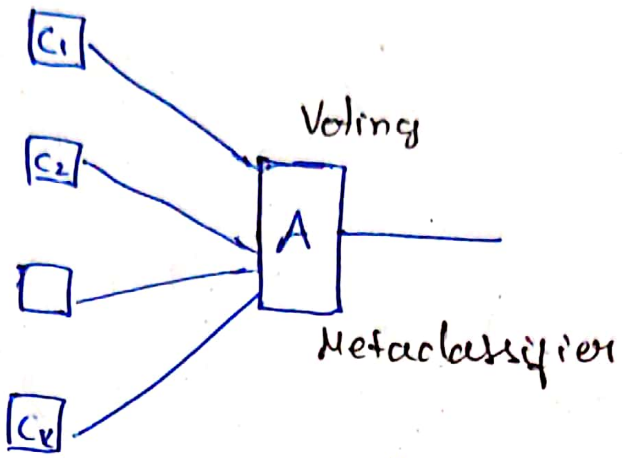
→ Ensemble Classifier :-



$C_1, C_2 \ldots\ldots$ → diversity
→ accuracy

1) Same Classifiers but trained on different training data sets.

2) Classifier is same, input features are different

3) Nature of classifier is also different.

Metaclassifier → Inputs are the outputs of the 'k' classifiers

→ It is trained on them and final output is given.

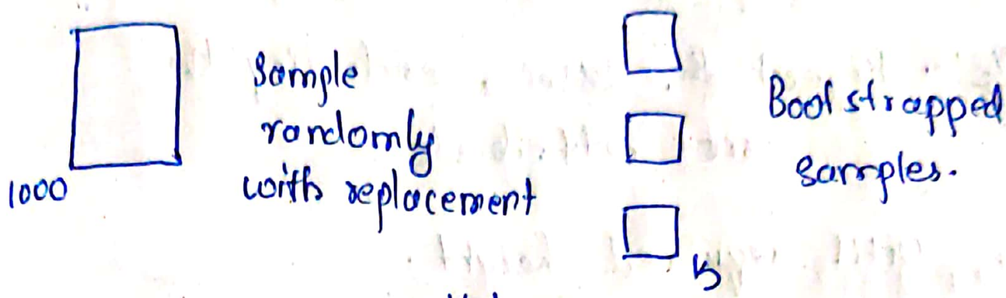
*) Training can be done in 2 ways :-

1) Train Cs and A together.

2) First train C's and then train 'A'.

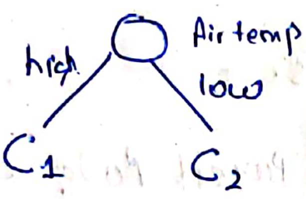
→ Bagging : (Bootstrapped Aggregation)

Bootstrap sample ⟶ getting multiple samples from a given single dataset.



Voting

→ Base classifier — weak classifier



⟶ decision stamp.
(better than randomization)

→ Instead of training a large classifier, training very large weak classifiers and combining them is good enough.

Random Forest (Extension of above) :-

→ Bootstrapped training set

→ random feature subsets.

1 [ ] — Height bounded decision tree.

2 [ ]

⋮

k [ ]

x_1 . . . . . x_d

↳ Randomly select the subset of features.

level 1
i.e randomly project 6-d data to 3-d etc.

→ On this randomized / projected data, create a decision tree.

level 2 :- Repeat the same, excluding the root attribute.

⋮

Until required height.

∴ T_1 (Bootstrap set) ⟶ BC_1 

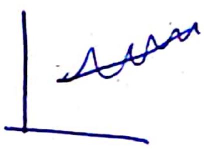
⋮

Get for all → 'k' trees.

→ Take a voting node ⤴ Parent node with 'k' children nodes.

→ Useful if input dimension is very high.

Random Forest ⟶ Random Projections
+
Bagging
+
Bayes Classifier (on top).

Ensemble
↳ bias + variance ↓ 
decrease
the variance by maintaining bias.

Bootstrap → Special case of Re-sampling data.

Bagging, Random Forest → Resampling + training + combining

- Bagging ↓ Resample on train set
- Random Forest ↓ resample on features also.

*) **Adaptive Re-Sampling** :-

↳ 1^{st} Practice set → Random

2^{nd} " " → where 1^{st} failed

3^{rd} " " → where 1^{st} & 2^{nd} failes.

i.e. k^{th} resample depends on its previous resamples & classifiers.

→ 1^{st} classifier is close a weak classifier that is never good on all features.

*) **AdaBoost** :- (Adaptive boosting)

→ Boosting.

$x_1, x_2, \ldots, x_n$

$w_{1i}, w_{2i}, \ldots, w_{ni}$ → change with iterations based on the failure set.

i.e weight of set where previous classifier fails is increased in next iteration.

Iteration 1 :- $\omega_1 = \omega_2 \cdots = \omega_n = \frac{1}{n}$

$\square_n$ → Samples with replacement.

choose a weak classifier as base classifier.

→ C_1 → ϵ (error)

↳ wrongly predicted.

Iteration 2 :-

$w_{i+1} = w_i e^{+\alpha}$ if x_i is misclassified by C_i

$= w_i e^{-\alpha}$ if x_i is correctly classified by C_i.

⋮

Repeat & combine $C_1 \cdots C_K$.

AdaBoost :-

Ensemble

$x_1 \quad x_2 \quad \cdots\cdots \quad x_n$

$y_1 \quad y_2 \quad \cdots\cdots \quad y_n$

$D_1^0 \quad D_2^0 \qquad D_n^0$, $\forall j \sum_{i=1}^{n} D_i^j = 1$.

j^{th} iteration.

$1/n \quad 1/n \qquad 1/n$.

↳ Probabilities that they are selected. (weights).

$\therefore S_1 = x_1 \quad x_2 \quad \cdots \quad x_n$ $\qquad C_1$ (weak classifier)

$C_1 \xrightarrow[\text{on}]{\text{train}} S_1$

weak classifier → slightly better than random classifier.

Now use C_1 on x to predict their output & compare with y.

let $\epsilon_1 = \sum_j D_j^0 \quad \forall j$ if x_j is misclassified

$\downarrow$

$\epsilon_1 < 1/2$ (as weak classifier).

Let $\alpha_1 = \frac{1}{2} \ln \left(\frac{1-\epsilon_1}{\epsilon_1} \right)$

↳ so α is +ve (as $\epsilon_1 < 1/2$).

$S_2 \quad D_1' \quad D_2' \quad \dots \quad D_n'$

if x_j is correctly classified by C_1

$D_j' = D_j^0 e^{-\alpha_1}$

else if misclassified.

$D_j' = D_j^0 e^{+\alpha_1}$

By proof of convergence

Construct $C_2 \xrightarrow[\text{on}]{\text{train}} S_2$.

$\epsilon_2 = \sum D_j' \quad \forall$ mis-classified

$\Rightarrow \alpha_2 = \frac{1}{2} \ln \left(\frac{1-\epsilon_2}{\epsilon_2} \right)$

↳ use for next iteration.

$C_1 \quad C_2 \quad \dots \quad C_T$

$\alpha_1 \quad \alpha_2 \quad \dots \quad \alpha_T$ → all +ve.

$\left(\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i} \right)$

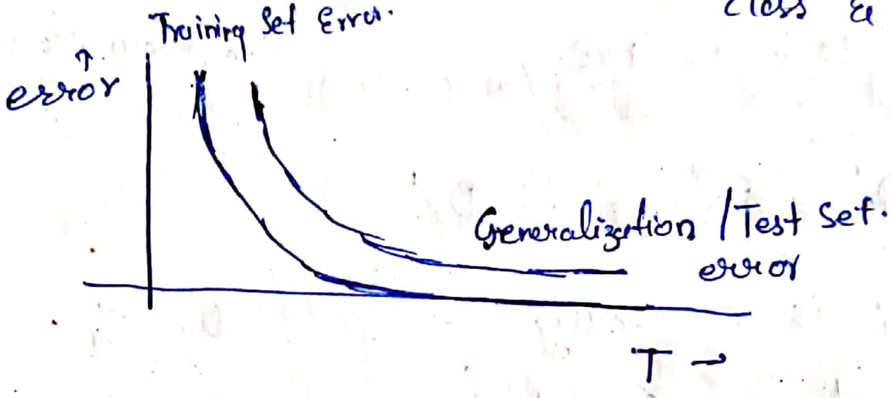
Final Classifier :-

$$C = \sum_{i=1}^{T} \alpha_i C_i$$

(ϵ_i is the class output values) , i.e lower 'ϵ', higher weight for classifier.

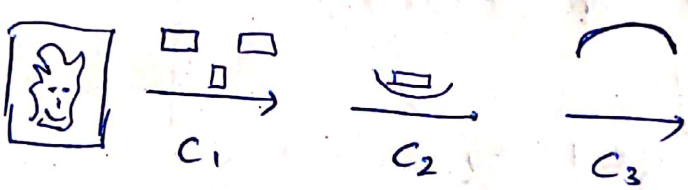
↳ Add all α's that predict particular class & compare.

↳ Adaboost Algorithm.



↳ Refer the Paper on Website for mathematical proof.

Viola-Jones Face recognition :-



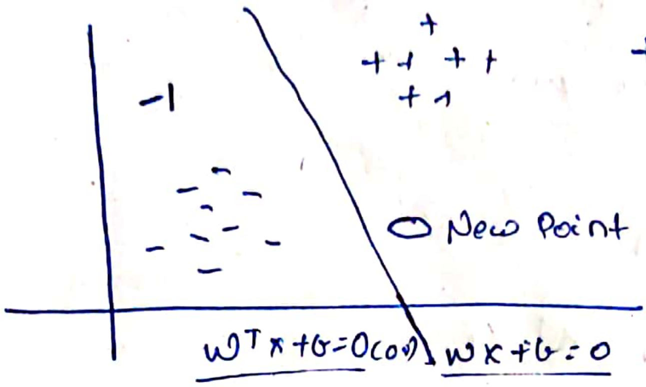
↳ combine these classifiers.

* All the classifiers are de-correlated. i.e no classifier aggrees on more than 50% of the examples.

*) New classifier need not be considered if similar to previous ones.

Support Vector Machine :-

Linear Discriminants.



→ to classify, look on which side it lies.

How to draw the discriminant?

→ $X \in \mathbb{R}^d$ i.e d-dimentional vectore (each point).

i.e

$X = [x_1 \ldots x_d]$.

→ class level $\in [-1, +1]$ → assumption (only 2 classes).

$S = [(x_1, y_1) \ldots (x_n, y_n)]$

$w = [w_1 \ldots w_d]$

$w \cdot x = w_1 x_1 + w_2 x_2 \ldots w_d x_d$

line = $w \cdot x + b = 0$

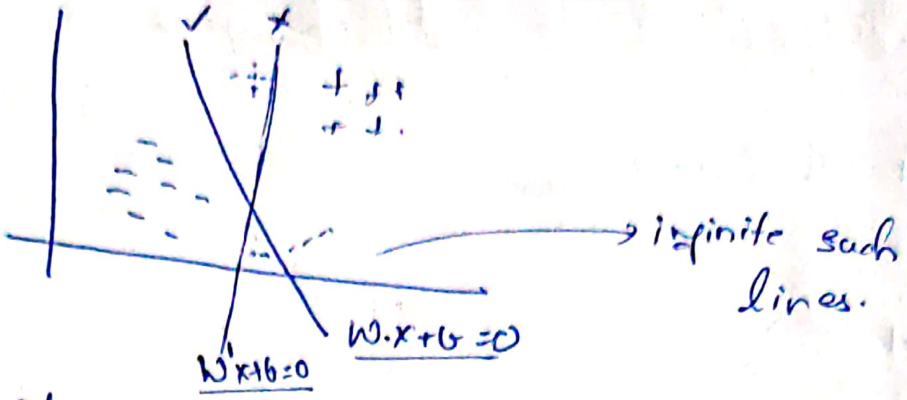
(or) $w^T \cdot x + b = 0$. ⇒ Notations.

+ve side → non-origin side.

$w \cdot z + b > 0$ if $y(z) = +1$

$w \cdot z + b < 0$ if $y(z) = -1$.

Scanned by CamScanner



$x_1 \; - \; - \; . \; x_n$
$y_1 \; - \; - \; . \; - y_n$

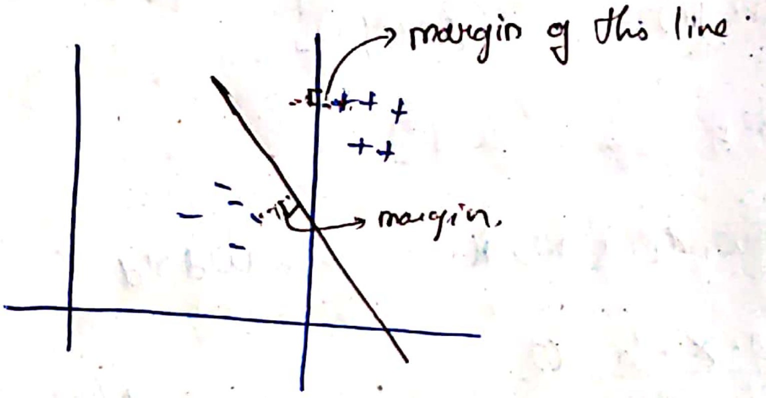
if $y_i = +1$ then $W \cdot x_i + b > 0$

if $y_i = -1$ then $W \cdot x_i + b < 0$.

$\forall i$

for all i, $y_i \cdot (W x_i + b) \geq 0 \cdot \forall i$

→ prefer more equi-distantly placed line.



Margin of the line → $\perp^{lar}$ distance from that classifying line to the closest point.

→ central line has more margin than the other.

Choose the one having highest Margin.

Scanned by CamScanner

→ See missed notes:

Primal Problem:-

Minimize $\frac{1}{2} w^T w$

such that $y_i(w \cdot x_i + b) \geq 1 \quad \forall i$.

Lagrange Multiplier:-

(Minimize) $L = \frac{1}{2} w^T w - \sum_{i=1}^{n} \lambda_i (y_i(w \cdot x_i + b) - 1)$

(L) s.t $\lambda_i \geq 0 \quad \forall i$

same solution as above. (optimization problem)

$\frac{\partial L}{\partial w} = 0, \quad \frac{\partial L}{\partial b} = 0,$

$w = \sum \lambda_i y_i X_i$ → direction factor.

and $\sum \lambda_i y_i = 0$ scalling factor.

$L = \sum \lambda_i - \frac{1}{2} \sum \sum \lambda_i \cdot \lambda_j \, y_i \, y_j \, X_i^T X_j$

$L = \Lambda U^T - \frac{1}{2} \Lambda H \Lambda^T$

$\Lambda = [\lambda_1, \ldots, \lambda_n], \; U = [1 \ldots \ldots 1]$

$U^T \rightarrow$ column of 1's

$H = [y_i y_j X_i^T X_j]_{n \times n}$

↳ Solving the Dual Problem.

→ finding λ's to optimize the problem.

λ's ↓ optimizing variable. (w & b can be derived from λ).

if $\lambda_i = 0$, then corresponding x_i will

not determine 'w'.

has a very sparse solution.

$\lambda_i > 0 \rightarrow x_i$ are called support vectors.

Most of the $\lambda_i = 0$ → Boundary solutions.

only few $\lambda_i > 0$ → interior solutions.

$$w = \sum \lambda_i y_i x_i$$

$$\lambda_i \geq 0.$$

Primal Problem ← Same solutions → Dual Problem.

Minimize $\frac{1}{2} w^T w$ $\quad\quad L = \frac{1}{2} w^T w - \sum \lambda (y_i (w \cdot x_i + b) - 1)$

s.T $y_i (w \cdot x_i + b) \geq 1$ ← $\boxed{\lambda_i = 0}$ $\boxed{\lambda_i > 0}$

Kahrusche Kuhn Tucker Condition:- (KKT)

In primal problem :-

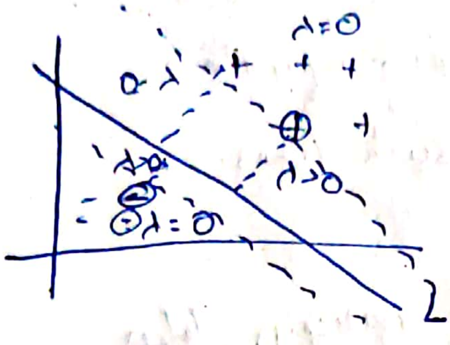
$y_i (w x_i + b) > 1$ ← $\lambda_i = 0$

$y_i (w x_i + b) = 1$ ← $\lambda_i > 0$

i.e the boundary & interior are in dual inverted in primal problem.

→ set of closest points.

i.e only the closest points have non-zero 'λ'
and only they determine 'w'.



both sides.

⇒ keep expanding the line, till H. touches a point.

$$w = \sum \lambda_i y_i \underline{x_i}$$ → support vectors.

↳ closest points.

$\therefore$ $w = \sum \lambda_j y_j x_j$
s.t x_j is a S.V.

How to find b?

We know for S.V,

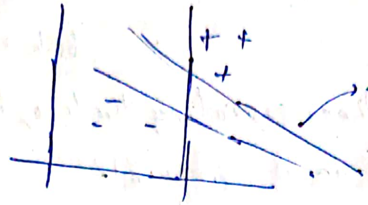
$y_j(w \cdot x_j + b) = 1$ → all known except 'b'.

↳ obtain 'b' from here.

Steps:-

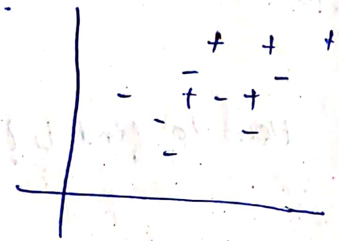
1) Prepare the $n \times n$ matrix.
2) Input that to a standard dual p solver.
3) Collect λ's > 0
4) find w & b.
5) New point can be classified using the line.

Assumptions :-

1) 

there are infinite such lines.

→ Only if there is a clear margin b/w the classes.



→ No line.

→ $y_i(w \cdot x_i + b) \geq 1$ for all.

So to ignore the overlapping points :-

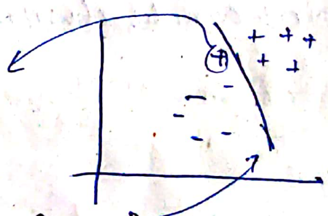
Consider $\boxed{y_i\,(w \cdot x_i + b) \geq 1 - \xi_i}$ for all.

Minimize :- $\frac{1}{2} w^T w + \underline{\underline{C}}$ and $\sum_{i=1}^{n} \xi_i^2$.

→ Allow & penalize relaxation.

~~$\xi_i(x_i)$~~ is different for various points.

$\xi_i\,(x_i)$



Allow with penalty.

Soft Margin.

Final dual P :-

$$L = \Lambda U^T - \frac{1}{2} \Lambda H \Lambda^T \qquad w = \sum \lambda_i y_i x_i$$

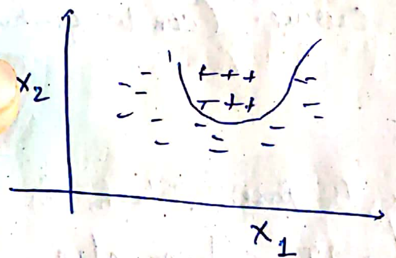
$$0 \le \lambda_i \le C$$

↳ only change is an upper limit for 'λ'.

$C \rightarrow$ Generalization constant.

→ No mathematical way to choose optimal 'C'. trial & error method.

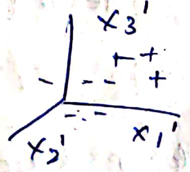
→ Curve Boundary :-



← Can we use the same optimization framework as before.

$2x_1^2 + 3x_2^2 + 5x_1x_2 + 7 = 0$ → example curve

→ Map it to a higher dimensional space.



$$x_1' = x_1^2$$
$$x_2' = x_2^2$$
$$x_3' = x_1 x_2$$

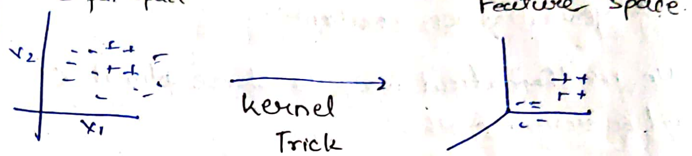
→ Map each point accordingly

Advantage ?

the new curve eqn $2x_1' + 3x_2' + 5x_3' + 7 = 0$

↳ linear Equation.

i.e the boundary is a line.



→ Introduce an axis for every term in the polynomial eq of the curve. (can be greater than 3-d.)
↳ Infinite.

→ Estimate the curve ie its possible term.
↳ ?? ← ↳ or else set infinite dimentions. (for every term possible)

SVM Training :-

Min $L = \Lambda U^T - \frac{1}{2}\Lambda^T H \Lambda$

Subject to $\Lambda = [\lambda_1 \ldots \lambda_n] \geq 0$

$H = [\,y_i y_j x_i' \cdot x_j'\,]_{n \times n}$ → dot product.

SVM Testing :-

$\text{sign}(w \cdot x_q' + b) \qquad w = \sum_{i=1}^{n} \alpha_i y_i x_i$

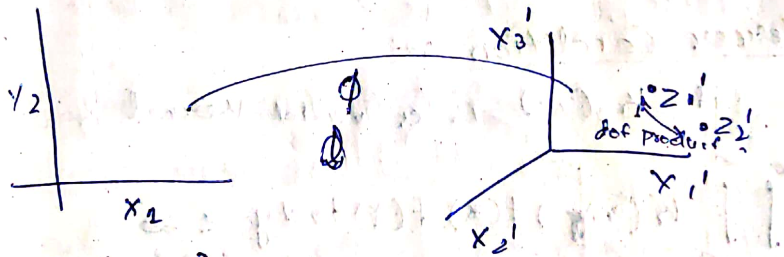
→ $\alpha_i = 0$ if x_i is not a support vector.

→ $\alpha_i > 0$ if x_i is a SV.

$$w = \sum_{x_i' \in SV} \alpha_i y_i x_i'$$

$$\text{sign}\Big(\sum_{x_i' \in SV} \alpha_i y_i \underbrace{x_i' \cdot x_q'}_{\text{dot product}} + b\Big)$$

so the only operation in both training & testing is dot product only.



$x = [x_1 \; x_2]$

$x' = [x_1' \; x_2' \; x_3']$

$x_1' = x_1^2 \; , \; x_2' = x_2^2 \; , \; x_3' = x_1 x_2$

→ $x' = \phi(x)$ → high dimension / Infinite dimentional vector.

$z_1' = \phi(z_1) \; , \; z_2' = \phi(z_2)$

$z_1' \cdot z_2' = \phi(z_1) \cdot \phi(z_2) \longrightarrow ?$

(kernel trick) $= K(z_1, z_2)$ → directly on $z_1 z_2$

$= k(z_1, z_2)$

→ Combination of dot product and ϕ.
(inner)

$H = \left[y_1 y_2 x_i' \cdot x_j' \right]^{n \times n}$ if 'K' is a valid kernel, then 'H' is positive definite.

$= \left[y_i y_j \underbrace{k(x_i, x_j)}_{\text{Scalar.}} \right]$

(and vice-versa)
↳ ??

$\text{Sign} \left(\sum_{x_i \in SV} \alpha_i y_i k(x_i, x_k) \right).$

→ H is a <u>positive</u> <u>definite</u>

$x^T H x \geq 0$ for all x.

<u>Mercer Condition</u> :-

if $k(\cdot,\cdot)$ is a valid kernel function,

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} k(x,y)\, f(x)\, f(y)\, dx\, dy < \infty$$

$\forall f(x)\, f(y)$ s.t $\int_{-\infty}^{\infty} f(x)\, dx < \infty$

functional analysis.

(square integrable / L2 functions)

Linear Kernel:

$K(X, Y) = X \cdot Y$ (dot product)

Quadratic kernel:

$K(X, Y) = X \cdot X + Y \cdot Y + X \cdot Y$

Polynomial kernel:

$K(X, Y) = (1 + X \cdot Y)^d$

Radial Basis Kernel (RBF) / Gaussian kernel:

$K(X, Y) = e^{-\|X-Y\|^2}$

↳ distance b/w the points.

↳ Some of the valid kernels.

→ Similarity can be defined between 2 objects (not only points) like graphs, complex objects etc.. using kernel functions.

↳ Classifier.

↳ Required to find? → Best kernel.

(Research → β-kernels)

*) If K_1 & K_2 are valid kernels, then

→ $\alpha K_1(X, Y) + (1-\alpha) K_2(X, Y)$ → is also valid

↳ (Composit kernels).

Ex:- $X = ATGGCAT$

$Y = AAGGCCT$

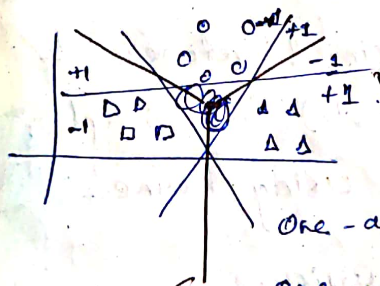
$K(X, Y)$ = No. of Common Sub-strings between X, Y

→ ~~All poss~~ One axis for each possible sub-string. No. of its occurences = its value on that axis. etc..

→ So if a kernel is defined, no need of an input vector

↳ (2-class SVM till here).

Multi-class SVM :-



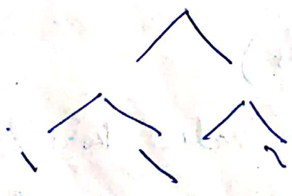
K-class ↳ lines → issue → points inside triangle.

One-against all

One against 1.

hyperplanes.

DAG:-

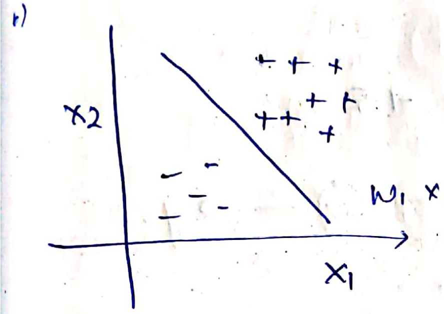


→ Like a tree of hyperplanes.

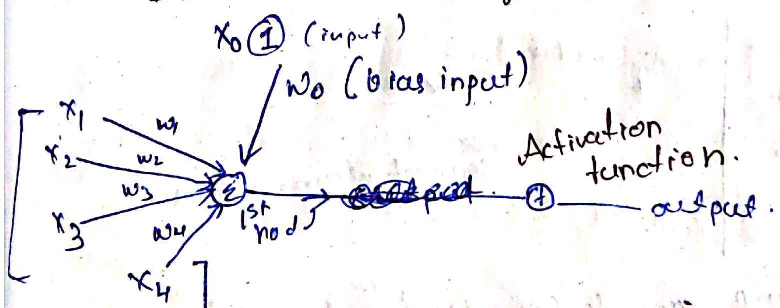
The discussed numerical methods previously have cubic complexity.

Optimizations to decrease complexity.

Ex:- Sequential Minimal Optimization (SMO) etc..



→ **Perceptron** :- / **Single Neuron** :-

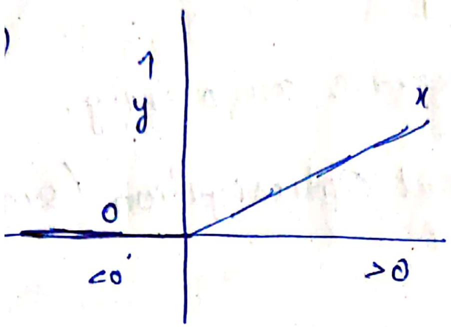


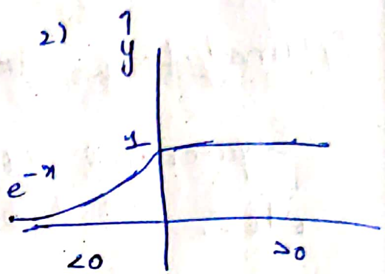
Input vector & weights.

The 1st node computes :- $\sum_{i=0}^{d} w_i x_i$, where $x_0 = 1$ & w_0 = bias.

$$\text{output} = f\left(\sum_{i=0}^{d} w_i x_i\right)$$

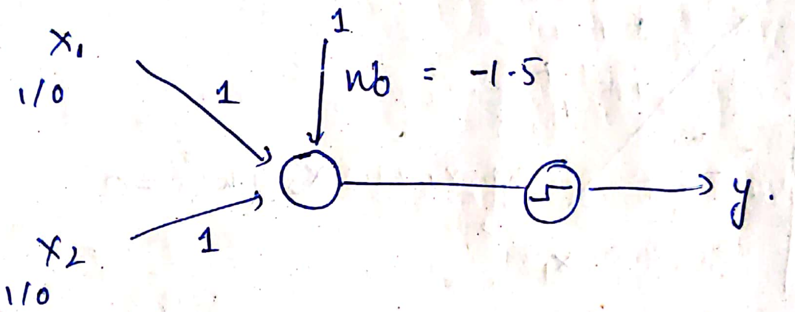
Activation f^n example :-



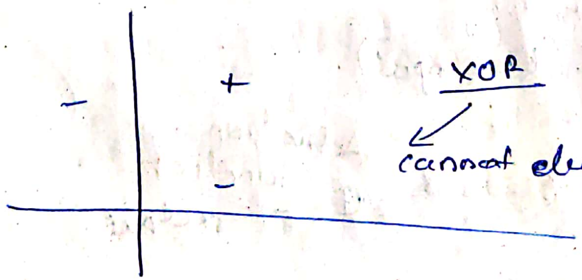


etc ...

X_1 AND X_2 :-



$y = X_1$ AND X_2

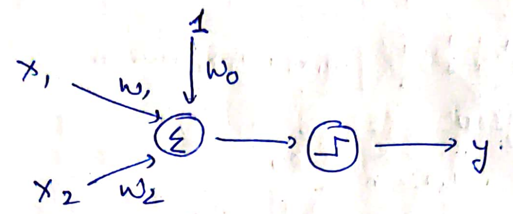


XOR

cannot draw line.

→ XOR cannot be realized using this.

Perceptron Learning Rule :-



→ Random initialization of weights.

Online -Training :-

→ Give the inputs & using the present weights, compute 'y' & match it with the desired output.

→ If they do not match, update weights else ~~leave~~ do not change.

$$w^{t+1} = w^t + \eta \Delta w$$

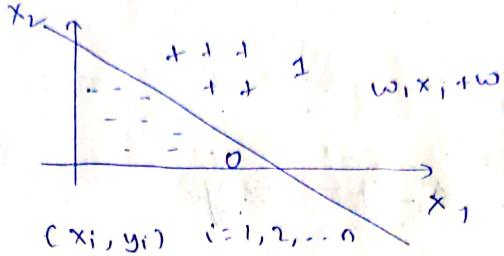
↑ learning rate ~ 0.1

$$\Delta w = (\underbrace{t}_{\text{Actual}} - \underbrace{y}_{\text{Prediction}}) \underbrace{X}_{\text{input.}}$$

→ Start with random weights, continue the learning till it realizes the function.

25/10/19

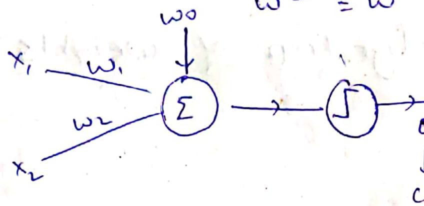
Perceptron



$w_1x_1 + w_2x_2 + w_0 = 0$

$(x_i, y_i) \quad i = 1, 2, \ldots n$

$w^{t+1} = w^t + \eta x_i (y_i - O_i)$ → output



O_i, y_i → actual output

O_i → calculated output

Delta Rule:

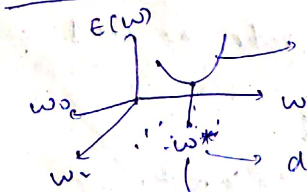
→ assume a value of w_0, w_1, w_2 & calculate the ~~given~~ output O_i for given input.

→ Error $e_i = \frac{1}{2}(y_i - O_i)^2$

total Error $E(w) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - O_i)^2$

Error is a function of w

→ Error landscape:



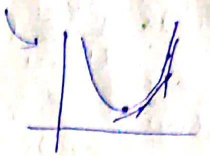
how error changes for diff. perceptrons

different perceptrons

minimizes the error.

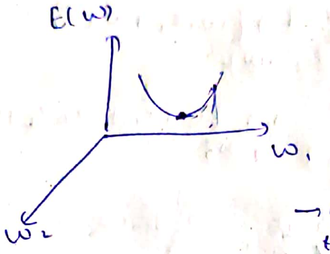
→ if E(w) has a parabolic shape, then we can find w^* (min. error)

- we can find w^* by Newton Raphson method



→ Newton Raphson generalized to an algorithm called as

Gradient - Descent Optimization algo



z-axis → error, other axes I n dim - w's

→ gradient descent algo takes us to the lowest point

→ direction of ~~steepest~~ steepest descent

$$(-)\nabla E = \left[\frac{\partial E(w)}{\partial w_1}, \frac{\partial E(w)}{\partial w_2}, \dots, \frac{\partial E(w)}{\partial w_d} \right]$$

d-dim vector ⇓ corresponds to dir. of tangent

→ reverse direction of that vector

$$\Rightarrow w^{t+1} = w^t + \eta(-\nabla E)$$

w^{t+1} ↓ new vector

w^t ↓ old vector

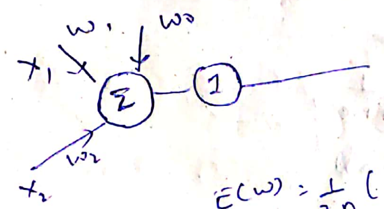
η ↓ learning rate

$(-\nabla E)$ → change vector.

$$\rightarrow E(w) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - O_i)^2$$

$$O_i = \sum w_i x_i$$

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



$$E(w) = \frac{1}{2n}\left(\sum_{i=1}^{n}\Big(y_i - \underbrace{\sum_{j=1}^{d+1} w_j x_i}_{O_i}\Big)^2\right)$$

$$\cancel{\nabla E = \sum_{i=1}^{n}(y_i - O_i)\frac{\partial(O_i)}{\partial w}}$$

$$E(w) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \underbrace{(w_1x_1 + w_2x_2 + w_0)}_{o_i})^2$$

$o_i \rightarrow$ output of perceptron

$$\nabla E(w) = \sum_{i=1}^{n}(y_i - o_i)\sum_{i=1}\left[\frac{\partial}{\partial w_1}[w_1x_1 + w_2x_2 + w_0] + \frac{\partial}{\partial w_2}[w_1x_1 + w_2x_2 + w_0] + \frac{\partial}{\partial w_0}[w_1x_1 + w_2x_2 + w_0]\right]$$

$$\frac{\partial E}{\partial w_1} = \frac{1}{n}\sum(y_i - o_i)x_{1i}$$

$$\frac{\partial E}{\partial w_2} = \frac{1}{n}\sum(y_i - o_i)x_{2i}$$

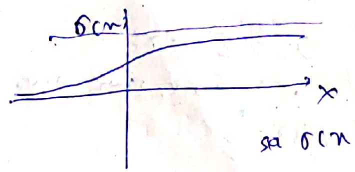
$$x_i = [x_{1i}, x_{2i}, x_{3i}]$$

$$\boxed{\nabla E(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - o_i)x_i}$$

$$\boxed{w^{t+1} = w^t - \eta\frac{1}{n}\sum_{i=1}^{n}(y_i - o_i)x_i}$$

Gradient Descent algorithm

$\rightarrow$ This is a batch algorithm unlike the previous one



sigmoid fxn

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

if we use sigmoid fxn then

$$w^{t+1} = w^t - \eta\frac{1}{n}\left(\sum_{i=1}^{n}(y_i - o_i)\cdot x_i\, \delta_i\right)$$

$$\delta_i = o_i(1-o_i)$$

WORKOUT

$X = [x_1 \;\; x_2 \;\; x_3 \;\; 1]$ | x_1, t_1

$W = [w_1 \;\; w_2 \;\; w_3 \;\; w_b]$ | x_2, t_2

| x_n, t_n

$$S = \sum_{i=0}^{3} w_i x_i$$

$y = \sigma(s)$ → activation function.

$\sigma(s) =$

$\sigma(s) =$

$\frac{1}{1+e^{-s}}$ sigmoid.

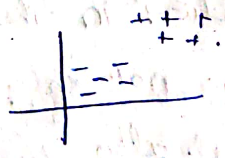
$$\sigma'(s) = \sigma(s)\,(1-\sigma(s))$$

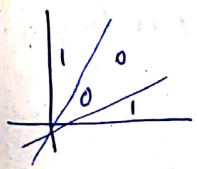
Gradient descent

$$E(w) = \frac{1}{2} \sum_{i=1}^{n} (t_i - y_i)^2$$

$$W^{t+1} = W^t - \eta \nabla E$$

$$= W^t - \eta \sum_{i=1}^{n} (t_i - y_i) X_i$$





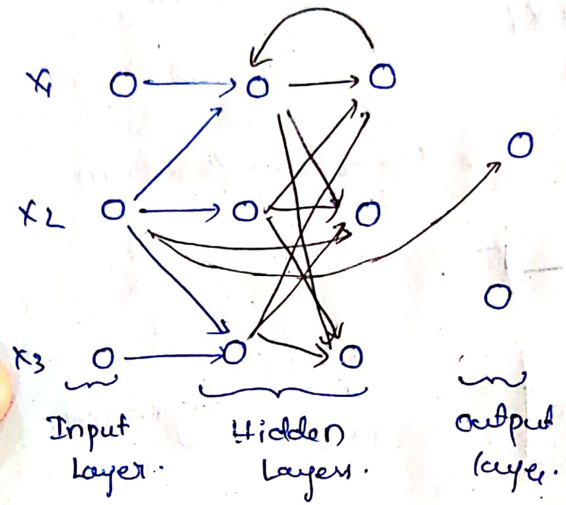
→ Single line cannot seperate them.

But 2 lines can.

Similarly more than one perceptron can be used to distinguish such cases.

Multi Layered Perceptron (MLP) :-

Architecture , Training Algorithm:



- feed forward Network.

→ Recurrent Network.

→ Fully connected Network

In a feed forward Network :-

1) Output of a layer , is the input to the next layer. Unidirectional flow.

In a Recurrent Network :-

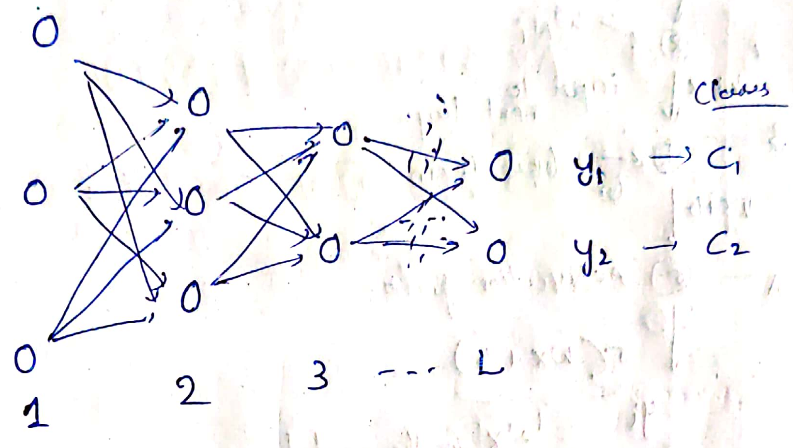
1) Data can flow backwards also.

→ Modifications in these network can have data flow not only between immediate layers, but also further (or) previous layers.

In a Fully connected N :-

→ Output of every node is input to every node of next layer.

→ MLP → fully connected feed-forward network



$X = [x_1 \; ... \; x_3]$; $y = [y_1 \; y_2]$

→ every node in a layer has a 'w' vector.

node → w_1 vector.

eg :- layer 2 → $\left[\begin{array}{c|c} w_1 & n_1 \\ w_2 & n_2 \\ w_3 & n_3 \end{array} \right.$

↳ Weight Matrix for layer 2.

→ Now every node has a bias term.

Soft max :-

$$P(C_1 | x) = e^{y_1} / e^{y_1} + e^{y_2}$$

Forward Propagation :-

Computation Graph

$$x \to \otimes \leftarrow W_{Ni}$$

$Wx \downarrow$ input to next layer.

B — $\oplus \leftarrow b$ bias term.

$Wx+b \downarrow$

A — σ Activation function.

$\downarrow$ $\underbrace{\sigma(Wx+b)}$

y_i ↳ 'x' for the next layer.

Gradient Descent :-

Back Propagation.

*) It was mathematically proved that even with one hidden layer, any mathematical function can be realized.

$$\frac{\partial y_i}{\partial x} = \frac{\partial y}{\partial A} \cdot \frac{\partial A}{\partial B} \cdot \frac{\partial B}{\partial x}$$

↳ By Chain Rule.

$$\frac{\partial y}{\partial A} = \sigma'(s) = \sigma(s)(1-\sigma(s))$$

$$\frac{\partial A}{\partial B} = \frac{\partial(Wx+b)}{\partial(Wx)} = 1$$

$$\frac{\partial B}{\partial x} = \frac{\partial(Wx)}{\partial(x)} = W.$$

$$w^{t+1} = w^t - \eta \nabla E.$$

→ $\frac{1}{2}(t_i - y_i)^2$

$$y = \sigma\left(\sum w_j x_j + B\right)$$

$$\nabla E = \sum_{i=1}^{n} \delta_{bi} X_{bi}$$

for output node :-

$$\delta_i = (t_i - y_i)\, \sigma'(s_i)$$

for hidden node:-

$$\delta_i = \left(\sum_j w_j \delta_j\right) \sigma'(s_i)$$

→ See Slides for perfect derivation

<u>Back Propagation</u> :- (<u>Stochastic</u>, <u>case</u>):-

→ Initialize all weights to small numbers.

→ for each training eg

compute network outputs.

for each output unit k:-

$$\delta_k \leftarrow O_k(1 - O_k)(t_k - O_k)$$

for each hidden unit h

$$\delta_h \leftarrow O_h(1 - O_h) \sum_{k \in outputs} w_{k,h} \delta_k.$$

for each weight

$$w_{j,i} \leftarrow w_{j,i} + \Delta w_{j,i}.$$

where $\Delta w_{j,i} = \eta\, \delta_j x_{j,i}$