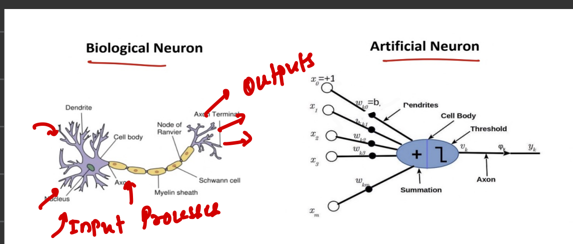
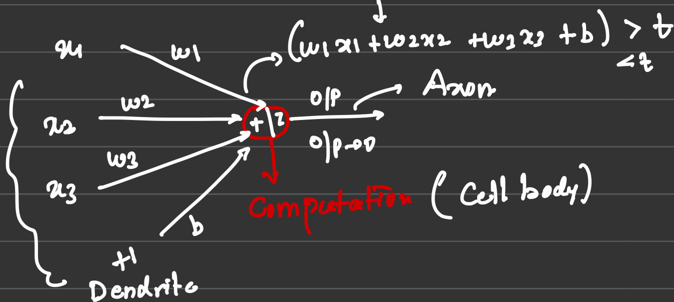
Perceptron

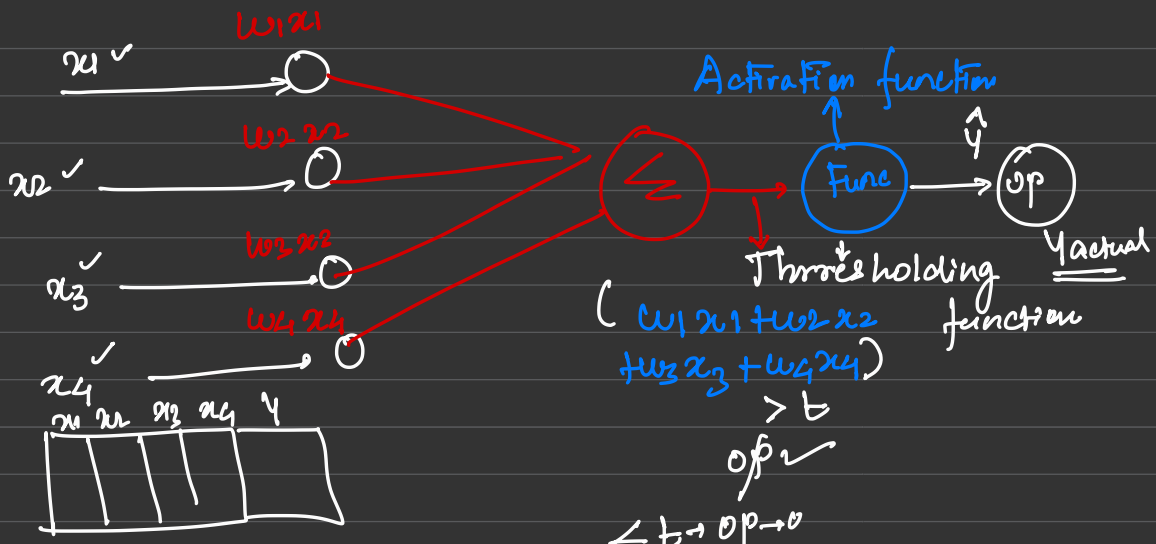
Biological Neuron

Artificial Neuron

owner rich
Businesses
estimate







$$loss = f(y, \hat{y})$$

We need to find the best set of weights which bring our $\hat{y}$ closer to y

↳ find the best set of weights which minimise the loss

Weight update Rule for a neuron

$$w_{new} = w_{old} - \alpha \times \partial h / \partial o$$

When our loss = mse

$$w_{new} = w_{old} + \alpha\,(target - predicted) \times input\ value$$

$$w_{new} = w_{old} + \alpha\,(y - \hat{y}) \times x$$

AND GATE

Objective: learn a perceptron weights such that it can function like AND Gate

A	B	O/P	$\hat{y}$
0	0	0	0
1	0	0	0
0	1	0	0
1	1	1	1

$w_1, w_2 = 0.6$, 1.2

$(w_1 A + w_2 B) > 1$

Case 1) $A \to 0, B \to 0$

$1.2 \times 0 + 0.6 \times 0 \to 0 < 1$

$\to 0$

Case 2) $A \to 1, B \to 0$

$\hat{y}$ $1.2 \times 1 + 0.6 \times 0 = 1.2 > 1$

$\to 1$

$y_{actual} \to 0$.

Case 3) $A \to 0, B \to 1$

$1.2 \times 0 + 1 \times 0.6$

$= 0.6 < 1$

$= 0$ ✓

Case 4) $A \to 1, B \to 1$

$1.2 \times 1 + 0.6 \times 1 = 1.8 > 1$

1

$w_{1new} = 1.2 + 0.5 (0-1) \times 1 \times A$

$= 1.2 - 0.5$

$= 0.7$

$w_{2new} = 0.6 + 0.5 (0-1) \times 0 \times B$

$= 0.6$

New weights

$w_{1new} = 0.7$

$w_{2new} = 0.6$

$(w_1 \times A + w_2 \times B) > 1$

Case 1) $A \to 0, B \to 0$

$0.7 \times 0 + 0.6 \times 0 = 0$ ✓

Case (2) $A \to 1, B \to 0$

$0.7 \times 1 + 0.6 \times 0 = 0.7 < 1$

$= 0$ ✓

Case (3) $A \to 0, B \to 1$

$0.7 \times 0 + 0.6 \times 1 = 0.6 < 1$

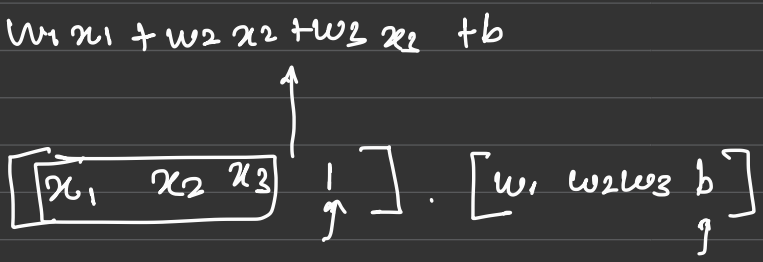
$= 0$ ✓

Case 4) $A \to 1, B \to 1$

$0.7 \times 1 + 0.6 \times 1$

$= 1.3 > 1$

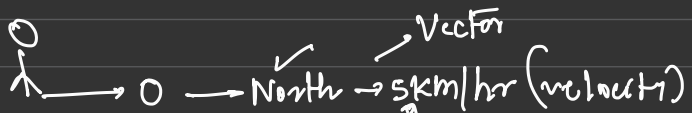
$\to 1$ ✓

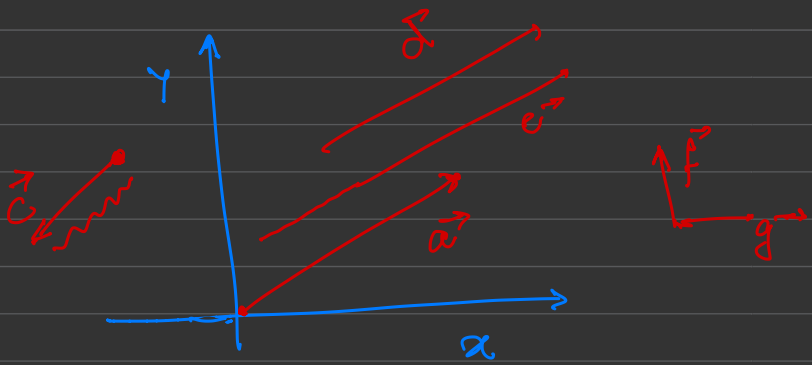

$w_1 x_1 + w_2 x_2 + w_3 x_3 + b$
x_1 x_2 x_3 1
w_1 w_2 w_3 b

Concept of Mathematics

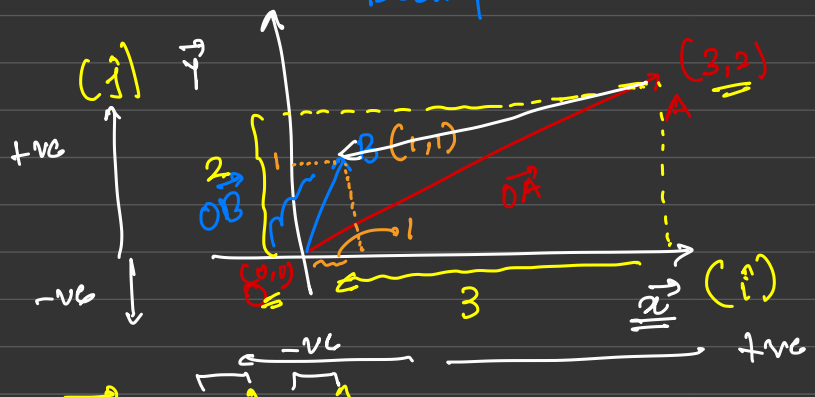
1. Vectors
2. Differentiation
3. Partial Differentiation
4. Gradient of a functions
5. maxima & minima of a function

Vector: it is an object which has a magnitude and a direction





Decomposed into its projections



$\vec{OA} = 3i + 2j$

$\vec{OB} = 1i + 1j$

$\vec{AB} = \vec{OB} - \vec{OA}$

$\vec{AB} = 1i + 1j - (3i + 2j)$

$\vec{AB} = -2i - 1j$

$|\vec{AB}| = \sqrt{(2)^2 + (-1)^2}$

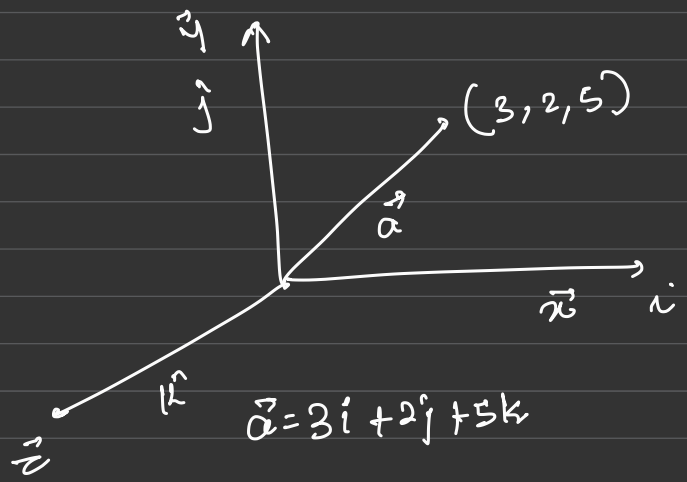
$|\vec{AB}| = \sqrt{5}$

$\vec{x} = a\hat{i} + b\hat{j}$

$|\vec{x}| = \sqrt{a^2 + b^2}$

$\vec{x} = a\hat{i} + b\hat{j} + c\hat{k}$

$|\vec{x}| = \sqrt{a^2 + b^2 + c^2}$

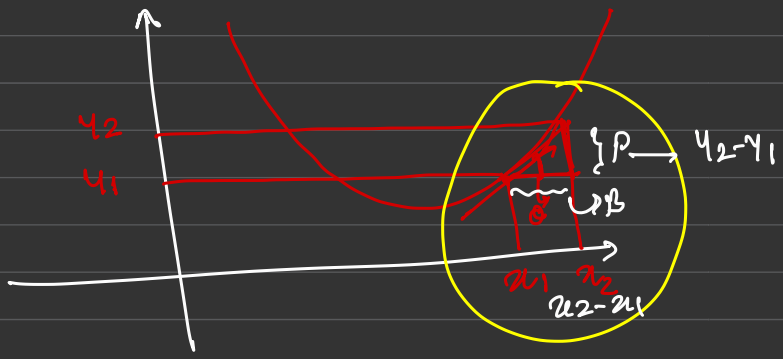


$\vec{a} = 3i + 2j + 5k$

Differentiation

$F(x)$

$\frac{dF(x)}{dx}$ → rate of change of the function w.r.t to the variable



$\tan\theta = P/B$

$\tan\theta = (y_2 - y_1)/(x_2 - x_1)$

$dx \to 0$ $\quad \frac{dF(x)}{dx}$ $\qquad$ $\Delta x \to 0$ $\quad \frac{\Delta y}{\Delta x} = dy/dx$

$$\frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

$$\lim_{h \to 0} \frac{f(x_1 + h) - f(x_1)}{x_1 + h - x_1}$$

Rules of Differentiation

① Power Rule.

$$f(x) = x^n$$

$$f'(x) = dy/dx = n x^{n-1}$$

$$f(x) = 5x^5$$

$$f'(x) = 5 \times \frac{d(x^5)}{dx}$$

$$= 5 \times 5 \times x^4$$

$$= 25x^4$$

② Product Rule

$$\frac{d(\sin x)}{dx} = \cos x$$

$$\frac{d\cos(x)}{dx} = -\sin x$$

$$h(x) = f(x) \times g(x)$$

$$h'(x) = f(x) \times g'(x) + f'(x) \times g(x)$$

$$h(x) = (x^4 + 2)\cos x$$

$$h'(x) = -(x^4+2)\sin x + \cos x\,(4x^3)$$

$$h'(x) = -\sin x\, x^4 - 2\sin x + 4x^3 \cos x$$

Partial Differentiation

$$f(x,y) = x^4 y$$

$\frac{\partial f(x,y)}{\partial x}$ / $\frac{\partial f(x,y)}{\partial y}$ → differentiate the function w.r.t y keeping x constant

↓

differentiate the function w.r.t x, keeping y constant

$$\frac{\partial (x^4 y)}{\partial y} = x^4$$

$$\frac{\partial (x^4 y)}{\partial x} = y \times 4x^3$$

Gradient

$f(x,y) \rightarrow \frac{\partial f(x,y)}{\partial y}$

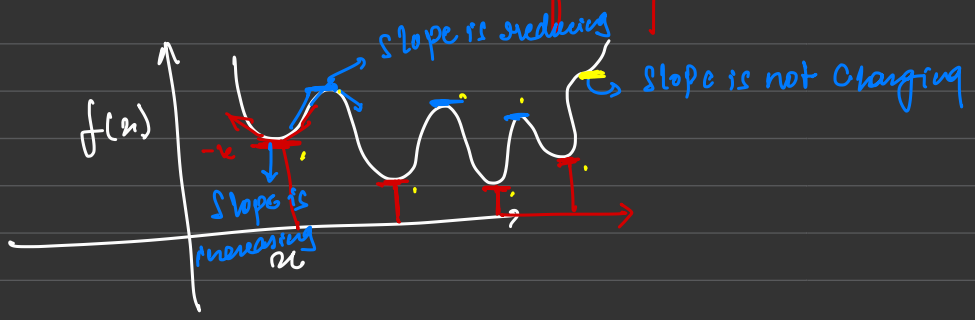
$\rightarrow \frac{\partial f(x,y)}{\partial x}$

$f(x,y) = 2x^2 + 4y$

$\nabla f = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{matrix} 4x \\ 4 \end{matrix}$

$\nabla f = \begin{bmatrix} 4x \\ 4 \end{bmatrix}$

Maxima & minima of a function



$f'(x) = 0$

Critical Points

- Points at which the slope of the function

→ i.e the slope of the tangent at that point is zero.

Red Points → minima

$f''(x) > 0$ Red

$\frac{d f'(x)}{dx}$

$f''(x) < 0$ → maxima
↳ Blue

$f''(x) = 0$ → inflection points

Calculation of the maxima & minima of a Bivariate Function.

$f(x,y)$ — local maxima and local minima points

$p = \frac{\partial f(x,y)}{\partial x}$ — Partial differentiation

$q = \frac{\partial f(x,y)}{\partial y}$

Solve for $p=0$ and $q=0$
Critical points.

$r = \frac{\partial^2 f(x,y)}{\partial^2 x}$

$s = \frac{\partial^2 f(x,y)}{\partial (xy)}$

$t = \frac{\partial^2 f(x,y)}{\partial y^2}$

for all the critical points (a,b)

if $rt - s^2 > 0$ and

$r > 0$ (a,b) is a local minima

$r < 0$ (a,b) is a local maxima

$rt - s^2 = 0$ test fails ✗

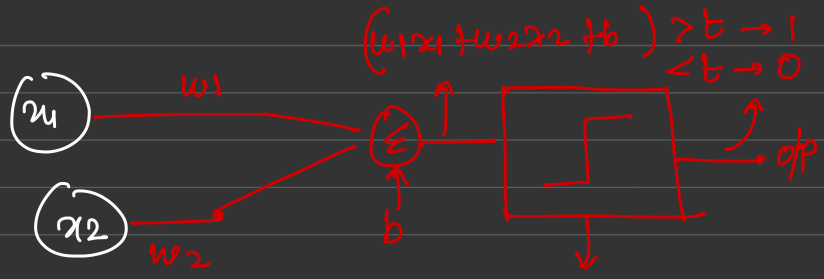
$rt - s^2 < 0$ it's a saddle point

Activation Functions

XOR gate

A	B	o/p
0	0	0
1	0	1
0	1	1
1	1	0

Perceptron couldn't train the logic on the XOR



Activation function ⟶ thresholding function

Cannot capture non-linear patterns

Activation Functions:

1. Sigmoid
2. Relu (Rectified Linear Unit)
3. Leaky Relu
4. tanh (hyperbolic tangent)
5. Softmax
6. Parameteric Relu

Sigmoid activation: Binary classification Problem

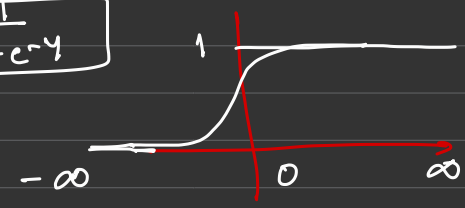
Squishing function

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

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

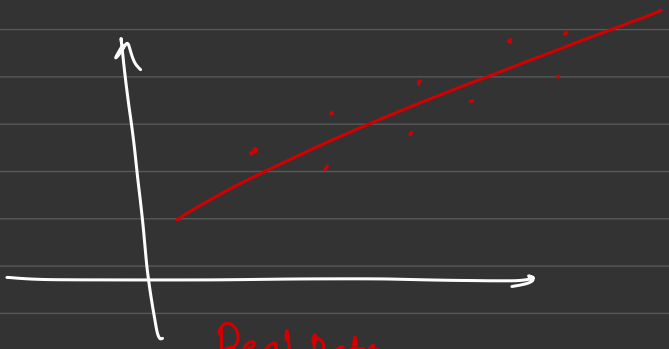
$$\sigma(y) = \frac{1}{1+e^{-y}}$$

0 - 1

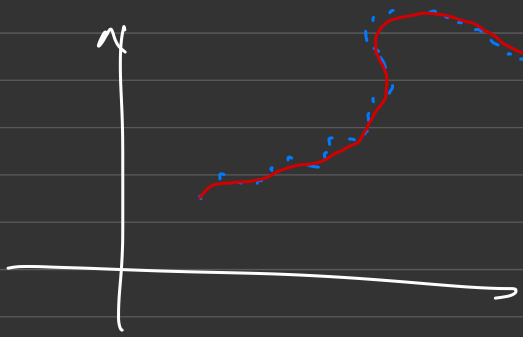


$-\infty \leq x \leq \infty$

$0 < \sigma(x) < 1$



Real Data



derivative of the sigmoid

$w_{new} = w_{old} - \alpha \, dJ/dw$

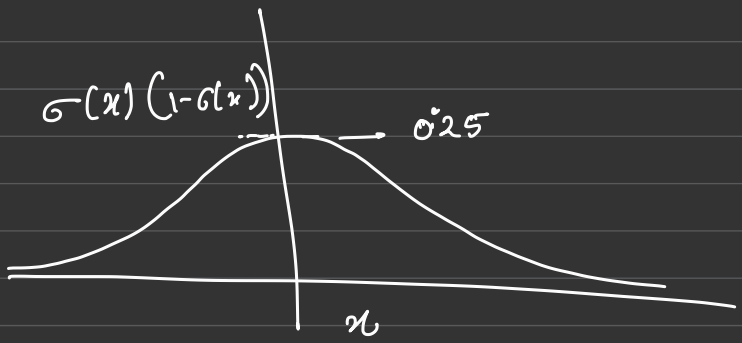
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

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

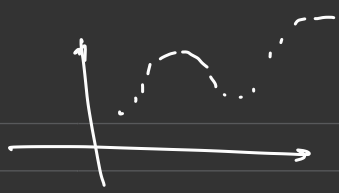
$$\sigma'(x) = (1+e^{-x})^{-2} \times (0 + e^{-x}) \times (-1)$$

$$\sigma'(x) = -(1+e^{-x})^{-2} \times e^{-x}$$

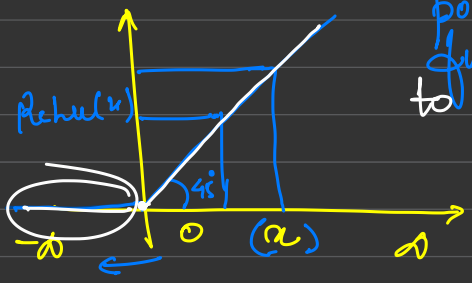
$$\sigma'(x) = \sigma(x) \times (1 - \sigma(x))$$



Relu → Rectified linear unit



$Relu(x) = max(x, 0)$ { $x \in (-\infty, \infty)$, $Relu \in (0, \infty)$



popularly used as an activation function in multi layer perceptrons to bring in non-linearity.

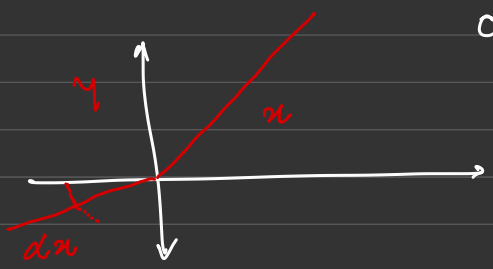
Parametric Relus:

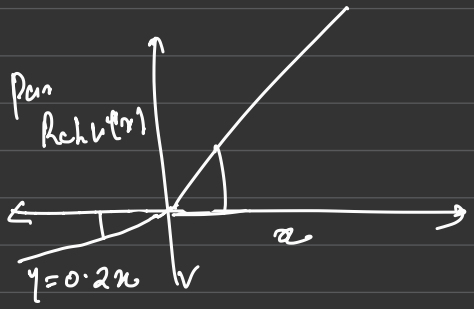
Relu $(x) \rightarrow \begin{cases} x & x \geq 0 \\ 0 & x = 0 \\ \alpha x & x < 0 \end{cases}$

$0 < \alpha < 1$

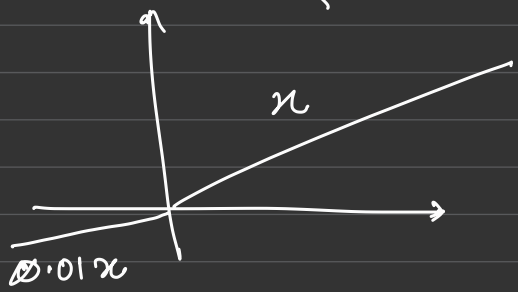
↓

learnt by the model.

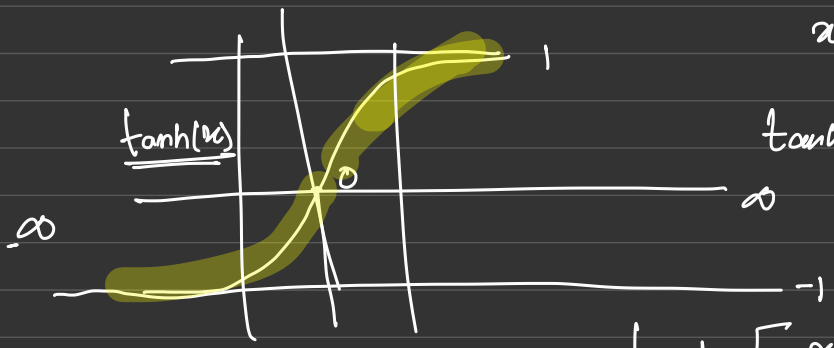




Leaky Relu(x) $\begin{cases} x & x > 0 \\ 0 & x = 0 \\ 0.01x & x < 0 \end{cases}$ → Leaky Relu



Hyperbolic Tangent (Tanh)

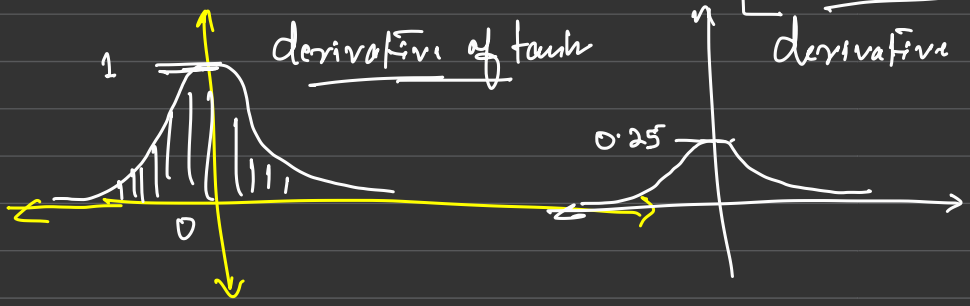


$x \in (-\infty, \infty)$

$\tanh(x) \in (-1, 1)$

$$\tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\tanh'(x) = 1 - \tanh^2(x)$$

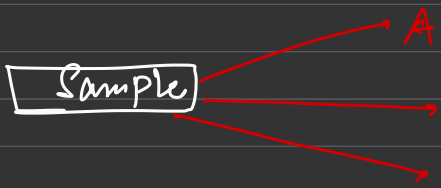


tanh is an activation function which is popularly used in the NN for creating non-linear features

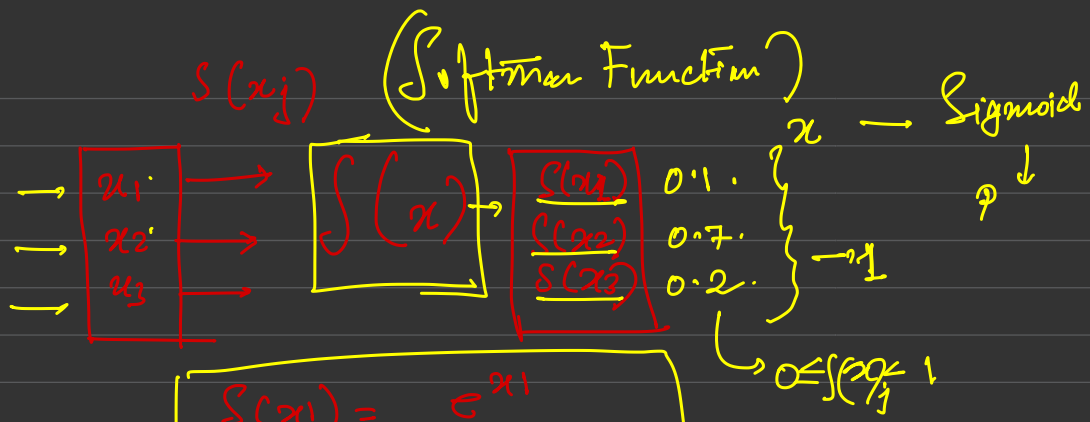
Softmax activation function

Special case of activation function

Multiclass classification problem



Softmax function
basically a more generalized form of sigmoid

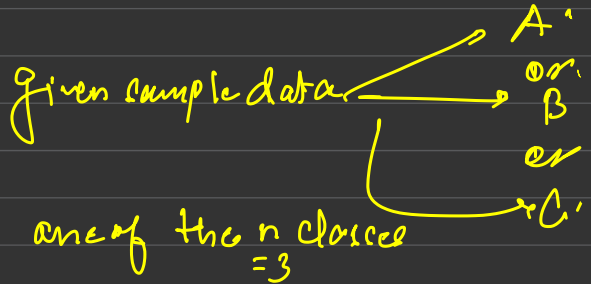


$$S(x_1) = \frac{e^{x_1}}{e^{x_1} + e^{x_2} + e^{x_3}}$$

$$S(x_2) = \frac{e^{x_2}}{e^{x_1} + e^{x_2} + e^{x_3}}$$

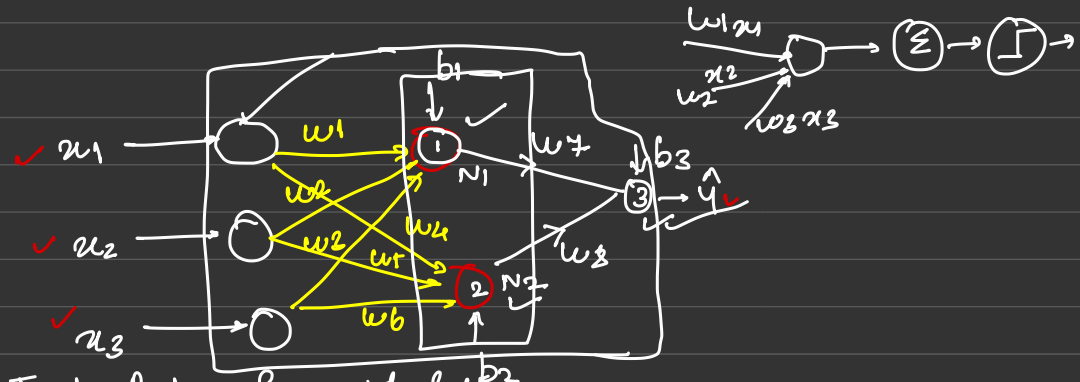
$$S(x_3) = \frac{e^{x_3}}{e^{x_1} + e^{x_2} + e^{x_3}}$$

$$S(x_j) = \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}$$



Output in a multiclass classification problem

Forward and Backward propagation



Tanh, Relu, Sigmoid, Softmax

$$N_1 = (w_1x_1 + w_2x_2 + w_3x_3 + b_1)$$

$$N_2 = (w_4x_1 + w_5x_2 + w_6x_3 + b_2)$$

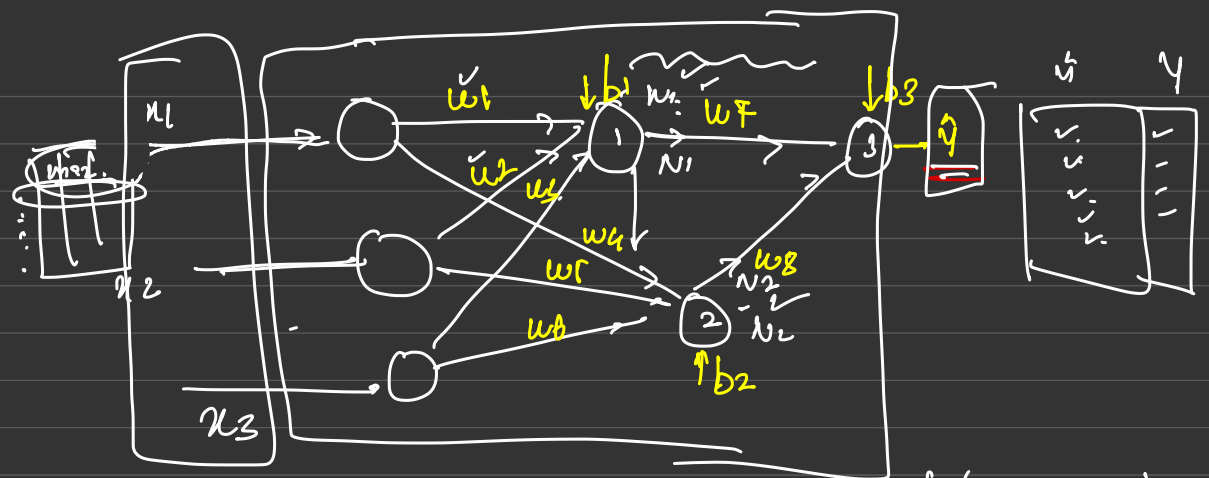
$$\hat{y} = (w_7N_1 + w_8N_2 + b_3)$$

x_1	x_2	x_3	y

$y = f(x_1, x_2, x_3)$

$\hat{y} \leftarrow$ MLP (x_1, x_2, x_3)

f will achieve (my $\hat{y}$ becomes closer to y) by manipulating the weights and biases



loss of my neural network (MLP)

$$\text{loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Objective: To manipulate my weights and biases to reduce the loss.

1) We are going to initialise my weights and biases randomly.

Forward Propagation

2) Based on the weights and biases initialised the network will calculate $\hat{y}$.

$N_1 = (w_1x_1 + w_2x_2 + w_3x_3 + b_1)$

$N_2 = (w_4x_1 + w_5x_2 + w_6x_3 + b_2)$

$\hat{y} = (w_7N_1 + w_8N_2 + b_3)$

3) We will calculate the loss of the network

$$\sum_{i=1}^{n}\frac{1}{n}(y_i - \hat{y}_i)^2$$

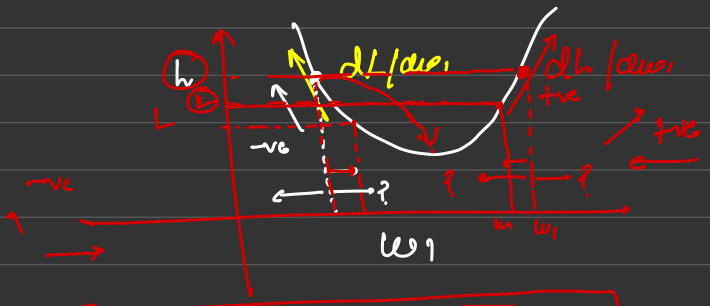
Backward Propagation:

① To minimize the loss by manipulating the weights and biases

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_1 x_{1i} + w_2 x_{2i} + b_3))^2$$

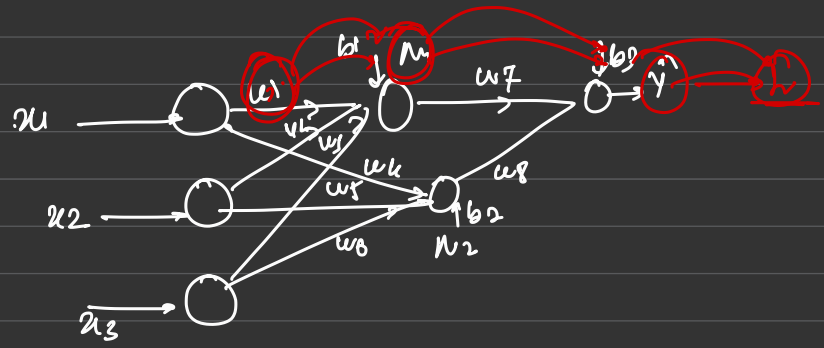
Gradient Descent:



$$w_1(t+1) = w_1(t) - \alpha \, dL/dw_1(w_1)$$

$\frac{dL}{dw_1}, \frac{dL}{dw_2}, \frac{dL}{dw_3} \cdots \frac{dL}{dw_p}$

$\frac{dL}{db_1}, \frac{dL}{db_2} \cdots \frac{dL}{db_p}$



$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_7 N_{1i} + w_8 N_{2i} + b_3))^2$$

$$\frac{dL}{dw_7} = \frac{1}{n}\sum_{i=1}^{n} -2\,(y_i - (w_7 N_{1i} + w_8 N_{2i} + b_3))\,N_{1i}$$

$\frac{dL}{dw_8}$

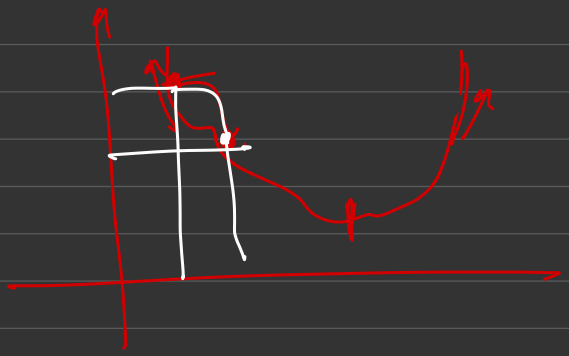
$\frac{dL}{dw_1}$

$\frac{dL}{dw_1}$ = Chain Rule

$$\frac{dL}{dw_1} = \frac{dL}{dN_1} \times \frac{dN_1}{dw_1}$$

$\frac{dL}{dw_2} = \frac{dL}{dN_1} \times \frac{dN_2}{dw_2}$

$N_1 = (w_1 x_1 + w_2 x_2 + w_3 x_3 + b_1)$

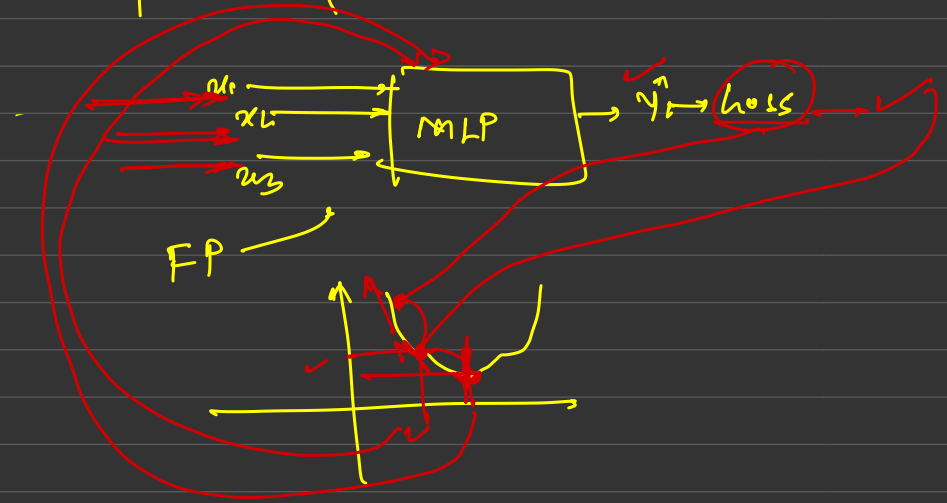


We initialise the weights and biases randomly.

given these weights and biases we calculate the loss by propagating through network → Forward Propagation

given the loss we calculate the derivative wrt to all the weights and biases and do the gradient descent to get new set of weights and biases

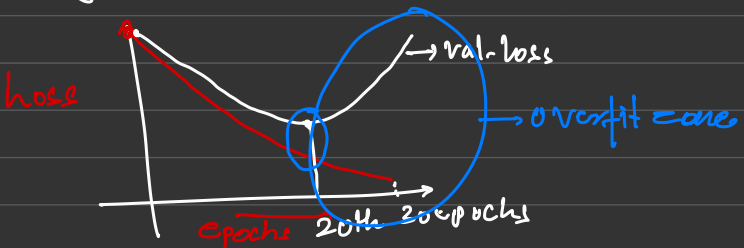
↓ Backward Propagation



Callbacks

1. Early stopping callback
2. model checkpoint callback
3. Tensorboard callback

Early Stopping Callback

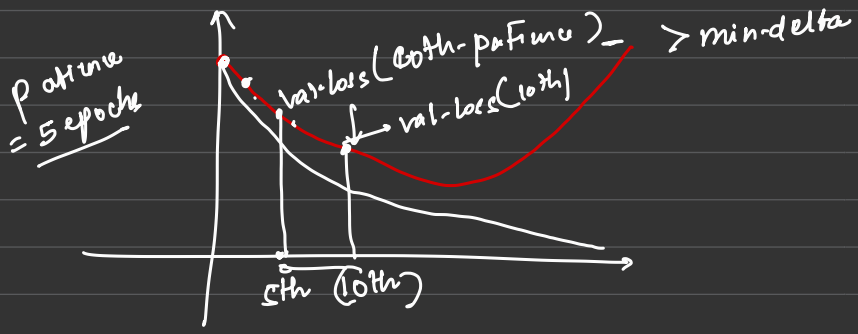


model.fit(epochs= epochs)

Early Stopping Callback

function that can monitor the improvement on a metric can stop the training if it does not improve beyond a certain threshold

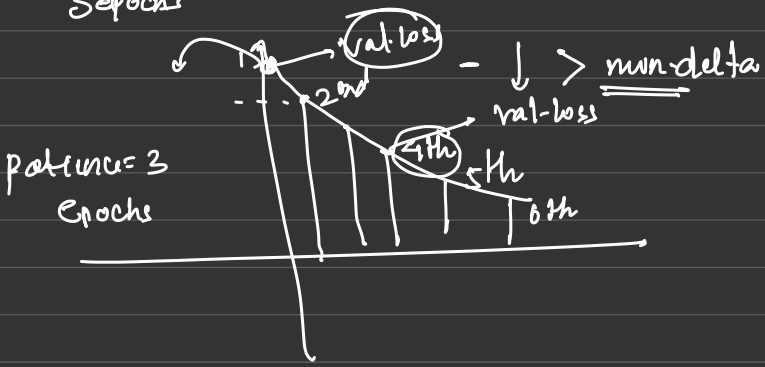
Early Stopping → min_delta
Patience



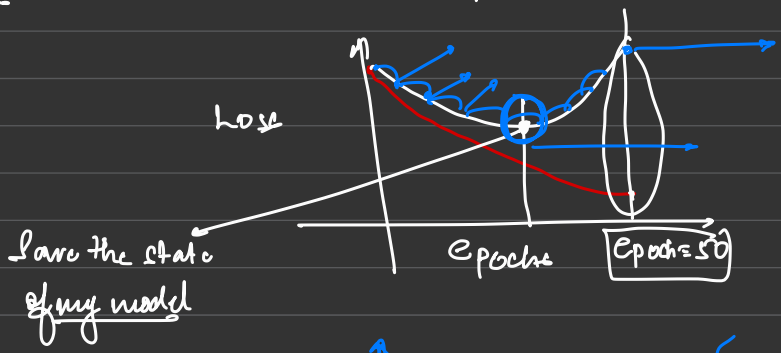
monitor my val-loss

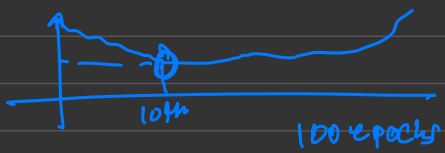
(patience, min-delta = 0.01)

5 epochs



(model Checkpoint call back.)





Early Stopping and model checkpoint → not to monitor to epochs

Tensorboard call back

Loss Functions

Regression losses

- Mean Squared error
- Mean Absolute Error
- Huber loss
- Pseudo Huber loss

Classification losses

- Binary Cross Entropy
- Categorical Cross entropy
- Sparse categorical cross entropy
- Hinge loss

Mean Squared error loss.

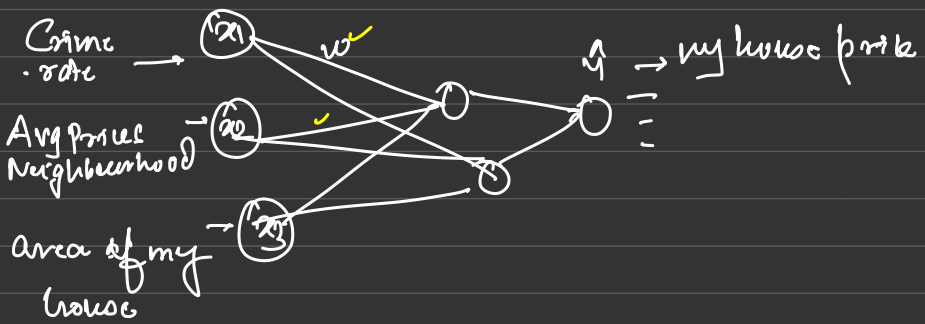
$$\sum_{i=1}^{n} \frac{1}{n} (y_i - \hat{y}_i)^2$$

Cost vs loss

x_1	x_2	x_3	y	$\hat{y}$
✓	✓	✓	✓	2

$(y-\hat{y})^2$

Cost function



	x_1	x_2	x_3	y	$\hat{y}$
✓	✓	✓	✗	90	70
✓	✓	✗	✓	100	120
✗	✓	✓	✓	110	100

$(y-\hat{y})^2$

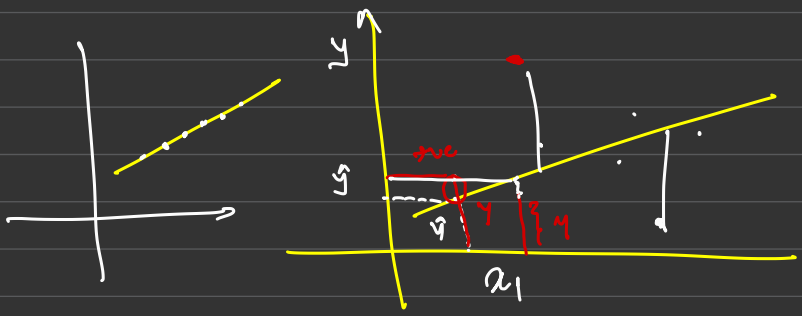
400

400

100

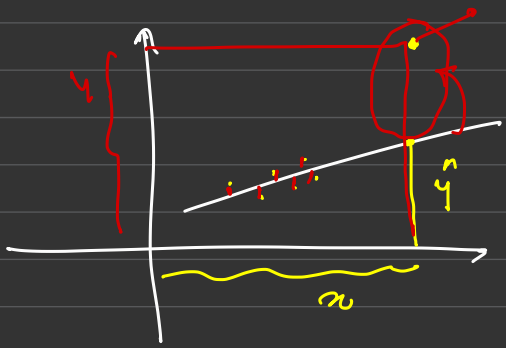
$loss = \sum_{i=1}^{n} \frac{1}{n} (y-\hat{y})^2$

$\frac{900}{3} = 300$



$(y - \hat{y})^2$

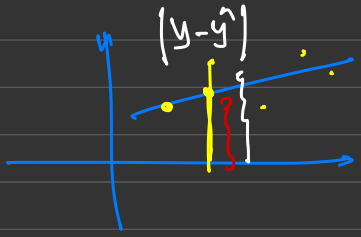
$(y - \hat{y})$



MSE is susceptible to outliers

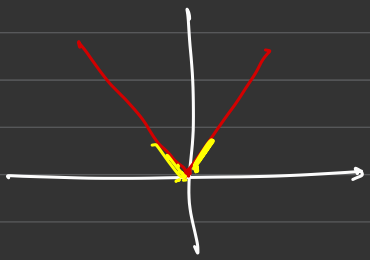
Mean Absolute error

$$\sum_{i=1}^{n} \frac{1}{n} (y_i - \hat{y}_i)$$



x_1	x_2	y	$\hat{y}$	Loss
~	~	20	25	$\|20-25\| = 5$
~	~	50	72	$\|50-72\| = 22$
~	~	70	90	$\|70-90\| = 20$

47/3



Huber loss

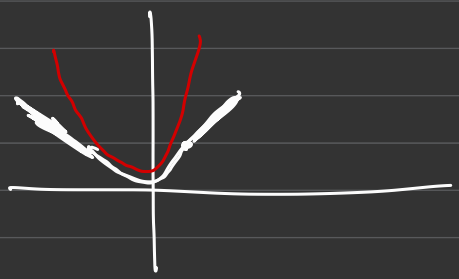
$$= \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |(y_i - \hat{y})| \le \delta \\ \delta |y_i - \hat{y}_i| - \frac{1}{2}\delta^2 & \text{if } (|y_i - \hat{y}_i|) > \delta \end{cases}$$

δ → Threshold parameter that determines the point of transition from quadratic to linear loss

Pseudo Huber loss:

$$\delta^2 \left(\sqrt{1 + \frac{(y_i - \hat{y}_i)^2}{\delta^2}} - 1 \right)$$

make the Huber loss differentiable



Classification losses.

Binary Cross Entropy loss:

x_1	x_2	y	$\hat{y}$
~	~	1	0.7
~	~	1	0.6
~	~	0	0.1

0.3 ~ 0.9

$$\log \text{loss (BCE)} = -\left[y_i \times \log(\hat{y}_i) + (1-y_i) \times \log(1-\hat{y}_i)\right]$$

$$= 1 \times \log(0.7) + 0 \times \log(0.3)$$

$$= 0.15$$

$$-\left[1 \times \log(0.3)\right] + 0 \times \log(0.7)$$

$$= 0.52$$

$$-1 \times \log(0.9) + 0 \times \log(0.1)$$

$$= 0.04$$

$$-y_i \times \log \hat{y}_i + (1-y_i) \log(1-\hat{y}_i)$$

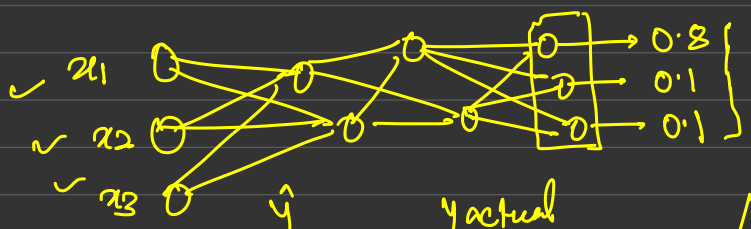
$$-\left[0 + (1) \times \log(0.9)\right]$$

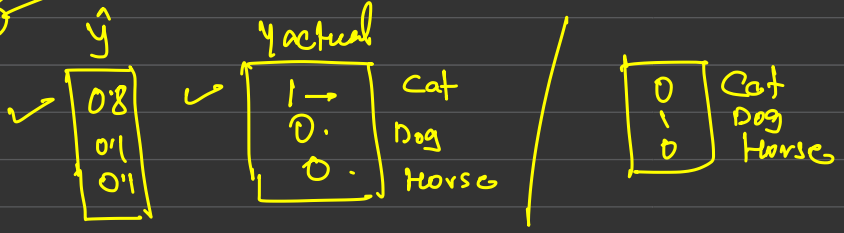
$$= 0.04$$

Categorical cross entropy

↓

multi classification problems



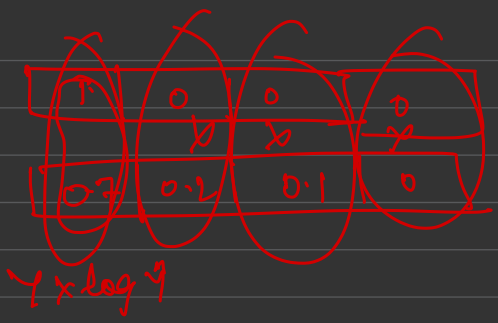


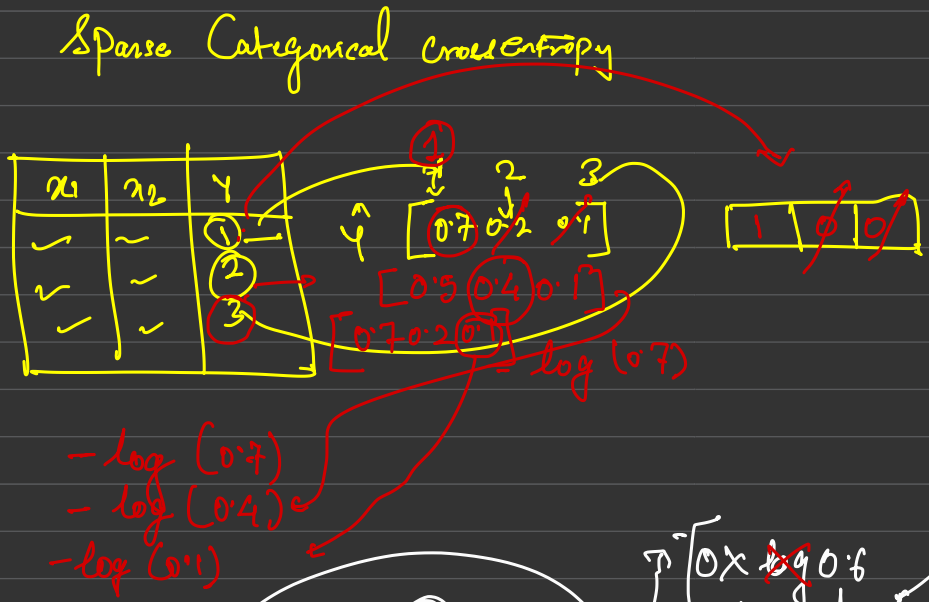
$$\text{loss } b/w(y, \hat{y})$$

$$-\sum_{i=1}^{n}\sum_{j=1}^{p} y_{ij} \times \log \hat{y}_{ij}$$

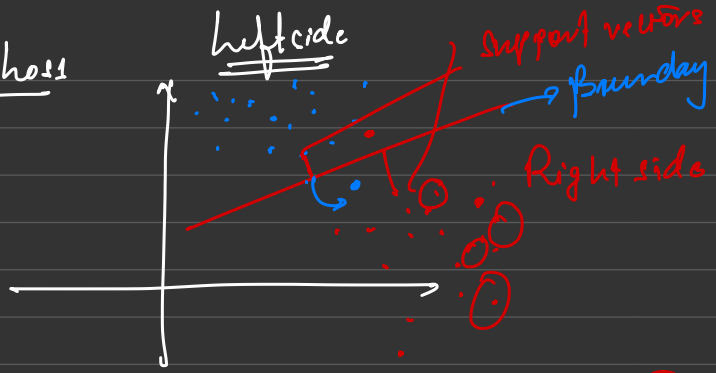
$$-[1 \times \log(0.7) + 0 \times \log(0.2) + 0 \times \log(0.1)]$$

$$\log(0.7) = 0.154$$


1
0
0
0
0.7
0.2
0.1
0
y x log y

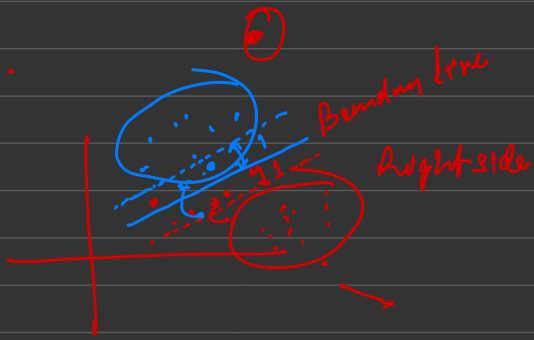

Sparse Categorical CrossEntropy
x1
x2
y
1
2
3
0.7 0.2 0.1
[0.5 0.4 0.1]
[0.7 0.2 0.1]
1 0 0
-log (0.7)
-log (0.7)
-log (0.4)
-log (0.1)
0 x log 0.6
+ 1 x log (0.3)
+ 0 x log (0.1)
0.6 0.3 0.1
-log (0.3)

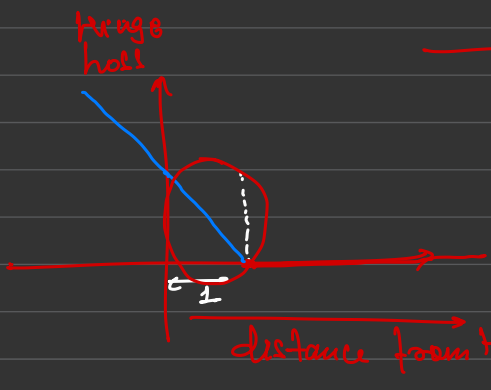
Hinge loss



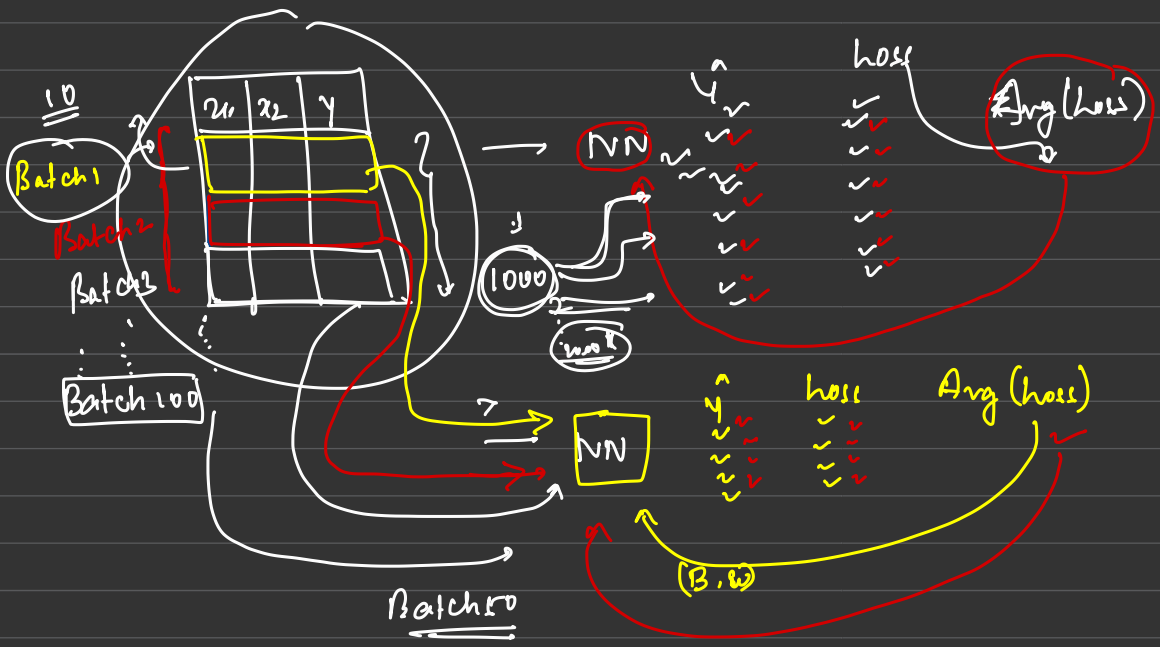
$\max(0, 1 - y \times \hat{y})$

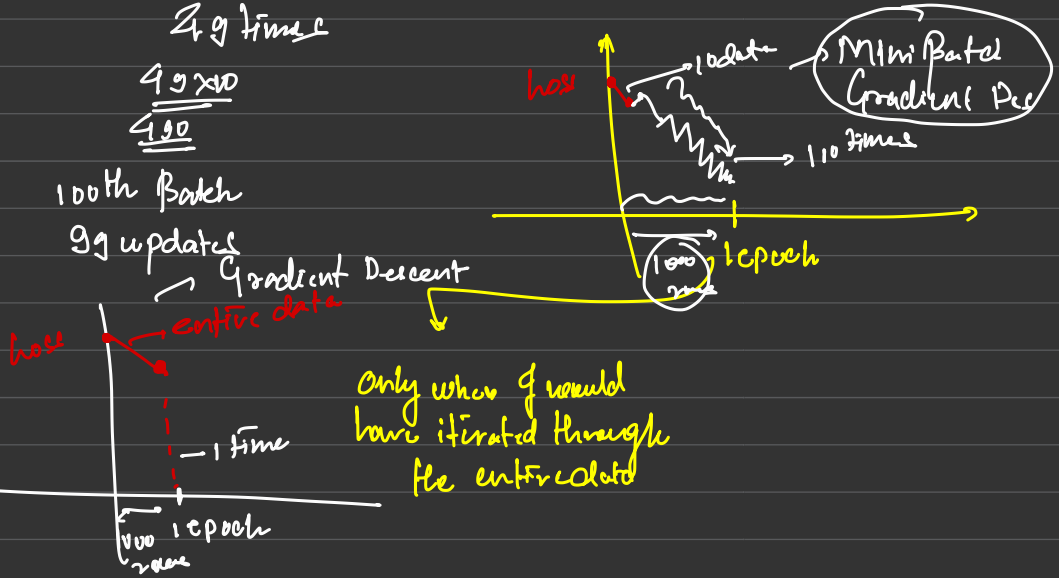
$y \in [-1 \,/\, 1]$



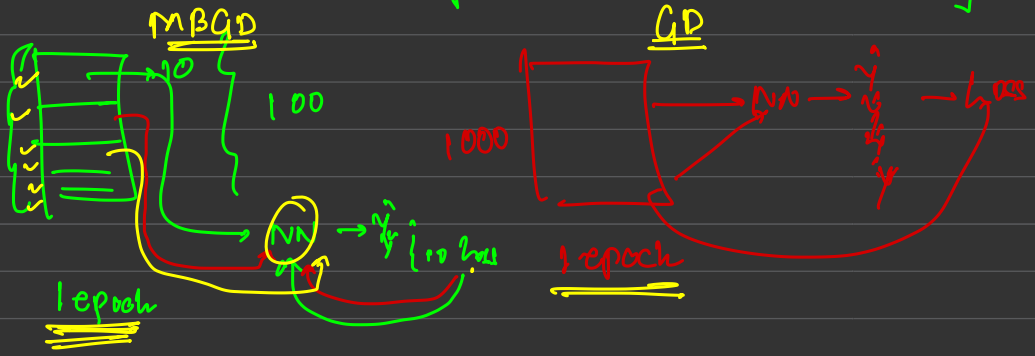


Batch Normalization

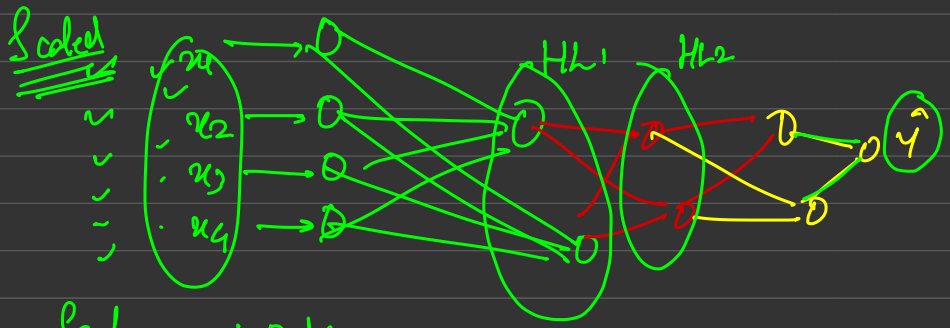




In a mini batch gradient descent, we divide the total data set into mini batches - for eg if my total no of samples is lets say 1000, and I choose the batch size = 10, then I will have 100 batches at my hand.



Batch Normalization



Scale our inputs

$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$

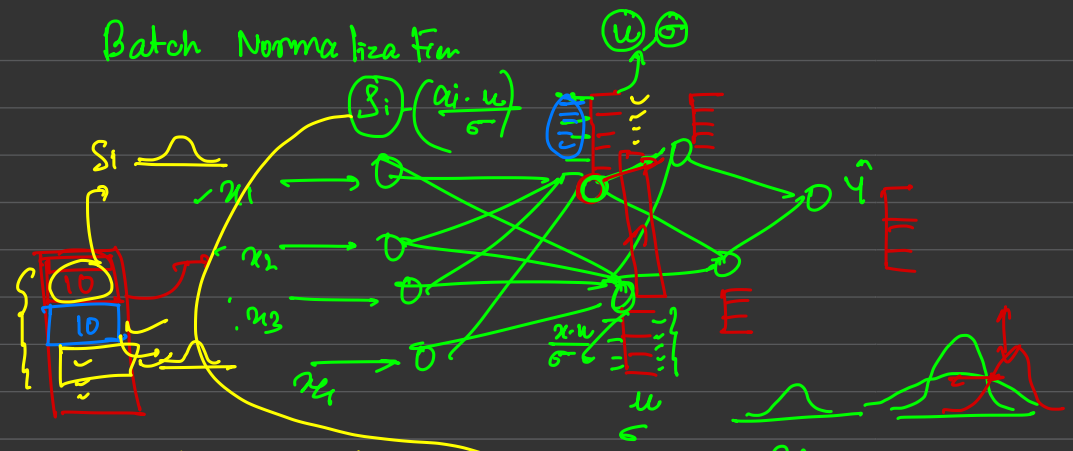
β_2

Ram
Honda City
50,000 kms
3rd Owner

Shyam
Honda City
52,000 kms
2000
1st Owner

Batch Normalization

Batch Normalization



putting too much of a constraint to my output

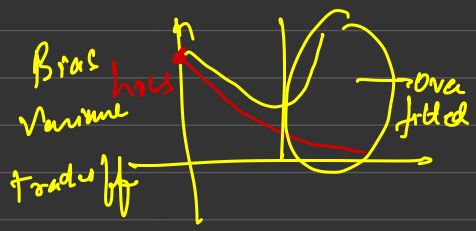
Ni = γ Si + β

Gamma

Beta

learnable parameters

Regularization



less robust (Variance High)

model changes predictions rapidly given slight changes in data

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

(1) $y = [100 + 5x_1 + 2x_2]$ — $\Delta x_1 = 1$ → low variance

$\Delta y = 5$, $\Delta x_2 = 1$, $\Delta y = 2$

$y = 100 + (100 x_1 + 1000 x_2)$ — High variance

if I see a unit change in x_1, 100 units change in y.

" " " x_2 1000 in y

To control / reduce the variance of a model, we can say that β's should not inflate

while training → the abs $|\beta|$ ↑

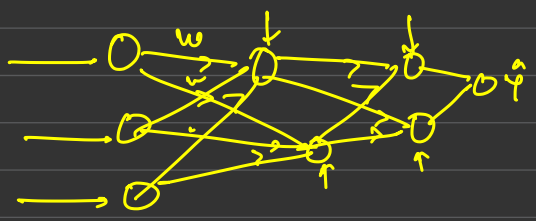
Very high abs $|\beta|$ → high variance

↳ weights and biases

Regularization

- Modifying the loss function
  - Dropout
  - Activation
    - Relu
  - lasso
  - Ridge
  - Elastic Net

Lasso Regression



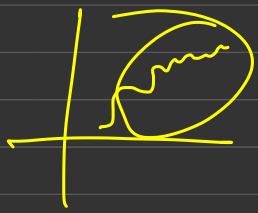
$$\checkmark \quad MSE = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}$$

During your epochs your MSE will reduce

$$\text{new loss} = MSE + \lambda \sum_{i=1}^{p} (|w_i|) \rightarrow \text{Lasso}$$

↓ weight and biases

$$\text{new loss} = MSE + \frac{1}{2} \sum_{i=1} (w_i^2) \rightarrow \text{Ridge}$$



MSE

Lagrangian parameter

①	50 +	②	= 52
②	40	5	= 45
③	32	7	= 39
4	25	11	= 36
5	24	18	= 42

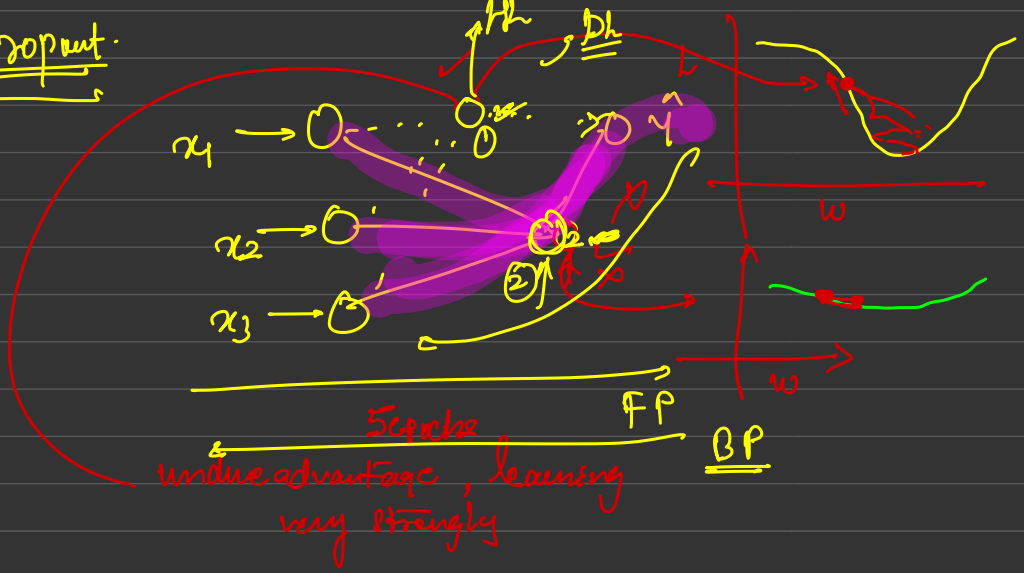
X

Modified loss Elastic Net

$$= \text{MSE} + \lambda\left(\alpha \sum_{j=1}^{p} |B_j|\right) + \left(\frac{1-\alpha}{2} \sum_{j=1}^{p} B_j^2\right)$$

Lasso — Ridge.

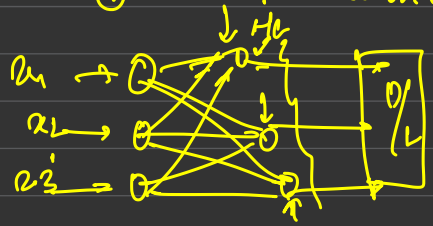
Dropout.



Randomly we dropout neurons from the training iteration.

H Neuron will stay (0.8)

(1) T Neuron will dropout (0.2)



Weight Initialization

1) Define your nn architecture, input-dim, HL → neurons
   o/p, activation, bias, matrices

2) weights, and biases in the nn → randomly initialized

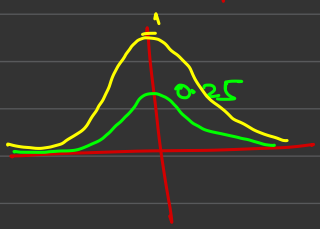
3) Training → FPE BP, the weights get trained

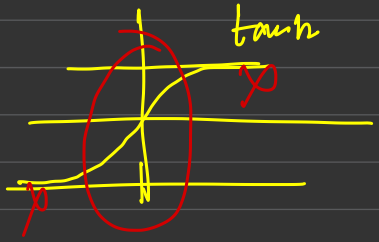
Zero Initialization :
↳ the entire network becomes symmetrical
→ ✗

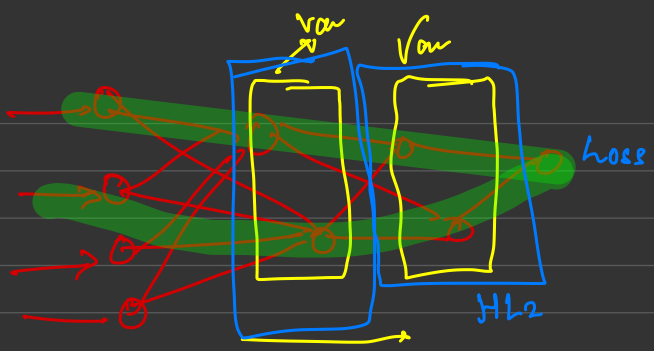
Random Initialization
↳ Generate weights which are very high
↳ activations high

Sigmoid

very low ← Vanishing gradient descent







forward flowing signals

$$Var\left(\frac{dh}{dHL2}\right) = Var\left(\frac{dh}{dHL1}\right)$$

to maintain that the neural network does not enter into a Vanishing or Exploding gradient problem.

Xavier initialization
He initialization

Xavier initialization (tanh)

$$W^{[l]} \sim N\left(\frac{N(0,1)}{n_{i+1}, n_i}\right) \times \sqrt{\frac{1}{n_i}}$$

n_i = n_{i+1}

$n_{i+1} \times n_i$

$$W^{[l]} \sim N(n_{i+1}, n_i) \times \sqrt{\frac{2}{n_{i+1}+n_i}}$$

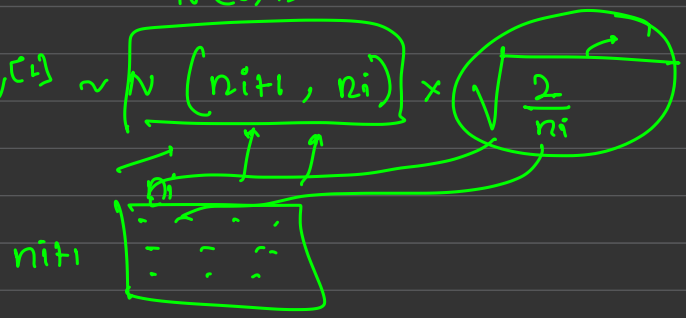
Normalized Xavier initialization

For Relu activation

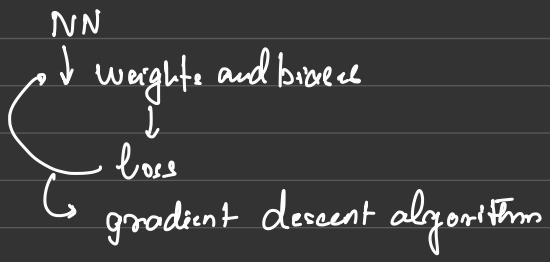
He initialization:

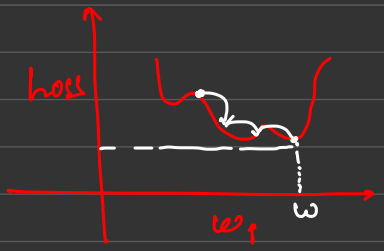
$N(0, \sigma^2)$

$W^{[l]} \sim N(n_{i+1}, n_i) \times \sqrt{\frac{2}{n_i}}$



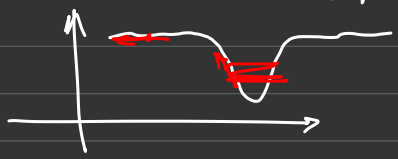
Optimizers





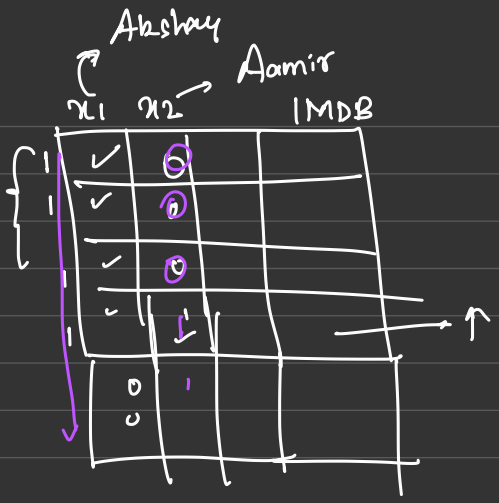
loss curves are extremely complicated functions → multiple minimas

:) be flat at certain regions
be very sharp at certain regions

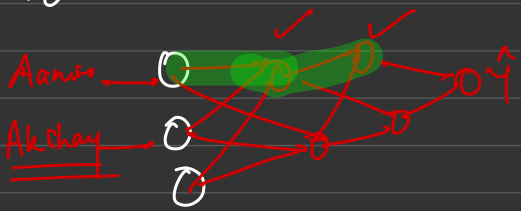


Gradient descent by design has various issues.

:) Multiple local minimas.
:) The gradients are not consistent
:) Sparse features create a problem in GD



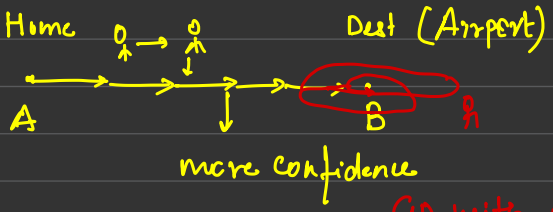
most of the rows of aamir khan's feature



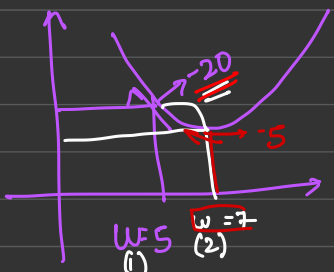
weights related to Aamir Khan
weights related to Akshay Kumar

$$w_{i+1} = w_i - \alpha \, dJ/dw_i$$

Momentum



GD



$w(2) = w(1) - \alpha \left(\frac{dh}{dw}\right)$, 0.1

$w(2) = w(1) - \alpha \times -20$

$w(2) = 5 + 2$

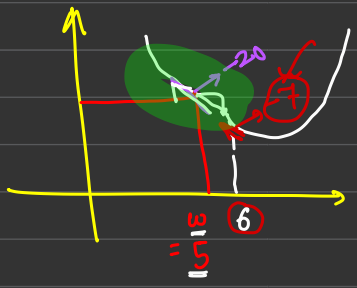
$w(2) = 7$

$w(3) = w(2) - 0.1 \times -5$

$w(3) = 7 + 0.5$

$= 7.5$

GD with momentum



$m(0) = 0$

$m(t) = \beta\, m(t-1) + (1-\beta) \times \frac{dL}{dwt}$

update → $w(2) = w(1) - \alpha(mt)$

$m(1) = 0.5 \times 0 + 0.5 \times -20$

$m(1) = -10$

$w(2) = 5 - 0.1 \times -10$

$w(2) = 6.$

$m(2) = 0.5 \times -10 - 0.5 \times -7$

$m(2) = -[5 + 3.5] = -8.5$

$w(3) = w(2) - 0.1 \times -8.5$

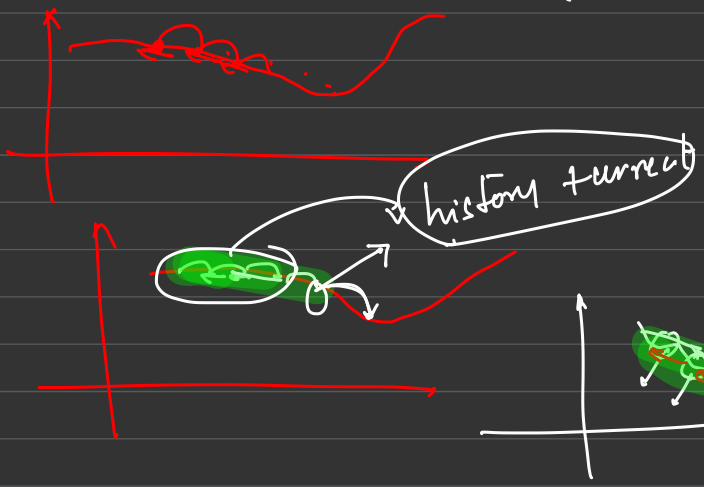
$w(3) = 6 + 0.85$

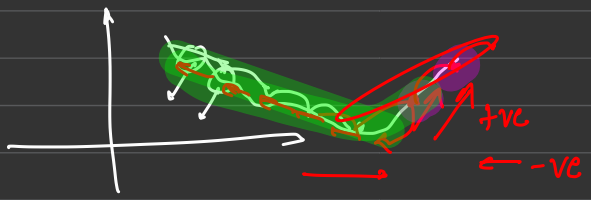
$= 6.85$

Momentum based gradient update actually takes into account the historical gradient × weightage + the current gradient × weightage.

↓

accelerates the process of the update





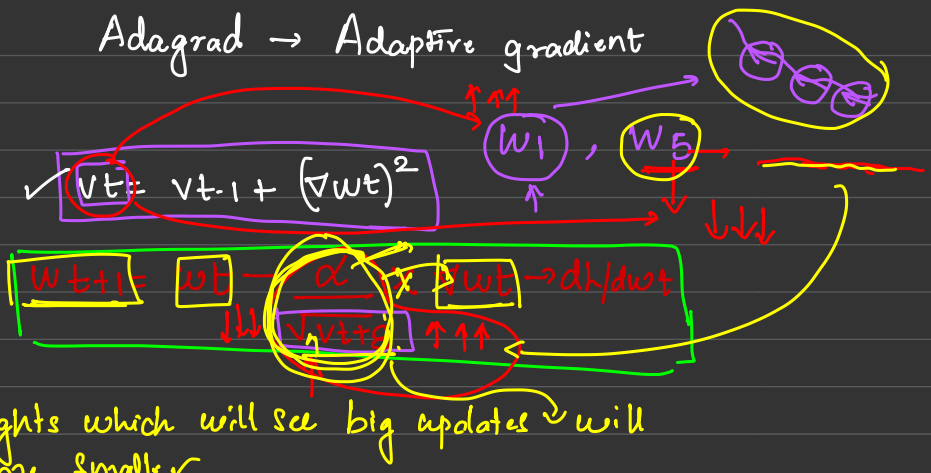
$$w_{t+1} = w_t - \alpha\, m_t$$

$$m_t = \beta \times m_{(t-1)} + (1-\beta) \times \frac{dL}{dw_t}$$

history — point / current

Even the if what min it takes longer U turns it is so fast it crosses / ignores local min

Adagrad → Adaptive gradient



Weights which will see big updates will become smaller

Adaptive gradients start punishing weight updates for weights which have already seen big gradient updates

→ dont punish/ reward weight updates for weights which have not seen big gradient updates

Rewarding (Aamir khan) ↑↑
Punish (Akshay kumar) ↓↓ → balanced learning

$$v_t = v_{t-1} + (\nabla w_t)^2$$ → too high too fast

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} \times dh/dw_t$$

Rms Prop.

$$V_t = \beta \times v_{t-1} + (1-\beta) \times (\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \varepsilon}} \times dh/dw_b$$

Epsilon

Adaptive moments

Adam optimizer.

momentum

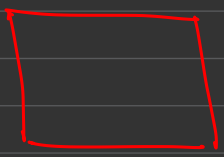
$$m_t = \beta_1 \times m_{t-1} + (1-\beta_1) \times \nabla w_t$$

$$v_t = \beta_2 \times v_{t-1} + (1-\beta_2) \times (\nabla w_t)^2$$ → RMS

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{v}_t + \varepsilon}} \times \hat{m}_t$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2}$$

Convoluted Neural Network



What is an Image

What type of problems (AI) using Images

ANN is a bad choice

CNN

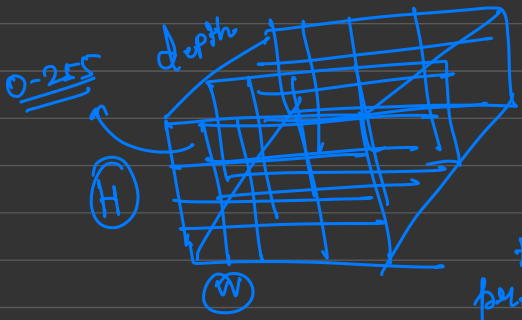
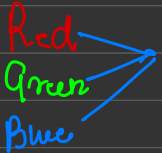


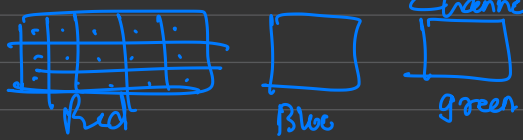
Image is a rank 3 tensor with dimensions of H x W x d.

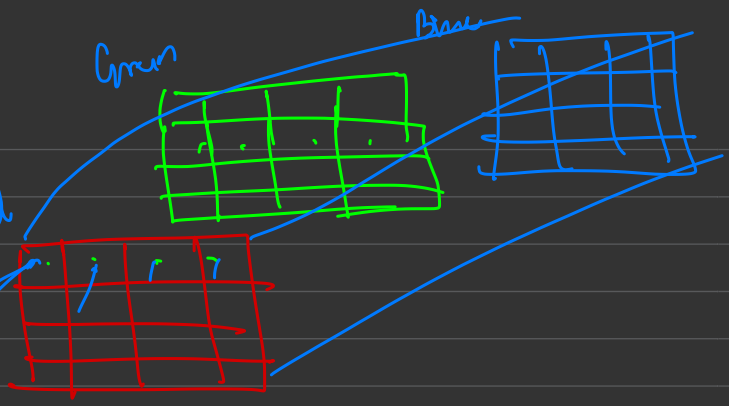
these many cells are present

within these cells we have some number (0-255)



Photons → Sensors → voltage (Red, Green, Blue channels)



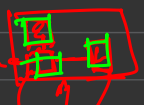


as (information content in an image)

→ Variation in the pixel values across the Red, Green, Blue Channel.

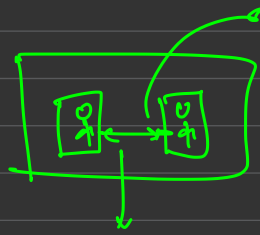
detect an object in an image
→ image classification

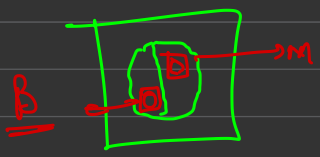
AI problems:
① Image classification
② Object detection and localization



CCTV Camera

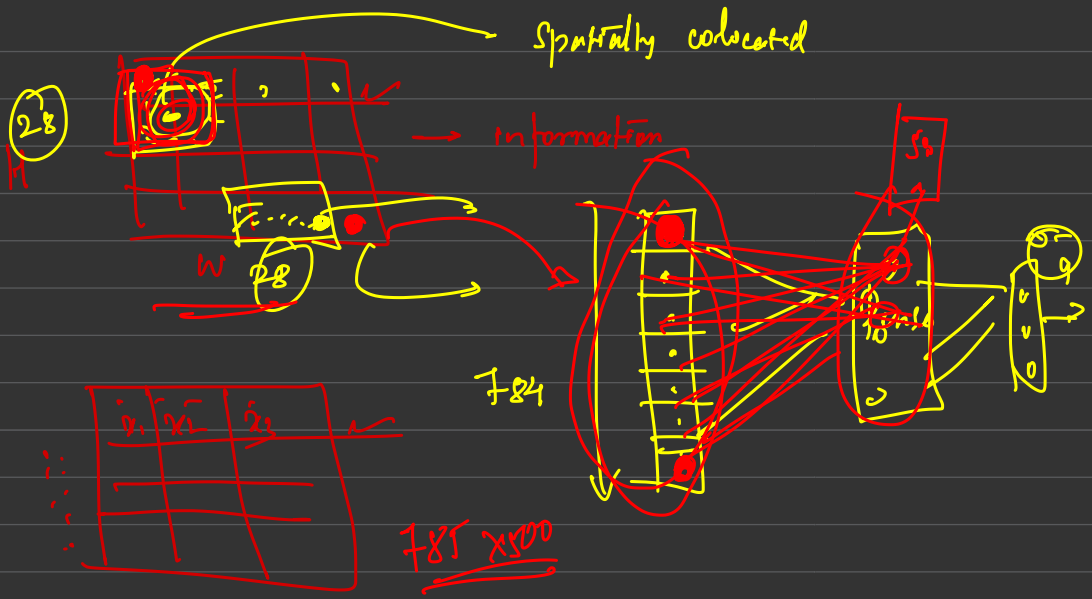
SD > 2.0m



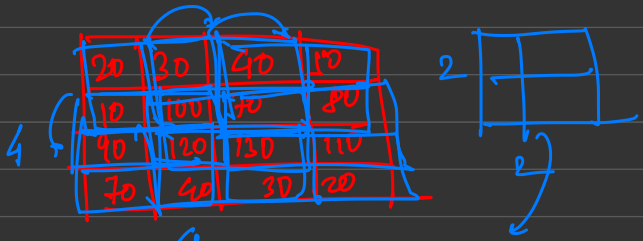


Facial Recognition System (Siamese network)

login phone

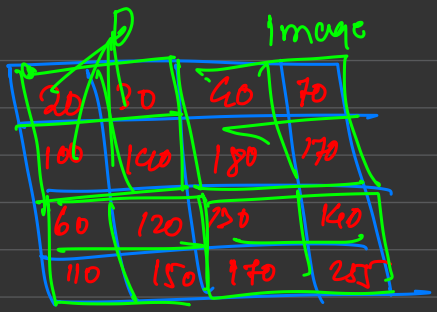


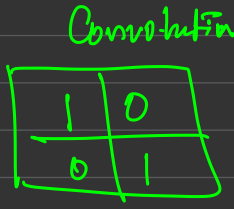
CNN



filter will move throughout the image and extract features

movement of the filter throughout the image is called Convolution





160

$20\times1 + 30\times0 + 100\times0 + 140\times1$

210

$40\times1 + 70\times0 + 180\times0 + 170\times1$

210

$60\times1 + 120\times0 + 110\times0 + 150\times1$

385

$130\times1 + 140\times0 + 170\times0 + 255$

Feature map →

160	210
210	255

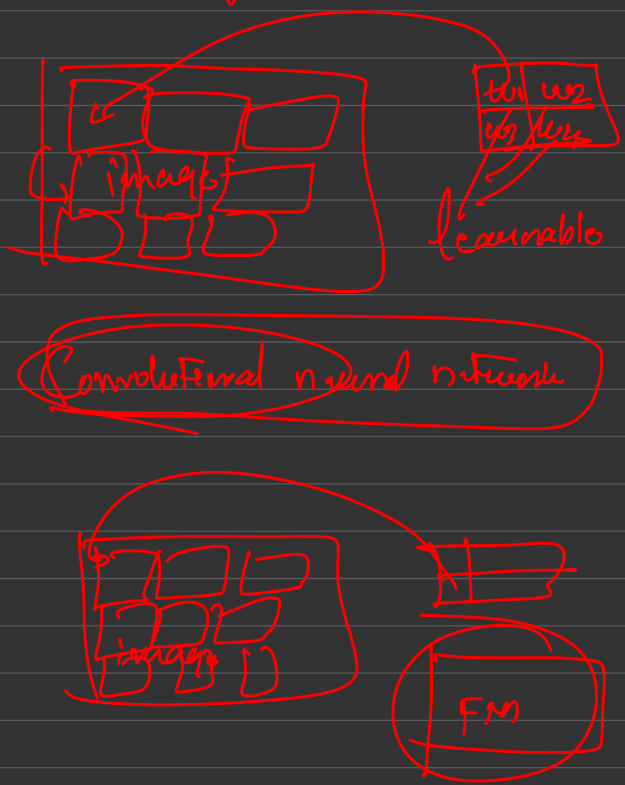
↓

Extracted feature from the image.

that the filter convolving on an image is basically a feature extraction process

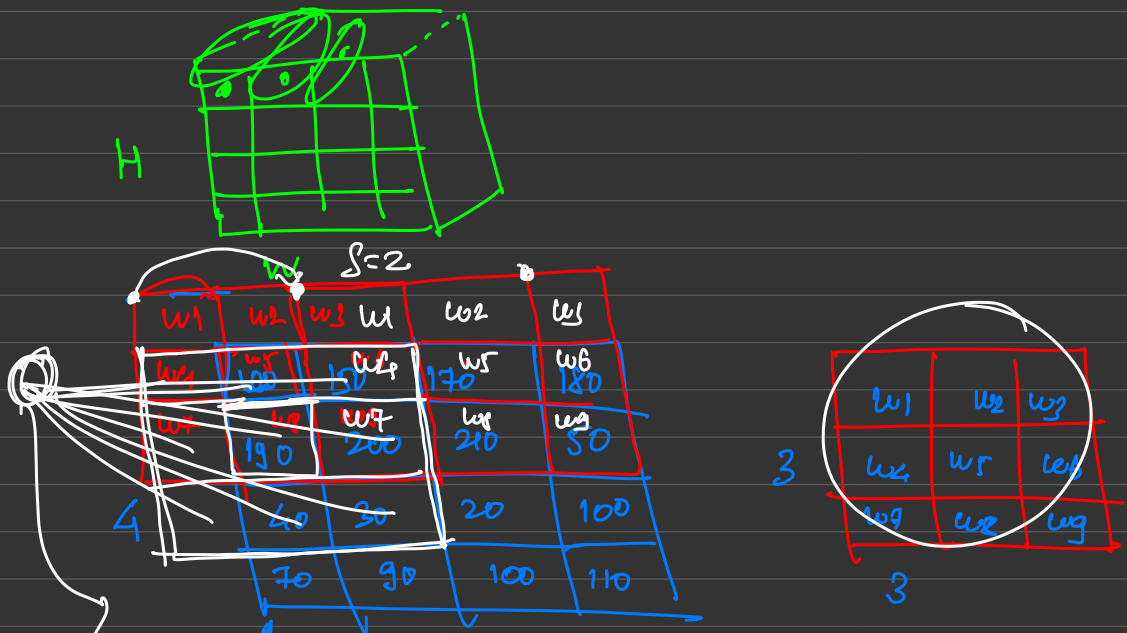
Edges
Boundaries
Curved
Complex Boundaries
Color variations

That the feature extraction



Convolutional neural network

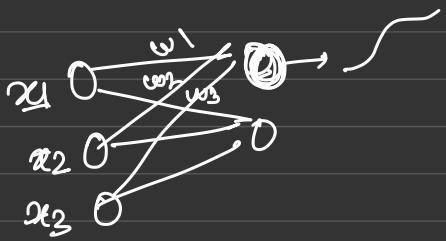
Image classification



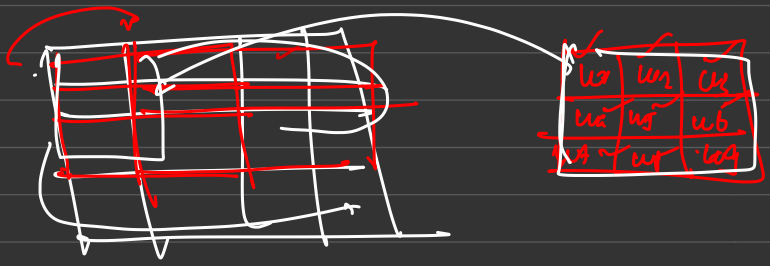
neuron is thereby trying to extract features from the pixel

activation $(w_5 \times 100 + w_6 \times 150 + w_8 \times 190 + w_9 \times 200)$

→ Feature map

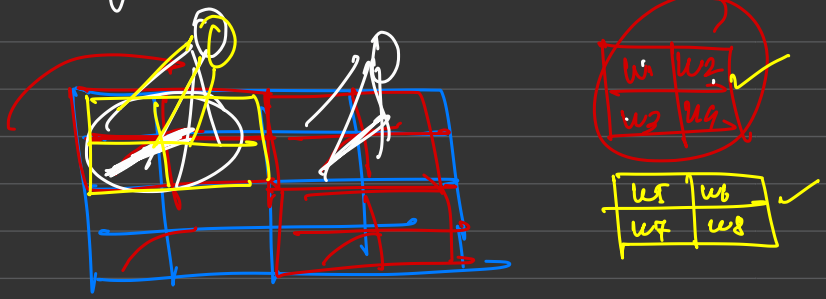


extracting features from an image

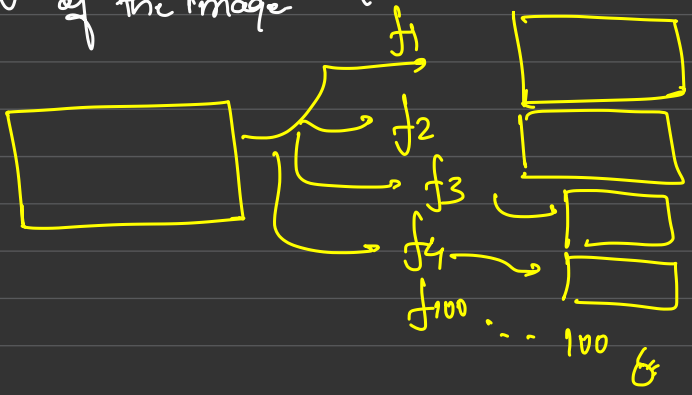


one iteration of the convolution,

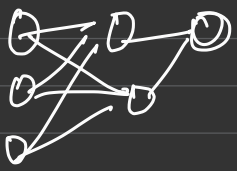
your filter weights don't change.



one filter is learning one kind of feature from the different regions of the image



Convo

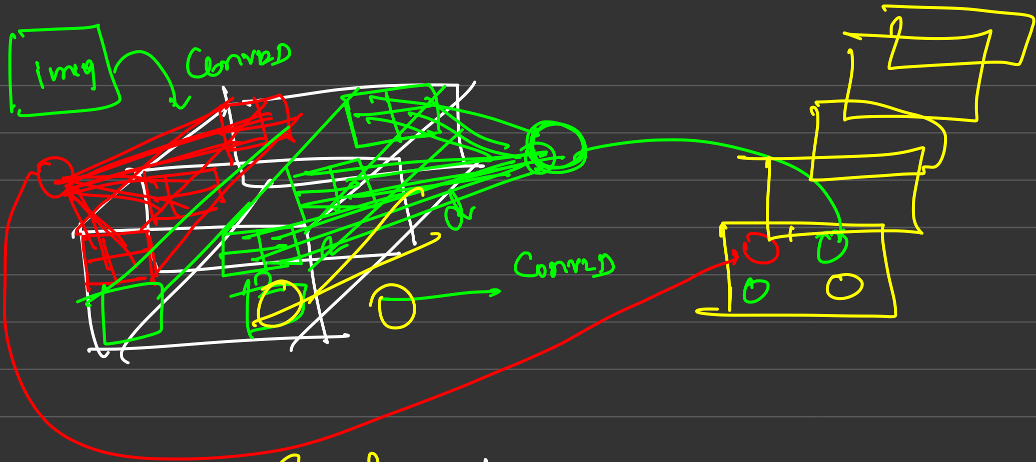


2x2

, neuron is going to learn a non-linear combination of all the features that were generated of the object placed on the top left corner of your original image

f f

one filter applied on an image learns one kind of feature at diff locations
many filters you will learn many features


img
Conv2D
Conv2D

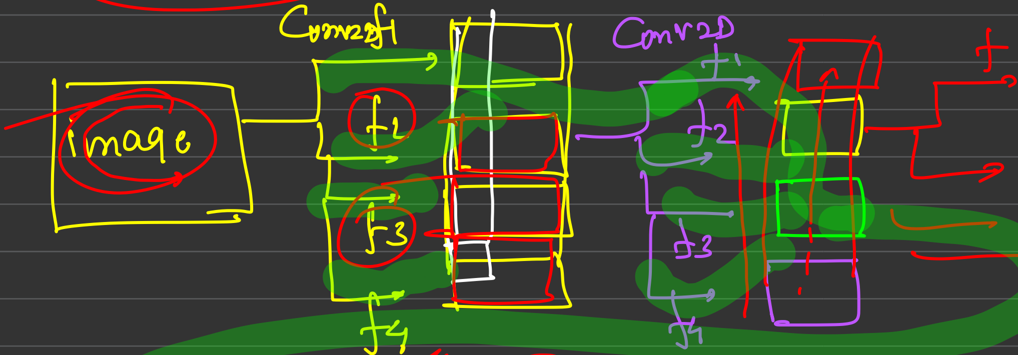


image
Conv2D
f1
f2
f3
f4
Conv2D
f2
f3
f4
f

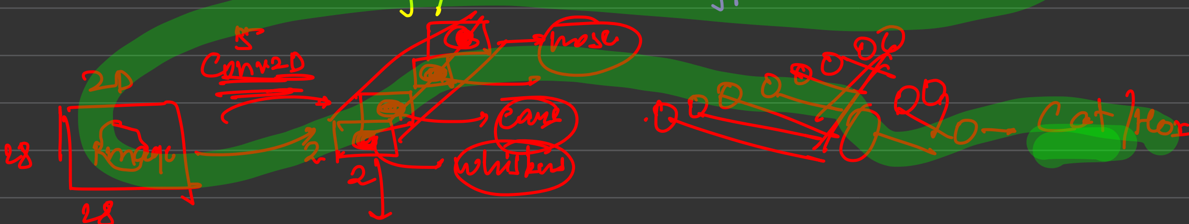
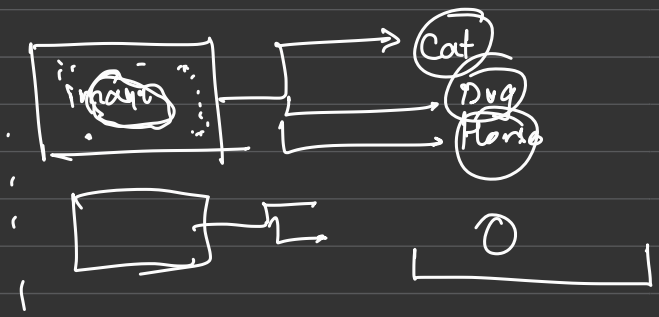

2D
Conv2D
28
28
nose
ears
whiskers
cat

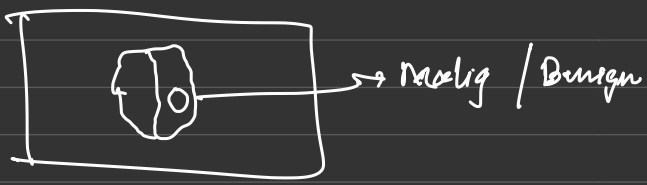

image
Conv2D
maxpooling
Conv2D

Convolutional neural network

extract information from images

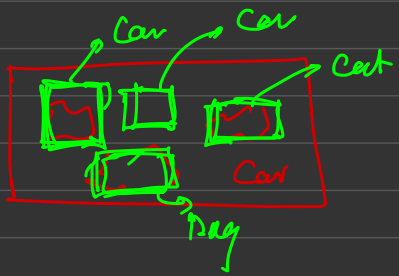
Image classification

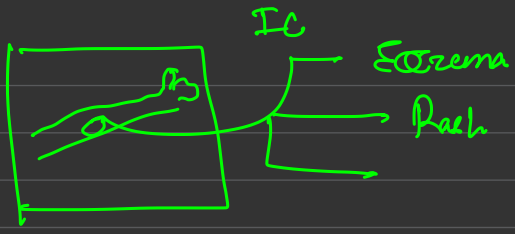


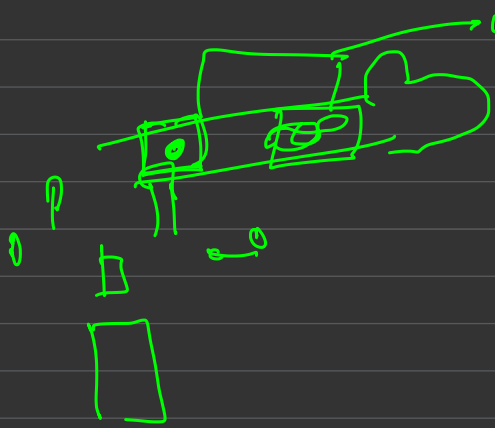


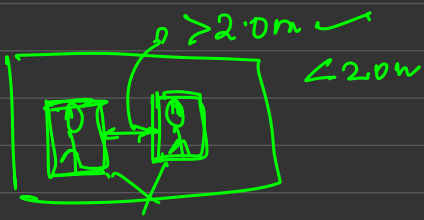
Object detection & localization

Car, Cat, Dog

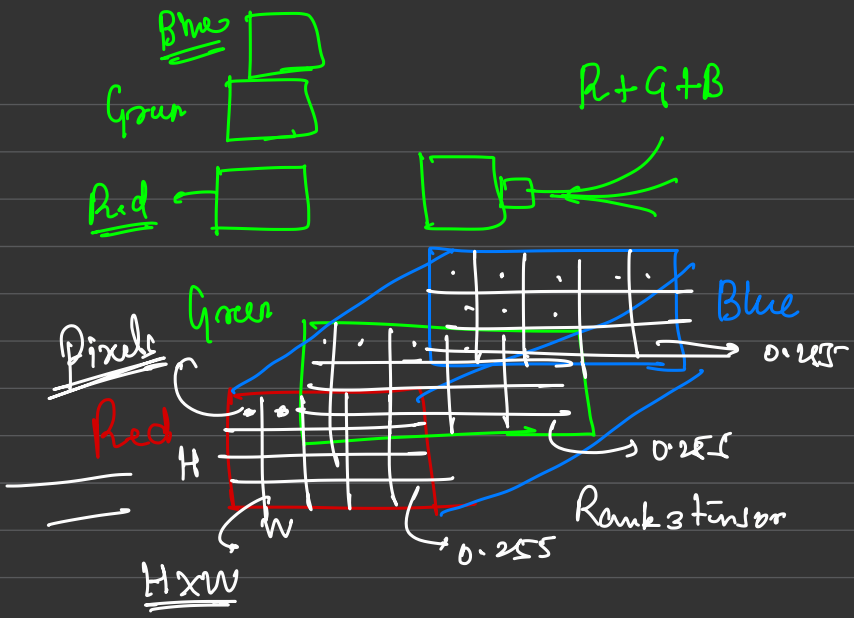






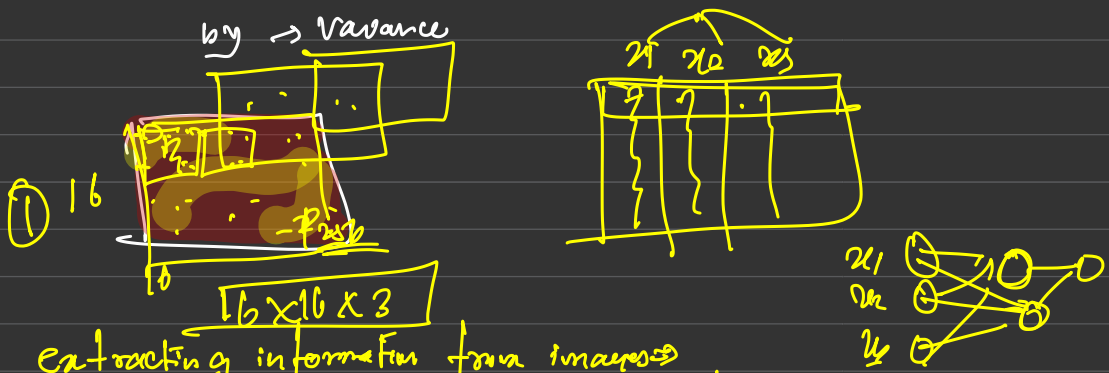


Siamese Network (Facial Recognition)


Blue
Green
Red
R+G+B
Pixels
Green
Blue
Red
H
W
HxW
0-255
0-255
0-255
Rank 3 Tensor

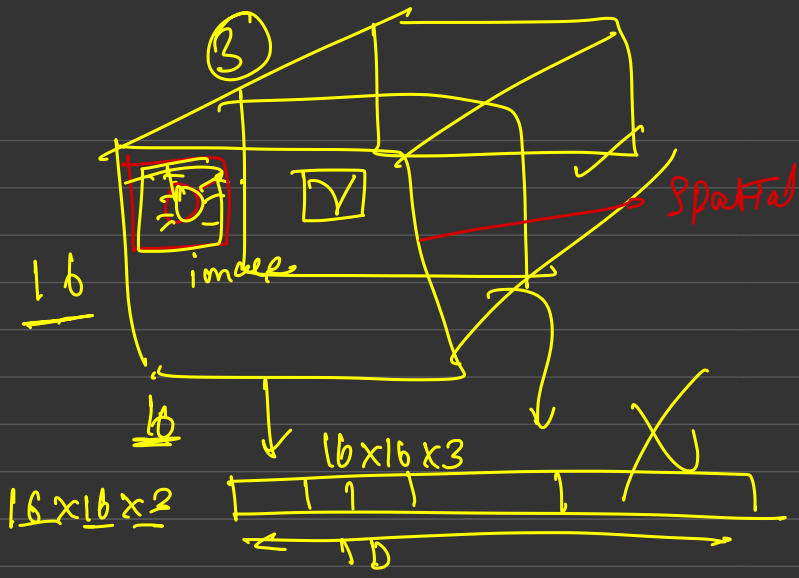
What is information in an image

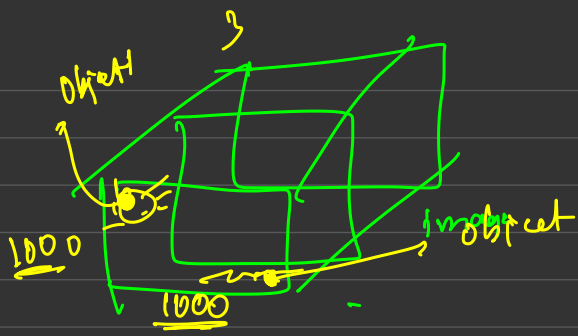
Variance in pixel values across all the 3 color channels is what we mean by → variance



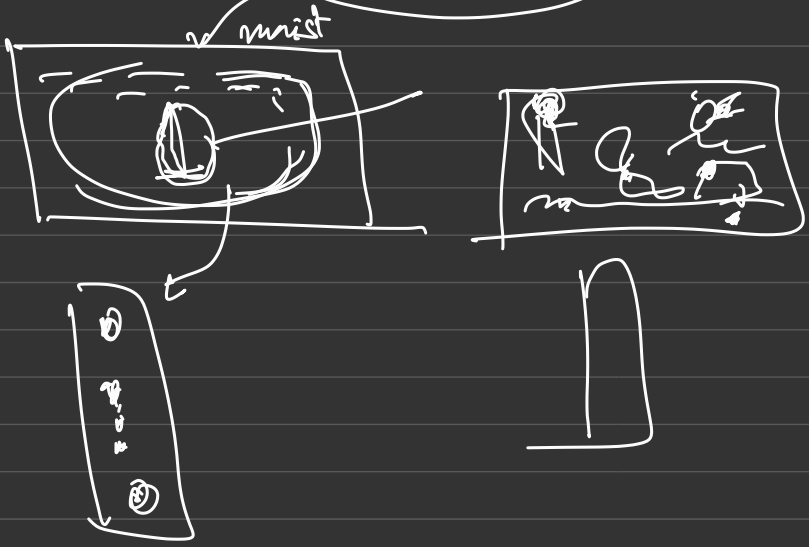
extracting information from images ⇒ extracting features / modifying pixel values

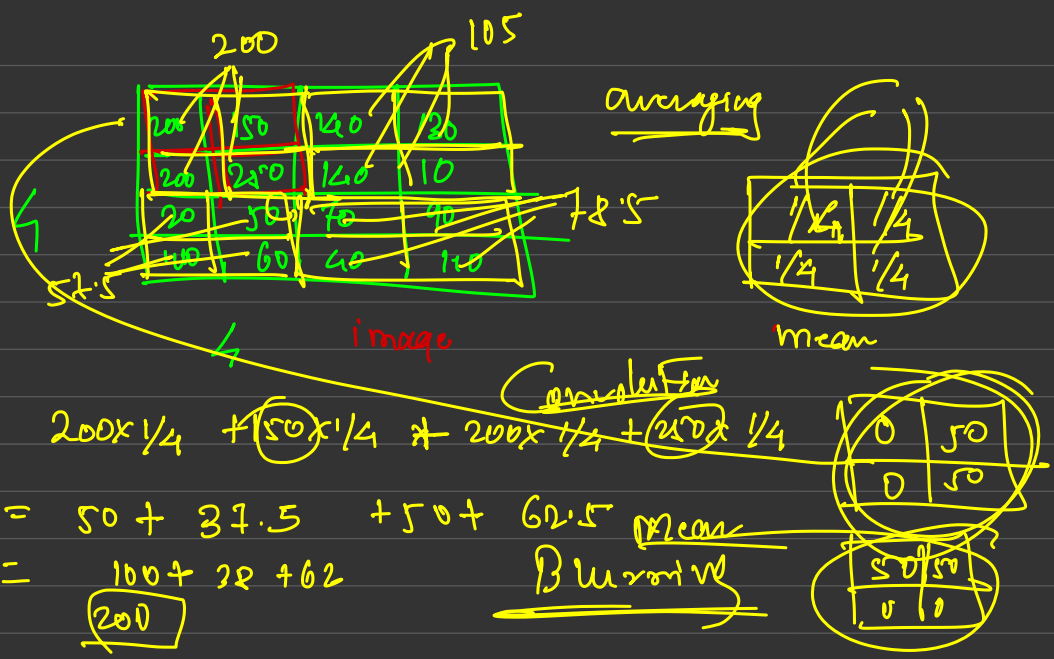
Image ∼ $\begin{pmatrix} R \\ (p_1 \dots p_{256}) \\ G \\ (p_1 \dots p_{256}) \\ B \\ (p_1 \dots p_{256}) \end{pmatrix}$


3
Spatial
image
16
16
16x16x3
16x16x2
10



· Convolution




200
105
200
150
140
30
200
250
140
10
20
50
70
40
140
60
40
140
37.5
78.5
image
averaging
mean
Convolution
200×1/4 + 150×1/4 + 200×1/4 + 250×1/4
= 50 + 37.5 + 50 + 62.5
= 100 + 28 + 62
201
Blurred
mean

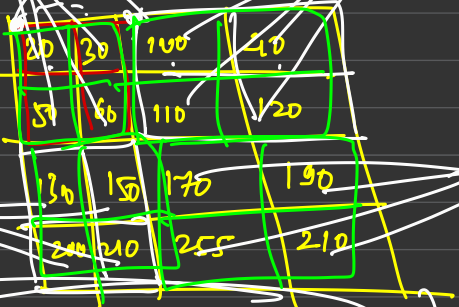
manual process of extracting features

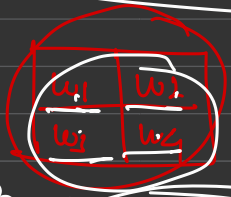
Convolution Neural network

tanh (20 × w1 + 30 × w2 + 50 × w3 + 60 × w4 + b)

Stride = 2

tanh (100 × w1 + 40 × w2 + 110 × w3 + 120 × w4 + b)





tanh (170 × w1 + 190 × w2 + 255 × w3 + 210 × w4 + b)

(130 × w1 + 150 × w2 + 200 × w3 + 210 × w4 + b)

Non-linearity (20 × w1 + 30 × w2 + 50 × w3 + 60 × w4 + b)

ReLU / Tanh

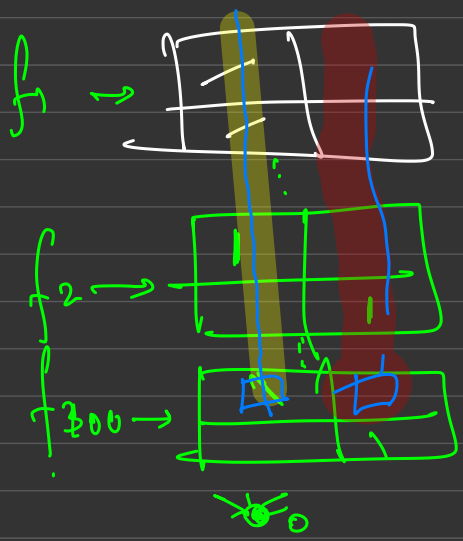
Capture a feature non-linear

Using this filter
I have identified one particular pattern across the image

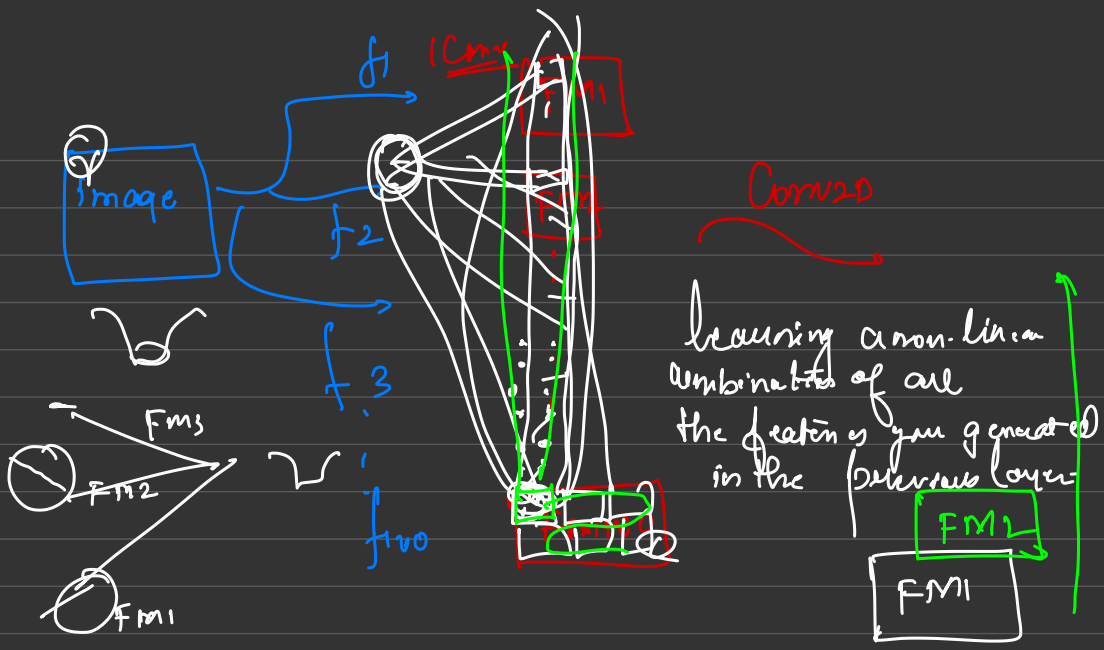
250	-150
60	-100

Feature map

w5	w6
w7	w8



feature maps are basically the deconstructed version of your object at that location


f1
1 Conv
FM1
image
f2
Conv2D
f3
FM3
FM2
FM1
learning a non-linear combination of all the features you generated in the previous layer
FM2
FM1

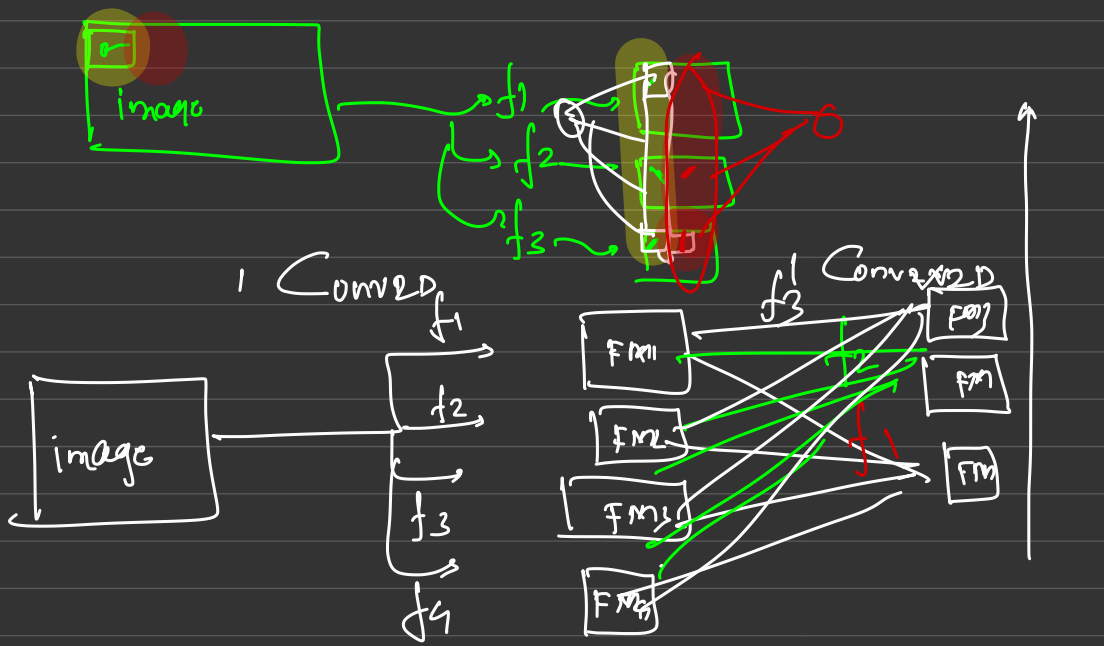
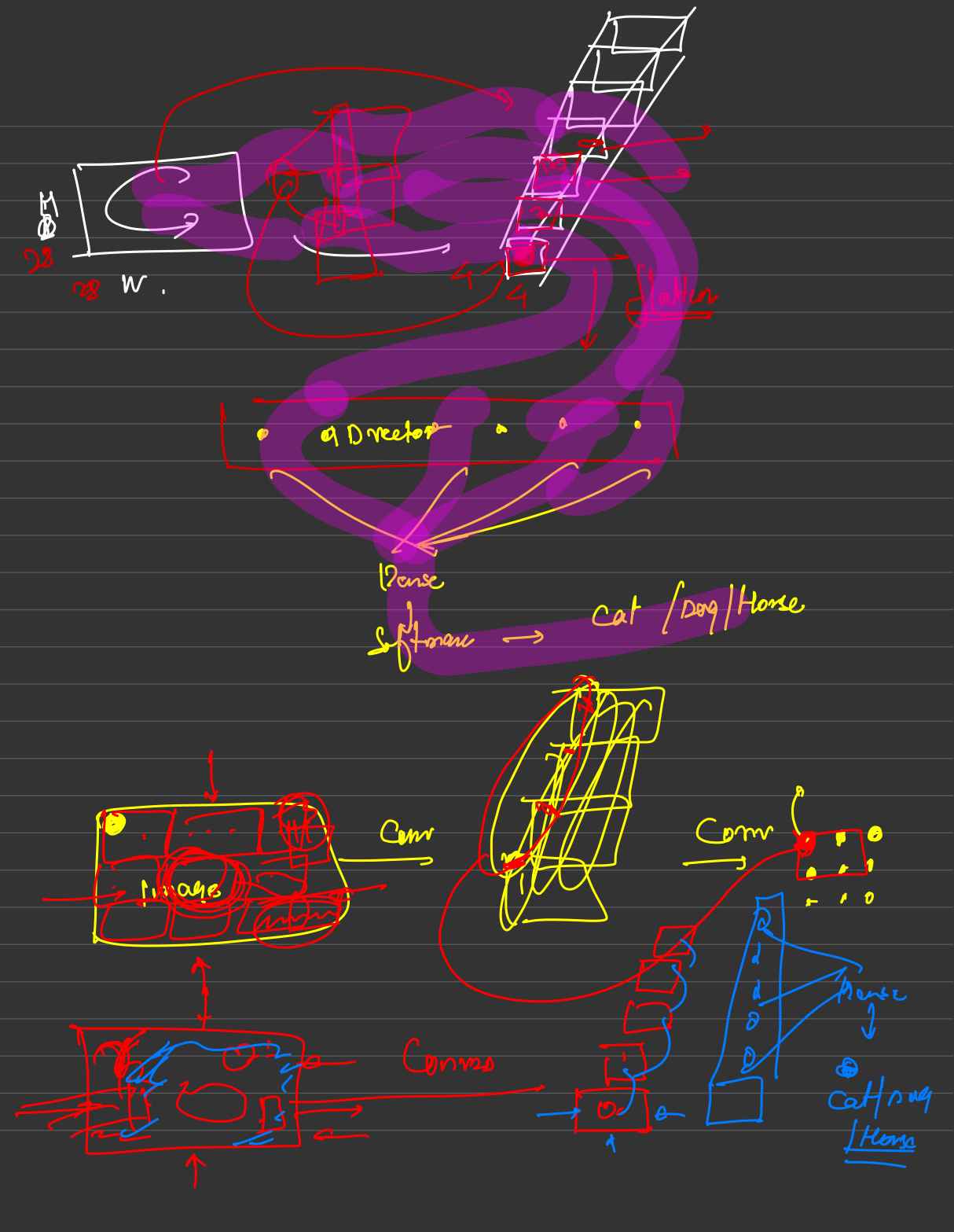


image
f1
f2
f3
1 Conv2D
f1
f2
f3
f4
image
1 Conv2D
f3
FM11
FM12
FM13
FM1
FM
FM


H
28
28 W.
4
4
Flatten
9 Dresses
Dense
Softmax
Cat / Dog / Horse
Image
Conv
Conv
Conv
Dense
Cat / Dog / Horse

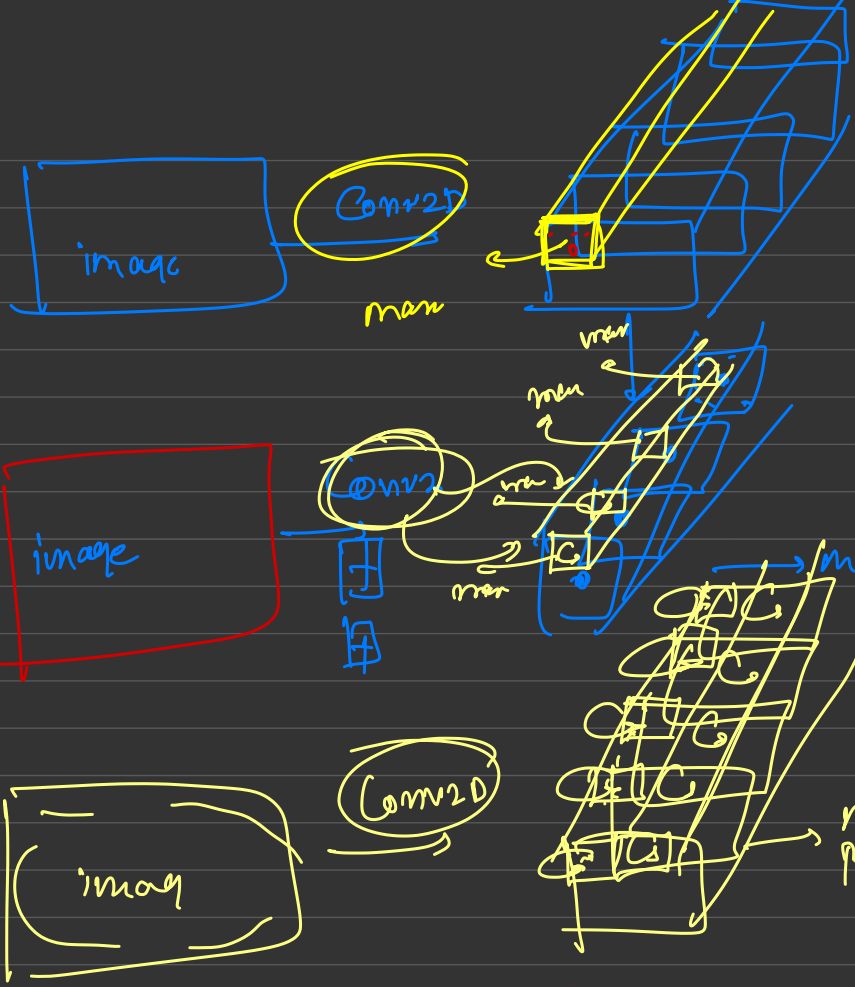


image
Conv2D
max
image
Conv2D
max
max
max
max
maxpooling
image
Conv2D
max pool2D
70
50
40
50
110
120
100
140
110
150
30
20
170
190
120
140
140
2x2
50
120
140
190
image
max
Conv2
max

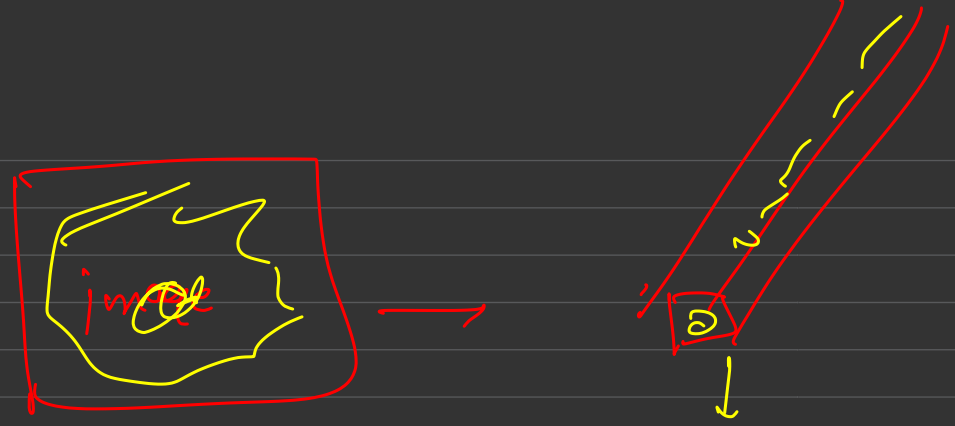
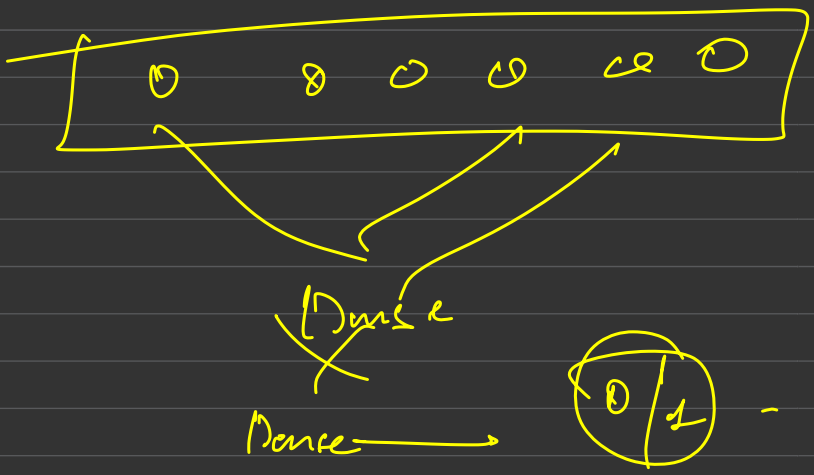
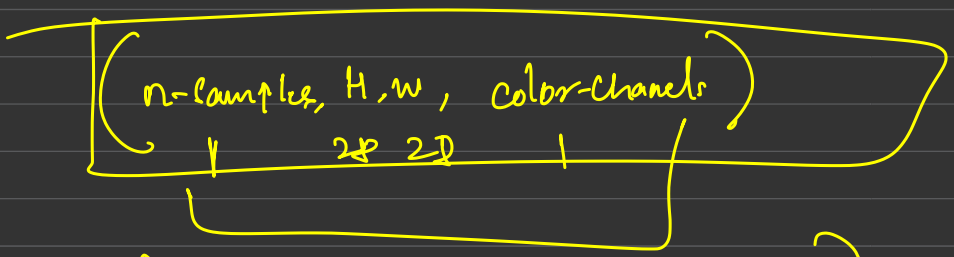


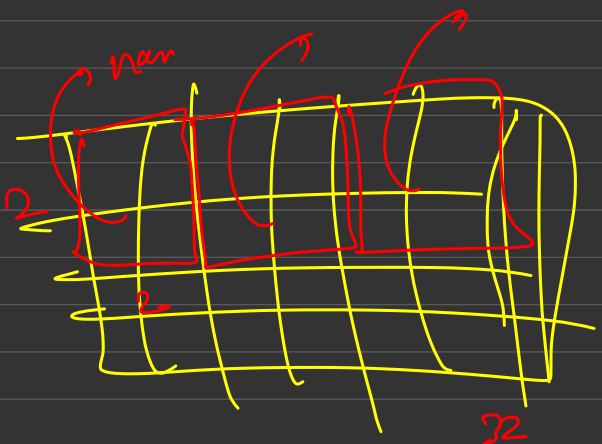
image
0
1

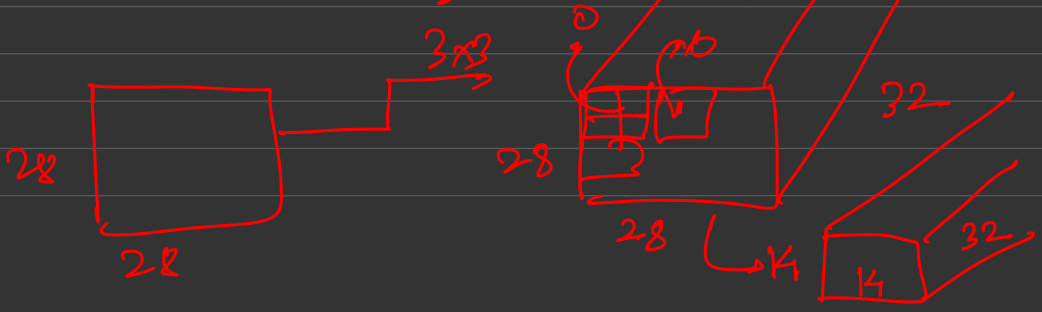

Dance
Dance
0
1

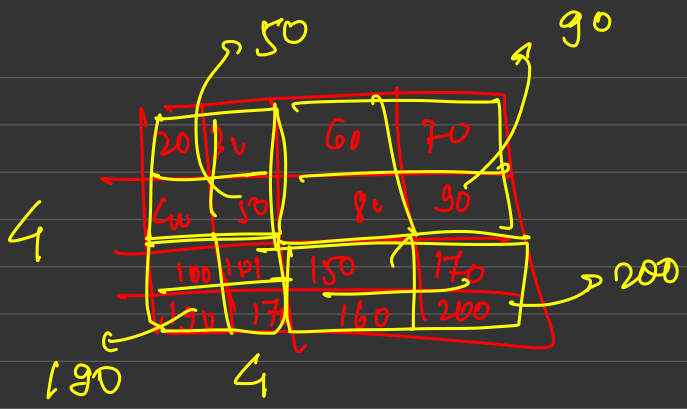
You will always have a image sent to a Conv layer / model as a 4D tensor

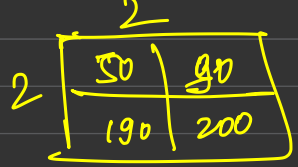


(28, 28) → Mishap (1, 28, 28, 1)






50
90
20
60
70
40
80
90
100
150
170
200
190
4
4


2
2
50
90
190
200