MLP VS CNN

MLP is referred as vanilla NN, when they have a single hidden layer.

① Sparse Connectivity

→ In case of CNN, the neuron in a layer will only be connected to a small region of the layer instead of all of the neurons in a fully connected manner in MLP.

② Dimension Information

CNN take into account the dimensional information of an image which gives them an advantage over MLPs for image classification tasks.

③ Better for Complex Images

CNNs perform better than MLP's for complicated images because they can understand spatial relationships between pixels.

④ Parameter Sharing

In CNN, the same filter is used across the entire image during convolution operation. This significantly reduces the no. of parameters that need to be learned.

In MLP, each neuron in a layer is connected to all neurons in the previous layer.

⑤ Reduced memory use

CNNs reduce memory usage by sharing weights among spatial dimensions.

⑥ Faster Computation

CNNs compute faster than MLPs because of the shared weights.

Input (32×32×3)

↓

Conv1 (5×5), s=1, 64 @ 28, 28

↓

Pool1 (MAX) 64 @ 14, 14

↓

Conv2 (f=5, s=1, 128) 128 @ 10×10

↓

Pool2 128 @ 5×5

Pooling :

→ Pooling layers aim to gradually reduce the dimensionality of the representation, and thus further reduce the no. of parameters and the computational complexity of the model.

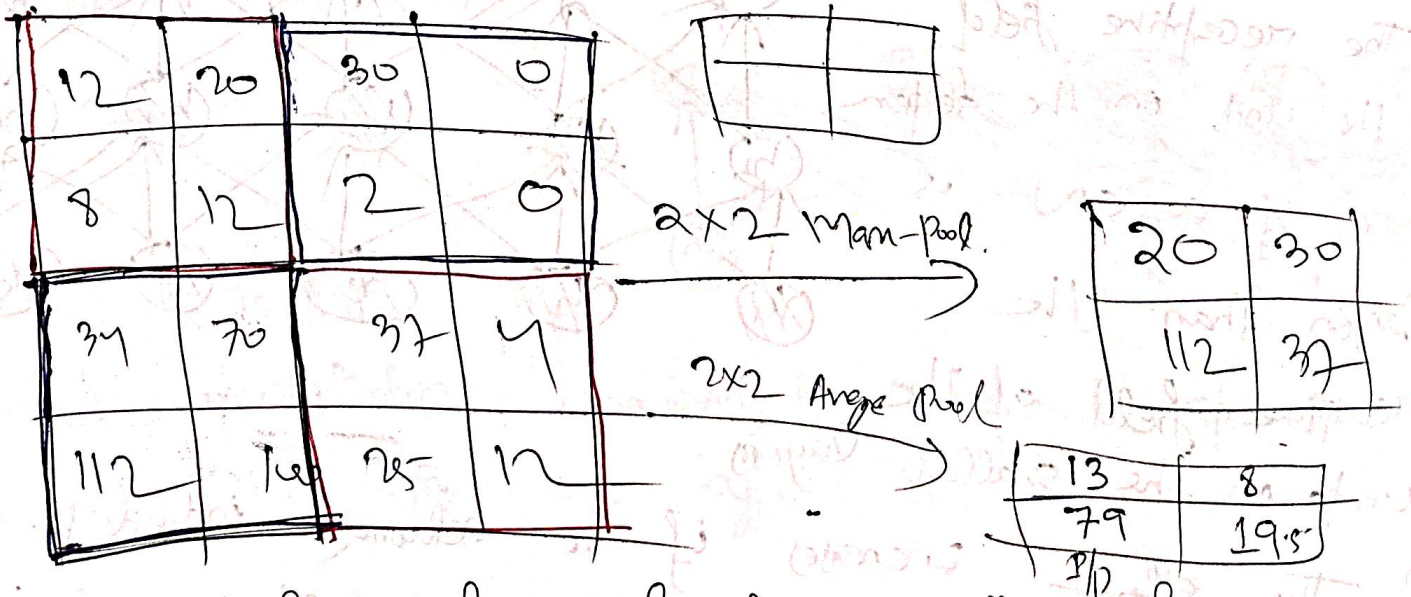
→ A pooling function replaces the o/p of the net at a certain location with a summary statistic of the nearby o/p.

① Max-pooling layers
② Average pooling
③ Global pooling

Pooling layers :

→ Max pooling condenses information in the I/p volume,

→ we take only the maximum entry of each window.



→ Average pooling condenses information in the volume.

→ we take average of entries of each window.

Padding:

Problem with simple convolution layers

→ For a gray scale $(n \times n)$ image and $(f \times f)$ filter/kernel the dimensions of the image resulting from a convolution operation is $(n-f+1) \times (n-f+1)$.

→ The pixel in the corner will only get covers one time but if you take the middle pixel it will get covered more than once basically what does that means is we have more info on that middle pixel so there are 2 main downsides.

* Shrinking outputs
* Loosing information on corners of the image.

To overcome this, we can introduce padding to an image.

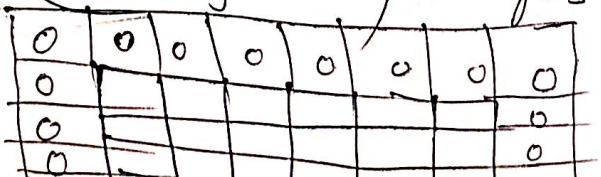
Padding:

⇒ Padding is simply a process of adding layers to zeros to our I/p images so as to avoid the problems.

This prevents shrinking.

→ If p = no. of layers of 0's added to the border of the image, then our $(n \times n)$ image becomes $(n+2p) \times (n+2p)$ image after padding.

→ So, applying convolution operation with $f \times f$ filter outputs $(n+2p-f+1) \times (n+2p-f+1)$ images.



[Zero padding]

Types of Padding :

⇒ Valid Padding :

→ It implies no padding at all. The I/p image is left in its valid/unaltered shape.

→ $[(n \times n) \text{ image}] * [(f \times f) \text{ filter}] \rightarrow [(n-f+1) \times (n-f+1) \text{ image}]$

Here * means convolution operation.

⇒ Same Padding :

→ In this, add 'p' padding layers such that the o/p image has the same dimensions as the I/p image.

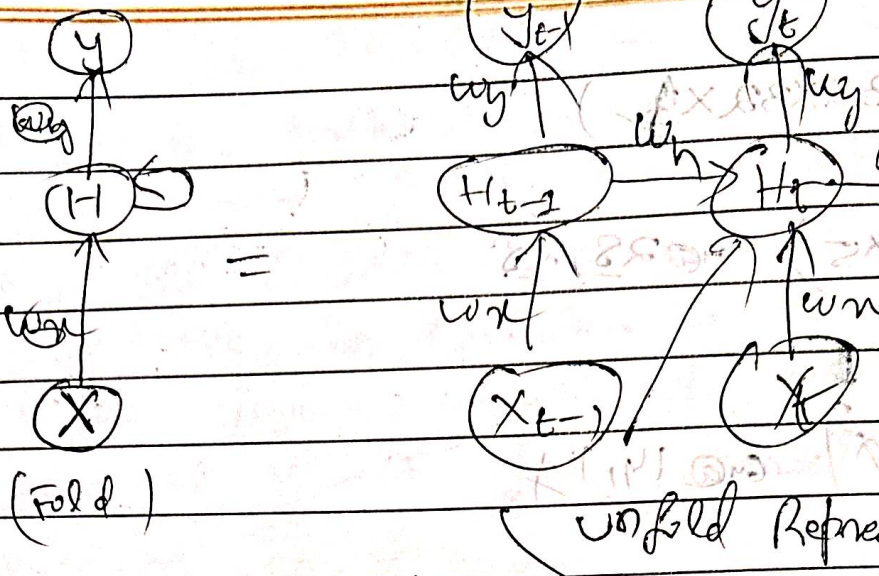
so, $[(n+2p) \times (n+2p) \text{ image}] * [(f \times f) \text{ filter}] \rightarrow [(n \times n) \text{ image}]$

→ In general, $p = (f-1)/2$ (because $n+2p-f+1 = n$)

stride x

Padding

RNN



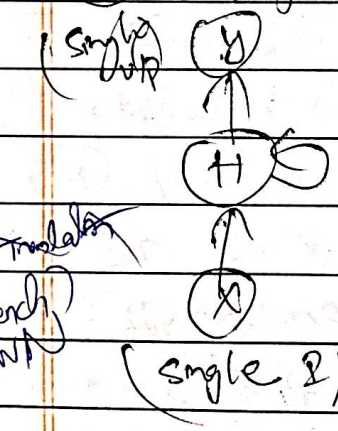
RNN is used to process sequential data and make predictions based on that data.

Types of RNN

→ uses a hidden layer that re
specific information about a seque
→ Has a memory that store all inform

(1) one to one

(2) one to many.



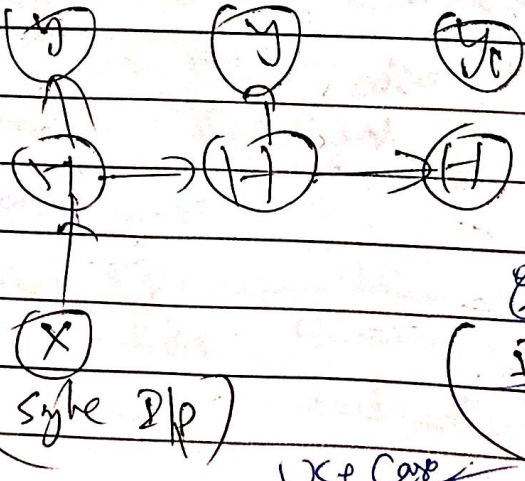
Use case: Image captioning

$$Z_h^{(t)} = [X_t W_x + h_{t-1} W_h] + b_h$$

$$h_t = a(Z_h^{(t)})$$

$$Z_y^{(t)} = (h_t * W_y) + b_y$$

$$y_t = a(Z_y^{(t)})$$



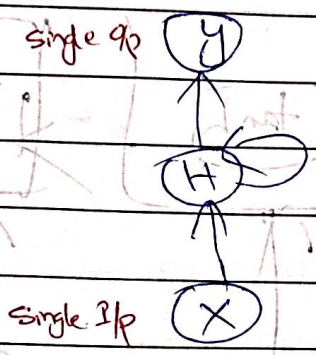
Eg: Image to text

Use Case: Image captioning I/p: Image, Text description of I/p

Types of RNN

1. One to one
2. One to many
3. Many to one
4. Many to many

⇒ One to one RNN :

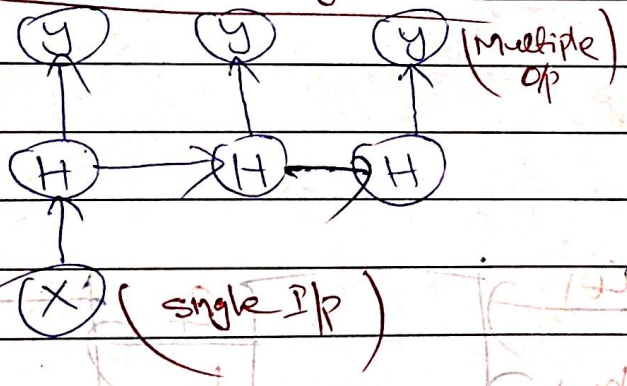


Use case

Vanila NN

Used for regular Time stamp data

⇒ One to many



Use case

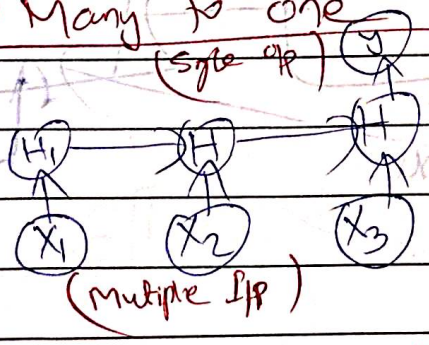
→ Image Captioning
→ Music generation

o/p is sequence

I/p :- Image

o/p :- Text description of I/p image.

⇒ Many to one



Use case

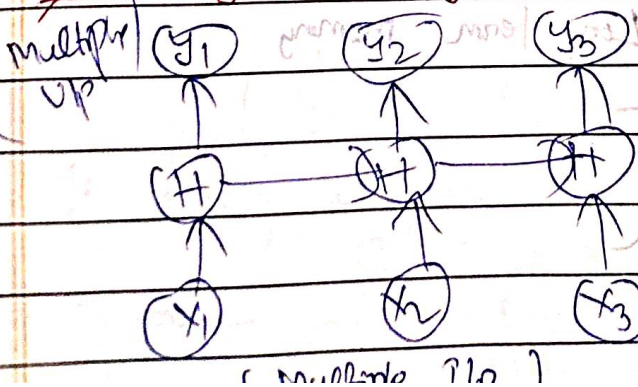
Sequency analysis

Mice eats cheese → English

I/p is sequence

o/p is fixed size vector.

⇒ Many to Many :



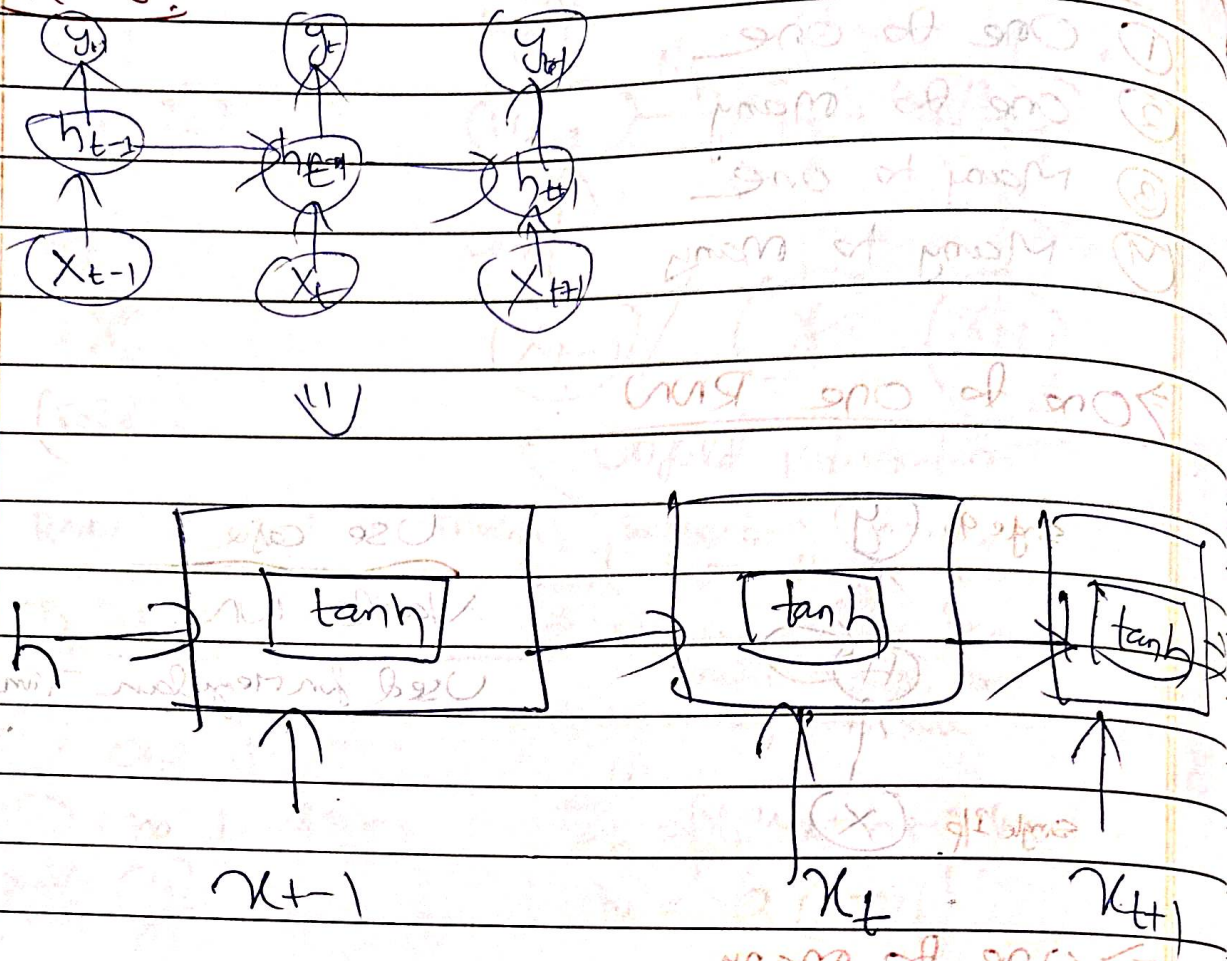
Use case

Langyge Translation

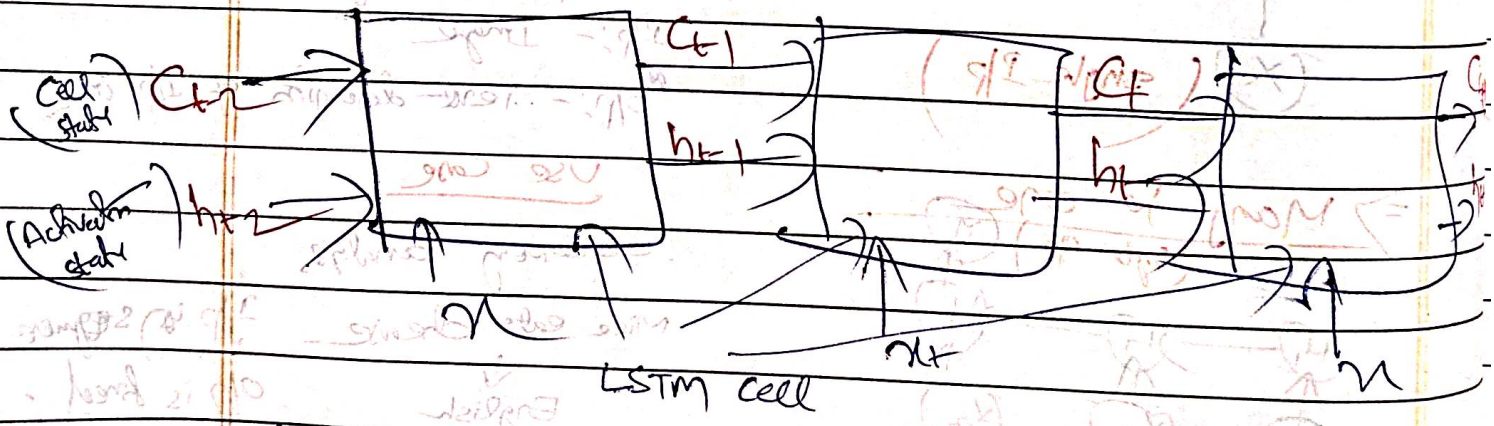
English → French.

Both I/p and o/p is sequence

RNN:

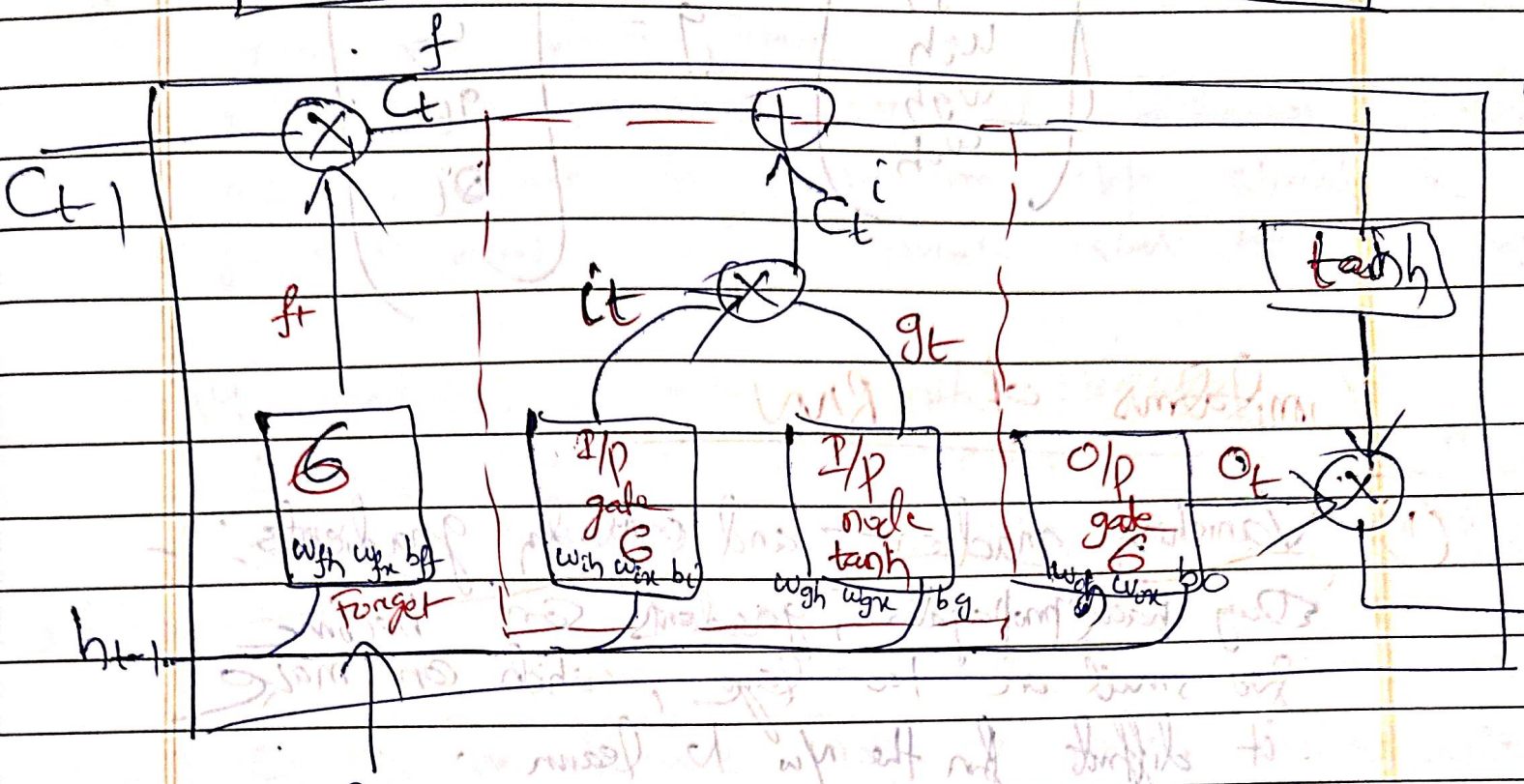


LSTM:



A LSTM cell contains

* A simple RNN cell
* Cell state = Long term memory
* Forget gate
* I/p gate
* O/p gate



$$f_t = G\left[(w_{fh} * h_{t-1}) + (w_{fx} + x_t) + b_f\right]$$

$$C_t^f = C_{t-1} * f_t$$

$$i_t = G\left[(w_{ih} * h_{t-1}) + (w_{ix} * x_t) + b_i\right]$$

$$g_t = \tanh\left[(w_{gh} * h_{t-1}) + (w_{gx} * x_t) + b_g\right]$$

$$C_t^i = i_t * g_t$$

$$\boxed{C_t = C_t^i + C_t^f}$$

$$o_t = G\left[(w_{oh} * h_{t-1}) + (w_{ox} * x_t) + b_o\right]$$

$$h_t = \tanh(c_t) * o_t$$

$$w = \begin{bmatrix} w_f \\ w_i \\ w_g \\ w_o \end{bmatrix}, \quad b = \begin{bmatrix} b_f \\ b_i \\ b_g \\ b_o \end{bmatrix}$$

$$h = \begin{bmatrix} w_{fh} \\ w_{ih} \\ w_{gh} \\ w_{oh} \end{bmatrix}, \quad g = \begin{bmatrix} f_t \\ i_t \\ g_t \\ o_t \end{bmatrix}$$

Limitations of RNN

(1) Vanishing gradient and exploding gradients :- During back propagation, gradients can become too small or too large, which can make it difficult for the n/w to learn.

(2) Difficulty with long sequences :- RNNs can have trouble capturing long term dependencies and often forget earlier information in long sequences.

(3) RNNs can be slow to train compared to feed forward n/w

(4) Difficulty interpreting o/p :- The o/p of an RNN can be difficult to understand especially while dealing with complex I/Ps.

Date
Page No.

Page No.

Advantages of LSTM

(1) Long term dependency learning

LSTMs can capture long-term dependencies and can capture complex patterns in sequential data.

(2) Vanishing gradient problem

→ LSTMs can avoid vanishing gradient problem.
→ LSTMs use gating mechanisms to control the flow of information and gradients, allowing them to learn and retain information over many time steps.

(3) Selective memory retention

→ LSTMs can selectively retain or discard information from the hidden state.
* The forget gate in LSTM cells determines which information from the previous time step should be forgotten and which information should be retained.

(4) Handling variable length sequence

LSTMs can handle variable length I/p sequences by dynamically adjusting their internal state.

(5) Handling noisy or missing data

LSTMs can handle noisy or missing data better than RNNs.

Attribute	CNN	RNN
① Datatype	Spatial data like image, signal	Sequential data like text, audio or time series data
② I/p & O/p	In CNN, the I/p size & O/p size are fixed	Here, I/p and O/p size can vary
③ Architecture	Feed forward NN	Feed forward and feedback
④ Processing	Parallel processing	Sequential processing
⑤ Memory	Limited memory	Retain memory of past I/ps
⑥ Use Cases	Facial recognition, medical analysis, image classification	NLP, sentiment analysis, speech analysis.

Batch Normalization (BN)

→ It is a technique in NN that improves speed and stability by converting data values into a common scale.

→ BN helps to reduce overfitting and improve the generalization ability of the model.

* It is particularly useful for training very deep networks, as it can help to reduce the internal covariate shift that can occur during training.

(Change in the distribution of NN activations due to the change in NN parameters during training)

Benefits of BN

→ improves the effy and reliably of the model.
→ Reduce the internal covariate shift
→ Decreases the drop out rate (amt of info lost from steps)
→ Allow for use of higher learny rates (η) without the risk of divergence.

Q- How non-linearity is introduced in CNN network?

→ Non linearity is introduced into a CNN through activation functions (ReLU) that are applied to the o/p of a neuron.
→ Activation functions are typically applied after each convolution layer and FC layer, but not after pooling layer.

Advantages of MLP

→ Complex modelling :- Handling non-linear relationships & complex patterns.
→ Flexibility :- Adaptable to diff. problem domains and achieves higher accuracy.

Batch Normalization

→ BN normalizes the I/Ps to each layer so that they have zero mean and unit variance during training.
→ It uses the statistics (mean and variance) of each mini batch to scale and shift activations through learnable parameters γ and β.
→ For a batch of activations x:

$$\hat{x} = \frac{x - \mu_{batch}}{\sqrt{\sigma^2_{batch} + \epsilon}}$$

$$y = \gamma \hat{x} + \beta.$$

where, $\mu_{batch}, \sigma^2_{batch}$ are the mean and variance computed over mini batch.
γ & β are learnable scale/shift parameters.

→ The I/p layer has no parameter i.e., 0.

→ The parameters on the conv layer

= (shape of width of filter * shape of height of filter * no. of filters in the prev. layer + 1) * no. of filters

→ Pooling layer has no parameter i.e., 0.

→ Parameters in FC layer = Current layer C (Current layer C * prev layer P) + 1 * C

Sr.	Example	shape	Activation size	# Parameters
1.	Input layer	32×32×3	3072	0
2.	CONV1 (f=5, s=1)	(28×28, 8)	6272	608
			((5×5×3)+1)*8	
3.	POOL1	(14, 14, 8)	1568	0
4.	CONV2 (f=5, s=1)	(10, 10, 16)	1600	3216
			((5×5×8)+1)*16	
5.	POOL2	(5, 5, 16)	400	0
6.	FC3	(120, 1)	120	48120
			(120×400)+(1×120)	
7.	FC4	(84, 1)	84	= 10164
			(84×120+1×84)	
8.	Softmax	(10, 1)	10	= 850
			10×84+1×10	

Advantages of MLP

→ MLP can learn complex patterns :- non-linear relationships in data (single layer perceptron can't learn)

→ Real time training :- MLPs can train models on real time

→ Can handle large amount of data.

→ MLPs can make quick predictions

→ MLPs can achieve accuracy with small samples.

→ MLPs are versatile tools and can be used for variety of tasks (image & speech recognition, classification and regression)

Batch Normalization in CNN

→ BN in CNN is normalizing the activations within each layer, thereby stabilizing the training process & significantly accelerating convergence.

Benefits of BN in DL.

(i) Improved training stability :- BN stabilizes the training process (by) reducing internal covariate shift, making it less sensitive to initial weights and learning rate

(ii) Faster Convergence :
Normalized activations allow the network to converge more quickly, reducing the no. of epochs.

(iii) Regularization :
BN introduces a slight regularization effect by adding noise to the activations, helping to prevent overfitting.

(iv) Enhanced generalization :
Model trained with BN often generalize better to unseen data, leading to improved performance on test sets.

Data Augmentation

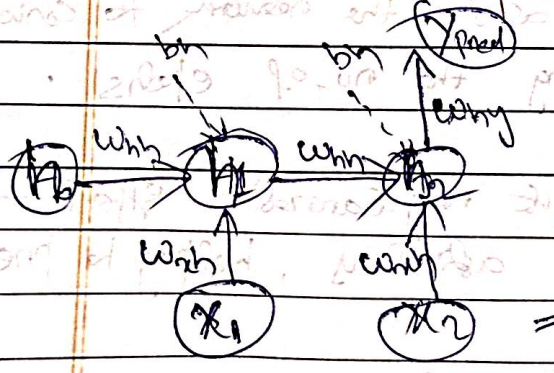
→ It is a technique used to artificially increase the training dataset by applying various transformations to the existing data.

eg:
- Flipping (horizontal / vertical)
- Rotation, zooming, cropping, color jittering (brightness, contrast), adding noise
- GAN (Generative Adversarial Network).

→ Data augmentation prevents overfitting and improve model generalization.

Q- Draw a RNN with I/P(2) → hidden(2) → O/P(2) calculate Y_{pred} for the following given parameter values in the table. with I/p $x_1 = [1, 0]$, $x_2 = [0, 1]$, $h_0 = [0, 0]$

Layer	weights	Bias
I/P to hidden	$\begin{bmatrix} 0.5 & 0.8 \\ -0.3 & 0.2 \end{bmatrix}$	-
Hidden to hidden	$\begin{bmatrix} 0.1 & -0.4 \\ 0.2 & 0.3 \end{bmatrix}$	0.0, 0.1
Hidden to o/p	[1.0, -1.0]	0.0



Step-1 Compute 1st hidden state h_1.

$$a_1 = W_{xh} \cdot x_1 + W_{hh} \cdot h_0 + b_h$$

$$= \begin{bmatrix} 0.5 & 0.8 \\ -0.3 & 0.2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.1 & -0.4 \\ 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 0.1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 \\ -0.3 \end{bmatrix}_{2\times 2} + \begin{bmatrix} 0.0 \\ 0.1 \end{bmatrix}_{2\times 2} = \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix}$$

$h_1 = \tanh(a_1) = [\tanh(0.5), \tanh(-0.2)]$

$\approx [0.462, -0.197]$

Step-2

Compute 2nd hidden state h_2

$$a_2 = W_{xh} \cdot x_2 + W_{hh} \cdot h_1 + b_h$$

$$= \begin{bmatrix} 0.5 & 0.8 \\ -0.3 & 0.2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.1 & -0.4 \\ 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 0.462 \\ -0.197 \end{bmatrix}$$

$$= \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 0.175 \\ 0.0333 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 0.1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.925 \\ 0.3333 \end{bmatrix}$$

$h_2 = [\tanh(a_2)] \approx [\tanh(0.925), \tanh(0.3333)]$

$\approx [0.729, 0.321]$

Step-3

Compute y_{pred}. $W_{hy} = [1.0, -1.0]$

$y_{pred} = W_{hy} \cdot h_2 + b_y$ (0.0)

$= [1.0 \ -1.0] \cdot [0.729 \ 0.321]$

$= 0.729 - 0.321 = 0.408$

Limitations of BN

① Sensitive to batch size

BN estimates mean/variance across mini batch
* with large batch sizes, estimates are stable.
* with small batch sizes (e.g. 1-4) stats became noisy and unreliable, harming training stability and performance.

eg.
In medical imaging or high resolution image models where GPU memory restricts batch size to small values, BN's estimation error increases, slowing learning or degrading accuracy.

② Training vs Inference mismatch

→ During training, BN uses mini-batch statistics but during inference, it uses running (moving average) statistics.
→ If the data distribution shifts between training and testing time, this mismatch can hurt performance.

③ Not Ideal for sequential/variable length data

In recurrent architectures (like RNNs or LSTMs) sequences may have different lengths and temporal dependencies, making it harder to compute meaningful batch statistics at each time step.

Layer Normalization

→ It normalizes all features of each individual sample, rather than across the batch.
→ It computes mean and variance per data point across its features, meaning it is independent of batch size.

How it works ?

→ Suppose a neural layer ops a feature vector for one sample.

$x = [3, 5, 2, 8]$

LayerNorm Computes

* Mean and variance across these 4 features for this single sample.
* Normalizes each activation accordingly.
* Applies learned scale/shift.

Benefits over BatchNorm

→ Works well even with batch size = 1
→ Ideal for RNNs and Transformers where sequence length varies
→ No mismatch between training/inference — same computations always.

Group Normalization

→ GroupNorm breaks channels into groups and normalizes within each group for each sample.

instead of

* Normalizing across batch (BN)
* Normalizing across all features (LayerNorm)

→ GroupNorm splits channels into G groups and computes statistics within each group for each sample. This balances feature locality and batch independence.

e.g

For a layer with 64 channels, and G = 8 groups.

* Each group has 8 channels.
* Compute mean/variance within each group across features in that group.
* Normalize and apply group specific learnable parameters.

Page No.

Benefits of GroupNorm

① Batch size Independence
GroupNorm compute statistics within a single sample rather than across a batch.
So, they work reliably even with small or varying batch sizes.

② Better for Sequences
Ideals for RNNs and transformers where sequence length varies.

③ stable during Inference
The same normalization is applied at training and inference time — no distribution mismatch.

Loss Functions in Auto encoder

① Reconstruction Loss:
This is the primary loss on standard auto encoders. It measures how well the network reconstructs the I/p.

(i) Mean Squared Error (MSE)
Used for continuous I/ps

$$L_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2$$

* It penalizes large reconstruction errors.
* It encourages the o/p to match the I/p closely.

(ii) Binary Cross Entropy (BCE)
It is used for binary or normalized data (e.g. pixel values in [0,1])

$$L_{BCE} = \sum_{i=1} x_i \log(\hat{x}_i) + (1-x_i)\log(1-\hat{x}_i)$$

* suitable when o/ps represent probability

Variants of Loss Functions

(2) Regularized Auto encoders

Some auto encoder variants add additional terms to the basic reconstruction loss.

(i) Sparse Auto encoder loss

Adds a sparsity penalty (e.g. L1) to encourage most latent activations to be zero.

$$L = L_{reconstruction} + \lambda \cdot \text{Sparsity Penalty}$$

This forces the model to pick only the most relevant features.

(ii) Denoising Auto encoder Loss

The I/p is 1st corrupted with noise x' and the model is trained to reconstruct the original x.

$$L = L(x, \hat{x})$$

where $\hat{x} = g(f(x'))$

This improves robustness to noise.