hidden layer

$n_H \geq d$ or $n_H < d$ both are possible.

'd' dimensional feature vector.

weight in general represented as →



"Target output is the classes"

Ideally 0, 1, 0 - - 0 should be the output but here we will talk about Probabilities,

Neural Networks Apps -

- Driving cars.
- function approximations.
- more appli other than classi fications.

Sometimes we do have a biased input node.

Suppose for training $t_1 - - t_c$ are values required. Thus we will require

$$J = \sum_{k=1}^{c} (t_k - z_k)^2 \times \frac{1}{2}$$

to be minimised, direction of gradient descent.

$$\Delta w = -\eta \frac{\partial J}{\partial w}$$ { w is the weight vector }

for Hidden Layer — [1 node]

① $net_j = \sum_{i=1}^{d} w_{ji} x_i$

② $y_j = f(net_j)$

③ $net_k = \sum_{j=1}^{n_H} w_{kj} \cdot y_j$

④ $z_k = f(net_k)$

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial w_0} \cdot \frac{\partial w_0}{\partial w_1}$$ } chain rule.

So J is a function of a function - - -.

a)- If a weight between OUT ↘ HIDDEN is changed only that node will be effected.

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}}$$

$$= \underbrace{-(t_k - z_k) \cdot f'(net_k)}_{\delta_k} \cdot y_j$$

$$\delta_k = (t_k - z_k) \cdot f'(net_k)$$

= (sensitivity of node) — we call

= [it captures the information that how will the error change if weights w_{nj} - (net j) is changed for that node]

$$\delta_k = \frac{-\partial J}{\partial Net_k}$$

and $\Delta W_{kj} = -\frac{\partial J}{\partial z_k}$

Layer 2

$$\Delta W_{kj} = \eta \delta_k y_j$$
$$\delta_k = (t_k - z_k) \cdot f'(net_k)$$

So pass (forward pass) ones and then find ΔW_{kj}, update the weights

$$W_{kj}(m+1) = W_{kj}(m) + \Delta W_{kj}$$

b) - $\frac{\partial J}{\partial W_{ji}} = \frac{\partial J}{\partial net_j} \cdot \frac{\partial net_j}{\partial W_{ji}}$

Now if the inputs weight are changed The output at both the forward nodes will change.

Write :→

$$\frac{\partial J}{\partial W_{ji}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial W_{ji}}$$

(K) $\quad \cdot f'(net_j) \quad \cdot x_i$

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{K}(t_k - z_k)^2\right]$$

$$\alpha = -\sum_{k=1}^{c} (t_k - z_k) \frac{\partial z_k}{\partial y_j}$$

$$\frac{\partial z_k}{\partial y_j} = \frac{\partial z_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j}$$

$$\frac{\partial net_k}{\partial y_j} = f'(net_k) \cdot w_{kj}$$

Hence $$\frac{\partial J}{\partial y_j} = -\sum_{k=1}^{c} (t_k - z_k) \cdot f'(net_k) \cdot w_{kj}$$

$$\frac{\partial J}{\partial y_j} = -\sum_{k=1}^{c} (t_k - z_k) f(net_k) w_{kj}$$

$$= \delta_k$$

Hence finally -

$$\frac{\partial J}{\partial w_{ji}} = -\sum_{k=1}^{c} \delta_k w_{kj} \cdot f'(net_j) \cdot x_i$$

$$-\delta_j = -\frac{\partial J}{\partial net_j}$$

↳ sensitivity of Hidden Node.

Always $[\Delta w = -\eta \text{ Gradient}]$

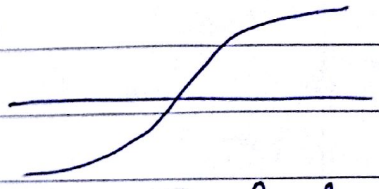
classmate Date Page

So $\Delta w_{ji} = \eta \delta_j x_i$

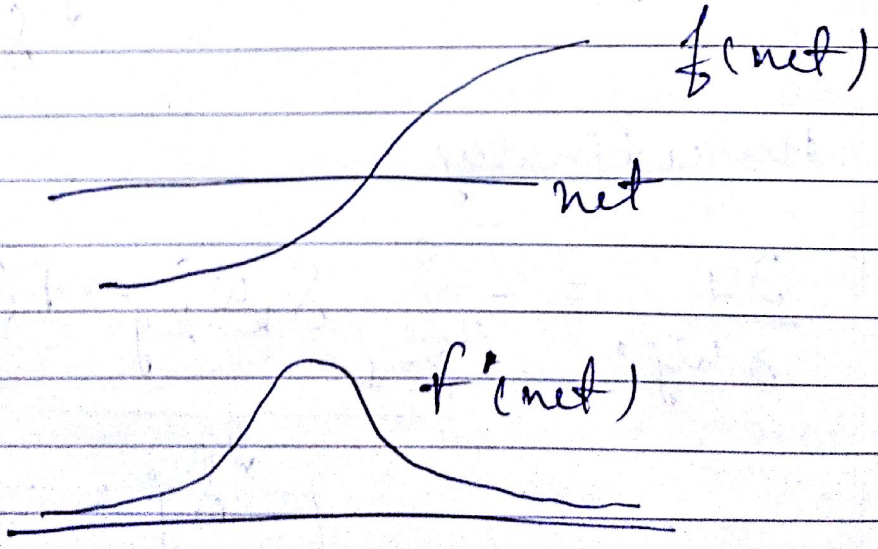
$\delta_j = \sum_{k=1}^{c} \delta_k w_{kj} \cdot f'(net_j)$ ← Layer 1

PROBLEMS:-

① - Activation function f:
↳ Like step fn

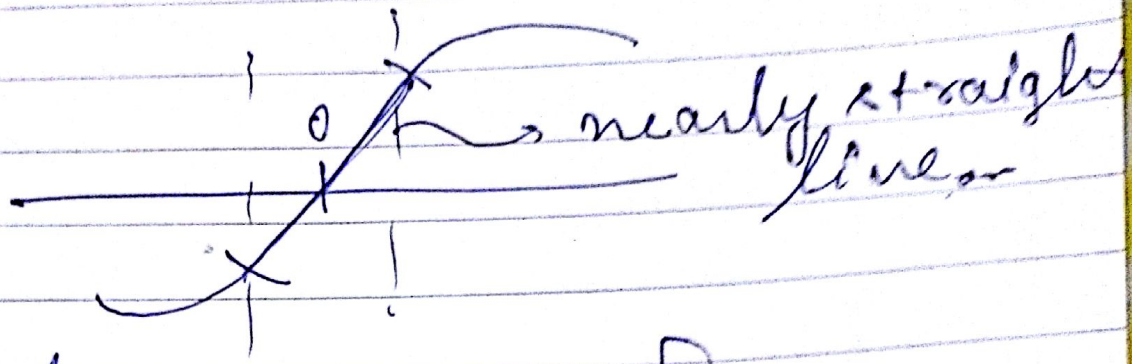


The function should be differentiable for whole process to work.

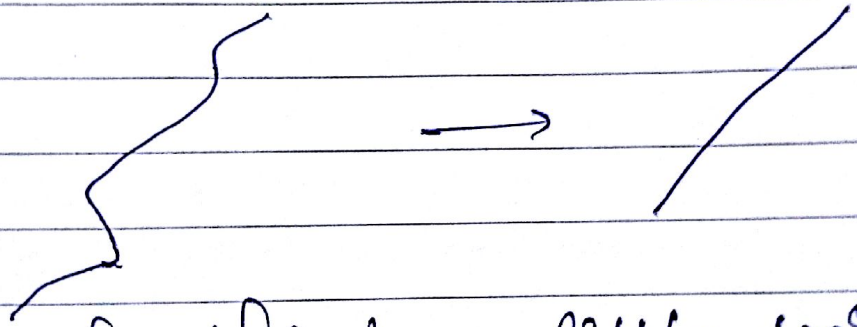


ii) - also f'(net) should not be zero higher number of times since update will be zero.

ii)- we would like to have functions like



If function is like this our curve will become less complex



for this do a little trick, in each update do weight decay, $\Rightarrow [w = 0.95\, w]$ update

iii)- tn should be determined based on f(net n) — {Ranges, do not put their asymptotically higher values}