MACHINE LEARNING
BSE: 5th SEMESTER
(After Mid)

6

17

3

5/12/24

⊙ Neural Network:

- Node : $z = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2$
- Layers : output, Hidden, output
  $x \in R^D$ D nodes in Input layer.
- Classification / Regression
- If NN is being used for C-class classification
  - $C = 2$ | output layer contains single node (Binary class classification).
  - $C > 2$ | " " " C nodes (multi-class classification).

⊙ Activation Functions: (Transformation)

(Hidden layers) (output layer) (binary)

1- Sigmoid Activation Function (0-1)

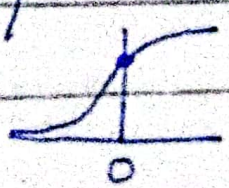
$$a = g(z) = \frac{1}{1+e^{-z}}$$

$z \in R$

$$g'(z) = \frac{d}{dz}(g(z)) = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$g'(z) = g(z)(1 - g(z))$$

$$= a(1-a)$$



① $z = 100$, $g(z) = 1$

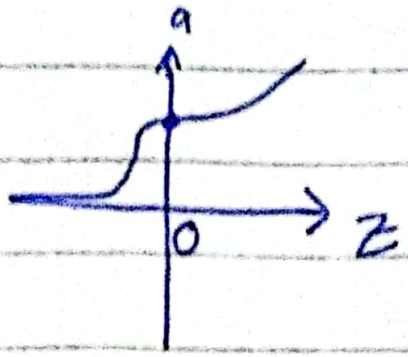
$$g(z) = \frac{1}{1 + e^{-100}} = \frac{1}{1} = 1 \quad , g'(100) = 0$$

→ $z = -100$

$$g(z) = \frac{1}{1 + e^{100}} = \frac{1}{\infty} = 0 \quad , g'(0) = 0$$

→ $z = 0$, $g(0) = 0.5$, $g'(0.5) = 0.5 \times (1 - 0.5)$

$= 0.25$



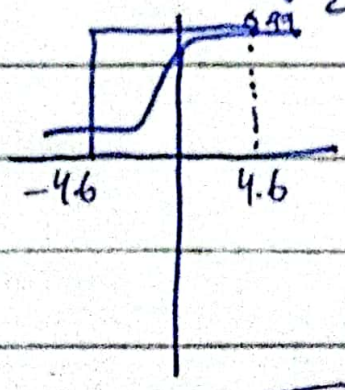
$$\text{Slope} = g'(z) = 0.25 = \frac{\text{change in } y}{\text{Changeins}} = \frac{\Delta y}{\Delta x} = \frac{\text{rise}}{\text{run}}$$

: slope decreases as we go left or right.

→ $g'(0.6) = 0.6 \times (1 - 0.6)$

$= 0.6 \times 0.4$

$= 0.24$



$$\theta_j = \theta_j - \boxed{\alpha \frac{\partial J(\theta)}{\partial \theta_j}}$$

→ Limitations:

: Limited range (0-1)

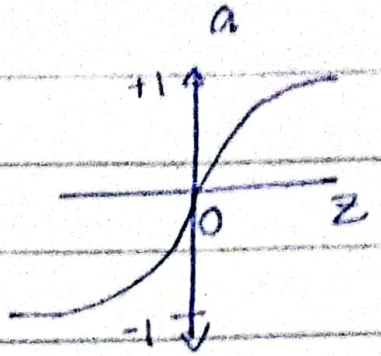
: Very Less Value (0.000099)

(New Activation Function)

② Tanh Activation Function:
(Hidden Layer)

(-1 ---- +1)
Range

$$a = g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



: $\tanh(0) = 0$

if $z = 100$, $g(100) = +1$

if $z = -100$, $g(-100) = -1$

if $z = 0$, $g(0) = 0$

$$g'(z) = \frac{d}{dz}\left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right) = \boxed{1 - (\tanh(z))^2}$$

$$= \boxed{1 - (g(z))^2}$$

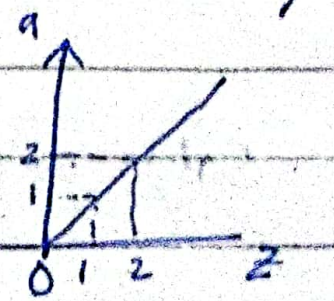
$$g'z = \boxed{1 - a^2}$$

→ One Problem solved of Range (-1, 1)

① Rectified Linear Unit (RELU):

(Formula) (Activation Function)
$$a = g(z) = RELU(z) = \max(0, z)$$

(Derivative)
$$g'(z) = \begin{cases} \text{if } z < 0 \text{ then } g'(z) = 0 \\ \text{if } z \geq 0 \text{ then } g'(z) = 1 \end{cases}$$



: Slope $= \frac{\Delta y}{\Delta x}$

Slope = 1

: 50% positive value

: 50% negative value

: Output → Sigmoid
(Binary classification)

: RELU (By default, it is used / common used)

⊙ Leaky RELU:

(multiplying by 1% of number)

→ $a = g(z) = \text{Leaky RELU}(z) = \max(0.01 \times z, z)$

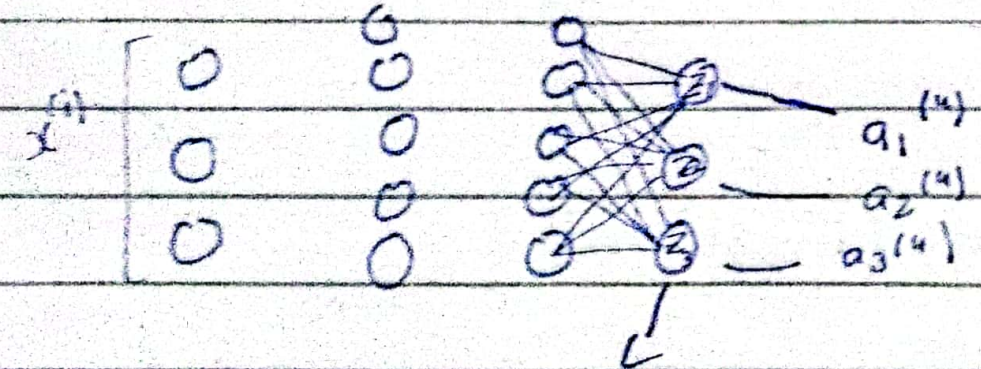
-0.05 (max), -5

→ $g'(z) = \begin{cases} \text{if } z < 0 & 0.01 \\ \text{if } z \geq 0 & 1 \end{cases}$ $\quad 0.01 \times z = 0.01$, $\quad 1 \times z = 1$

⊙ Activation Fuctions for output Layer:

1- Binary classification : Sigmoid

2- Multiclass classification: Softmax (Takes all z)



: Logit (Final value of layer which not passed to softmax)

→ $z_1^{(4)} = \theta_{10}^{(3)} a_0^{(3)} + \theta_{11}^{(3)} a_1^{(3)} + \theta_{12}^{(3)} a_2^{(3)} + \theta_{13}^{(3)} a_3^{(3)} + \theta_{14}^{(3)} a_4^{(3)}$

⊙ $\underset{4}{\text{Softmax}}\left(z_1^{(4)} + z_2^{(4)} + z_3^{(4)}\right)$

$t_1 = e^{z_1^{(4)}}$

$t_2 = e^{z_2^{(4)}}$

$t_3 = e^{z_3^{(4)}}$

: $t_1 = 10$
$t_2 = 20$
$t_3 = 30$

: Softmax (world wide use)

→ $a_1^{(4)} = t_1 / (t_1+t_2+t_3)$ $a_1^{(4)} = 0.17$

$a_2^{(4)} = t_2 / (t_1+t_2+t_3)$ $a_2^{(4)} = 0.33$

$a_3^{(4)} = t_3 / (t_1+t_2+t_3)$ $a_3^{(4)} = 0.5$

return $a_1^{(4)}, a_2^{(4)}, a_3^{(4)}$

]

: softmax gives probabilities

: Hardmax will return only one.

: $a_3^{(4)} > a_2^{(4)} > a_1^{(4)}$
(max)

$x^{(1)} = a_3^{(4)}$

④ Assignment (Raw Data → cleanse → Interface ← Deploy ← Evaluate ← Test → Evaluate → Cloud Upload.

12/12/24

→ Neural Networks Multiclass Classification:

⊙ For Multiclass Classification we use Softmax

⊙ Data/Structure (set) ⊙ Architectures (More class more dataset) ⊙ Changing

⊙ Cost Function ⊙ Activation Function (softmax)

⊙ $y^{(i)}$ rectors. To minimize errors

⊙ Cost Function:

→ Logistic regression:

: K-term new added.

⊙ Backward Propagation: Optimization (Error Computation)

Forward Propagation: Result / Activation computation

⊙ Partial derivatives are important for finding optimization of dataset.

(For Training it is important)

⊙ z(Logit) computed as back iteration and multiplying it with Θ_1.

: nodes = Number of z equal

rows = Depend on number of nodes.

⊙ **Error Term:**

$\delta_j^{(l)}$ = error of node j in layer l

∴ For output layer, finding error will be easy,

$\delta_j^{(4)} = a_j^{(4)} - y_j$ (True value)

: In node error there is participation of every node.

$a_5^{(3)}$ = Being multiplied by weights.

(*) When subscript is given, it is single value.

(*) $a^{(3)}$ is computed by $a^{(4)}$ and $\delta^{(4)}$ is computed by $\delta^{(3)}$ (Inverse for Delta)

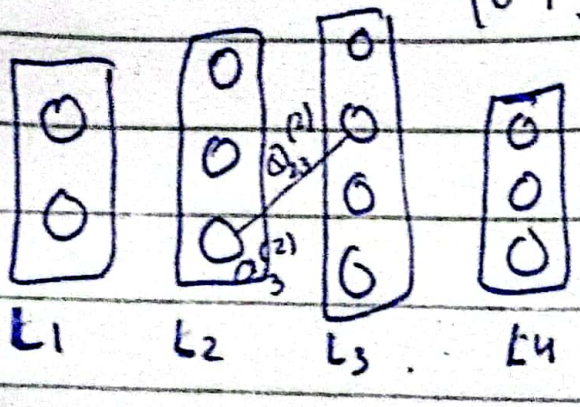
$$\delta^{(3)} = \theta^{(3)T} \delta^{(4)} \cdot g(z^{(3)})$$

$$g'(z^{(3)}) = g'(z^{(3)})(1 - g(z)^{(3)})$$

$$= a^{3}_{6x1}(1 - a^{3}_{6x1})$$

$$= * a_{6x1}\left(\begin{bmatrix}1\\1\\1\end{bmatrix} - a^{3}_{6x1}\right)$$

(*)



$$\frac{\partial J(\theta)}{\partial \theta^{(2)}_{23}} = a^{(2)}_3 \delta^{(3)}_2$$

$$\delta^{(3)}_2 = ?$$

(Big Delta) $\Delta^{(2)}_{23} = a^{(2)}_3 \delta^{(3)}_2$

$$\theta^{(2)}_{23} = \theta^{(2)}_{23} - \alpha \Delta^{(2)}_{23}$$

⊙ Steps

(i) Forward Propagation
(ii) Backward Propagation
(iii) Compute Partial Derivatives of each weight
(iv) Updates all the weight

⊛ Feature Extraction (Python Libraries Functions)

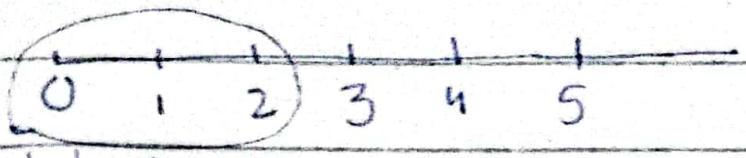
⊛ Main Important Tasks:

(i) Wearable Sensors Data Reading

(ii) Other will be Feature Extraction (Feature Engineering)

⊛ 55 Class classification property. (Behavioral Pattern) (Standing to Sit)

⊛ Transition location:



First Half

: 4 Sec (Duration)

: JINS (2 Sensor Glasses)

: Microsoft (2 Sensor band)

: Nexus (5 Sensor)

⊙ **Phone:**

→ Phone Accelerometer (200/s) (Speed)

→ Gyroscope (Rotation) (200)

→ Magnetometer (Orientation) (50)

→ Gravity (Towards Earth) (200)

→ Linear Accelerometer (200)

⊙ **MS. Watch:**

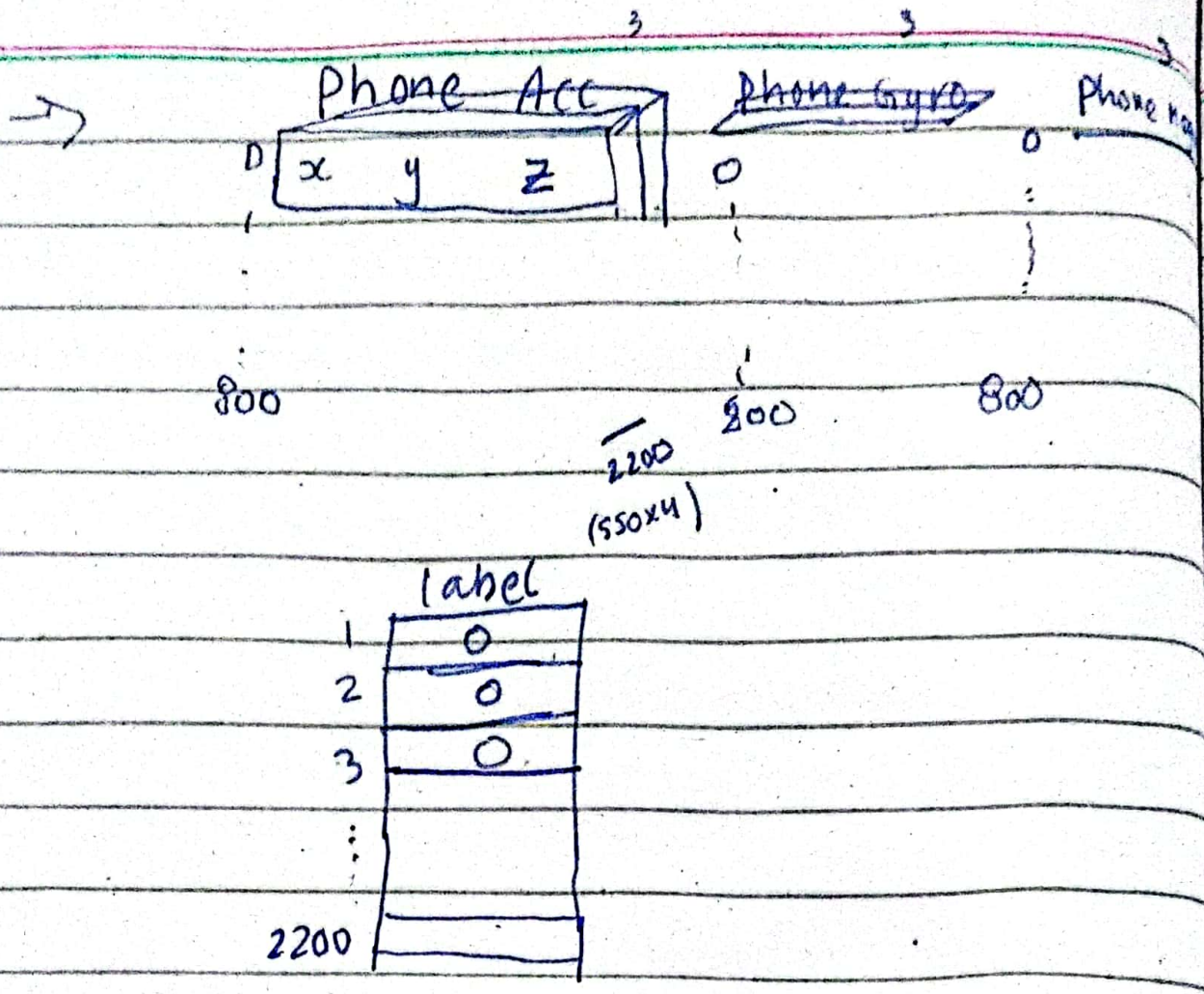
→ MS Accelerometer (67)

→ MS Gyroscope (67)

⊙ **JINS:**

→ JINS Acc (20)

→ JINS Gyro (20)



④ Feature Extraction:

: compress data

Features - phone Acc

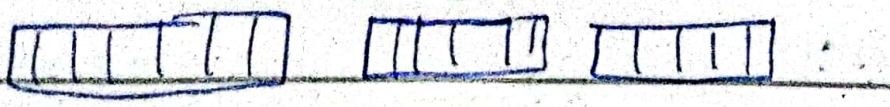
x	y	z
Max	Max	Max
Min	Min	Min
Aver	Aver	Aver
Std	Std	Std

800 compress to 12

: 12 Matrix

: 12x9 = 108

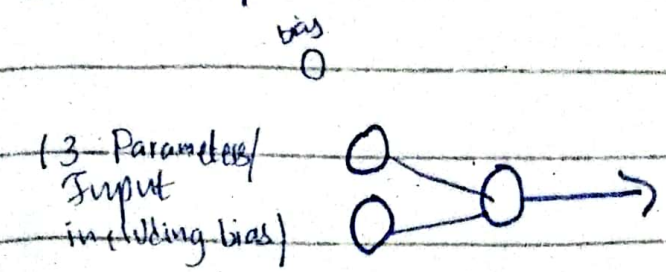
: For 54x9 = 486 Features



→ All classification except last requires Matrix (SVM, LR, DT.....)

19/12/24

⊛ Simple Neural Network:



→ Code → .pynb → Jupiyter

→ pytorch, numpy, matplotlib

→ Generate Data.

① Cluster (Add or subtract value to make data appear around value).

→ This and self in coding is same.

→ Input Layer is itself activation.

→ ① Forward → ② Compute Loss → ③ Error → Gradient ④ → Add Gradient ⑤ in before weight

epochs: 800

→ torch.nn, torch.optim

→ Numpy Arrays into Torch Arrays. (Tensors)

→ Fully connected Neural Network

① Bias Node is always computed by program itself.

② Then apply nn.sigmoid.

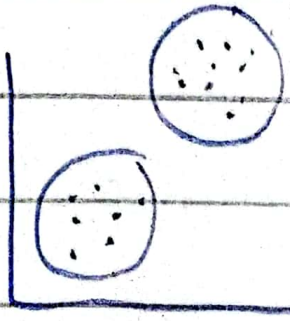
Using Pytorch

→ model → optimize → Loss Function

31/12/24

⊙ Clustering:

→ Close to classification but class association is not provided.
→ Unsupervised machine learning.
→ Labelling not provided we identify from its property.
→ Cluster central point centroid.



Cluster = K = 2

→ Testing for data or new data is done by compairing with mev.

⊙ Clustering K-means algorithm:

→ Two steps:

⊙ Cluster Assignment (i) Randomly generate two points (Red and Blue)
(ii) check for other points distance from red and blue and make cluster.

(ii) Centroid movement step
(Take point average of two clusters)
and then move
(Till the old and new value
centroid difference is less (threshold))

→ Input for K-Means:

⊙ Weights ⊙ Number of clusters.

→ Eucleadian Distance:

⊙ By square ⊙ By absolute.

→ Distance/cost will be distance of points from its centroids.

→ Number of clusters should be less than number of examples.

→ ci (cluster of index)
μk (centroid Position).

→ Distortion (For optimization).

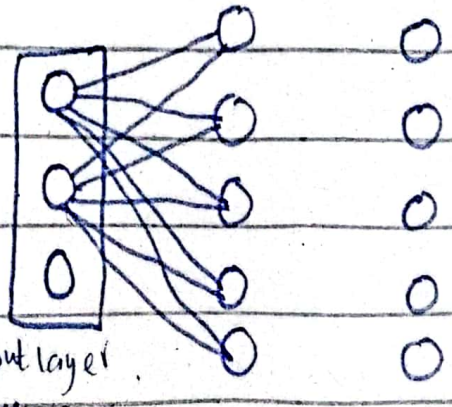
→ Random Initialization
(Local optima)

→ Data Generate → Plot → Solution of Two clusters

⊛ Clustering is unsupervised machine learning is used in grouping. (which helps us to group the number of examples)

⊙ Deep-Learning Algorithm:

→ Neural Network



(Fully Connected Tensor Flow | Linear)

⊙ Auto-Encoder:

→ Unsupervised Machine Learning Neural Network

→ Forward, Loss, Backward, Weight update, Gradient.

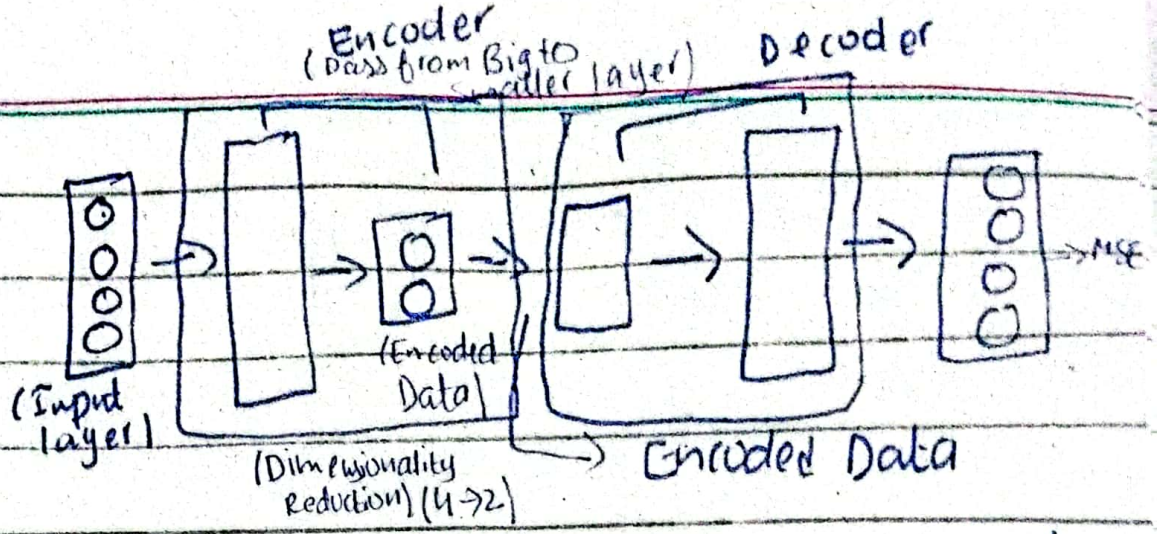
→ Feed Forward NN: (Get result by one time forward propagation) (Not recurrent NN).

⊙ Fully Connected NN

⊙ Formulation NN

Convutional NN

Recurrent ⊙ GRM ⊙ LSSM ⊙ RNN



→ Auto-Coder is encoder or decoder Encodes encodes inputs into smaller number of nodes dimensions

① (Nodes at input layer is equal to nodes at output layer).

② Hidden Layer are dense.

→ Decoder decodes smaller dimension into actual number of dimension.

→ Auto Encoder do Clustering and Dimensionality Reduction.

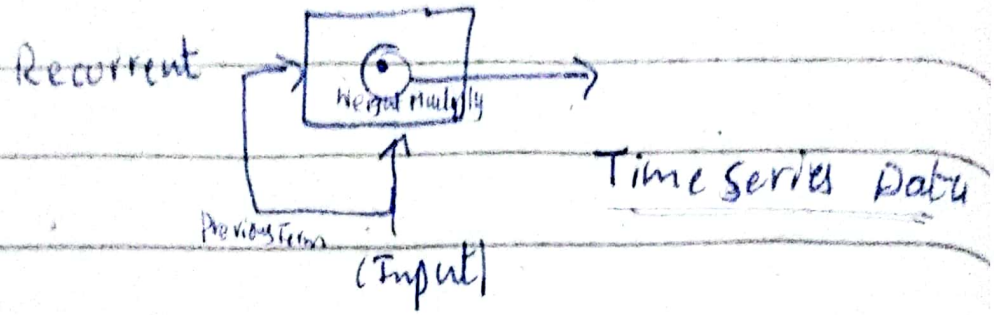
→ Encoded Data gives us reduction.

(*) For check of encoded input and output layer, we use MSE:

$$\frac{1}{m} \sum_{i=1}^{m} (\text{Actual Data} - \text{Predicted Data})^2$$

(*) Gradient Descent is used for convex function used for global minimum and maximum.

(*) We called error detecting phase ~~is called~~ "converging".



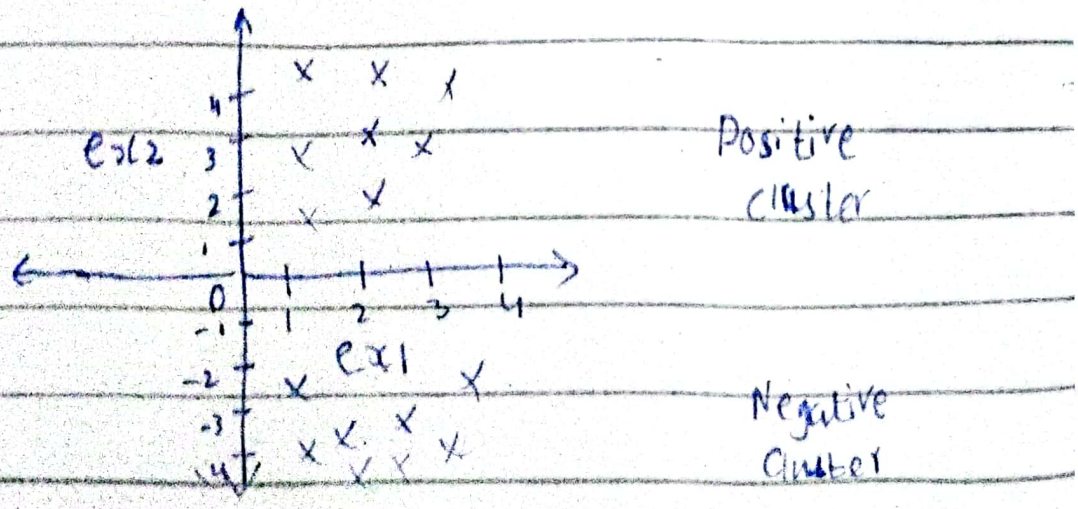
-> FE :

(i) FCNN (ii) CNN

-> Recurrent:

(i) RNN, (ii) LSSM (iii) GRU

(*) 4D into 2D:



-> Training -> Back Propagation