Perceptron Loss Function

* Need of Loss Function Over Perceptron Trick :-

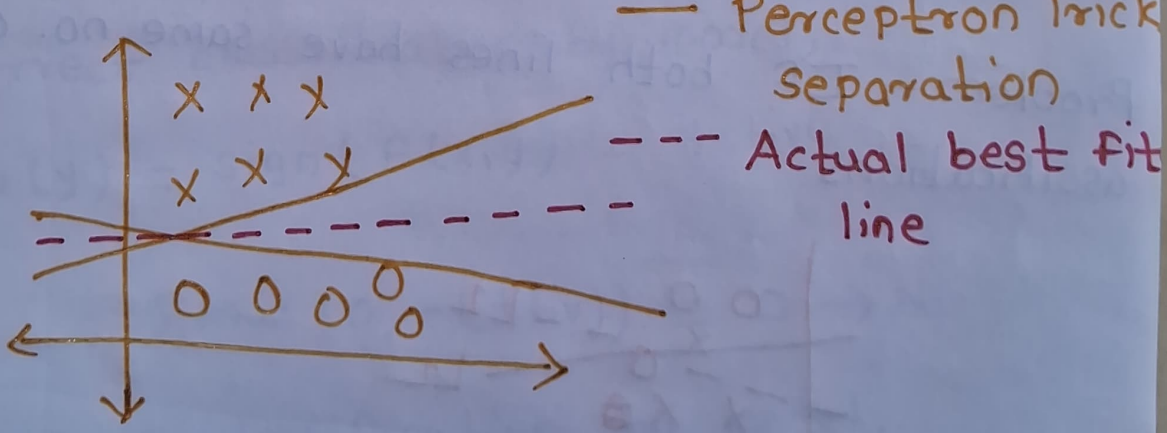
① Perceptron Trick – A heuristic method to update weights.

It stops as soon as it separates the points.

It cannot distinguish between "best separtion" and "barely good" separation.

In rare cases it fails to converge because of selection of the random point repeatively.

It does not provide any proof (i.e number) how the bad is model performing.



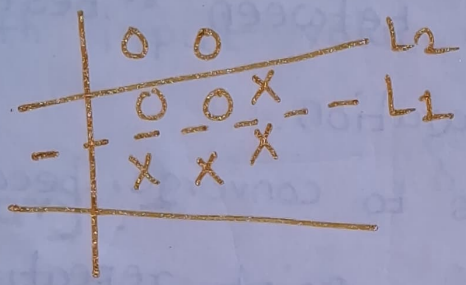
② Loss Function – A mathematical function based on weights (ω) and bias (b)

It give single representing number that tells how bad our model is performing.

The goal is to find such values of w & b that minimizes this number.

* Designing of perceptron Loss Function :-

Approch (1) – Count the misclassified points by the line and choose the line having minimum misclassified points.

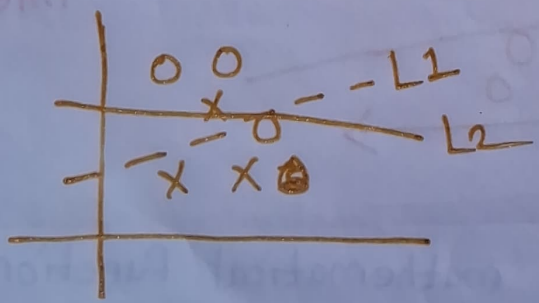


So, here L1 is better than L2

because,

misclassification (L1) < misclassification (L2)

Problem – If both lines have same no. of misclassifications.



Here,

misclassification (L1) = misclassification (L2)

Approch ② - Calculate the distances of miscla-ssified points from line.

So,
less magnitude (missclassification) → better line.

Problem - Too expensive (by time)

Approch ③ - Hinge Loss

$$L = \sum \max(0, -y_i \cdot f(x_i))$$

where,
y_i = actual value (-1 or 1)
$f(x_i)$ = predicted value or score $(w \cdot x + b)$

So,
If correct classification by model,

$$\text{sign}(y) = \text{sign}(f(x_i)) = +\text{tive}$$

$$\& \quad L = \sum \max(0, -(+\text{tive})) = 0 \rightarrow \text{No Loss.}$$

& If wrong classification by model,

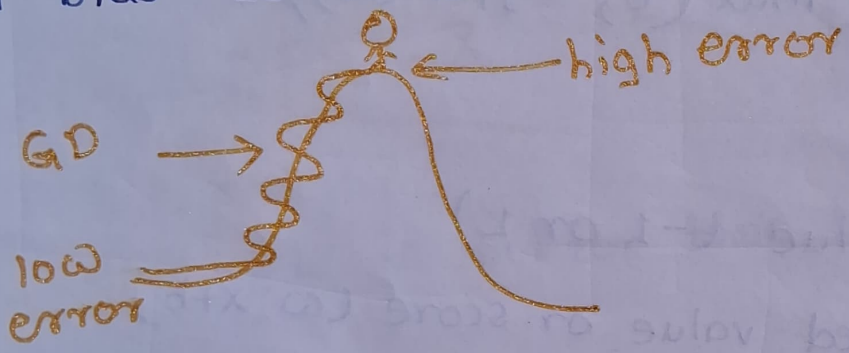
$$\text{sign}(y) \neq \text{sign}(f(x_i)) = -\text{tive}$$

$$L = \sum \text{Max}(0, -(-\text{tive})) = \text{some +tive loss}$$

* Gradient Descent to minimize loss :-

* (1) Gradient Descent – Gradient Descent is the iterative algorithm used to minimize the loss function.

Imagine standing on a hill and taking steps downward to reach valley (low error), we calculate derivative of loss function w.r.t weights and bias to know which direction is to step.



(2) steps –

(i) Initialize random weights

(ii) Loop through data points

(iii) calculate $\frac{\partial L}{\partial w}$

(iv) Adjust weights using learning rate η

$$W_{new} = W_{old} + \eta(\cancel{y_i x_i})\frac{\partial L}{\partial w}$$

if correct pred. then $\frac{\partial L}{\partial w} = 0 \therefore W_{new} = W_{old}$

if wrong pred then $W_{new} = W_{old} + \eta\left(\frac{\partial L}{\partial w}\right)$