learning → Supervised : predicting using the available data with labels.

Un-supervised : When the data is provided without a label.

analysing the data to predict

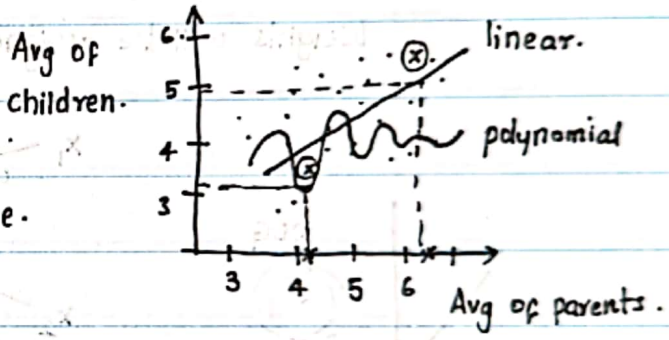
Supervised → Regression (Trying to map input var to continuos out fn)

classification → gives discrete output

Suppose, we were given data of (average of heights of parents) and (Avg of heights of children)

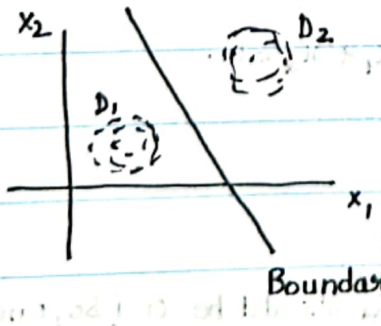
we can establish a relation between 2 datasets. it can be linear / polynomial.

And in both cases, it can be error-prone.

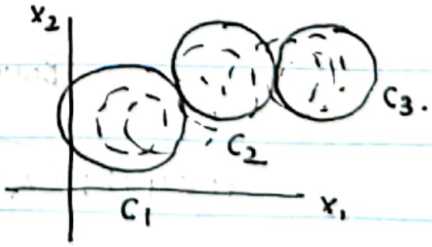


To avoid errors, we need more data but processing more data can be difficult.

→ In supervised learning, in order to classify the data we need a boundary that seperates the datasets. where as, in unsupervised learning we form clusters to determine / classify.



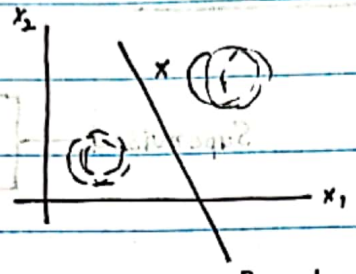
(Supervised)



C_1, C_2, C_3 : Clusters

(unsupervised).

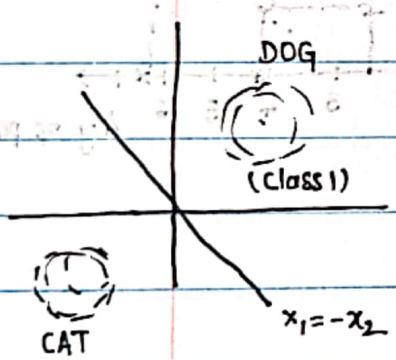
Supervised learning: provided the data with labels, we need a boundary to classify the input data.

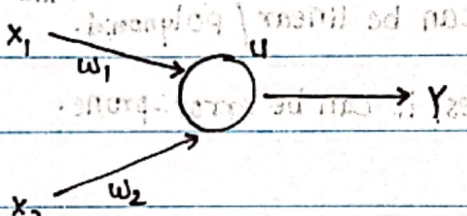


input lies left side of Boundary
↳ belongs to dataset 1

Right side of boundary
↳ belongs to D2.

Weights will be assigned to each dataset (w_1, w_2).





$U = x_1 w_1 + x_2 w_2$

if $U \geq 0$: $Y = 1$

else $Y = 0$.

When $U = 0$; we get a boundary line.

$x_1 w_1 + x_2 w_2 = 0$ ...

(if $w_1, w_2 = 1$) $\Rightarrow$ $x_1 = -x_2$

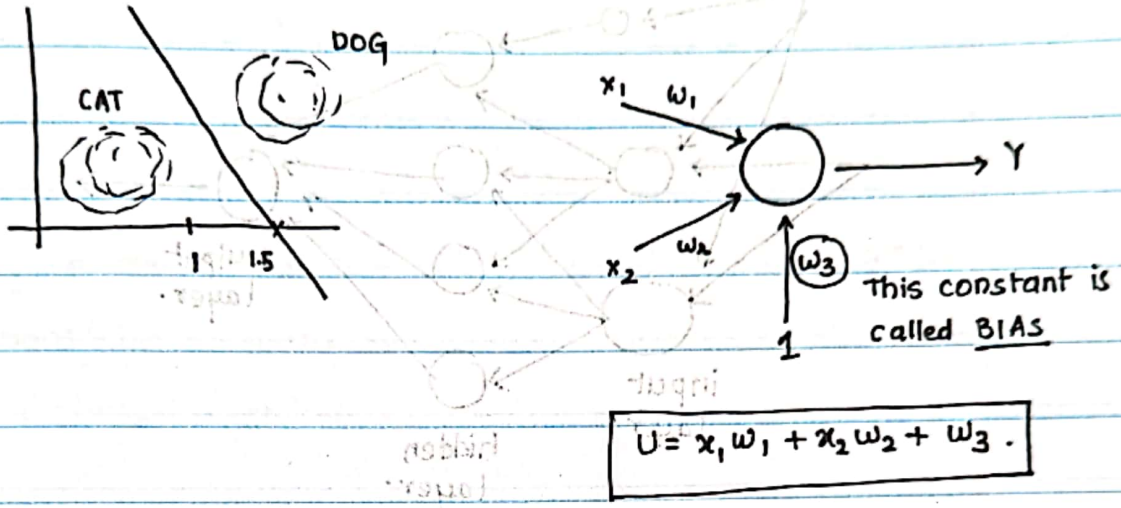
Boundary line : $x_1 + x_2 = 0$.

For $y = 1$; $x_1 + x_2 = 1$

$x_1 w_1 + x_2 w_2 = 1$

when $w_2 = 1$; x_2 should be 0 (So, one dataset will be insignificant).

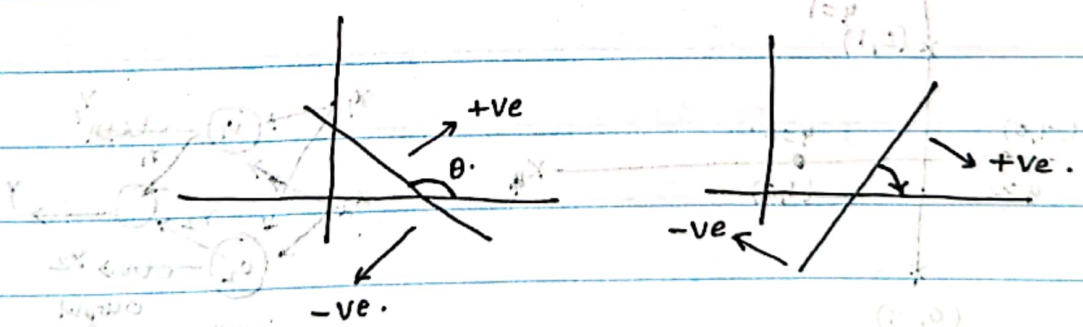
When the boundary isn't passing through the centre, we need an additional constant to differentiate with the boundary.



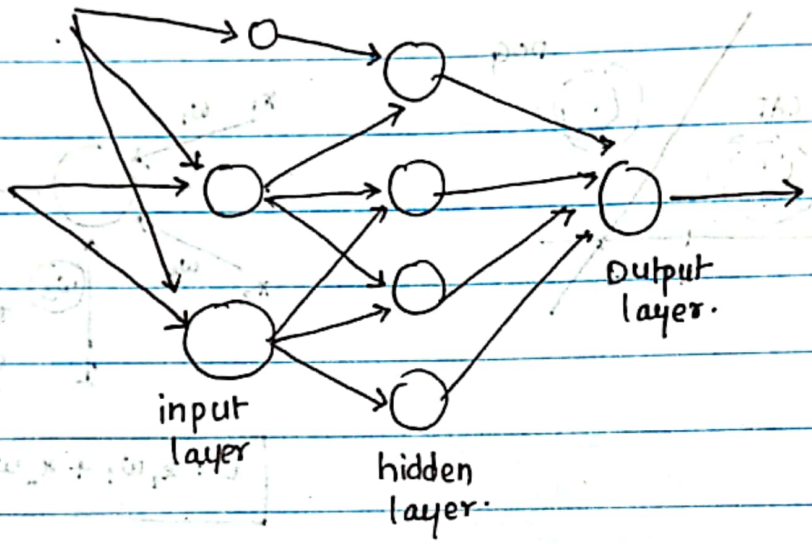
$$U = x_1 w_1 + x_2 w_2 + w_3 .$$

→ (For the line, slope always +ve)

Slope will be calculated w.r.t x-axis, so the values towards x-axis angle is +ve.

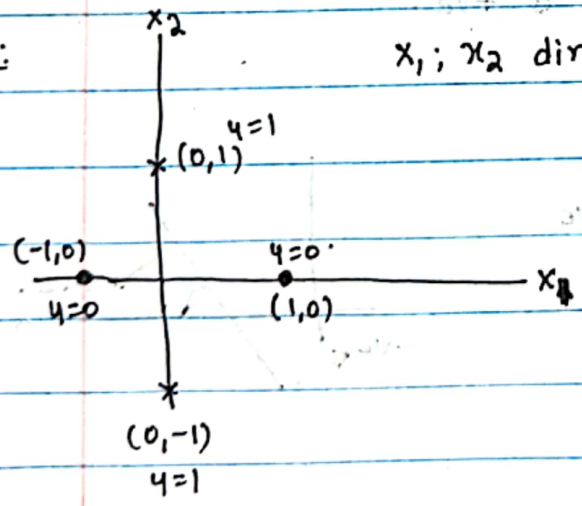


The layers which we don't have access to is called hidden layer.

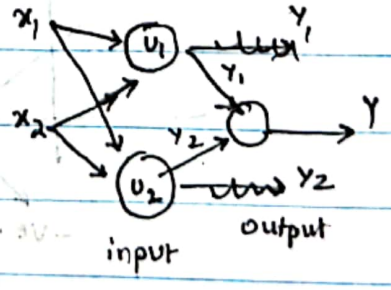


When the result can be given without any lines (neurons) i.e there are no hidden layers we don't need neurons (lines).

Ex:



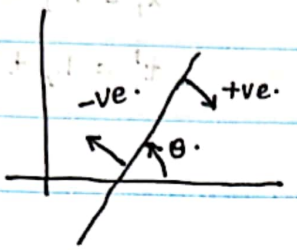
x_1 ; x_2 directly gives the desired output.



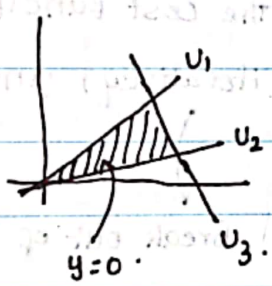
(no hidden layers)

→ So, no neurons in hidden layer

The angular region (region between x-axis and our reference line) will be positive and remaining part will be negative.



when finding common region to distinguish datasets ($y=0$ or $y=1$), consider all the neurons/lines making up region into same sign. so that it will be easier to distinguish/get line equations.

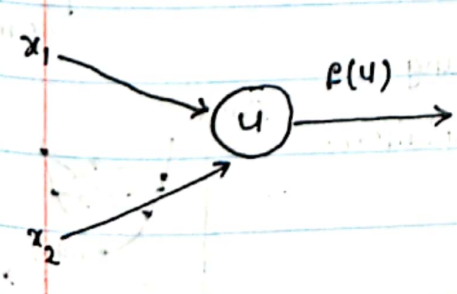


else $y=1$

towards inside (for region) consider -ve for all neurons/lines.

if it's positive, multiply the line equation with -1 to change sign.

Gradient Descent Algorithm: In order to get the minimum errors, we calculate cost function $J(w_1)$ [when there's only one variable] and randomly assume values for w and calculate cost value. And, for value which cost is minimum, that 'w' value will be considered.



$$f(u) = \begin{cases} 1 \text{ if } u \geq 0 \\ 0 \text{ if } u < 0 \end{cases}$$

$m \Rightarrow$ # of samples

$x^i \rightarrow i^{th}$ sample.

$t^i \rightarrow$ target/output

$x_i \rightarrow i^{th}$ attribute

$n \rightarrow$ Dimension.

Example:

x_1	x_2	target(t)
1	0	1
0	1	0

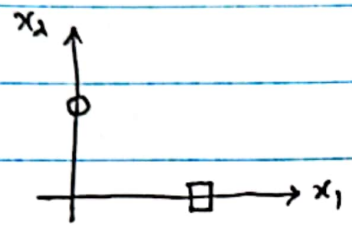
$m = 2$

$x^1 = (1,0)$

$x^2 = (0,1)$

$x_1^1 = 1, \; x_2^1 = 0$

$t^1 = 1, \; t^2 = 0.$



Error / Cost Function / Loss Function = $J(w) = \frac{1}{2m} \sum_{i=1}^{m} (f(v^i) - t^i)^2.$

Gradient Descent: Based on random selection of w_1, w_2 we calculate the cost function. And differentiate the cost function (iteratively) until you get the minimum Cost / Error value.

Break out of the loop when

i) Error is very minimal. (like $\varepsilon < 0.0001$)

(and)

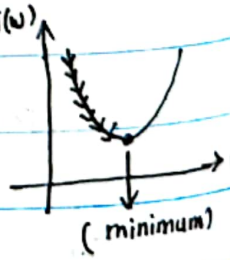
ii) There's no change in Δw. (change in w).

$w_1 = w_1 - \rho \Delta w_1$

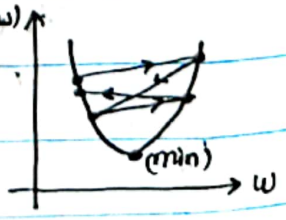
$w_2 = w_2 - \rho \Delta w_2.$

where $\rho \rightarrow$ learning rule / learning rate.

→ if ρ is small, the steps we take to reach the solution will be very small. (Takes more time).

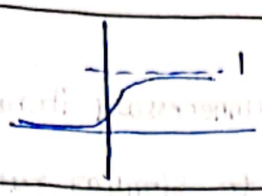


→ if ρ is high, w_1/w_2 values will change drastically and we might not get a solution.



Sigmoid function.

$$f(u) = \frac{1}{1+e^{-u}}$$



deciding function we use to determine y value.

→ in case of non-linear classification, we take the data to the higher dimension in order to classify the data.

Example: $x_3 = x_1^2 + x_2^2$

(or) $ax^2 + by^2 + c^2 = 0 \Rightarrow$ change this to $ax_1 + by_1 + c^2 = 0$

where $x_1 = x^2$; $y_1 = y^2$

Using Chain rule, $\frac{\partial}{\partial w_j}(J(w)) = \frac{\partial J}{\partial f} \cdot \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial w_j}$

$$\frac{\partial J}{\partial f} = \frac{\partial}{\partial f}\left(\frac{1}{2m}\sum_{i=1}^{m}(f(u^i) - t^i)^2\right) = 2(f(u) - t)$$

$$\frac{\partial f}{\partial u} = \frac{\partial}{\partial u}\left((1+e^{-u})^{-1}\right) = -1\left(e^{-u}\right)\cdot\left(1+e^{-u}\right)^{-2} = \frac{e^{-u}}{1+e^{-u}} \cdot \frac{1}{1+e^{-u}}$$

$$= 1 - \frac{1}{1+e^{u}} \cdot \frac{1}{1+e^{-u}}$$

$$\frac{\partial f}{\partial u} = (1 - f(u)) \cdot f(u)$$

$$\frac{\partial u}{\partial w_j} = \frac{\partial}{\partial w_j}\left(\sum_{i=1}^{n}(x_j w_j)\right) = x_j$$

$$\Rightarrow \boxed{\Delta w = \frac{\partial (J(w))}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}\left[(f(u^i) - t^i) \cdot f(u^i)(1 - f(u)^i) \cdot x_j^i\right]}$$

$$\boxed{w_j = w_j - \alpha \Delta w}$$

α : Learning rate.

Scaling: In order to avoid the unnecessary iterations / to speed up the procedure we scale our values to similar ranges.

↳ we scale our values to a standard unit

Example:

x_i	Deviation	Squared	standard unit
1	-2.5	6.25	$-2.5/1.7 = -1.47$
2	-1.5	2.25	$-1.5/1.7 = -0.88$
3	-0.5	0.25	$-0.5/1.7 = -0.29$
4	0.5	0.25	$0.5/1.7 = 0.29$
5	1.5	2.25	$1.5/1.7 = 0.88$
6	2.5	6.25	$2.5/1.7 = 1.47$
	Φ	Avg: $17.5/6 = \boxed{2.9}$	

Average : $\frac{21}{6} = 3.5$

variance.

→ we can't consider deviation as standard unit, as in most cases Summation will be zero which doesn't help us.

Standard deviation $= \sqrt{2.9} \simeq 1.7$

↳ we use, $\frac{x_i - \mu_i}{sd_i} = \boxed{\frac{x_i - \text{Average}}{\text{standard deviation}}}$ as our standard unit. to scale our data.

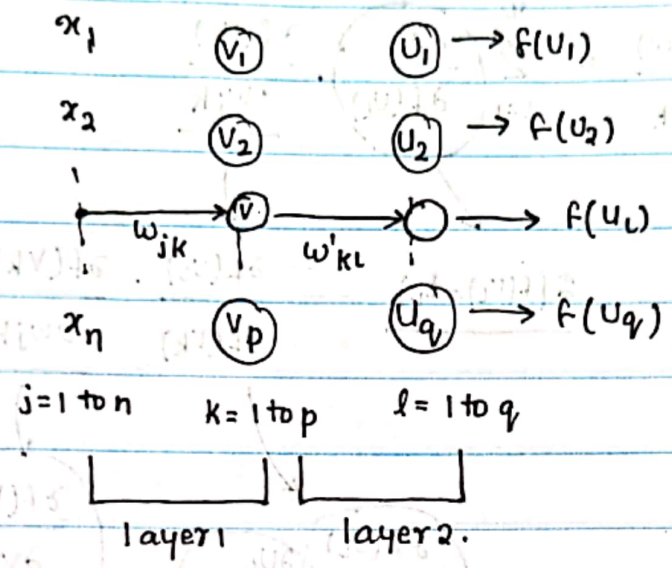
x_i : input values for feature (i)

μ_i : Average of feature (i)

sd = standard deviation of feature (i)

↳ This is called MEAN NORMALIZATION method.

Training Two Layer Network:



m : number of samples.

cost Function $J(w) = \sum_{i=1}^{m} \cdot \sum_{l=1}^{q} (f(u_l) - t_l)^2$

$$\Delta w'_{kl} = \frac{\partial J(w)}{w'_{kl}} = \frac{\partial J(w)}{\partial F(u_l)} \cdot \frac{\partial F(u_l)}{\partial u_l} \cdot \frac{\partial u_l}{\partial w'_{kl}}$$

$u_l = \sum_{k=1}^{p} f(v_k) \cdot w'_{kl}$.

$2(f(u_l) - t_l)$ $(f(u_l) \cdot (1 - f(u_l))$ $f(v_k)$:

$$\Delta w'_{kl} = 2(f(u_l) - t_l) \cdot (f(u_l) \cdot (1 - f(u_l)) \cdot f(v_k)$$

Layer 02

$$\Delta w_{kl}' = \sum_{i=1}^{m} 2 f(v_k^i) \cdot \sum_{l=1}^{q} (f(u_l^i) - t_l^i)(f(u_l^i) \cdot (1 - f(u_l^i)))$$

For Layer 01 :

$$\Delta w_{jk} = \frac{\partial J(w)}{\partial w_{jk}} = \sum_{l=1}^{q} \left(\frac{\partial J}{\partial f(u_l)} \right) \cdot \frac{\partial f(u_l)}{\partial w_{jk}}$$

$$\frac{\partial J}{\partial f(u_l)} = 2(f(u_l) - t_l)$$

$$\frac{\partial f(u_l)}{\partial w_{jk}} = \frac{\partial f(u_l)}{\partial f(v_k)} \cdot \frac{\partial f(v_k)}{\partial w_{jk}}$$

$$\frac{\partial f(u_l)}{\partial f(v_k)} = \frac{\partial f(u_l)}{\partial u_l} \cdot \frac{\partial u_l}{\partial f(v_k)}$$

$$\frac{\partial f(u_l)}{\partial u_l} = (f(u_l) \cdot (1 - f(u_l))$$

$$\frac{\partial u_l}{\partial f(v_k)} = w_{kl}'$$

$$\frac{\partial f(v_k)}{\partial w_{jk}} = \frac{\partial f(v_k)}{\partial v_k} \cdot \frac{\partial v_k}{w_{jk}}$$

$$\frac{\partial f(v_k)}{\partial v_k} = (f(v_k) \cdot (1 - f(v_k))$$

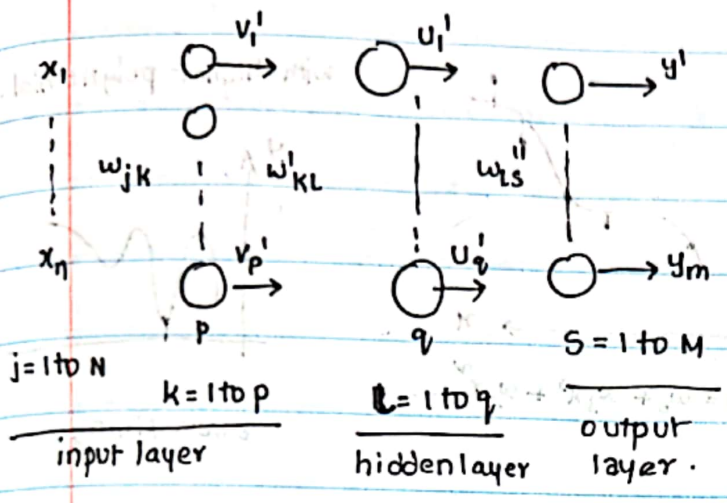
$$\frac{\partial v_k}{w_{jk}} = x_j$$

$$\boxed{\Delta w_{jk} = 2 x_j \left(f(v_k) \cdot (1 - f(v_k)) \sum_{l=1}^{q} (f(u_l) - t_l) \cdot f(u_l)(1 - f(u_l)) \cdot w_{kl}' \right)}$$

→ So, we have to calculate the Δw for layer 2 and then we have to calculate for the layer 1 until convergence.

↳ we call this **Back propogation Training** Algorithm.

Training 3 layers : ↳ Back propogation.



$$\boxed{\Delta w_{LS}'' = \rho\, u_L'\, y_S\,(1-y_S)\,(y_S - t_S)}$$

let $y_S(1-y_S)(y_S-t_S) = \delta_S$

$$\Delta w_{KL}' = \rho\, v_K'\, u_L'(1-u_L') \sum_{S=1}^{M} w_{LS}''\, y_S(1-y_S)(y_S-t_S)$$

$$\boxed{\Delta w_{kl}' = \rho\, v_K'\, u_L'(1-u_L) \sum_{S=1}^{M} w_{LS}''\, \delta_S}$$

let $u_L'(1-u_L') \sum_{S=1}^{M} w_{LS}''\, y_S(1-y_S)(y_S-t_S) = \delta_L'$

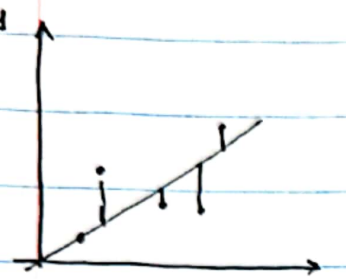
$$\boxed{\Delta w_{jK} = \rho\, x_j\, v_K'(1-v_K') \sum_{l=1}^{q} w_{KL}'\, \delta_L'}$$

let $v_K'(1-v_K') \sum_{L=1}^{q} w_{KL}'\, \delta_L' = \delta_K''$

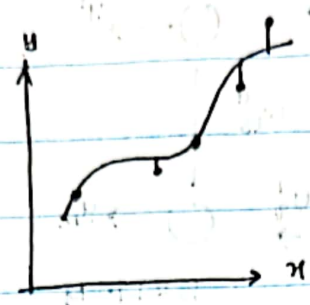
$$\underline{\Delta w_{ij} = \rho\, x_j\, \delta_K''}$$

↳ Overfitting may minimise the error data and may give error response for the new data.

Linear Regression.

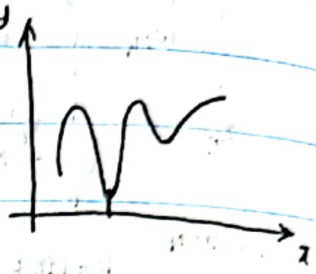


$y = w_0 + w_1 x$



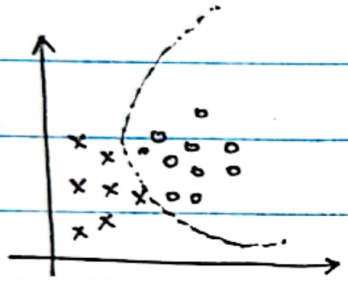
$y = w_0 + w_3 x^3 + w_2 x^2 \ldots$

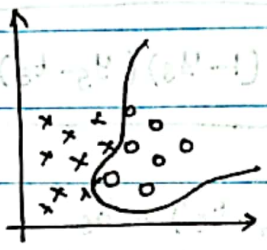
with higher polynomial.



over-fitting.

Logistic regression





(over fitting)

Regression Line :

$$\text{standard unit} = \frac{x - \text{Avg of } x}{\text{standard deviation}} \quad \left[\frac{x-\mu}{\delta}\right]$$

$$\boxed{r_{est} = r \cdot x_{given}}$$

in Original scale :

$$Y_{estimate} = r \cdot \frac{SD_y}{SD_x} \cdot X_{given} + Y_{average} - r \cdot \frac{SD_y}{SD_x} \cdot X_{average}$$

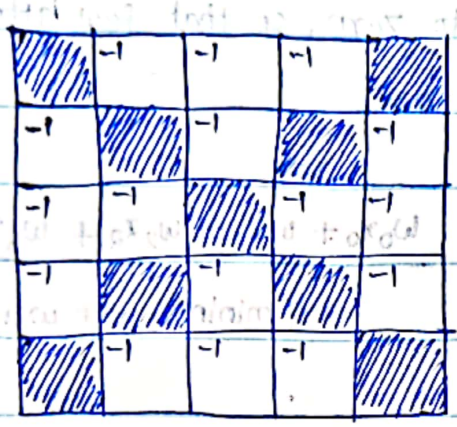
r : Co-relation

Slope of the regression line.

Intercept of the regression line

Convolution Network :

Consider a 5x5 image, with a pattern (let's say X)



let dark pixels value = 1
and other pixels = -1

consider 2 Filters.

1	-1
-1	1

-1	1
1	-1

use each Filter, traverse through the image (by each column) and do the elementary multiplication and update the values.

→ do this for both filters.

With Filter 01,

1	-0.5	0.5	0
-0.5	1	0	0.5
0.5	0	1	0
0	0.5	0	1

use max-pooling (Take max value in each 2x2 block)

1	0.5
0.5	1

FLAT

1
0.5
0.5
1
0.5
1
1
0.5

With Filter 02;

-1	0.5	-0.5	1
0.5	-1	1	-0.5
-05	1	-1	0.5
1	-0.5	0.5	-1

max-pooling.

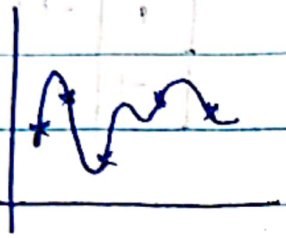
0.5	1
1	0.5

(max pooled pixels to Flat featured vectors).

→ In training perceptron models, with images, pixels acts as features and filters acts as weights.

⊙ To adjust the error response caused by over fitting of trained data, we would make higher polynomial weights to zero, so that few attributes are minimized/removed.

Example:



$w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3^2 + w_4x^4$

(minimize the weights.)

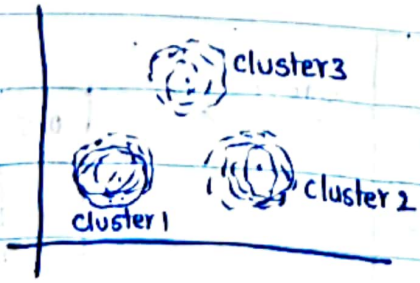
Text Classification: To identify/train a model to identify a statement of string to be True/False, we convert the words to vector.

Example: I like reading book in night.

like → high; reading → centre; book → high

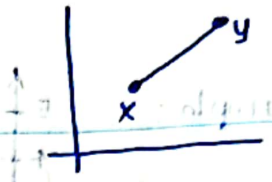
→ We focus mainly on middle word and its neighbouring words to be converted to feature vector.

Heirarchial Clustering: To group/cluster any data we measure the distance between data points to identify the cluster they belong to.



Distance Measurement :

Euclidean Distance : $\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$

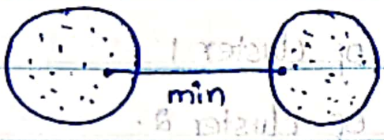


Squared Euclidean Distance $\sum_{i=1}^{n}(x_i - y_i)^2$

Manhattan Distance : $\sum_{i=1}^{n}|x_i - y_i|$

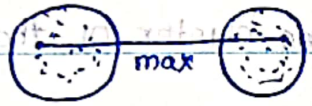
Measure Similarity between clusters :

Single Linkage :



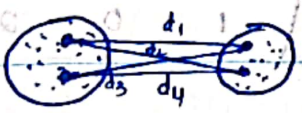
minimum / closest distance between points from 2 clusters.

Complete Linkage :



maximum distance between points from 2 clusters.

Group Linkage :



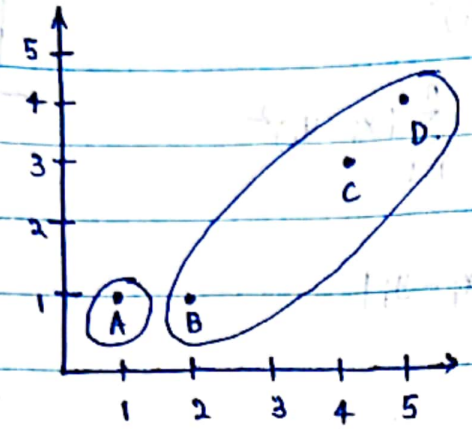
= average of the points

$$\frac{d_1 + d_2 + d_3 + d_4}{4}$$

Average Group Linkage :

Distance between centre data points of the clusters.

Example:



A (1,1)
B (2,1)
C (4,3)
D (5,4)

: Clustering of given data points A, B, C, D.

lets consider number of clusters = 2

C_1 = (1,1) centre of cluster 1

C_2 = (2,1) centre of cluster 2.

If distance of any point is minimum to any centre then that data point belongs to the corresponding cluster of that centre.

Iteration 0:

$$D^0 = \begin{matrix} \\ C_1 \\ C_2 \end{matrix} \begin{matrix} \begin{matrix} A & B & C & D \end{matrix} \\ \begin{pmatrix} 0 & 1 & 3.61 & 5 \\ 1 & 0 & 2.83 & 4.24 \end{pmatrix} \end{matrix}$$

Clustering

$$\begin{matrix} \\ C_1 \\ C_2 \end{matrix} \begin{matrix} \begin{matrix} A & B & C & D \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \end{matrix}$$

Find new centres after clustering:

New values of C_1 & C_2;

$C_1 = (1, 1)$

$C_2 = \left(\frac{2+4+5}{3}, \frac{1+3+4}{3} \right) = (11/3, 8/3)$

Iteration 01 :

$$D_1 = \begin{array}{c} \\ C_1 \\ C_2 \end{array} \begin{pmatrix} A & B & C & D \\ 0 & 1 & 3.61 & 5 \\ 3.14 & 2.36 & 0.41 & 1.89 \end{pmatrix}$$

Clustering :

$$\begin{array}{c} \\ C_1 \\ C_2 \end{array} \begin{bmatrix} A & B & C & D \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

New values for C_1 & C_2 ;

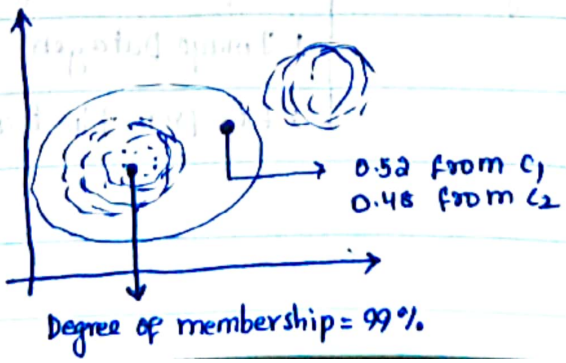
$$C_1 = \left(\frac{1+2}{2}, \frac{1+1}{2} \right) = (1.5, 1)$$

$$C_2 = \left(\frac{4+5}{2}, \frac{3+4}{2} \right) = (4.5, 3.5)$$

Repeat the iterations until C_1 and C_2, become consistant / doesn't change.

k-Means clustering : Here any given data point belongs to particular cluster (100%) as distance would be min to centre of that cluster than others.

Fuzzy clustering : if any given data point is at equal distances from centres of different clusters then any one of the cluster would be pull the data point based on degree of membership.



CATS & DOGS

Define Dataset ⟷ images, classes (KerasImage Datagenerator)

model sequential

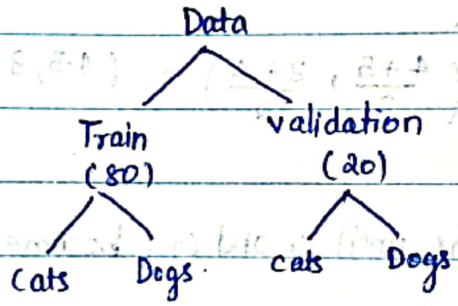
model. Compile

model. fit (Training data, Testing Data)

history

model. predict.

Divide the dataset (into folders) : [80-20]



→ image dataset can be diverse i.e: differ in background / angle of capture / transformation (Rotation, Zoom etc...) / Effects (B/w, etc...)

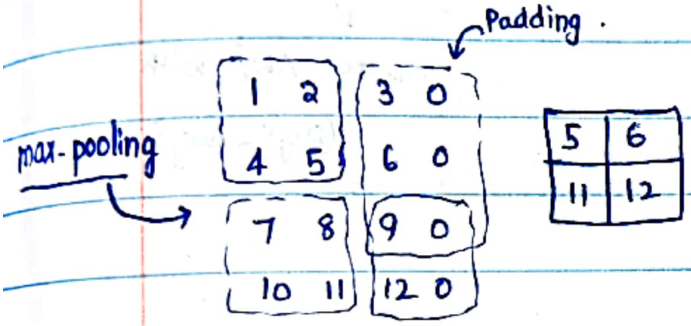
Split the data into different folders. [path/train/cats
path/train/Dogs]

KerasImage Datagenerator → Define the changes
→ Image Datagen
→ Flow from Directory

Training/model:

Sequential:

Conv2D layer: Conv2D (num of filters,
filter size,
activation function, input size)



Takes important data.

Flatten: Converts input to a 1D-array/vector.

Dense: Neuron Layer.

→ Then compile the model with relevant optimizer.

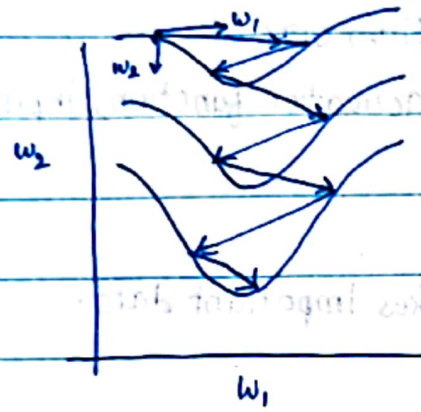
and then train the data [model.fit()]

→ if we have multiple convolution layers, maxpooling layer output of previous layer will be the input of next convolution layer.

k- Nearest Neighbourhood (Algorithm)

Returns the closest result to the sample.

Improve Gradient Descent: (using optimization Techniques).



$$w_j^{t+1} = w_j^t - \Delta w_j^t$$

replace Δw_j^t with $(\Delta w_j^t - \Delta w_j^{t-1})$

contour graph for gradient descent.

Momentum: $w^{t+1} = v^t + w^t$.

β : momentum co-efficient

$$v^t = \beta v^{t-1} - \alpha \left(\frac{\partial \varepsilon}{\partial w}\right) G^t.$$

$$v^3 = 3\left[3(-\alpha G^1) - \alpha G^2\right] - \alpha G^3$$

Root Mean Square propogation (RMSprop):

$$W^{t+1} = W^t - \frac{\beta}{\sqrt{v^t + \varepsilon}} \cdot G^t$$

ε is very small number which we include to avoid denominator being 0.

Ex: $\varepsilon \approx 0.00001$

$$v^t = \rho v^{t-1} + (1-\rho)(G^t)^2$$

For $v^0 = \phi$ (lets say).

$$v^1 = (1-\rho)(G^1)^2$$

$$v^2 = \rho(1-\rho)(G^1)^2 + (1-\rho)(G^2)^2$$

$$v^3 = \rho\left[\rho(1-\rho)(G^1)^2 + (1-\rho)(G^2)^2\right] + (1-\rho)(G^3)^2$$

Depending on each slope (during descent), for each weights change will be dynamic.

Adaptive Moment optimization [ADAM]

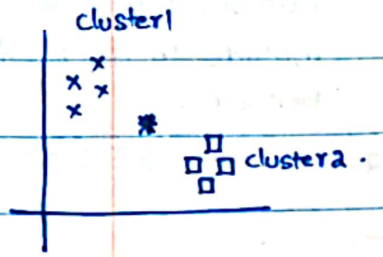
ADAM combines both momentum and RMSprop.

$$w^{t+1} = w^t - \beta \cdot \frac{s^t}{\sqrt{v^t + \varepsilon}} \cdot G^t$$

where $v^t = \rho v^{t-1} - (1-\rho)(G^t)^2$

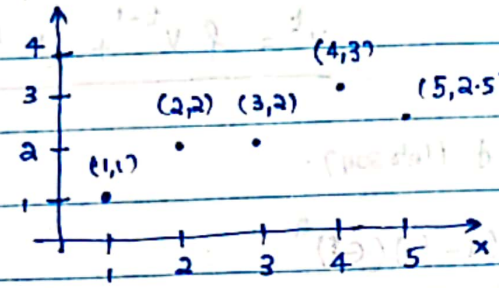
$$s^t = \rho s^{t-1} - (1-\rho) G^t$$

Principal Component Analysis (PCA): Technique used to project higher dimensional data to lower dimension.



Consider the average point, and shift origin to that point.

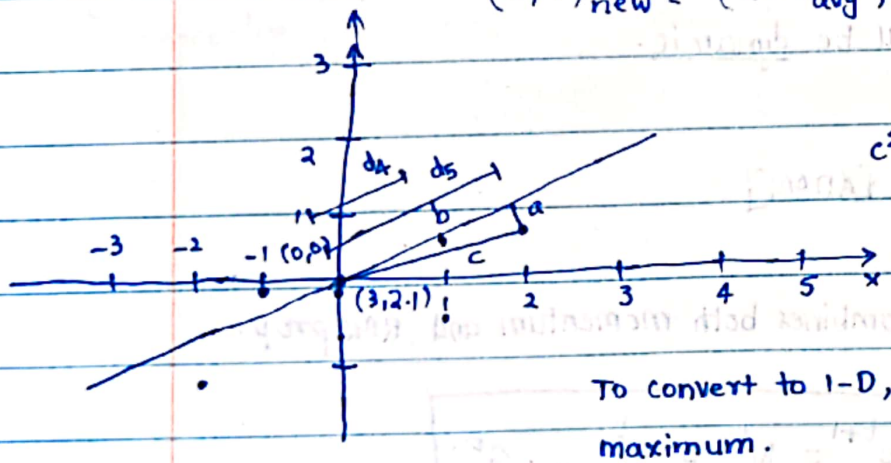
Ex:



$x_{avg} = \frac{1+2+3+4+5}{5} = 3$; $y_{avg} = \frac{1+2+2+3+2.5}{5} = 2.1$

shift origin to $(x_{avg}, y_{avg}) = (3, 2.1)$

$(x, y)_{new} = (x - x_{avg}, y - y_{avg})$



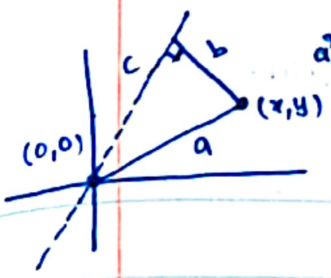
$c^2 = a^2 + b^2$

To convert to 1-D, we make $(d_1^2 + d_2^2 + d_3^2 + d_4^2 + d_5^2)$ maximum.

(SS)

we draw random line, through origin and find if the line fits the data or not.

$\{d_1, d_2, \ldots, d_5\}$ is the distance between origin to the points projected on to the random line.



$a^2 = b^2 + c^2$

in PCA, (x,y) point projected on to the line.

if b gets bigger, c gets smaller.

(or) if b gets smaller, c gets bigger.

So, PCA can minimize the distance to the line (or) maximize the distance of the projected point to the origin.

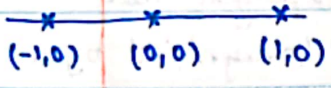
Consider the line, which has maximum value of SS.

we call this line principal component 1 (PC1)

and PC2 will be 1^{lr} to PC1.

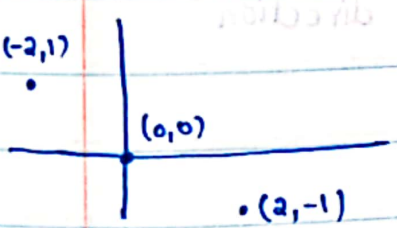
Calculate Co-variance Matrix based on PC1 & PC2.

Ex:



variance : $\frac{(-1)^2 + (0)^2 + (1)^2}{3} = 2/3$.

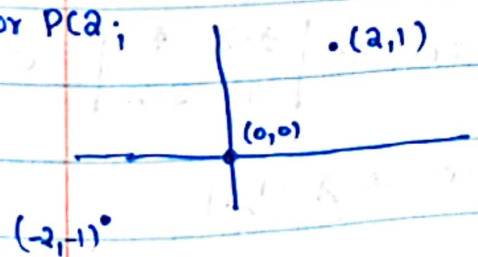
① (-2,1)



$x_{var} = \frac{(-2)^2 + (0)^2 + (2)^2}{3} = 8/3$

$y_{var} = \frac{(1)^2 + (0)^2 + (-1)^2}{3} = 2/3$.

For PC2;



$x_{var} = 8/3$

$y_{var} = 2/3$

variances are same for both PC1 & PC2.

Co-variance : Take average of product of x & y values.

① $\frac{(-2 \times 1) + (0 \times 0) + (2 \times -1)}{3} = -4/3$ | ② $\frac{(2 \times 1) + (0 \times 0) + (-2 \times -1)}{3} = 4/3$

Co-variance matrix gives us the direction.

Matrix transformation: Takes the given data to new dimension.

$$\text{Cov Mat} = \begin{bmatrix} \text{var}(x) & \text{cov}(x,y) \\ \text{cov}(x,y) & \text{var}(y) \end{bmatrix}$$

Ex: $\text{Cov Mat} = \begin{bmatrix} 9 & 4 \\ 4 & 3 \end{bmatrix}$

Transformation: $(x \quad y)\begin{bmatrix} 9 & 4 \\ 4 & 3 \end{bmatrix} = (9x+4y,\ 4x+3y)$

$(1,0) \Rightarrow (9,4)$

$(0,1) \Rightarrow (4,3)$

$(2,1) \Rightarrow (22,11)$: $(2,1)$ multiplied by (11) → Eigen value.
(magnified by 11 times)

Eigen vectors: Vectors that doesn't change direction.

Eigen vector (v): $Av = \lambda v$

λ: Eigen value.

$$(A-\lambda I)v = 0.$$

$$\begin{pmatrix} 9-\lambda & 4 \\ 4 & 3-\lambda \end{pmatrix} v = 0 \Rightarrow \begin{vmatrix} 9-\lambda & 4 \\ 4 & 3-\lambda \end{vmatrix} = 0.$$

$$\Rightarrow (9-\lambda)(3-\lambda) - 16 = 0 \Rightarrow \lambda = 1, 11$$

Solve the equations for v.

$$\begin{pmatrix} 9 & 4 \\ 4 & 3 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = 11\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \qquad \begin{pmatrix} 9 & 4 \\ 4 & 3 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = 1\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

$$\begin{pmatrix} 9v_1+4v_2 \\ 4v_1+3v_2 \end{pmatrix} = \begin{pmatrix} 11v_1 \\ 11v_2 \end{pmatrix} \qquad \begin{pmatrix} 9v_1+4v_2 \\ 4v_1+3v_2 \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

For $\lambda = 11$, $V = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$

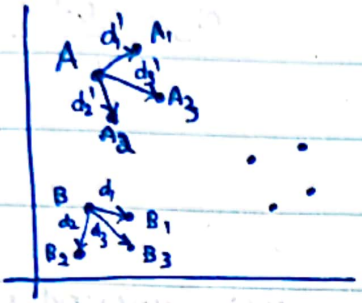
For $\lambda = 1$, $V = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$

T-SNE (T-Distrubution Stochastic Neighbour Embedding)

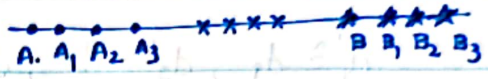
↳ Technique to project higher dimension data to lower dimension.

Ex: projecting 2D data into 1D.

Actual Data.



Target projection

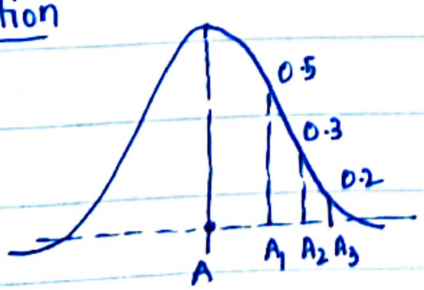


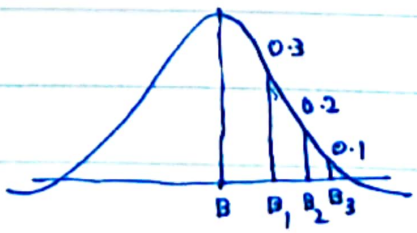
Stochastic distrubution of the given data.
(random distrubution)

1D data of stochastic distrubution tries to change as points from same group moves closer and points from different group move away from each other. This process happens millions of times to attain the 'Target projection'.

To move the points around within the cluster in specific order we consider simillarity among points to be same between groups i.e, Simillarity from A to A_1 should be same as B to B_1.

t-Distrubution





Similarity

$$d_1' = \frac{0.5}{0.5+0.3+0.2} = 0.5$$

For cluster 2:

$$d_1 = \frac{0.3}{0.3+0.2+0.1} = 0.5$$

$$d_2' = \frac{0.3}{1} = 0.3$$

$$d_2 = \frac{0.2}{0.6} = 0.3$$

$$d_3' = \frac{0.1}{1} = 0.1$$

$$d_3 = \frac{0.1}{0.6} = 0.16$$

$d_1' \cong d_1$, $d_2' \cong d_2$, $d_3 \cong d_3$.

$\therefore$ points within different groups move around with same force because of having equal similarities.

Regularization: To overcome the problem of overfitting, we use Regularization.

i. More Data

ii. Data Augmentation

iii. Early stop.

m : # of samples

n: # of features

$h_w(x) : w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$

$$\text{Error / cost } J(w) = \frac{1}{2m} \sum_{i=1}^{m} (h_w(x^i) - t^i)^2$$

in order to minimise the weights of few features (to zero), we introduce regularization factor (λ).

updated cost function

$$J(w) = \frac{1}{2m}\left[\sum_{i=1}^{m} (h_w(x^i) - t^i)^2 + \lambda \sum_{j=1}^{m} w_j^2\right]$$

$$w_j = w_j - \frac{\alpha}{m}\left[\sum_{i=1}^{m} (h_w(x^i) - t^i)\, x_j^i + 2\lambda w_j\right]$$

$$w_j = \left(1 - \frac{2\alpha\lambda}{m}\right) w_j - \frac{\alpha}{m}\left[\sum_{i=1}^{m} (h_w(x^i) - t^i)\, x_j^i\right]$$

$\left(1 - \frac{2\alpha\lambda}{m}\right)$ also called as decay-rate.

in this approach, instead of taking w_j, we take fraction of w_j which helps us to minimise w_j faster.