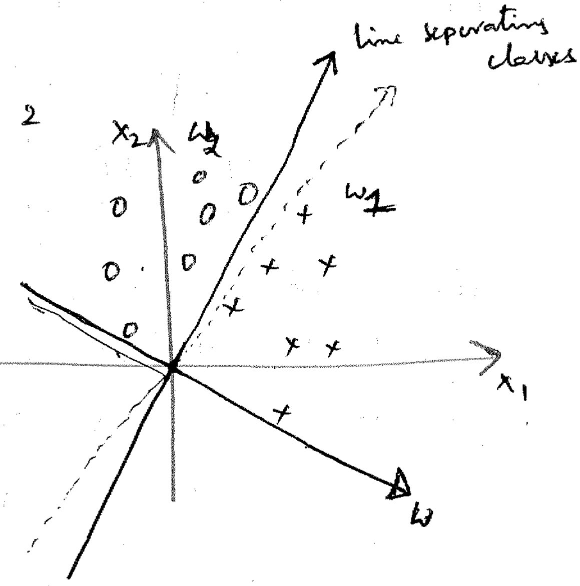
Perceptron learning algorithm:

Consider Samples of class 1 & class 2

Let there samples are linearly seperable. The idea here is to search for the hyperplane that linearly seperates two class data samples such that



(i.e) $w^t x > 0$ for class 1

and $w^t x < 0$ for class 2.

* one such direction 'w' for the given set of samples of class '1 & '2' is shown. * Note that direction of 'w' will be $\perp$er to line seperating two classes. * Also note that the line seperating two classes is not unique. (i.e even the dotted line also seperates class 1 & 2).

* ~~Let~~ the miss classified ~~vech~~ samples of class '1 & class 2' constitute the total cost in misclassification, which is given by

$$J(w) = \sum_{x \in \text{missclassified } w_1 \& w_2} \delta_x w^t x \qquad \delta_x = -1 \text{ for class 1}, \quad 1 \text{ for class 2.}$$

Note:
This can be extendended to lines not passing through origan i.e

$w^t x + w_0 = 0$ by using $w' = [w \; w_0]$; $x' = [x, 1]$

and comparing $w'^T x' > 0$ for class 1

$w'^T x' < 0$ for class 2.

The cost function associated with linear discriminant function 'w' is given by

$$~~J(w) = \sum_{x \in \text{mis classified}} \delta_x w^t x~~ \qquad J(w) = \sum_{x \in \text{misclassified}} \delta_x w^t x \quad \text{— (0)}$$

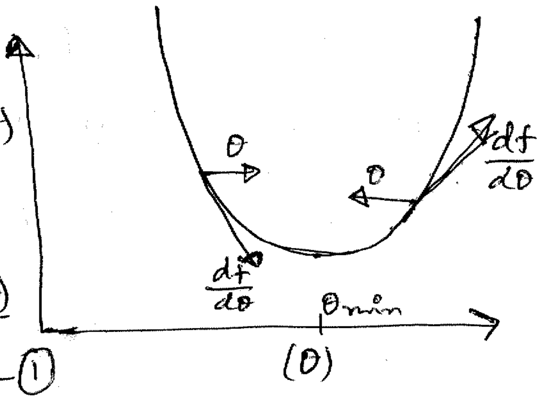
* ~~But~~ ~~minimi~~ Note that $J(w)$ is a piece wise linear function but has discontinuous gradiants when set of misclassified vectors 'x' change. Thus the minima of $J(w)$ can not be obtained by direct differentiation.

*) But never the less gradiant descent approach can be applied to find minima of $J(w)$.

⇒ Gradiant descent can be applied to ~~find~~ reach minima ~~by~~ iteratively.

for example we update 'θ' value $f(\theta)$ with following scheme

$$\theta(\text{new}) = \theta(\text{old}) - \rho \frac{df(\theta)}{d\theta} \quad \text{— (1)}$$



⇒ i.e we move in the direction opposite to (sign) of gradiant.

It can be show that for 'ρ' satisfying some properties (i.e < 1 & is decreasing real ~~no.~~ no. from iteraction to iteration

eq. (1) converges to θ_{min}.

Now by applying gradiant descent to eq (0) we get

$$w(t+1) = w(t) - \rho(t) \frac{dJ(w)}{dw}.$$

since $\frac{\partial (w^t x)}{\partial w} = x$

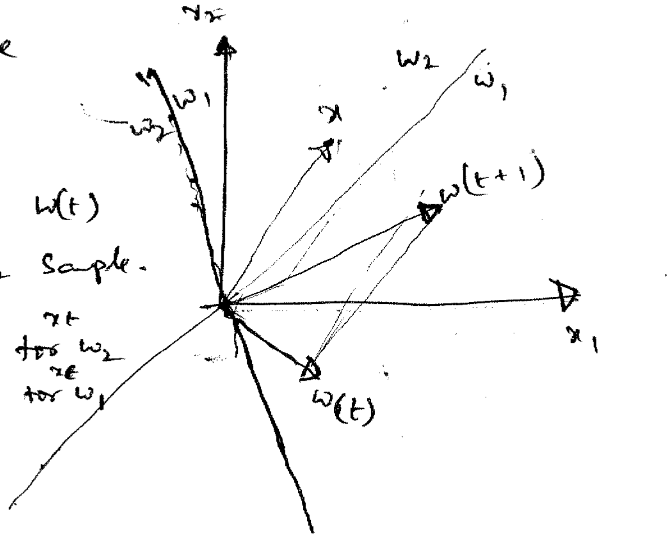
or

$$\boxed{w(t+1) = w(t) - \rho(t) \cdot \sum_{x \in \text{mis classified}} \delta_x x}$$

$\rho(t)$ – is decreasing ~~fact~~ wrt (t) & $\rho(0) \le 1$

Perceptron learning: An example:

Actually $x \in \omega_1$ But at t, $w(t)$ classify it as ω_2



Let x be misclassified Sample at time 't'. ~~i.e w(t+1)~~

i.e $x \in \omega_1$ But classified by $w(t)$ as ω_2 Sample.

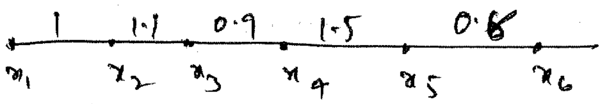
$\therefore \quad \delta_x x = -x$, as $\delta_x = 1$ for $x \in \omega_2$, -1 for $x \in \omega_1$

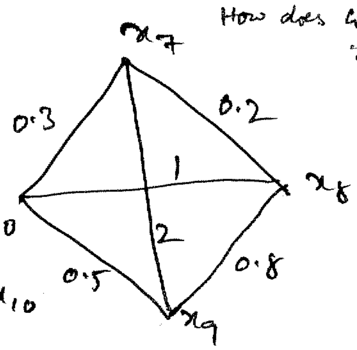
$\therefore \quad w(t+1) = w(t) - \rho_t \sum_{x \in Y} \delta_x x$

Let $\rho_t = 1$ & x - is only misclassified Sample.

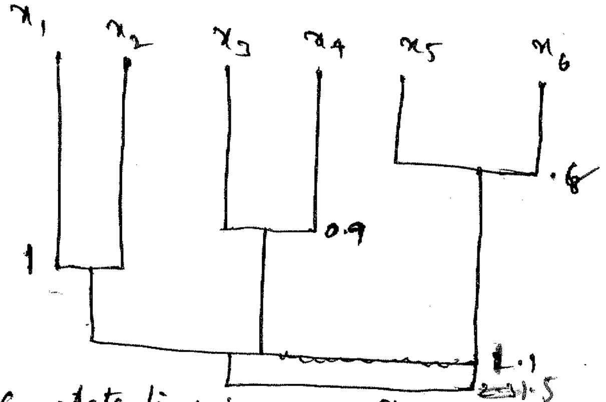
$w(t+1) = w(t) + x \Rightarrow$ By graphical addition direction of $w(t+1)$ & corresponding classification are shown.

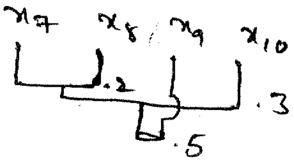
Example: Let $\{(0,0), (0,-1), (-1,0)\} \in \omega_1$ and $\{(1,0), (0,1), (1,1)\} \in \omega_2$ apply linear perceptron to evaluate ω, with initial $w_0 = (-1,-1)^t$ & $b = 0$ & $\rho = 0.5$. How does convergence effect if $\rho = 1$.





Single-link:

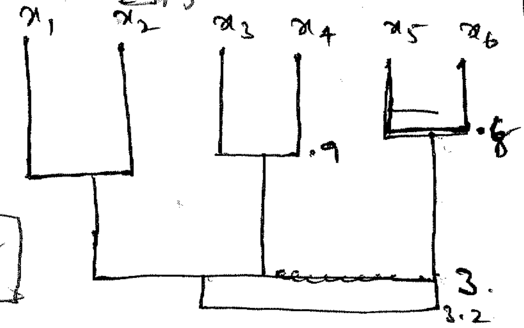


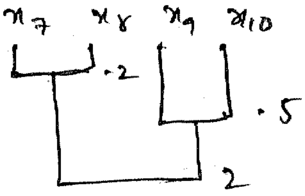


Breaking into two clusters:

$\{x_1, x_2, x_3, x_4\}$, $\{x_5, x_6\}$ ~~or~~ $\{x_7, x_8, x_{10}\}$, x_9

Complete link:





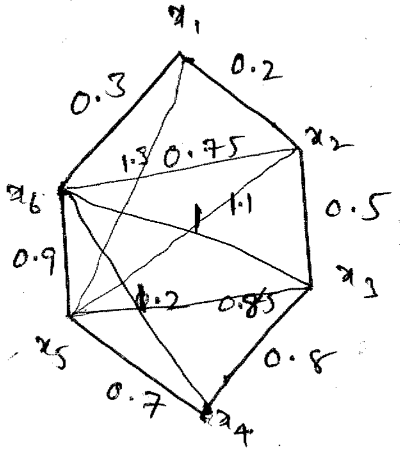
$D_{min}(D_i, D_j) = \min_{x \in D_i, x' \in D_j} \|x - x'\|^2$

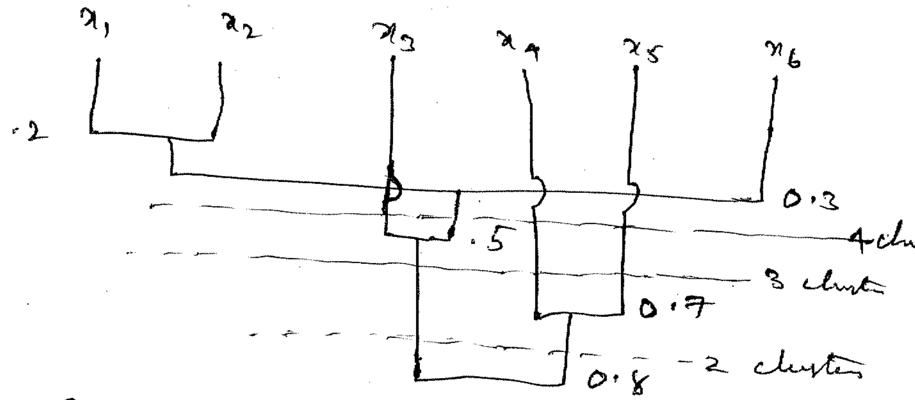
$D_{max}(D_i, D_j) = \max_{x \in D_i, x' \in D_j} \|x - x'\|^2$

$\{x_1, x_2, x_3, x_4\}$, $\{x_5, x_6\}$.

$\{x_7, x_8\}$, $\{x_9, x_{10}\}$.

Single link:





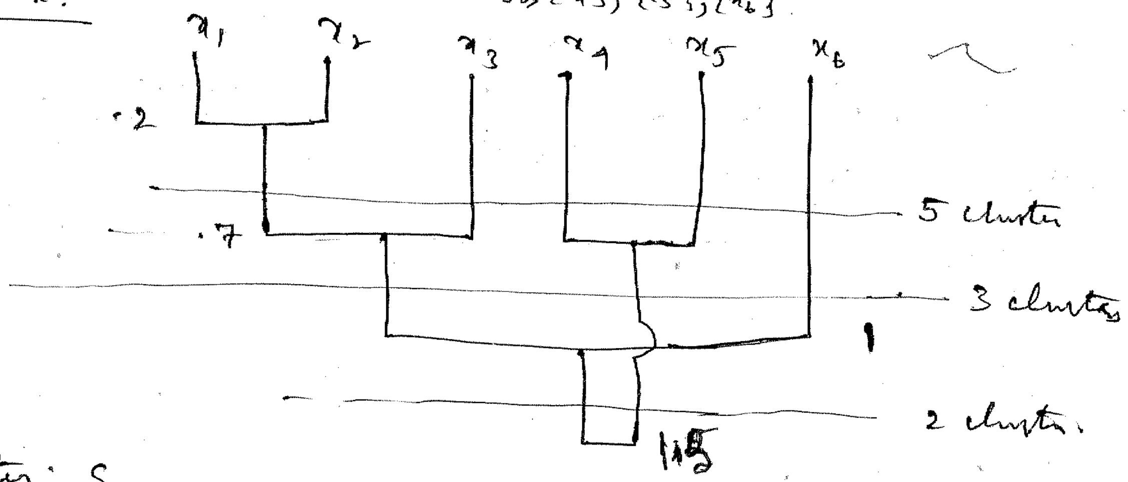
Two clusters: $\{x_1, x_2, x_3, x_6\}$, $\{x_4, x_5\}$

Three clusters: $\{x_1, x_2, x_3, x_6\}$, $\{x_4\}$, $\{x_5\}$

4-clusters: $\{x_1, x_2, x_6\}$, $\{x_3\}$, $\{x_4\}$, $\{x_5\}$.

5-clusters: $\{x_1, x_2\}$, $\{x_3\}$, $\{x_4\}$, $\{x_5\}$, $\{x_6\}$

Complete link:

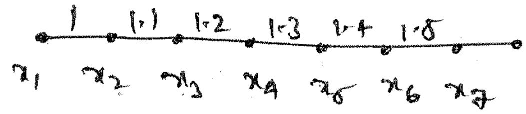


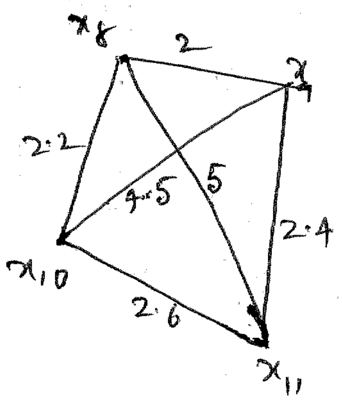
Two clusters: $\{x_1, x_2, x_3, x_6\}$, $\{x_4, x_5\}$

3 clusters: $\{x_1, x_2, x_3\}$, $\{x_4, x_5\}$, $\{x_6\}$.

5 clusters: $\{x_1, x_2\}$, $\{x_3\}$, $\{x_4\}$, $\{x_5\}$, $\{x_6\}$.

Problem:

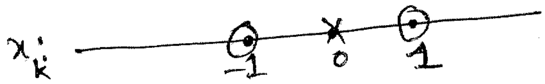




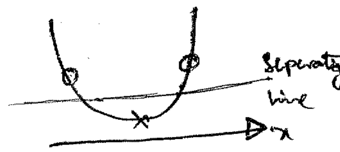
→ Apply single link & complete link algorithms for clustering the points x_1 to x_{11}.

Introduction:

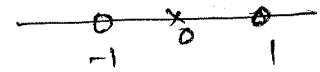
Neural Networks: The Short coming of the linear perceptron is that it can classify a data only if the samples of different classes are linearly seperable.

* One of the approachs for classifying the non seperable data is to apply kernal based methods i.e non linear mapping of data in to a space where they are linearly seperable. ex: x_k:  if $x \in \omega_1$, $o \in \omega_2$

than data samples x_k are not linearly sperable

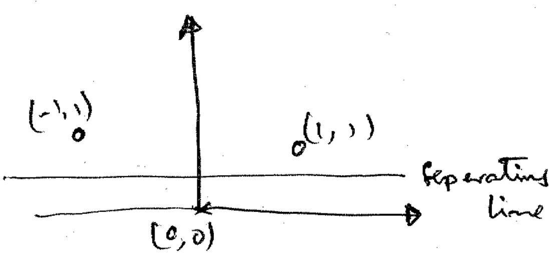
But if we consider $\phi(x_k) = e^{-\frac{x_k^2}{2\sigma^2}}$, than $\phi(x)$ 

$\phi(x_k)$ are seperable!

* Another approach is to increase the dimensionality so that the samples in the higher dimention are linearly seperable. Considering same example x_k: 

$x \in \omega_1$, $o \in \omega_2$.

define $y = \begin{bmatrix} x \\ x^2 \end{bmatrix}$ Then $\{y_k\} = \left\{ \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$

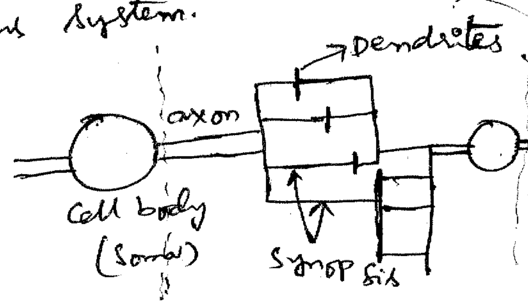


and $\{y_k\}$ are linearly seperable.

However the choice of kernal function or arriving at higher dimensional reperasentation ~~where~~ so that the samples are linearly seperable is not trivial to generalize and ~~depe~~ dependens ograately on the data samples.

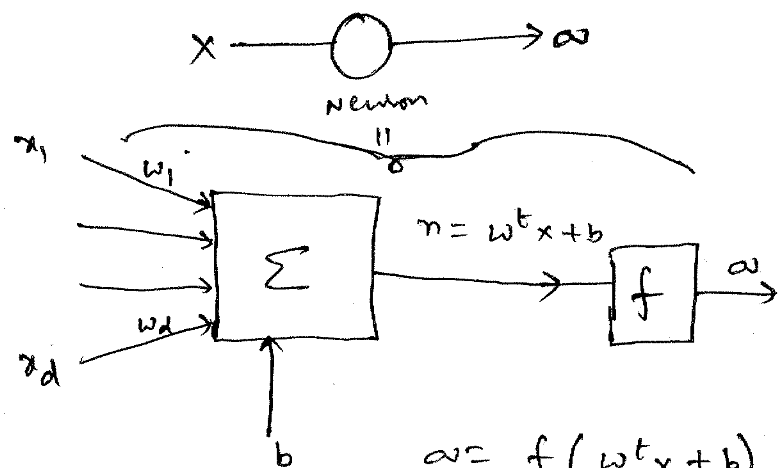
~~The~~ As against the techniques discussed so far, the Neural networks ~~based~~ classification is motivated or inspired by ~~bio~~ Biological capability of neurons (in the ~~human or~~ animal brain). i.e their Nervous system.

* Neurons have cell body, axon, synopsis & Dendrites etc.



* The spikes travelling along the axon triggers the release of neuro transmitter substance at the synapse of pre-synaptic neuron.
* These neurotransmitters cause excitation or inhibition at the dentrites of post synaptic neuron.
* The integration of (weighted) excitations & inhibitions signals produce spikes in axon of post-synaptic neuron, which is modulated by cell body before transmitted to next neuron.
* The contribution of signals depends on the strength of synaptic connections.
* Now if we wish to develope a similar model we note the following similarities:

Neuron	its model
Strength of Synopsis	weight
Cell body	Transfer function and Summation
Signal on ~~out put~~ axon of neuron	output



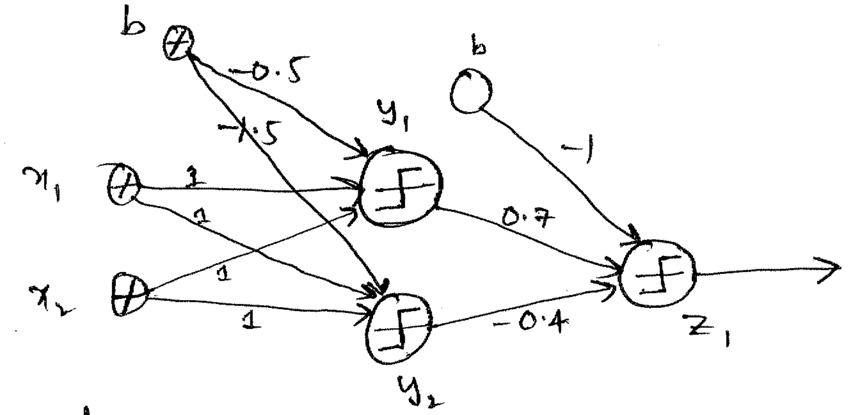
$$a = f(w^t x + b)$$

output of Neuron $a = f\left(\sum_{i=1}^{d} w_i x_i + b\right)$

Feed Forward Neural network:

* We consider following <u>example</u> to see the ability of neural network in classifying the non seperable XOR data

~~In this~~

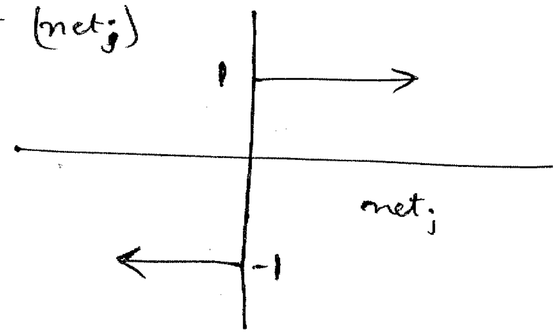


At hidden nodes

Let $net_j = \sum_{i=1}^{d} w_{ij} x_i$ → weighted sum of inputs at j^{th} node/neuron

$y_j = f(net_j)$ Here f is given as threshold activation function.

⇒ $f(net_j)$



i.e $f(net_j) = 1$ for $net_j \geq 0$

-1 for $net_j < 0$

Similarly we defined at output node $net = \sum_{j=1}^{2} y_j w_j$ (here only one output & 2-hidden)

$z = f(net)$; f again being threshold activation function.

Note $net_1 = x_1 + x_2 - 0.5$

$net_2 = x_1 + x_2 - 1.5$

and $net = 0.7 y_1 - 0.4 y_2 - 1$

Now XOR	x_1	x_2	net_1	y_1	net_2	y_2	net	z	NN. classification
w_2 ←	0	0	-.5	-1	-1.5	-1	-1.3	-1	w_2
w_1	0	1	.5	1	-.5	-1	0.1	1	w_1
w_1	1	0	.5	1	-.5	-1	0.1	1	w_1
w_2	1	1	1.5	1	0.5	1	-0.7	-1	w_2

Feed forward neural network: As from the above example it is clear that a linear combination of perceptrons or an ~~set~~ orgnized set of neurons can seperate the samples that are not linearly seperable.

→ However, the question arise (i) how do we arrive at those weights that optimally seperate the samples of two or more classes. And

→ (ii) what type of activation function to ~~you~~ use at every neuron

(iii) ~~How~~ to choose the number of neurons their placement & number of layers etc.

Feed forward neural network has an input layer, one-or-more hidden layers & an ~~op~~ output layer.

* No of input layer neurons is proportional to no. of features
* No. of output layer neurons depends on no. of classes.

$$y_j = f_1(net_j^h)$$

$$z_k = f_2(net_k^o)$$

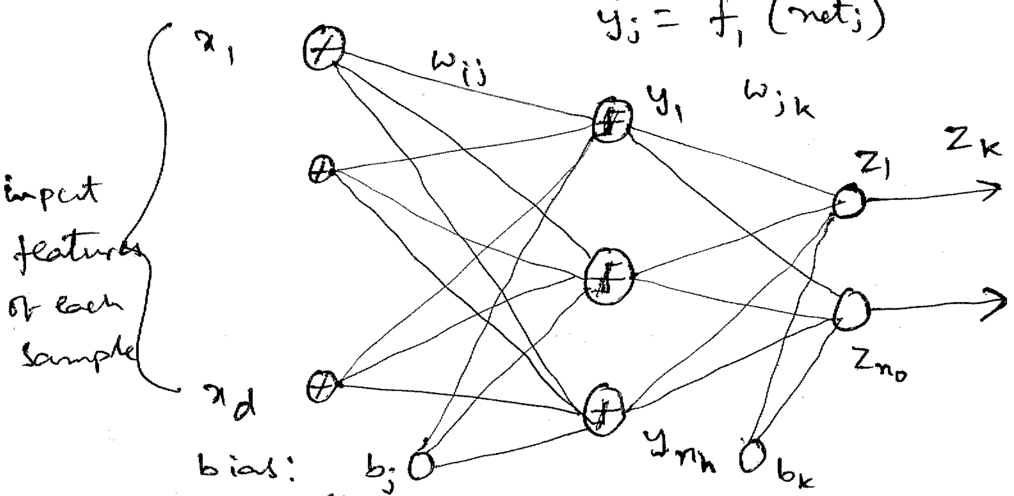


Fig 1

Single - ~~lay~~ hidden layer feed forward neural network.

looking at the neural network of Fig 1.
we can write the following:

$$\text{Eq 1} \begin{cases} net^h_j = \sum_{i=1}^{d} x_i w_{ij} + b_j \\ y_j = f_1 (net^h_j) \\ net^o_k = \sum_{j=1}^{n_h} y_j w_{jk} + b_k \; ; \quad z_k = f_2 (net^o_k). \end{cases}$$

where
→ f_1 is the activation function at all neurons of ~~first la~~ hidden layers.
→ f_2 is the " " " " of output layers.

eq 1 defines the mapping or the relation between input features & output lable (or value), In terms of the synaptic weights. (& biases) & activation fn.

i.e

$$Z_k = f_2 \left[\sum_{j=1}^{n_h} w_{jk} f_1 \left(\sum_{i=1}^{d} w_{ij} x_i + b_j \right) + b_k \right]$$

→ Now we address the problem of how to arrive at the weights that result in optimum classification.

→ There are several approches for finding or learning the weights from training samples. The well known & widely used being the error Back propagation algoritham.

→ As the name indicates we quantify the error at output neurons and back propagate it to update the weights by gradiant descent.

Let d_k ~~$d_k(n)$~~ be the desired or target values at output ~~node~~ neuron for $(n)^{th}$ sample, then ~~total~~ mean square error at output layer of NN is (for given set of weights)

$$J(w) = \frac{1}{2} \sum_{k=1}^{n_0} (d_k - z_k)^2 \quad \text{or} \quad \frac{1}{2} \|\bar{d} - \bar{z}\|^2$$

factor $\frac{1}{2}$ is introduced for later simplifications (for convinience).

→ The backpropagation learning rule is based on gradiant descent. The weights are ~~twice~~ Initialized with random values, and then changed in the direction that reduce the error (i.e opposite to direction of error gradiant

$$\Delta w = -\eta \cdot \frac{\partial J}{\partial w} \quad \text{or} \quad \Delta w_{pq} = -\eta \frac{\partial J}{\partial w_{pq}}.$$

η – is learning rate $(=\rho)$. (in component form)

Then the weights are iteratively updated as

$$w(t+1) = w(t) + \Delta w(t).$$

→ To obtain the Δw we need to find $\frac{\partial J}{\partial w}$ $\forall\, w_{ij}$ & w_{jk}.

→ Starting from output layer,

$$J = \frac{1}{2} \sum_{k=1}^{n_0} \left[d_k - f_2(net_k)\right]^2$$

we need

$$\cancel{\frac{\partial J}{\partial w_{kj}}} = -\frac{\partial J}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}} \cancel{\left[d_k - f_2(net_k)\right]} \; \cancel{f_2'(net_k)}.$$

$$net_k = \sum_{j=1}^{n_h} w_{kj}\, y_j + b_j \qquad \frac{\partial J}{\partial net_k} =$$

P.T.O

→ Starting from output layer: at n^{th} or t^{th} iteration:

we have $J(\theta) = \frac{1}{2} \sum_{k=1}^{n_o} \left[d_k - f_2(net_k) \right]^2$

we can write

$e(n) = d_k - f_2(net_k)$. for finding Δw_{jk}

we find $\frac{\partial J}{\partial w_{jk}} = \frac{\partial J}{\partial net_k^o} \cdot \frac{\partial net_k^o}{\partial w_{jk}} = -\delta_k \frac{\partial net_k^o}{\partial w_{jk}}$

$net_k^o = \sum_{j=1}^{n_h} w_{jk} y_j \Rightarrow \frac{\partial net_k^o}{\partial w_{jk}} = y_j$

$-\delta_k^o = \frac{\partial J}{\partial net_k^o} = \frac{\partial}{\partial net_k^o} \left[\frac{1}{2} \sum_{k=1}^{n_o} \left[d_k - f_2(net_k^o) \right]^2 \right]$

$= - \left[d_k - f_2(net_k) \right] f_2'(net_k^o) = -(d_k - z_k) f'(net_k)$ —(1)

$\Rightarrow \frac{\partial J}{\partial w_{jk}} = -(d_k - z_k) \cdot f'(net_k) \cdot y_j$

$\Delta w_{jk} = \cancel{\eta\, d_k}\ \eta e_k \cdot f'(net_k) \cdot y_j$

→ Now for updating the weights of inputs to hidden layer

neurons: we need $\Delta w_{ij} = -\eta \frac{\partial J}{\partial w_{ij}}$

$\frac{\partial J}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \cdot \frac{1}{2} \sum_{k=1}^{n_o} \left[d_k - f_2 \left(\sum_{j=1}^{n} w_{jk} \underbrace{f_1(net_j^h)}_{y_j} + b_j \right) \right]^2$

$= \frac{\partial J}{\partial net_j^h} \cdot \frac{\partial net_j^h}{\partial w_{ij}} = -\delta_j^h \frac{\partial net_j^h}{\partial w_{ij}}$ —(2)

$\frac{\partial J}{\partial net_j^h} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial net_j^h}$; $\frac{\partial J}{\partial y_j} = \underbrace{\sum_{k=1}^{n_o} (d_k - z_k) \cdot f_2'(net_k^o) \cdot w_{jk}}$

$\frac{\partial y_j}{\partial net_j^h} = f_1'(net_j^h)$ —(4) $= -\sum_{k=1}^{n_o} \delta_k w_{jk}$ $\frac{\partial J}{\partial y_j}$ from (1) ⟶ (3)

$$\frac{\partial\, net_j}{\partial w_{ij}} = x_i \quad — (5)$$

from (1), (3), (4) & (5), we have

$$\frac{\partial J}{\partial w_{ij}} = -\underbrace{\sum_{k=1}^{n_o} \delta_k^o\, w_{jk}\; f_1'(net_j^h)}_{\delta_j^h} \cdot x_i$$

$$\therefore \quad \Delta w_{ij} = -\eta \frac{\partial J}{\partial w_{ij}} = \eta\, \delta_j\, x_i$$

$$= \eta \left[\sum_{k=1}^{n_o} w_{jk}\, \delta_k^o\right] \cdot f'(net_j^h) \cdot x_i$$

Thus based on these rules we can update the weights at different layers.

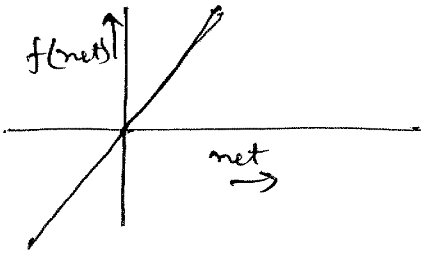
→ Now coming to the selection of activation functions 'f' (or f_1 & f_2) (i) as required in weight update, it must be differentiable.

(ii) to bring proportional change in the output, from its inputs, it must be monotonically increasing.

(iii) it must not shoot up or too low → well with in range that suffic to take a decision.

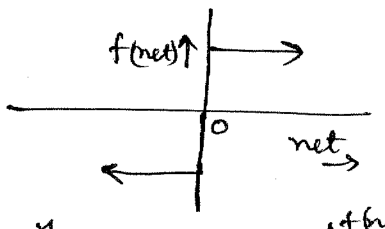
A few common and widely used activation functions:

(i) linear: $f(net) = net$.



(ii) Threshold $f(net) = 1$ for $net \geq 0$

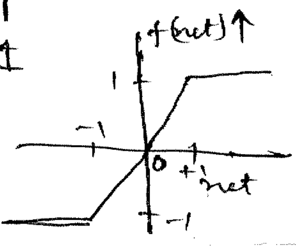
0 for $net < 0$



(iii) Piece wise linear; $f(net) = net$ for $-1 < net < 1$

$= -1$ for $net < -1$

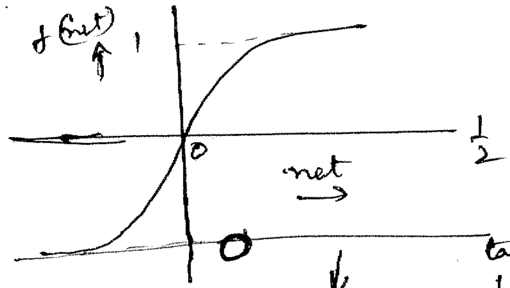
$= 1$ for $net > 1$



Activation function:

(iv) Sigmoid function: $f(net) = \frac{1}{1+e^{-net}}$ $-\infty \leq net \leq \infty$

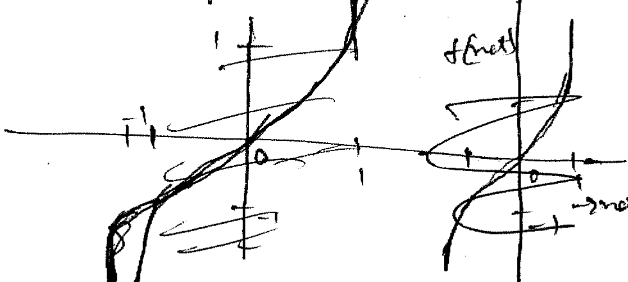
or also called logistic function.

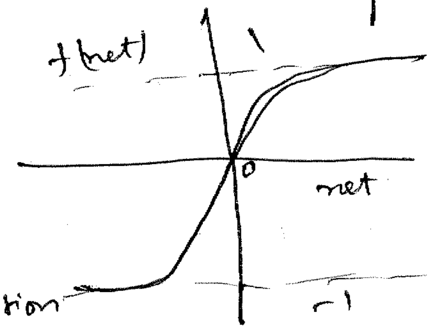


(v) tan hyperbolic function:

$$f(net) = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}}$$

$$\tanh(net) = \frac{e^{2net} - 1}{e^{2net} + 1}$$





Choice of no. of hidden neurons:

→ Higher the ~~~~ complexity of decision boundaries, larger the no. of hidden neurons required

→ As we have discussed & if the samples (of (two classes) are linearly seperable single perceptron or (neuron with threshold activation function) is sufficient to classify correctly.

→ However as the samples become more and more intermingled or not seperable linearly, larger no. ~~~~ hidden neurons are needed

Choice of input & output neurons: Choice of input neurons usually depends on the dimentionality of features of each sample. if d-features are attributed with each sample d- input neuron are used. ~~~~ while the no. of output neurons depends on the no. of classes / clusters in the data. If we

use sigmoid output neuron more ~~no of output~~ even with one or few output neuron more classes can be mapped.
However if we use a threshold activation fuction with output neurons, then each output neuron can classify only two class

No. of hidden layers: depends on the problem ~~domain~~, if we want to incorporate/model operations like Translation
~~Settling in Local minimas~~
Rotation & Scaling each can be modeled by one-layer.
Holistic

Weight update: ~~se~~ Even if we use large network with large no. of neurons, the weight update

$$w(m) = (1-\epsilon)\, w(m)$$, where ϵ in $0<\epsilon<1$

helps to nullify the smaller, possibly unimportant weights

~~No. of~~ Choice of no. of hidden layers: usually single hidden layer NN with arbitarily large no. of neurons must be sufficient to model any nonlinearity in mapping or classification. However if the problem/~~a~~task at hand demands few indepent or various operation like translation rotation & scaling to be incorporated through training processes, than the Multiple hiden layers NN is ~~pre~~ prefered. Each layer can be traind to effectively model ~~o~~ each operation.

Overfitting: If we increase the no. of hidden neurons or layers to a very large extent than there is a possibity of getting the decision boundries/mapping more tuned only to the trainin samples. In that case we may have good ~~test~~ performance with training set, but fails and does not provide even ~~minimum~~ performance on the test set. The decision boundries will be more local than needed. This to some extent can be ~~to~~ over comed by ~~using~~ ~~[illegible]~~ trains with cross validation ~~How to~~ and continuing to min no of neuros that give fairly good performance.

→ Heuristic weight update: will also helps in avoiding over fitting. This eliminates the role of uneffective neuros (or makes them dummy neuros) by decreasing their weigh (heuristically) to almost zero values. This is done by

$$w(m) = \epsilon\, W(m) \qquad 0 < \epsilon \leq 1$$

near to 1

By this smaller weights gets smaller & smaller & subsequently eliminated. where as effective weights pick up well in gradiat decent update process.

Local minima vs global minima: If the cost function has several local minima than the NN may end up ~~with~~ by reaching local minima, based on how the weights are intialised (and how they are updated).

Momentum term: In error surface there can be regions where $\frac{dJ}{dw} \approx 0$ to over come ~~local minima~~ these (Plateaus) as momentum term may be added as ~~w(m+1) = w(m-1) +~~

$$w(m+1) = W(m) + \Delta w_d(m) + \alpha \Delta w(m-1)$$

$\Delta w_d(m)$ - is gradiant decent update

while $\Delta w(m-1) = \cancel{\Delta w(m)}\; w(m) - w(m-1)$.

Then the moment of weights will be more smoother.

$0 < \alpha < 1$.

8. <u>Standardizing input features:</u> Before feeding into the neural network, the input features are to be ~~the~~ scaled & demeanded so as to ensure that all features are in same range. If they are not the features at higher dynamic range / scale will be given more contribution in update as ($net = \sum_{i=1}^{d} w_{ij} x_i$). ~~at~~ weights are uniformly initialized. So effectively we may need to apply whitening transform i.e make all features zero mean & unit variance, before feeding to NN.

<u>Initializing weights:</u> ~~wt~~ weights are usually randomly initialized. (from back propagation update eq. it is clear that if $w_{jk} = 0$ then $\Delta w_{ij} = 0$ and update does not take place. So they can't be zero). usually the weights are uniformly distributed in such a way that $-1 < \underbrace{\sum_{i=1}^{d} w_{ij} x_i}_{net} < 1$ as activation function become linear in ~~contd~~ this range. if weights are high than $|net| > 1$ & saturation occurs. Thus we prefer to ~~to~~ initialize the weights in such a way that $\left| \sum_{i=1}^{d} w_{ij} x_i \right| < 1$. if we have d- dimensions it can be shown that

~~Let~~ if $\frac{-1}{\sqrt{d}} < w_{ij} < \frac{1}{\sqrt{d}}$ $\forall i,j$ then

$$-1 \le \sum_{i=1}^{d} w_{ij} x_j \le 1 .$$

Thus we initialize all weights, by uniform distribution having in range $\left[-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}\right]$.

for hidden to output neurons we initialize in range $\left[-\frac{1}{\sqrt{n_h}}, \frac{1}{\sqrt{n_h}}\right]$.

Non uniform convergence & Bayes Rate:

If the weights are non uniformly initialized there is a possibility that few ~~factors~~ reach their optimal value faster than others & non uniform convergence of NN occurs, this makes the output error to differ markedly from Bayes error rate. we use the above discussed ~~into~~ uniform distribution based initialization to over come this.

Training protocols:

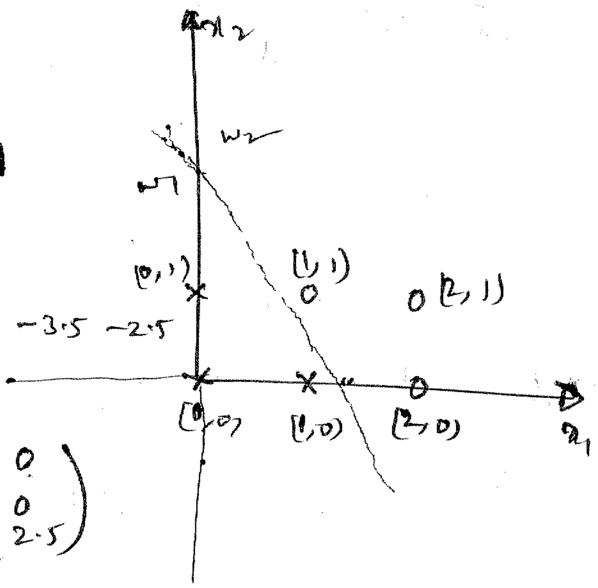
There are three broad categories of Training NN. They are stochastic, batch and on line learning

In stochastic training we present ~~in~~ training pattern randomly from training set and the network weights are updated for each pattern. In batch ~~train~~ training we present all the patterns are presented to network once, ~~before~~ corresponding weight changes are summed up then only each weight update takes place.

In both of these trainings we must ventually make several passes through the training set each pass is called an epoch. In online training each pattern is presented once & only once.

4.

$W = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \quad w_0 = 0.5, \quad \rho = 1$



$W^t x = \quad .5 \quad \underbrace{-.5 \quad -.5}_{\text{mis classified}} \quad -2.5 \quad -3.5 \quad -2.5$

$$W = \begin{pmatrix} -1 \\ -1 \\ 0.5 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 2.5 \end{pmatrix}$$

$$W = \begin{pmatrix} 0 \\ 0 \\ 2.5 \end{pmatrix} - \begin{pmatrix} 5 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} -5 \\ -2 \\ -.5 \end{pmatrix} \qquad -.5 \quad -2.5 \quad -5.5$$

$$W = \begin{pmatrix} -5 \\ -2 \\ -.5 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix} = \begin{pmatrix} -4 \\ -1 \\ 2.5 \end{pmatrix} \qquad \cancel{-ve} \quad 2.5 \quad \underline{-1.5} \quad 1.5 \quad -6.5 \quad -ve \quad -ve$$

$$W = \begin{pmatrix} -4 \\ -1 \\ 2.5 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -3 \\ -1 \\ 3.5 \end{pmatrix}$$

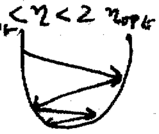
$w^t x + w_0 = 0 \Rightarrow \quad -3x_1 - x_2 + 3.5 = 0$

$\Rightarrow \quad 3x_1 + x_2 - 3.5 = 0 \qquad x_2 = 0 \Rightarrow x_1 = \frac{+3.5}{3}; \quad x_1 = 0 \Rightarrow x_2 = 3.5$

<u>Stopping criterion;</u> $\|\nabla J(w)\| < \theta$, where $J = \sum_{p=1}^{n} J_p$,

<u>optimal learning rate:</u> $\eta_{opt} = \left(\frac{\partial^2 J}{\partial w^2}\right)^{-1}$, Converge for $\eta < 2\eta_{opt}$

$\eta_{opt} < \eta < 2\eta_{opt}$



Slow for $\eta < \eta_{opt}$ or $\eta_{opt} < \eta < 2\eta_{opt}$

diverge for $\eta > 2\eta_{opt}$

<u>activation function prop:</u> Saturation → outputs mapped to posteriors

monotonicity →

Polynomial activation, Radial basis/Gaussian

Sigmoid activation, $\tanh = a\,\frac{e^{b\,net} - e^{-b\,net}}{e^{b\,net} - e^{-b\,net}}$ nonlinear, differentiable → Imp.

Parameter selection →

<u>generating/ creating/ manufacturing training data:</u> Adding d-dimensional Gaussian noise

ex by Rotation, Scale, etc in character recognition.

expressive power: no. of weights

Pattern Recognition

VI – Physical science Assignment – 5 18-4-2012

1. Let $p(\mathbf{x}|\omega_i)$ be arbitrary densities with means $\boldsymbol{\mu}_i$ and covariance matrices $\boldsymbol{\Sigma}_i$— not necessarily normal — for $i = 1, 2$. Let $y = \mathbf{w}^t\mathbf{x}$ be a projection, and let the induced one-dimensional densities $p(y|\omega_i)$ have means μ_i and variances σ_i^2
   (a) Show that criterion function $J_1(\mathbf{w}) = (\mu_1 - \mu_2)^2/(\sigma_1^2 + \sigma_2^2)$ is maximized by $\mathbf{w} = (\Sigma 1 + \Sigma 2)^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$.
   (b) If $P(\omega_i)$ is the prior probability for ω_i, show that $J2(\mathbf{w}) = (\boldsymbol{\mu}1 - \boldsymbol{\mu}2)^2 / (P(\omega 1)\sigma_1^2 + P(\omega 2)\sigma_2^2)$, is maximized with $\mathbf{w} = [P(\omega 1)\Sigma 1 + P(\omega 2)\Sigma 2]^{-1}(\boldsymbol{\mu}1 - \boldsymbol{\mu}2)$.
2. Given samples $x_1 = [0,0]$, $x_2 = [1,0]$, $x_3 = [0,3]$, $x_4 = [-2,0]$, $x_5 = [-3,0]$, $x_6 = [-3,3]$. Find the dissimilarity matrix, (i) using Euclidian distance (ii) City block distance (defined as $d(x_1,x_2) = |dx|+|dy|$, where dx, dy are distances in x and y, directions respectively), and find the similarity matrix using (iii) cosine angle or correlation measure and (iv) tonimato measure.
3. Apply single link and complete link clustering algorithms to samples in above problem based on the (i) Euclidian distance based dissimilarity matrix and (ii) Tanimato measure based similarity matrix. Break in to clusters that contain less than or equal to 3 elements in each cluster. Comment on the kind of clusters resulted by these methods.
4. Given samples $x_1 = [0,0]$, $x_2 = [1,0]$, $x_3 = [0,1]$, $x_4 = [2,0]$, $x_5 = [2,1]$, $x_6 = [1,1]$ where first 3 samples belongs to class1 and next 3 samples to class2, and initial direction of the linear discriminant function $w_0 = [-1,-1]$. Find the direction of linear discriminant function that separates the class 1 and 2 samples, by perception learning, (i) with learning rate of 1 and 0.5 (assume the samples on the separating hyper-plane as correctly classified). (ii) Repeat learning or find decision hyper-plane, with a learning rate of 0.5, if the samples on the separating hyper-plane are assumed to be mis-classified.

   take bias $w_0 = .5$
5. Represent a single neuron as perception and discuss its limitations. Draw a neural network that can be used for XoR classification task.
6. Take a 2-2-1 neural network, take same weights and bias at all neurons as used in class, but use sigmoid activation function at all hidden neurons and threshold activation function at output neuron, and verify whether it solves XoR problem or not?
7. Represent a feed forward neural network with d-input, n-hidden and c-output neurons, designate connection weights, activation functions and outputs. Assume you have a training set with known desired outputs d_k, k = 1:c. Derive the weight update equations by back propagation algorithm.
8. Comment on the choice of number of input neurons, output neurons and hidden neurons. How do you initialize weights and if the input features ~~or~~ are at different scales/have different dynamic range how will you present them to NN? What is the problem of over fitting in NN and how will you minimize it? How do you asses performance of your NN?
9. What is the importance of momentum term and how do we chose activation function. Give few typical choices. What are different types of training NNs.

Support Vector Machine / Maximum margin classifier:

Linear discriminant function: $y(x) = w^t x + w_0$

x assigned to class 1 if $y(x) \geq 0$
else assigned to class 2.

$\therefore$ decision boundary is $y(x) = 0$ or $w^t x + w_0 = 0$. It can be line, plane, or hyper plane.

It can be seen that for two points on hyper plane, x_A, x_B, $y(x_A) = 0$, $y(x_B) = 0 \Rightarrow w^t (x_A - x_B) = 0$ & w is $\perp$er to every line on decision plane.

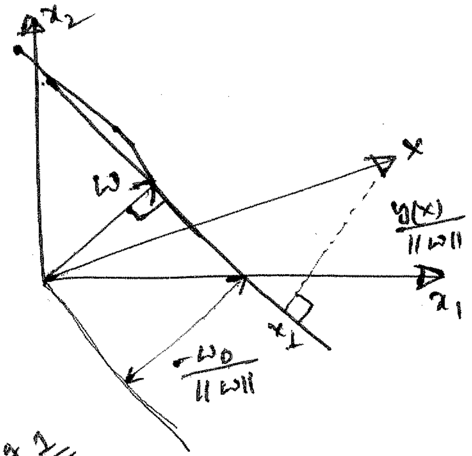


Fig 1

Similarly if 'x_0' is a point on decision surface, then the normal distance of decision surface from origin is given by $\frac{w^t x_0}{\|w\|} = -\frac{w_0}{\|w\|}$

Since $y(x_0) = 0$. Thus the bias parameter determines the location of decision plane.

→ if 'x' is not on decision surface, then $\frac{y(x)}{\|w\|}$ gives the signed $\perp$er distance of point from decision plane.

This can be proved as follows.

if $x = x_\perp + r \frac{w}{\|w\|}$, where $x_\perp$ is on decision plane.

~~$y(x) = y(x_\perp) + x$~~ multiplying both sides by w & ~~two~~ adding w_0 we get

$$\underbrace{w^t x + w_0}_{y(x)} = \underbrace{w^t x_\perp + w_0}_{y(x_\perp)} + r \frac{w^t w}{\|w\|}$$

$$\Rightarrow r = \frac{y(x)}{\|w\|}.$$

Maximum margin classifier:

$$y(x) = w^t \phi(x) + b$$

we decide ω_1 if $y(x) > 0$, ω_2 otherwise.

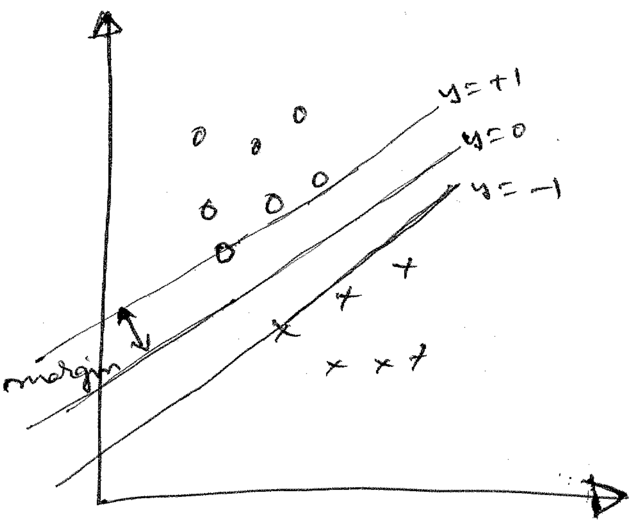
Let the training data set comprises of $x_1, \ldots, x_N$, with lables or corresponding target values $t_1, \ldots, t_N$, where $t_n \in \{-1, 1\}$

such that $t_n = 1$ for $y(x_n) > 0$

$t_n = -1$ for $y(x_n) < 0$

assuming all the points are linearly seperable, i.e there exist at least one 'w' & b to seperate class 1 & 2 data samples.

observe that $t_n y(x_n) \geq 0$ ~~always~~ for all data points



In general several solutions can exist for 'w', that seperates the two class data. Here we are intrested in such a solution that maximize the 'margin' w.r.to two class data points. Here margin is defined as the $\perp^{er}$ distance from closest point (of both classes) to the plane seperating the two class data. In Fig 1

We quantified the signed $\perp^{er}$ distance of any point to hyper plane $y(x) = 0$ as $\frac{y(x_n)}{\|w\|}$. Thus the (absolute) distance of point x_n to the decision surface is given by

$$\frac{t_n \, y(x_n)}{\|w\|} = \frac{t_n \left(w^t \phi(x_n) + w_0\right)}{\|w\|}$$

This margin ~~$t_n(w^t\phi(x_n)+b)$~~

$$\frac{t_n\left(w^t\phi(x_n)+b\right)}{\|w\|}$$

gives the $\perp$er distance from plane to closest point (x_n) in the data set, we wish to optimize wrt ~~the~~ parameters 'w' & 'b' in order to maximize the margin. The maximum margin solution is found by solving

$$\underset{w,b}{\arg\max}\left\{\frac{1}{\|w\|}\cdot\min_n\left(t_n\, w^t\phi(x_n)+b\right)\right\}$$

maximizing margin.

for closest point x_n

if we set $w \to kw$ & $b \to kb$ than the distance with x_n to decision surface $\frac{y(x_n)}{\|w\|}$ will not change thus we utilize this freedom to set

$$t_n\left(w^t\phi(x_n)+b\right)=1 \qquad -①$$

for the closest point. For all other points

$$t_n\left(w^t\phi(x_n)+b\right)\geq 1 \qquad -②$$

The constraint is said to be active, if the equality holds. There will always be one such closest point. And when we maximize the margin (wrt two class data points) there will at least be two data points ~~two~~ (one from each class) for which the constraint ② is active. On the other hand we need to ~~minimize~~ maximize $\|w\|^{-1}$, or we can minimize $\|w\|^2$. So we have to solve the optimization problem ~~arg min~~ $\underset{w,b}{\arg\min}\ \frac{1}{2}\|w\|^2$, wrt constraint ②. factor '$\frac{1}{2}$' is for mathematical convinience.

(4)

~~To~~ To do this we construct a Lagrange multiplier function

$$L(w,b,\underline{a}) = \frac{1}{2}\|w\|^2 - \sum_{n=1}^{N} a_n \{ t_n (w^t \phi(x_n) + b) - 1 \} \quad \text{— (4)}$$

Where $\underline{a} = (a_1, \ldots, a_N)^t$.

-ve sign in front of second term is because, though we need to minimize wrt w & b, we are maximizing wrt 'a_n'. Setting derivative of $L(w,b,\underline{a})$ wrt w & b to zero we get

$$w = \sum_{n=1}^{N} a_n t_n \phi(x_n) \quad \text{— 5}$$

and

$$\sum_{n=1}^{n} a_n t_n = 0 \quad \text{— 6}$$

using (5) & (6) & eliminating w & b from $L(w,b,a)$ gives dual representation of the maximum margin problem. ~~which~~

$$\tilde{L}(w,b,a) = \frac{1}{2}\langle w \cdot w\rangle - \sum_{n=1}^{N} a_n t_n (w^t \phi(x_n) + b) + \sum_{n=1}^{N} a_n$$

$$\tilde{L}(a) = \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m \langle \phi(x)\cdot\phi(x)\rangle$$

$$- \sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_m t_n \langle \phi(x_n)\cdot\phi(x_m)\rangle + 0 + \sum_{n=1}^{N} a_n$$

$$\tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2}\cdot\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m \, k(x_n, x_m) \, \cancel{\langle \phi(x_n)\cdot\phi(x_m)\rangle}$$

where $k(x_n, x_m) = \langle \phi(x_n)\cdot\phi(x_m)\rangle = \phi(x_n)^T \phi(x_m)$

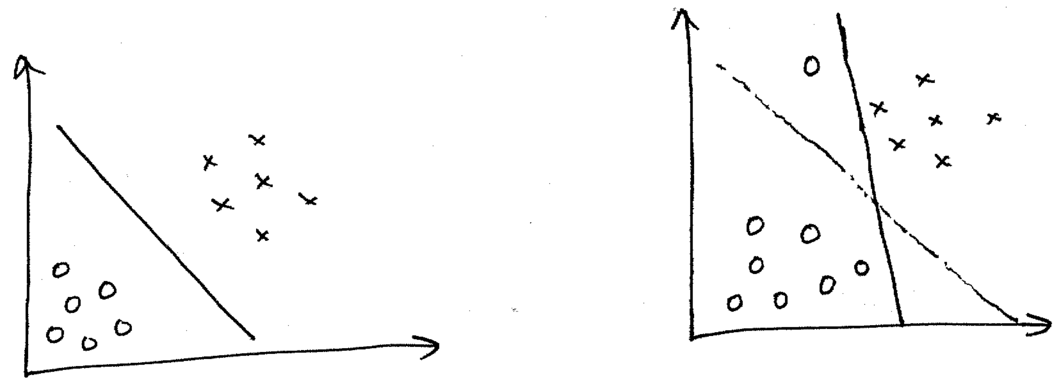
This dual we need to maximize with the constraints

$a_n \geq 0 \quad \forall \; n = 1, \ldots, N$ and $\sum_{n=1}^{N} a_n t_n = 0$.

In order to classify new data point, with trained model, we have $y(x) = w^t x + b = \sum_{n=1}^{N} a_n t_n k(x, x_n) + b$.

Support vector Machine: Regularization in non Seperable case:

While mapping data to high dimensional feature space via ϕ does generally increase the likelihood that the data is seperable, but does not guarantee that it always will be so. Also, in some cases it is not clear that finding a separating hyperplane is exactly what we would want to do, since that might be susceptible to outliers. For instance, the left figure below shows an optimal margin classifier, and when a single outlier is added in the upper left region (right figure), it causes the decision boundary to make a dramatic swing, and the resulting classifier has a much lesser margin.



To make the algorithm work for non-linearly seperable data sets as well as be less sensitive to outliers, we reformulate our optimization (using l_1 regularization) as follows:

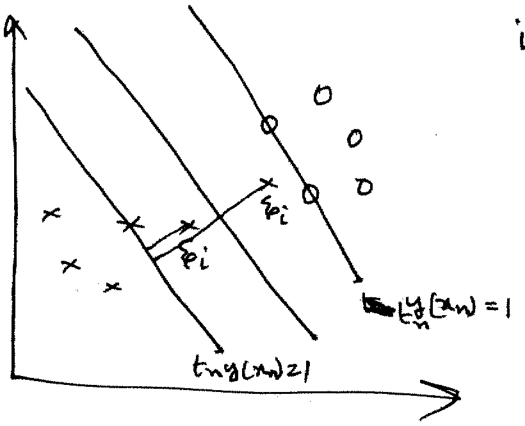
$$\min_{w,b} \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i$$

such that $$t_n\left(w^t \phi(x_n) + b\right) \geq 1 - \xi_n, \quad n = 1, \ldots, m.$$

$$\xi_n \geq 0, \quad n = 1, \ldots, m.$$

Thus examples are now permitted to have functional margin less than 1. And if an example has a functional margin $1-\xi_n$

(with $\xi > 0$), we would pay a cost of the objective function being increased by $C\xi_n$.



if for a sample $t_n y(x_n) = 1 - \xi_i$, $\xi_i > 0$

$C\xi_i \rightarrow$ is its associated cost.

if $0 < \xi_i < 1$ it is correctly classified but in between the separating hyperplane & margin. If $\xi_i > 1$ then it is an ~~out lair~~ outlier, wrongly classified with associated cost $C\xi_i$.

The Parameter C controls the relative weighting between the twin goals of making the $\|w\|^2$ small (which makes the margin large) and of ensuring that most examples have functional margin at least 1.

For this non-seperable case, we can form the Lagrangian:

$$L(w, b, \xi, \alpha, r) = \frac{1}{2} w^t w + C \sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i \left[t_i (\phi(x)^t w + b) - 1 + \xi_i \right] - \sum_{i=1}^{m} r_i \xi_i \quad — (D)$$

Here, the α_i's & r_i's are Lagrange multipliers (constrained to be ≥ 0). After setting the derivatives wrt ξ_i, w & b to zero as before, substituting them back in, & simplifying, we get following dual form of the problem:

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} t_i t_j \alpha_i \alpha_j \langle \phi(x_i), \phi(x_j) \rangle$$

Such that $0 \leq \alpha_i \leq C$, $i = 1, \ldots, m$

$$\sum_{i=1}^{m} \alpha_i t_i = 0$$

as before $w = \sum_{i=1}^{m} \alpha_i t_i \phi(x_i)$ and $y(x) = w^t x + b$

$$= \sum_{i=1}^{m} \alpha_i t_i \langle x_i, x \rangle + b$$

$$y(x) = w^t x + b$$
$$= \sum_{i=1}^{m} \alpha_i t_i \langle x_i, x \rangle + b.$$

we note that, some what surprisingly, in adding l_1 regularization the only change to the dual problem is that what was originally constraint that $0 \le \alpha_i$ has now become $0 \le \alpha_i \le C$.

This dual problem can be solved by sequential minimal optimization (SMO).

Sequential ~~optimal~~ minimal optimization (SMO): The SMO algorithm gives an efficient way of solving dual problems arising from the derivation of SVM. It is based on coordinate ascent optimization algorithm.

coordinate ascent: consider trying to solve the unconstrained optimization problem $\max_{\alpha} W(\alpha_1, \alpha_2, \ldots, \alpha_m)$

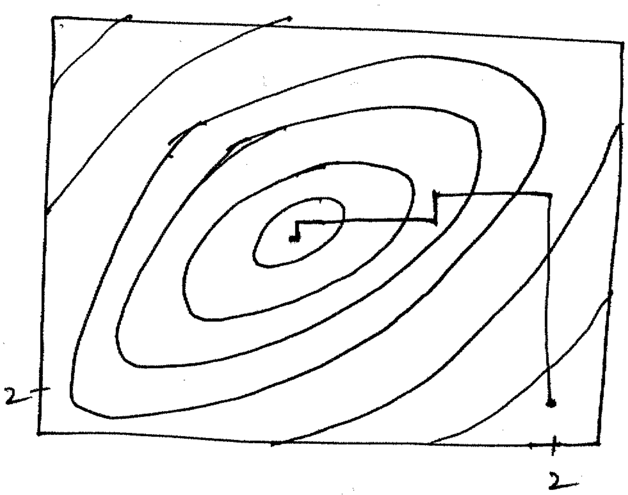
Here, we think of W as just some function of the parameters α_i's. and for now we ignore any relationship between this problem & SVMs. The coordinate ascent can be summarized as

Loop until convergence {
  for $i = 1, \ldots, m$ {
    $\alpha_i : \arg\max_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$.
  }
}

Thus, in the innermost loop of this algorithm, we will hold all the variables except for some α_i fixed, and re-optimize W with respect to just the parameter α_i. Finally algorithm re-optimizes wrto $\alpha_1, \ldots, \alpha_m$, each at a time holding all others fixed.

A more sophisticated version might choose the ordering. For instance, we may choose the next variable to update according to which one we expect to allow us to make the largest increase in $W(\alpha)$.

When the function W happens to be of such a form that the "arg max" in the inner loop can be performed efficiently then coordinate ascent can be a fairly efficient algorithm. Here is a picture of coordinate ascent in action:



The elipses in the figure are the contours of a quadratic function that we want to optimize. Coordinate ascent is initialized at $(2,-2)$ and also plotted in figure is the path it took on its way to the ~~global~~ global maximum. We note that coordinate ascent takes a step that's parallel to one of the axes, since only one variable is optimized at a time.

<u>SMO</u>: Now we give details of SMO algorithm to solve dual problem in SVMs. Here is the dual optimization problem that we want to solve. (assume $N = m$)

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j}^{N} \alpha_i \alpha_j t_i t_j \langle x_i, x_j \rangle \quad -(1)$$

$$\text{s.t.} \quad 0 \le \alpha_i \le C, \quad i = 1, \dots, N \quad -(2)$$

$$\text{and} \quad \sum_{i=1}^{N} \alpha_i t_i = 0 \quad -(3)$$

Lets say that we have α_i's satisfying 2 & 3. Now suppose we want to hold ~~$\alpha_2, \ldots, \alpha_m$~~ $\alpha_2, \ldots, \alpha_N$ fixed, and take a coordinate ascent step and re-optimize the objective with respect to α_1. Can we make any progress? The answer is no, because the constain 3 ensures

$$\sum_{i=1}^{N} \alpha_i t_i = 0 \Rightarrow \alpha_1 t_1 = -\sum_{i=2}^{N} \alpha_i t_i$$

$$\cancel{\Rightarrow t_1 = -\alpha_1 \sum_{i=2}^{N} \alpha_i t_i}$$

(since ~~$\alpha_1^2 = 1$ as $\alpha_1 = \{-1, 1\}$ we have this~~). ~~Thus~~

$$\alpha_1 = -t_1 \sum_{i=2}^{N} \alpha_i t_i$$

(Since $t_1 \in \{-1, 1\}$, $t_1^2 = 1$, we have this). Thus α_1 is completely determined by $\{\alpha_2, \ldots, \alpha_N\}$. Thus if we hold $\{\alpha_2, \ldots, \alpha_N\}$ we cant make any change in α_1, without violating the constraint 3 in the optimization.

Thus if we want to optimize some subset of the α_i's we must update at least two of them simultaneously in order to keep satisfying the constraints. This motivates the SMO algorithm which simply does the following:

~~Repete~~ Repeat till convergence {

1. Select some pair ~~α_i & α_j~~ α_i & α_j to update next (using a heuristic that ~~allows~~ tries to pick the two that allow us to make the biggest progress towards global maximum).

2. Re optimize $W(\alpha)$ with respect to α_i & α_j, while holding all other α_k's $(k \neq i, j)$ fixed.

}.

To test the convergence of the algorithm we ~~test~~ check Karush-Kuhn-Tucker (KKT) conditions, whether are satisfied with in a tolerence are not. KKT conditions ~~can be~~ are given by

$$\alpha_i = 0 \Rightarrow t_i(w^t x_i + b) \geq 1$$

$$\alpha_i = c \Rightarrow t_i(w^t x_i + b) \leq 1$$

$$0 \leq \alpha_i \leq c \Rightarrow t_i(w^t x_i + b) = 1.$$

See the last page for derivation of these conditions.

The key reason for SMO to be an efficient algorithm is that the update to α_i, α_j can be computed very efficiently. The brief sketch of the SMO ~~is as follows~~ for deriving efficient update is as follows.

Suppose we have decided to hold $\alpha_3, \ldots, \alpha_N$ fixed and want to re-optimize $W(\alpha_1, \ldots, \alpha_N)$ wrt α_1 & α_2, subject to the constraint 2 & 3.

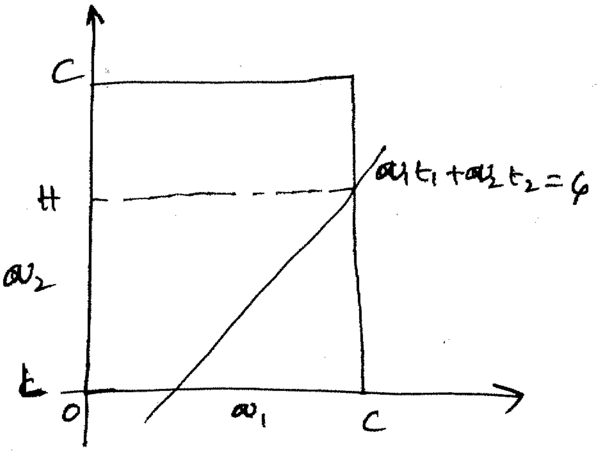
From 3 we can write

$$\alpha_1 t_1 + \alpha_2 t_n = \underbrace{-\sum_{i=3}^{N} \alpha_i t_i}_{\zeta}$$

Since RHS is fixed we let denote it by a constant ζ.

Then $\cancel{\alpha_1 = t_1(\zeta - \alpha_2 t_2)}$ $\alpha_1 t_1 + \alpha_2 t_2 = \zeta$. — 4

We can picture the constraint α_1 & α_2 as follows:



~~from to~~

From constraint (2) we know that ~~α_1, α_2~~ α_1, α_2 must lie within the box $[0,C] \times [0,C]$ shown. also plotted is the line $\alpha_1 t_1 + \alpha_2 t_2 = \zeta$ on which α_1, α_2 must lie. We note that from these two constraints $L < \alpha_2 < H$.

Now we can rewrite eq (4) as

$$\alpha_1 = t_1 (\zeta - \alpha_2 t_2) \quad \text{— (5)}$$

Hence the objective $W(\alpha)$ can be written as

$$W(\alpha_1, \alpha_2, \ldots, \alpha_m) = W(t_1(\zeta - \alpha_2 t_2), \alpha_2, \alpha_3, \ldots, \alpha_N)$$

Treating $\alpha_3, \ldots, \alpha_m$ as constants, this is just some quadratic function in α_2. i.e this can also be expressed in the form $\alpha \alpha_2^2 + \beta \alpha_2 + \gamma$ ~~for~~ for some appropriate α, β, & γ. If we ignore the box constraint (or $L \le \alpha_2 \le H$) we can easily maximize this quadratic function by setting its derivative to zero & solving. Let that be $\alpha_2^{new, unclipped}$.

But we want to maximize '$W(\alpha)$' ~~want~~ subject to box constraint & $L \le \alpha_2 \le H$, in that case we have

$$\alpha_2^{new} = \begin{cases} H & \text{if } \alpha_2^{new, unclipped} > H \\ \alpha_2^{new, unclipped} & \text{if } L \le \alpha_2^{new, unclipped} < H \\ L & \text{if } \alpha_2^{new, unclipped} < L. \end{cases}$$

Now with this α_2^{new} we can use (5) to find α_1^{new}. Here few considerations to be made are the choice of the ~~box~~ heuristics used to

Select the next ~~α_i, α_j~~ α_i, α_j to update and other is how to update 'b' as the SMO algorithm runs.

derivation of Karush - Kuhn - Tucker Conditions (KKT):

we have ~~for~~ the Lagrangian ~~cost~~ function for non seperable case as

$$L(w, b, \xi, \alpha, \gamma) = \frac{1}{2} w^t w + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i \left[t_i (x_i^t w + b) - 1 + \xi_i \right] - \sum_{i=1}^{N} \gamma_i \xi_i \quad \text{--- } (0)$$

KKT conditions for this are

$$\gamma_i \xi_i = 0 \quad \text{------ } (1)$$

$$\alpha_i \left[t_i (x_i^t w + b) - 1 + \xi_i \right] = 0$$

or

$$t_i (x_i^t w + b) - 1 + \xi_i \geq 0 \quad - (2)$$

and we have ~~$C - \alpha_i - \xi_i = 0$~~ $\frac{\partial L}{\partial \xi_i} = 0 \Rightarrow C - \alpha_i - \gamma_i = 0$

(i) $\alpha_i = 0 \Rightarrow C - \gamma_i = 0 \quad \gamma_i > 0 \Rightarrow \xi_i = 0$

$$\Rightarrow t_i (x_i^t w + b) - 1 \geq 0$$

$$\Rightarrow t_i (x_i^t w + b) \geq 1 \quad - (i)$$

(ii) $0 < \alpha_i < C \Rightarrow \gamma_i > 0 \Rightarrow \xi_i = 0$

$$t_i (x_i^t w + b) - 1 = 0 \Rightarrow t_i (x_i^t w + b) = 1 \quad - (ii)$$

(iii) $\alpha_i = C \Rightarrow \gamma_i = 0 \Rightarrow \xi \geq 0$

$$t_i (x_i^t w + b) - 1 + \xi_i = 0$$

$$t_i (x_i^t w + b) = 1 - \xi_i \Rightarrow t_i (x_i^t w + b) \leq 1 \quad - (iii)$$

derivation of dual form in non seperable case:

The ~~cost~~ Lagrangian cost function to be minimized is given by

$$L(\omega, b, \xi, \alpha, \gamma) = \frac{1}{2}\omega^t\omega + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\left[t_i\left(x_i^t\omega+b\right)-1+\xi_i\right] - \sum_{i=1}^{N}\gamma_i\xi_i \quad - (0)$$

$$\frac{\partial L}{\partial \omega} = 0 \Rightarrow \omega - \sum_{i=1}^{N}\alpha_i t_i x_i = 0 \quad - (1)$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{N}\alpha_i t_i = 0 \quad - (2)$$

$$\frac{\partial L}{\partial \xi_i} = 0 \Rightarrow C - \alpha_i - \gamma_i = 0 \quad - (3)$$

we have $\alpha_i \geq 0$ & $\gamma_i \geq 0$ from (3) $\gamma_i = C - \alpha_i$

$\gamma_i \geq 0 \Rightarrow C - \alpha_i \geq 0 \Rightarrow C \geq \alpha_i$

$\Rightarrow \quad 0 \leq \alpha_i \leq C.$

KKT conditions are

$$\alpha_i\left[t_i\left(x_i^t\omega+b\right)-1+\xi_i\right] = 0$$

$$\gamma_i\xi_i = 0$$

$\xi_i \geq 0$ &

$$t_i\left(x_i^t\omega+b\right)-1+\xi_i \geq 0$$

Substituting (1), (2), (3) in (0) yeilds dual form which is to be maximized

$$\boxed{\begin{array}{l}\text{maximize} \quad W(\alpha) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_i\sum_j\alpha_i\alpha_j t_i t_j \langle x_i, x_j\rangle \\ \text{Subject to} \quad \sum_{i=1}^{N} t_i\alpha_i = 0 \quad \& \quad 0 \leq \alpha_i \leq C, \ i=1,\ldots,N.\end{array}}$$