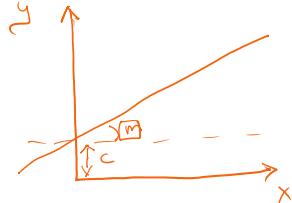


Gen AI - Generate Content

AI - Artificial Intelligence

ML - Machine learning → understand data

DL - Deep learning → processing power



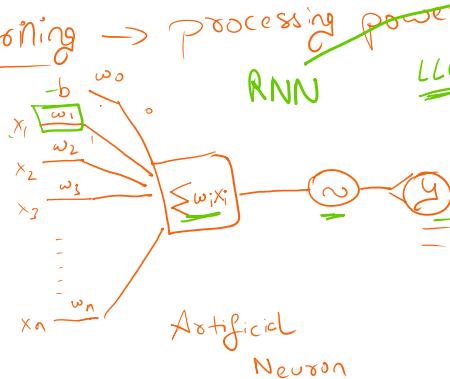
$$y = mx + c$$

Linear deg.

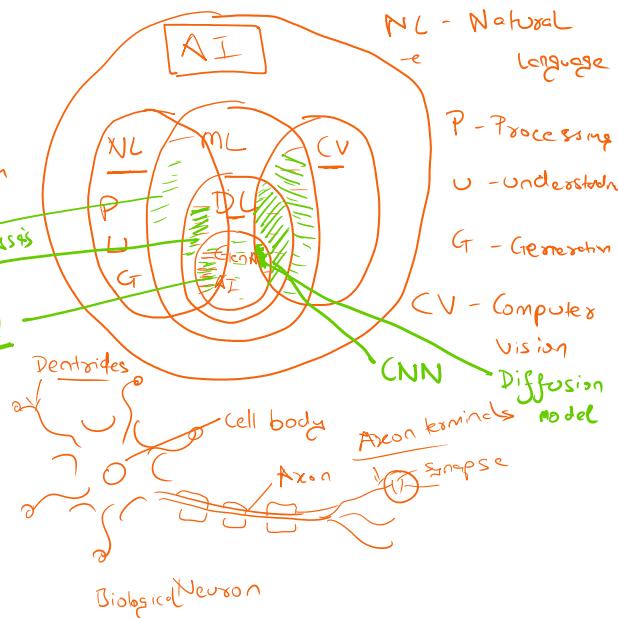
$$y = m_1x_1 + m_2z_2 + \dots + m_nx_n + c$$

$$y = m_1x_1 + m_2z_2$$

Data collect.,

 w = weights b = bias, c = constan., w_0 x_i = feature y = output

Ⓐ - Activation function

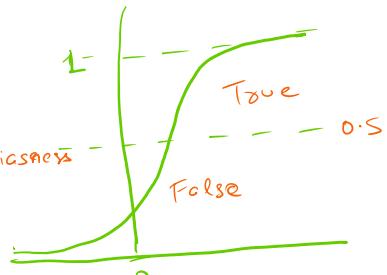
Activation function

- Step func.

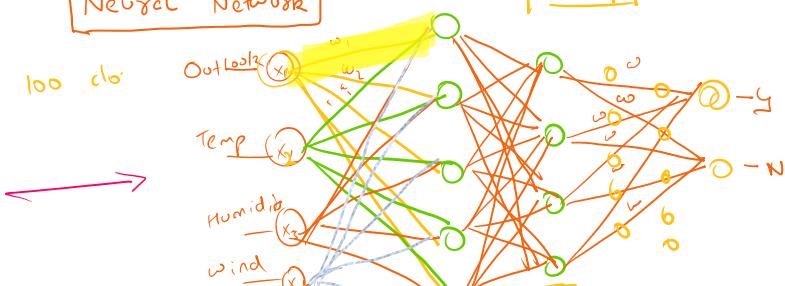
$$\boxed{\text{Sigmoid func}} = \frac{1}{1 + e^{-z}} = [0, 1]$$

- ReLU

- tanh

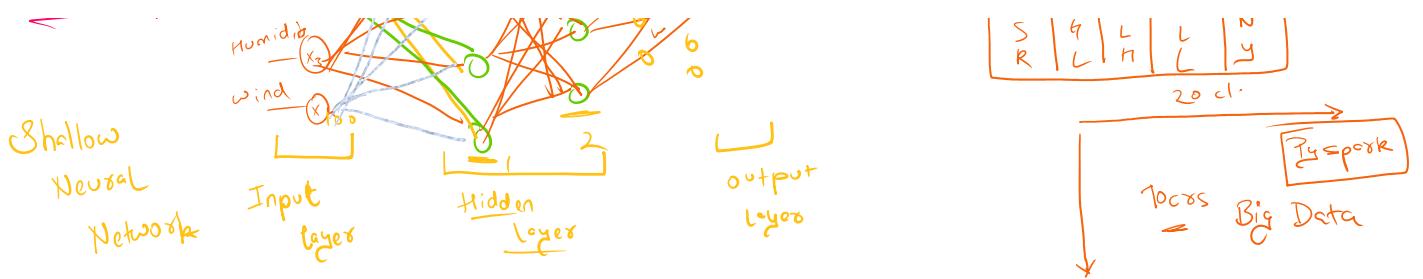
 x = feature w = weights b = (helper value) biasnessBuilding block
of DL

Artificial Neuron
Precptone

Neural NetworkDeep Neural Network

Outlook	Tc	H	W	Y/N
S	H	L	H	Y
R	L	H	H	N
S	G	L	L	N
R	L	H	L	Y

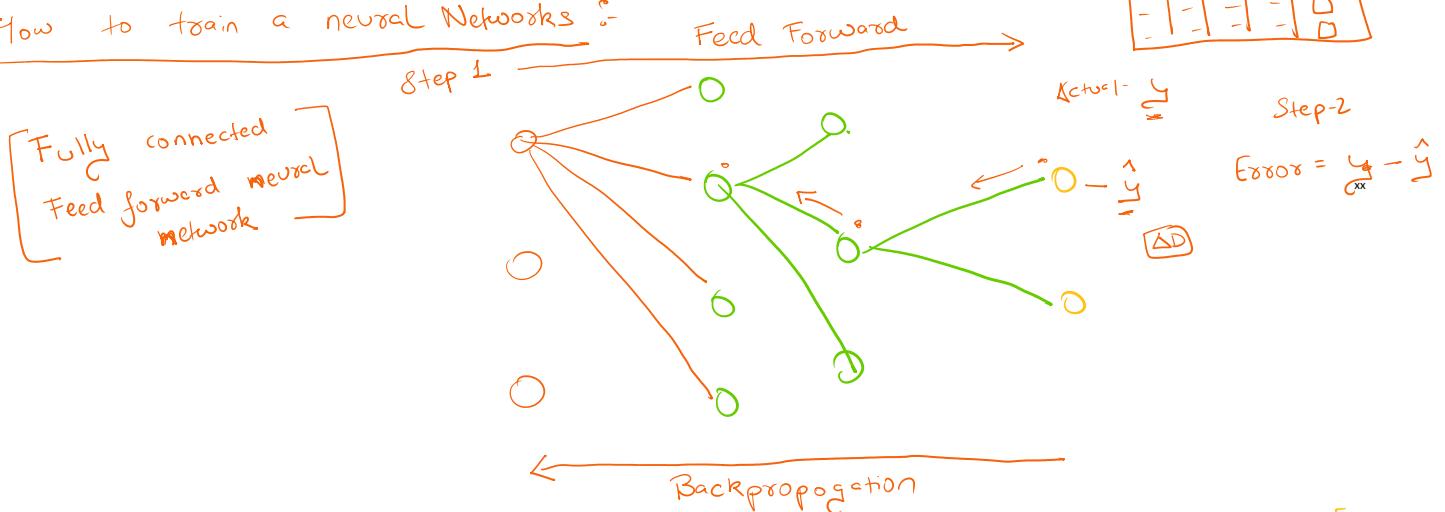
20 cl.



Why we need Deep Learning instead Machine learning ?

- Machine
 - ◦ Structured data
 - ◦ Feature Engg.
 - Not good for high dimensional data
- Deep learning
 - can work unstructured data - audio, text, img. - High Dimensional data
 - High dimensional data
 - Automatic Feature Selection
 - High computation cost

How to train a neural Network :-



Feed Forward

$$\begin{aligned} z &= \sum x_i w_i \\ y &= f(z) = \frac{1}{1+e^{-z}} \end{aligned}$$

$$\begin{aligned} \text{Step-1} \\ a_1 &= (\omega_{13} \times x_1) + (\omega_{23} \times x_2) \\ &= (0.1 \times 0.35) + (0.8 \times 0.8) = 0.755 \end{aligned}$$

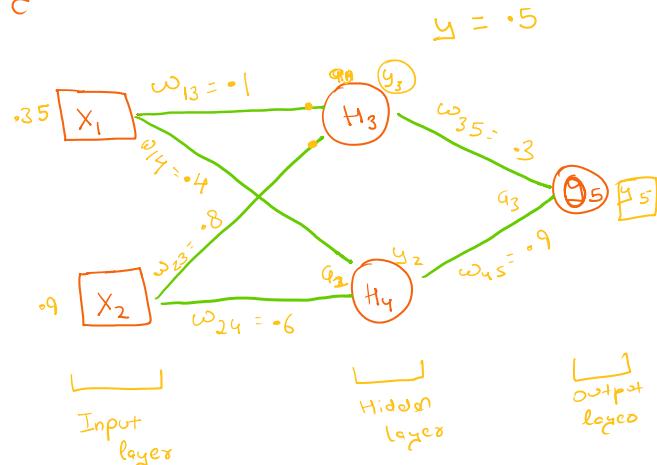
$$y_3 = \frac{1}{1+e^{-0.755}} = 0.68$$

$$a_2 = (0.4 \times 0.35) + (0.9 \times 0.6) = 0.68$$

$$y_4 = \frac{1}{1+e^{-0.68}} = 0.6637$$

$$a_3 = (0.68 \times 0.3) + (0.6637 \times 0.9) = 0.8013$$

$$y_5 = \frac{1}{1+e^{-0.8013}} = 0.6902$$



Step-2

$$\begin{aligned} \text{Error} &= \hat{y} - y \\ 0.5 - 0.69 &= -0.19 \end{aligned}$$

Step -3

Backpropagation

$$\Delta \omega_{ji} = \eta * \delta_j O_i$$

$$\delta_j = O_j(1-O_j)(t_j - O_j) \quad \text{if } j \text{ is output}$$

$$\delta_j = O_j(1-O_j) \geq \delta_k \omega_{kj} \quad \text{if } j \text{ is hidden neuron}$$

η = learning rate

δ_j = error measure of joint

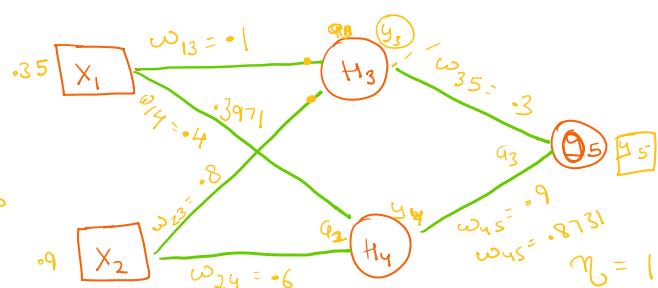
t_j = correct output

O = output of neuron

For output neuron

$$\delta_s = \hat{y}(1-\hat{y})(y - \hat{y})$$

$$= 0.6902(1-0.6902)(0.5 - 0.6902) = -0.0406$$



$$\delta_3 = y_3(1-y_3)(\omega_{35} \times \delta_s)$$

$$= 0.68(1-0.68)(0.3 \times -0.0406) = -0.00265$$

$$\delta_4 = y_4(1-y_4)(\omega_{45} \times \delta_s) =$$

$$0.6637(1-0.6637)(0.9 \times -0.0406) = -0.00815$$

Updating weights

$$\Delta \omega_{di} = \eta \delta_i y_i$$

$$\Delta \omega_{45} = 1 \times -0.0406 \times 0.6637 = -0.0269$$

$$(\text{new}) \omega_{45}^{(1)} = \Delta \omega_{45} + \omega_{45} (\text{old}) = -0.0269 + 0.9 = 0.8731$$

$$\Delta \omega_{14} = \eta \delta_4 y_4$$

$$= 1 \times -0.0082 \times 0.35 = -0.0287$$

$$\omega_{14} = 0.3971$$

$$\begin{aligned} \Delta \omega_{45} &= -0.026 \\ \Delta \omega_{14} &= -0.0082 \\ \Delta \omega_{35} &= -0.0026 \\ \Delta \omega_{24} &= -0.000034 \end{aligned}$$

-2nd Iteration

-0.19

→ Feed Forward

→ Error - 0.15

→ Backpropagation

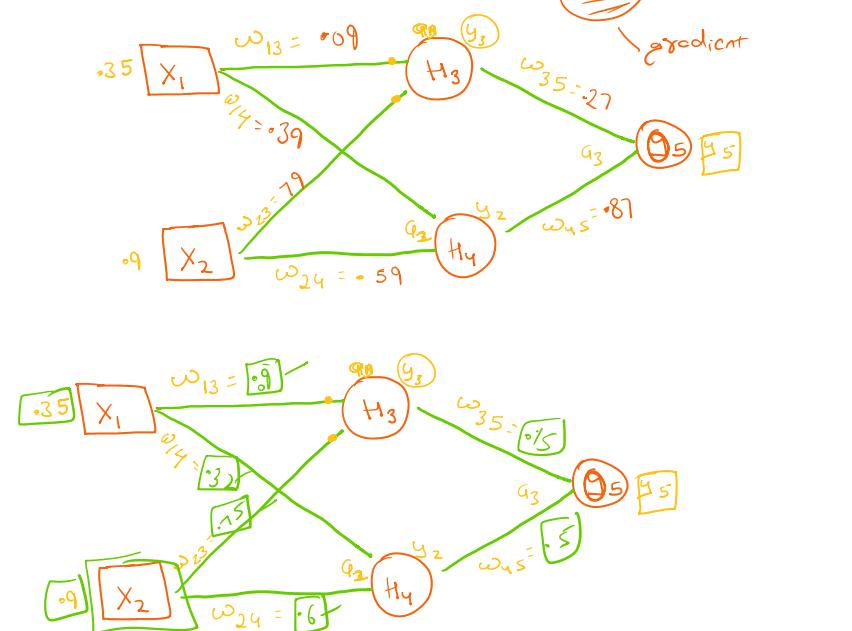
3rd Iteration

Feed Forward

$$\rightarrow E_{\text{Error}} = 0.09 \rightarrow 14.9 \downarrow$$

Y_{th}

$$E_{\text{Error}} = 0.1495$$



Different types of Neural Networks:

- o
- c

Type of Neural Networks

- Fully Connected Neural Network :- (FCNN)
 - ◦ Convolutional Neural Network :- (CNN)
 - ◦ Recurrent Neural Network (RNN)
 - ◦ Long Short term memory : (LSTM)
- ★ Interview question
- GAN - Generative Adversarial Network
 - Autoencoder
- 37 connection

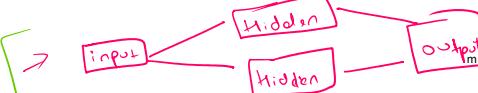
→ Different Architecture of Neural Network :-

◦ one input one output :-

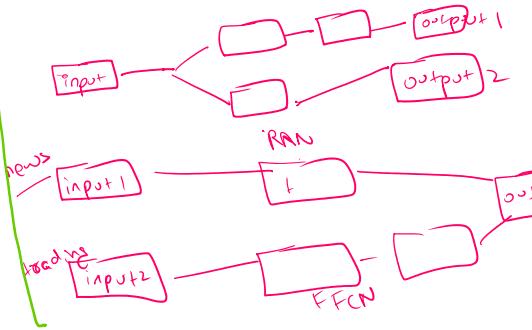


◦

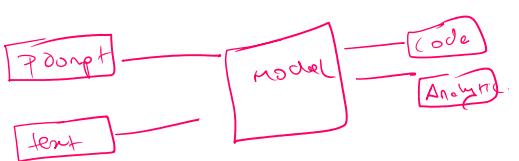
◦ One input multiple output :-



◦ multiple input one output :-



◦ multiple input multiple output :-



$$\Delta w = -0.026$$

$$\Delta w = 1$$

→ Activation function :-

◦ Sigmoid :- $\frac{1}{1 + e^{-z}}$

Range :- (0 to 1)

→ Vanishing Gradient

→ Exploding Gradient

→ use case:-

- binary classification

→ Pros :-

probability based values which are interpretable

→ Cons:-

- Vanishing Gradient

◦ Hyperbolic tan → $\tanh = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Range : $[-1, 1]$

Use Cases:-

- Generally used in hidden layer

Pros:-

- Zero centered output helps optimizer
- less likely to saturate

Cons:-

Vanishing gradient

→ ReLU

$$\text{ReLU}(x) = \max(0, x)$$

Rectified Linear Unit

Range = $[0, \infty)$

Use Cases :-

Default activation for hidden layers

Pros:-

- Computationally efficient
- Mitigates vanishing gradient

Cons:-

Dying ReLU Problem , Not zero centered

4) Leaky ReLU $[0, \infty)$

Parametric ReLU pReLU

ELU - Exponential Linear Unit

$$LR = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

$$\begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

$\alpha =$ is trained

$$\begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

Cons - outliers don't -
- class are mutually exclusive -

→ Head Sigmoid

$$= \min_{\bar{x}} \left(1, \max_{x} (0, 0.2x + 0.5) \right)$$

$$\text{Range} = (0, 1)$$

Use:

faster approach value

Project

computationally faster

Const

less smooth than regular sigmoid

→ soft plus

$$= \log(1 + e^x)$$

Range $\rightarrow (0, \infty)$

User

smooth approx of Relu

Pos

No dead neurons

Coas 81 new

- optimizer

- ## Gradient Descent is (GD)

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \nabla J(\theta)$$

- o Use

small datasets

• Pros :-

Simple & effective on convex problem

o Const:

S: slow convergences & computationally expensive
so better local

θ = weight
 η = learning rate
 $\nabla \delta(\theta)$ = loss function



- Cons: slow convergences & computationally expensive on large dataset

- Stochastic GD : (SGD)

$$\theta = \theta - \eta \nabla f(\theta; x_i, y_i)$$

(x_i, y_i) - individual sample

Use Case :-

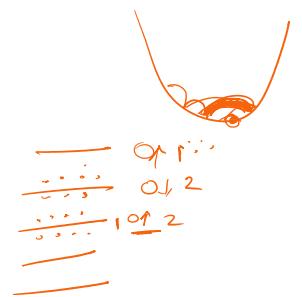
Suitable for large datasets

Pros:

Fast processing & efficient on large dataset

Cons:

Noisy update, may not reach global minima.



- Mini-batch GD :

batch - (mini-batch)

$$\theta = \theta - \eta \nabla f(\theta; \text{batch})$$

Use:

hybrid SGD & GD for large dataset, more stable than SGD

Pros:

- Reduce Noise & variance SGD
- Faster than GD

Cons:

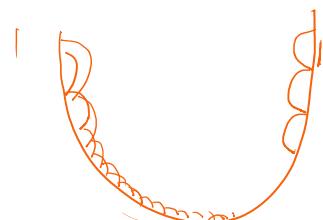
Very large dataset

→ Momentum

$$\vec{v} = \gamma \vec{v} + \eta \nabla f(\theta)$$

$$\theta = \theta - \vec{v}$$

γ = momentum factor =



Use:

useful in training deep network

Pros:

Speeds up the convergences

Pros:

- Speeds the convergence
- Reducing the oscillation

Const:

- Overshoot if momentum is high
- Requires tuning of γ

→ NAG - Nesterov Accelerated Gradient

$$\underline{v} = \underline{v} + \eta \nabla J(\theta - \underline{v})$$

$$\theta = \theta - \underline{v}$$

Use:

it looks ahead before computing the gradient

Pros:

- better than momentum
- prevent overshooting

Const:

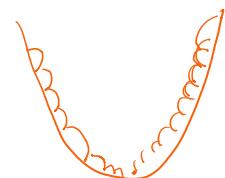
- more complex.

→ Adagrad

$$\theta = \theta - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla J(\theta)$$

G_t = sum of sq of past gradient

ϵ = small constant to avoid zero division.



User:

Suitable for sparse data like NLP, image

Pros:

Adapting learning rate based on freq. of parameter



Const:

Learning rate keeps decaying.

- RMSprop : Root Mean Square propagation

$$G_t = \beta G_t + (1 - \beta)(\nabla J(\theta))^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla J(\theta)$$

$$\beta = \text{decay rate} = 0.9$$

$$\begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \end{array} \quad \begin{array}{c} \uparrow 0.9 \\ \uparrow 0.9 \\ 0.9 \end{array}$$

Pros:

Addressing the adagrad decay prob.

Use :-

Good for non-stationary prob. like in RNN's

Cons :-

Sensitive to β value

- Adam :- Adaptive moment Estimation

First moment estimation

$$m_t = \beta_1 m_t + (1 - \beta_1) \nabla J(\theta)$$

$$\beta_1 = 0.9 - 0.99$$

second moment estimation

$$v_t = \beta_2 v_t + (1 - \beta_2) \nabla^2 J(\theta)$$

Bias correction :-

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

Update :-

$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t}} \cdot \hat{m}_t$$

use case

widely used in almost scenarios.

Pros :-

- Combining adaptive learning & momentum learning
- Efficiently works with large datasets & noisy data.
- Faster convergence

Cons :-

- Multiple tunable parameters
- it can overshoot sometimes

Adamax

$$v_t = \max(\beta_2 v_{t-1}, |\nabla J(\theta)|)$$

$$\theta = \theta - \frac{\eta}{v_t} \cdot m_t$$

Use :-

very large dataset

Pros :-

more stable updates

Cons

Nadam - Nesterov-accelerated adam

$$m_t = \beta_1 m_t + (1 - \beta_1) \nabla J(\theta)$$

$$\theta = \theta - \frac{\eta}{\sqrt{v_t} + \epsilon} \cdot m_t$$

Use + Pros

adaptive learning of adam + fast convergence nature of NAG

Cons

More sensitive to tuning η parameters

Adamax

$$m_t = \beta m_{t-1} + (1-\beta) \nabla J(\theta)$$

$$v_t = \max(\beta_2 v_{t-1}, |\nabla J(\theta)|)$$

$$= \theta - \frac{\eta}{v_t} \cdot m_t$$

Use +

B) Better for model that sparse updates

Pros

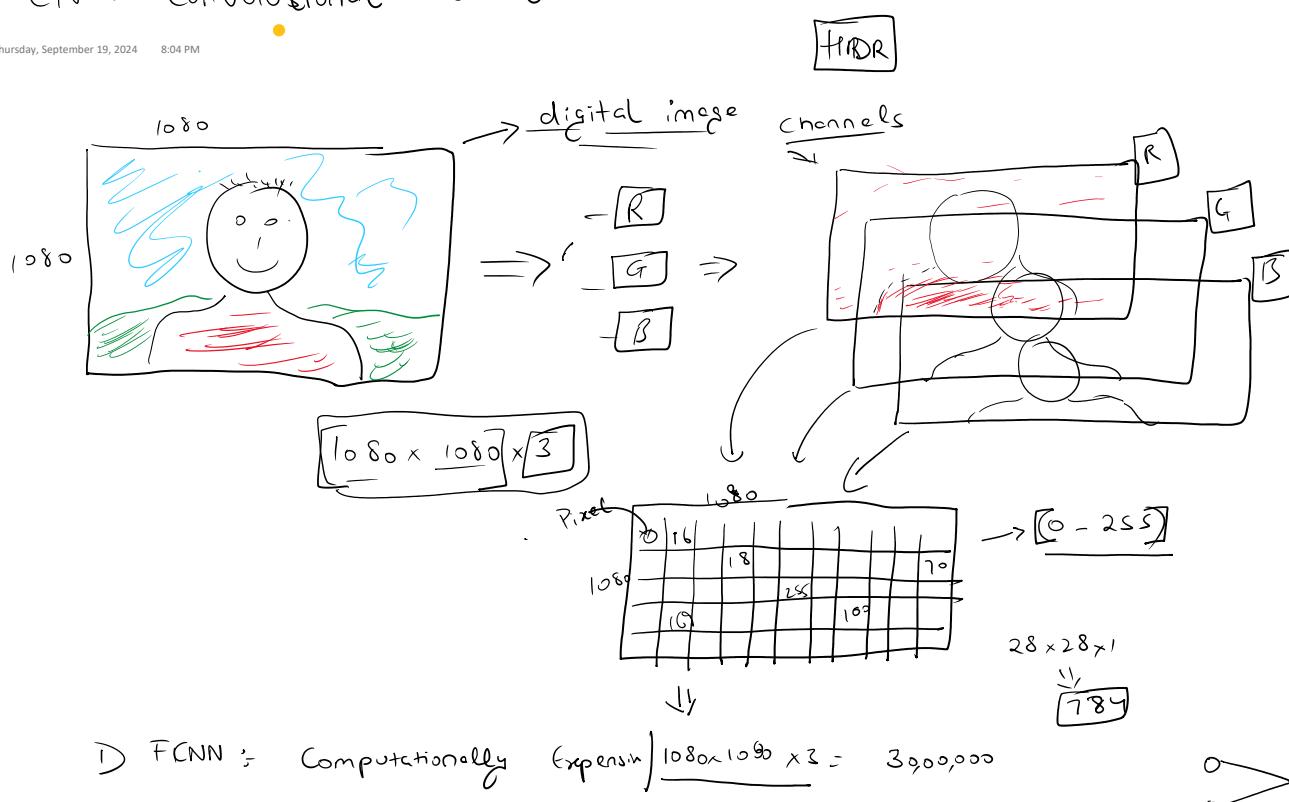
more stable than Adam

Cons

slightly slower than Adam.

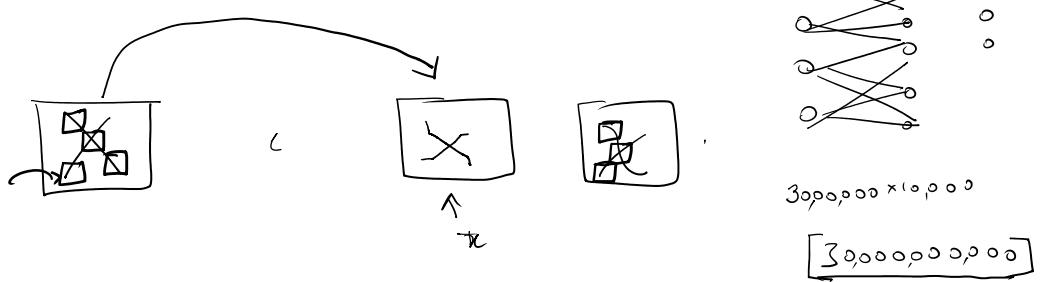
CNN - Convolutional Neural Networks

Thursday, September 19, 2024 8:04 PM



$$\text{D) FCNN} \rightarrow \text{Computationally Expensive} \left| \frac{1080 \times 1080 \times 3}{\text{ }} = 3,900,000 \right.$$

2)



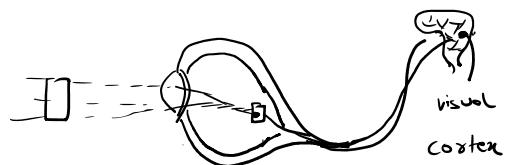
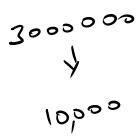
CNN -

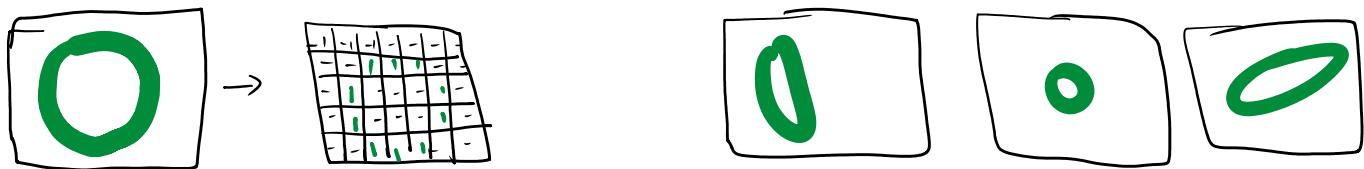
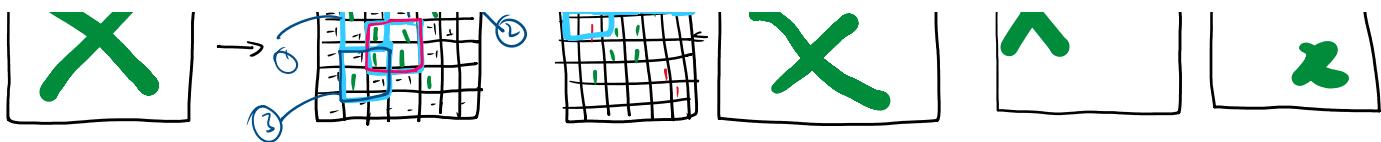
- local area Network \rightarrow computationally less expensive
- automatically learns spatial features in data

How CNN works

- [

 - Convolutional layers
 - Relu layers
 - Pooling
 - FENN —





Convolution

feature \rightarrow

n - size of image (5×5)

f - size of kernel (3×3)

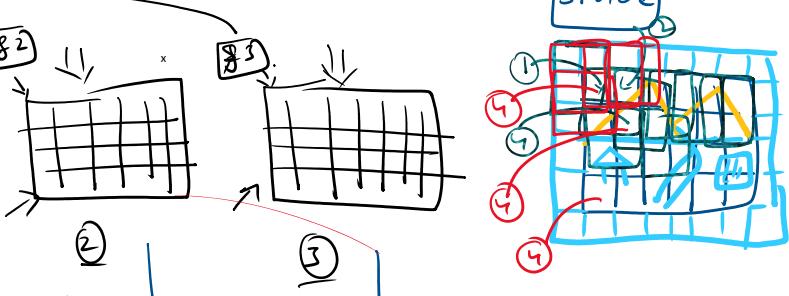
$(n-f+1)$ $(5-3+1) = 3$

$(f-1)$ $(3-1) = 2$

$(n-f+1) \times (f-1)$ $(5-3+1) \times (3-1) = 2 \times 2 = 4$

Convolution layer

$$\begin{array}{l} \text{Input: } \begin{matrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{matrix} \\ \text{Kernel: } \begin{matrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{matrix} \\ \text{Output: } \frac{1+2+3+7+11+15}{4} = 10 \rightarrow \text{Conv}(3, (2,2), 1) \\ \text{Padding: } P_{\text{out}} = P_{\text{in}} \end{array}$$

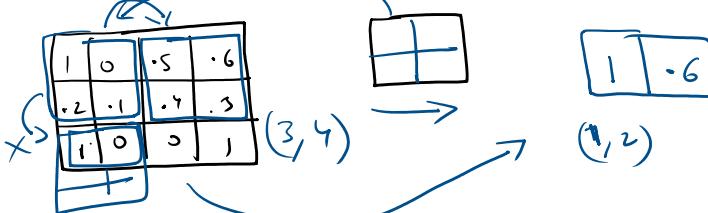


Relu - Making all $\neq 0$ to 2^{32} \downarrow

$(4, 6, 3)$

Relu()

\rightarrow Max Pooling
Pooling layer:
(Dimension red.)



Max - to get sharp edges
Average when we want to retain more info.

.1 .6
.2 .8
.11

Fraction

