Code: 20CS4701A

PVP-20

PRASAD V POTLURI SIDDHARTHA INSTITUTE OF TECHNOLOGY

(AUTONOMOUS)

IV B. Tech- I Semester- Regular Examinations- DECEMBER 2025

DEEP LEARNING

Duration: 3 Hours **Max. Marks :** 70

UNIT-I

1 A) Explain the following Activation Function

i) Linear ii) Sigmoid iii) Tanh iv) ReLu V) Leaky ReLu

[CO1-L2] [7M]

Ans:

i) Linear Activation Function: The linear activation function, also known as **"no activation,"** or "**identity function",** is one of the **most straightforward activation functions**, where the **output is identical to the input**.

Equation: Linear function has the equation similar to as of a straight line i.e. **y = x**

Range: -inf to +inf

The **linear activation function** typically **adds the weighted sum with bias and passes it as output.**

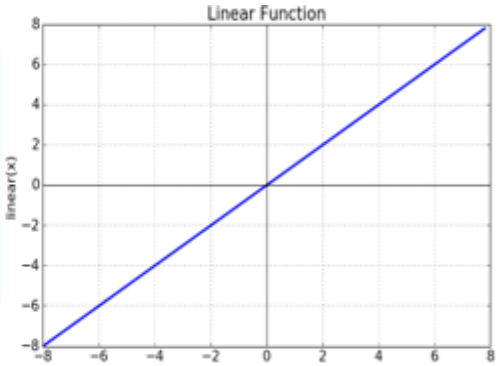
Limitations: It cannot provide Multi Value Outputs. **i.e, it cannot be used in Multi Class Classification Problems.**

Mathematically, it can be defined as:

$$f(x) = x$$

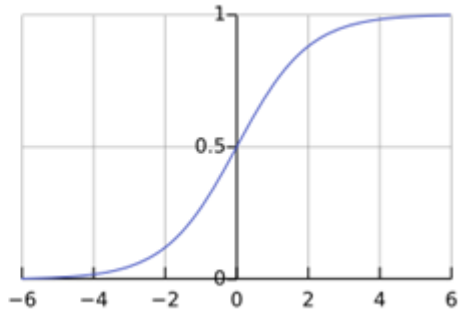
where :

$$x = \sum Input * Weight + Bias(b)$$



ii) Sigmoid Activation Function: This function is a probabilistic approach towards decision making and output range is between 0 and 1. The Sigmoid function, often represented as **(x),** features an **S-shaped curve.** Sigmoid finds the binary classification problems, like logistic regression, where outputs represent probabilities. The sigmoid function, also known as the **squashing function**, takes the **input from the previously hidden layer** and **squeezes it between 0 and 1.** Such as "**yes" or "no", or 1 or 0.** The sigmoid function can **output a probability between 0 and 1,** which can be used to predict the likelihood of a particular class.

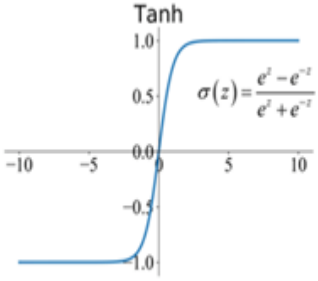
$$f(x) = \frac{1}{1 + e^{-x}}$$

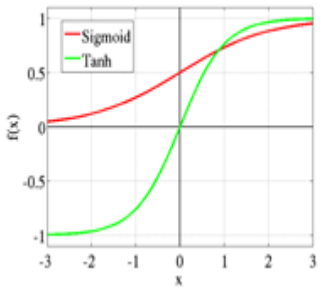


iii) Hyperbolic Tangent Activation Function(Tanh): It overcomes the disadvantage of the sigmoid activation function by *extending the range to include -1 to 1*. It is also having the same **S-shape** with the difference in **output range of -1 to 1.** In Tanh, the larger the input (more positive), the closer the output value will be to 1.0, here as the smaller the input (more negative), the closer the output will be to -1.0. Tanh function can be used when the input data has a range between **negative and positive values.**

Tanh

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

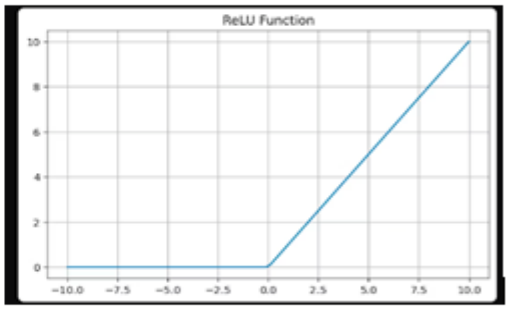




iv) ReLU Activation Function: ReLU is nothing **but Rectified Linear Unit**. ReLU is a **piecewise linear function** that results in the **input directly if it is positive**. Otherwise**, it outputs zero.**

Equation:- It gives an output x if x is positive and 0 otherwise.

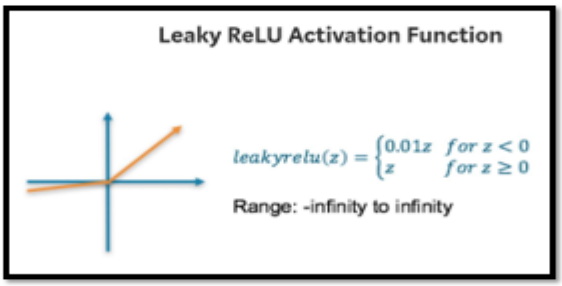
$$f(x) = \max(0, x)$$

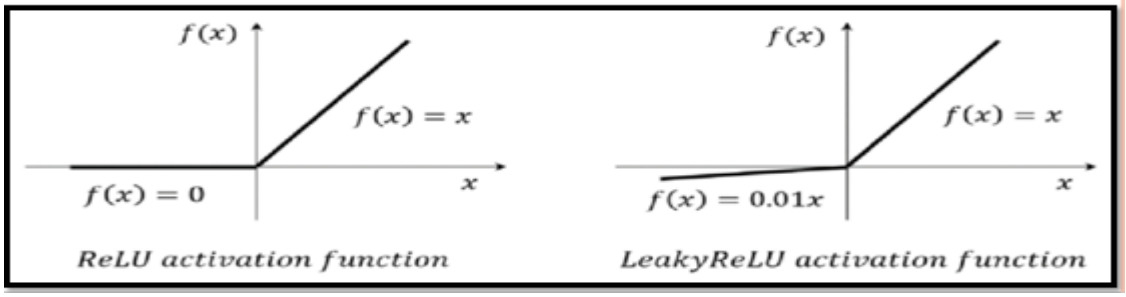


- **ReLU is most commonly** used in **Large Deep Neural Networks** with **lots and lots of hidden layers**. Since ReLU only **requires some threshold operations**, it is computationally **efficient** even though the **dataset and the network are large.** Also, ReLU is a great choice to **avoid the Vanishing Gradient Problem**

- This is the most frequently used activation unit in deep learning. **R(x) = max(0, x) . Thereby, if x < 0, R(x) = 0 and if x ≥ 0, R(x) = x**.

v) **Leaky ReLU Activation Function:** Leaky ReLU function is nothing but an improved version of the ReLU function. As we saw that for the ReLU function, the gradient is 0 for x<0, which would deactivate the neurons in that region. Leaky ReLU is defined to address this problem. Instead of defining the Relu function as 0 for negative values of x, we define it as an extremely small linear component of x.





1B) Write short notes about each Loss Function given below:

i) Mean Absolute Error Loss

ii) Mean squared Log Error Loss

iii) Mean Absolute Percentage Error Loss **[CO1-L2] [7M]**

Ans:

i) **Mean Absolute Error (MAE) Loss :** Mean Absolute Error (MAE) measures the average of the absolute differences between the actual values and the predicted values.

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

- It calculates how far predictions are from actual values without considering the direction of the error.
- All errors are treated equally (linear penalty).
- It is **robust to outliers** compared to squared error losses.

ii) **Mean Squared Log Error (MSLE) Loss:** **Mean Squared Log Error (MSLE)** measures the mean squared difference between the logarithms of actual and predicted values.

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

- Focuses on **relative differences** rather than absolute differences.
- Penalizes **underestimation more than overestimation**.
- Reduces the effect of large target values

iii) **Mean Absolute Percentage Error (MAPE) Loss:** **Mean Absolute Percentage Error (MAPE)** measures the average percentage difference between actual and predicted values.

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

- Expresses error as a **percentage**, making it easy to interpret.
- Sensitive when actual values are **close to zero**.
- Larger penalties for errors on smaller actual values.

(OR)

2 A) Explain Various Loss Functions for Classification and reconstruction

[CO1-L2] [7M]

Ans: Loss functions play a crucial role in machine learning and deep learning models. They quantify the difference between the predicted output of a model and the actual target values. During training, the model tries to minimize this loss to improve performance. Based on the nature of the task, loss functions are broadly classified into Classification Loss Functions and Reconstruction Loss Functions.

i) Loss Function for Classification: Classification loss functions are used when the output variable is categorical (class labels). These loss functions measure how accurately the model predicts class probabilities.

1. Binary Cross-Entropy Loss (Log Loss) : Binary Cross-Entropy loss is used for binary classification problems where the target variable has two classes (0 or 1).

Formula: $\mathbf{L = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]}$

Where:

y = actual class label (0 or 1)

$\hat{y}$ = predicted probability

Example: Email spam detection (Spam / Not Spam)

2. Categorical Cross-Entropy Loss: Categorical Cross-Entropy is used for multi-class classification problems.

Formula: $L = -\sum y_i \log(\hat{y}_i)$

Example: Handwritten digit recognition (0–9)

ii) Loss Function for Reconstruction: Reconstruction loss functions measure how accurately a model can reproduce its input. These are mainly used in auto encoders, image reconstruction, and signal processing.

1. Mean Squared Error (MSE) : Mean Squared Error (MSE) is a commonly used loss function for regression and reconstruction problems. It measures the average of the squared differences between the actual values and the predicted values produced by a model.

Formula: $MSE = (1/n) \sum (y_i - \hat{y}_i)^2$

Example: Image reconstruction in autoencoders.

2. Mean Absolute Error (MAE): Mean Absolute Error (MAE) is a widely used loss function in regression and reconstruction problems. It measures the average of the absolute differences between the actual values and the predicted values.

Formula: $MAE = (1/n) \sum |y_i - \hat{y}_i|$

Example: Signal reconstruction and time-series prediction.

3. Binary Cross-Entropy Loss for Reconstruction : Used when reconstruction output is binary in nature.

Formula: $L = -[x \log(\hat{x}) + (1 - x) \log(1 - \hat{x})]$

Example: Reconstruction of black-and-white images.

---

2 B) Explain Various Hyperparameters tuning methods [CO1-L2][7M]

Ans: **Hyperparameters** are external parameters that control the learning process of a model and are set before training begins. Examples include learning rate (η), regularization parameter (λ), number of hidden layers, batch size, and number of trees. Hyperparameter tuning is the process of selecting the best Hyperparameter values to minimize the **objective (loss) function** and improve model performance.

1. Manual Search Method: In the manual search method, hyperparameters are selected based on experience and repeated experimentation. The model is trained multiple times using different values, and performance is compared.

Objective Function:

$$\theta^* = \arg\min_{\theta \subset H} L(\theta)$$

where H is the set of hyperparameters and L is the loss function.

2. Grid Search Method: Grid search is a systematic approach where a predefined grid of hyperparameter values is created. The model is trained and validated for **every possible combination** using cross-validation.

Cross Validation Score:

$$CV = \frac{1}{k}\sum_{i=1}^{k} Accuracy_i$$

3. Early Stopping Method: Early stopping acts as an implicit tuning method by stopping training when validation loss starts increasing.

Stopping Condition: $L_{val}(t+1) > L_{val}(t)$

UNIT-II

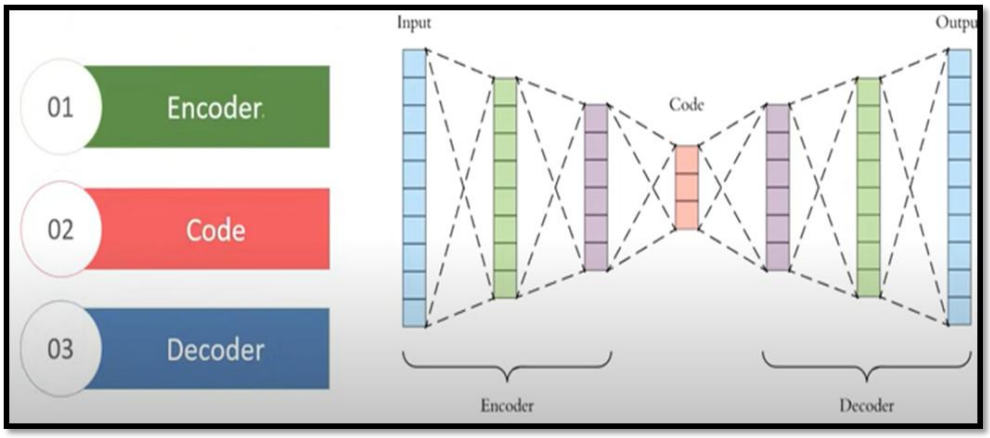
3A) Draw a neat diagram to illustrate an Autoencoder. Explain the purpose of each component in the architecture and how it will work [CO1-L2][7M]

Ans: Autoencoders fall under unsupervised learning algorithms as they learn the compressed representation of the data automatically from the input data without labels.

- In Autoencoders Name,
  - AUTO means =self
  - ENCODERS means- Convert into different form (reduced Dimensions)

Autoencoders (AEs) are a type of neural network architecture that is able to find a *compressed representation* of the input data such as image, video, text, speech, etc. The **aim of an autoencoder** is to **learn a lower-dimensional representation (encoding)** for a **higher-dimensional data**, typically for **dimensionality reduction**, by **training the network to capture the most important parts of the input image.**

Architecture: It has 3 parts



An autoencoder replicates the data from the **input to the output** in an **unsupervised manner** and is therefore sometimes referred to as a **replicator neural network.** A generic way to define an autoencoder using a mathematical notional will **be f(x) = h, where x is the input data and h is the latent variables in the information bottleneck**. This formula denotes the encoder part of the network. The basic goal of an autoencoder is to make the **output x as close to the input x as possible**

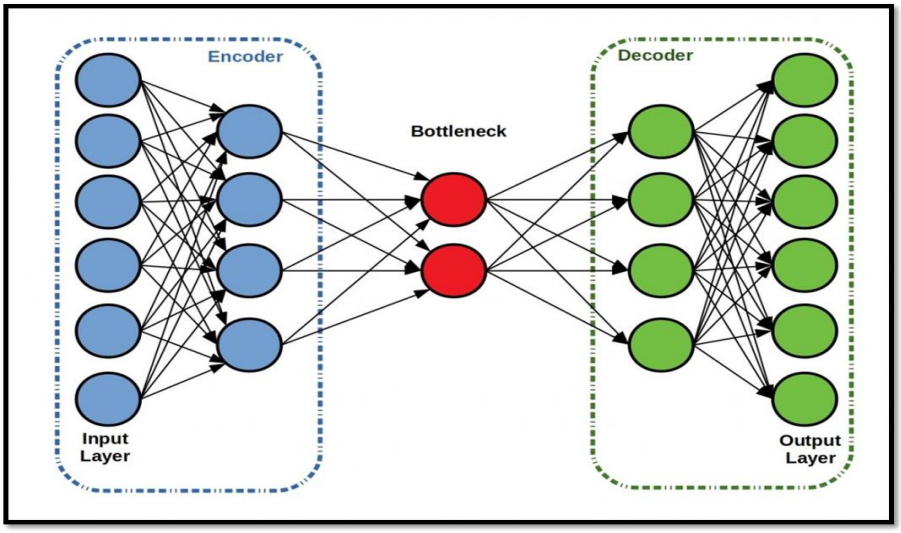
1. **Encoder:** An encoder is a feed forward, fully connected neural network. The encoder is the first part of the autoencoder and is responsible for transforming the input data into a compressed representation. This compressed representation is often referred to as the latent space or bottleneck.

$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

2. **Code:** Code is the representation of **compressed input** which applies **to the decoder**. This part of the network is also refer as **bottleneck**. It **balances two factors** such as **which part of information to be taken** and **which part of information to be discarded.**

3. **Decoder:** This layer decodes the **encoded image back to the original dimension.** The decoded image is a **lossy reconstruction of the original image** and it is **reconstructed from the latent space representation.**

$$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b}')$$



Mathematical Representation: Essentially, we split the network into two segments, the encoder, and the decoder

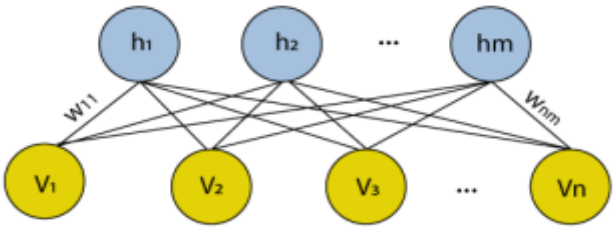
$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$
$$\psi : \mathcal{F} \rightarrow \mathcal{X}$$
$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \, ||X - (\psi \circ \phi)X||^2$$

- The **encoder functio**n, denoted **by ϕ,** maps the **original data X,** to a **latent space F,** which is present at the **bottleneck.**
- The **decoder function**, **denoted by ψ**, maps the **latent space F** at the **bottleneck to the output.**
- The **output,** in this case, is the **same as the input function**. Thus, we are basically **trying to recreate the original image** after some **generalized non-linear compression.**
- The **loss function** can then be written in terms of **these network functions,** and it is this **loss function that we will use to train the neural network** through the standard **back propagation procedure.**

$$\mathcal{L}(x, x') = \|x - x'\|^2 = \|x - \sigma'(W'(\sigma(Wx + b)) + b')\|^2$$

3B) Draw a neat diagram to illustrate Major parts of Restricted Boltzmann Machine and then Explain how each part will function [CO1-L2] [7M]

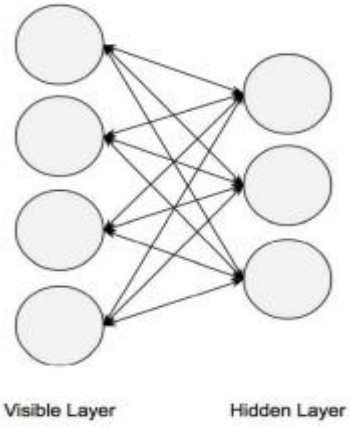
Ans: Restricted Boltzmann Machine is an undirected graphical model that plays a major role in Deep Learning Framework in recent times. It is an algorithm which is useful for dimensionality reduction, classification, regression, collaborative filtering, feature learning, and topic modeling. The "restricted" part of the name "Restricted Boltzmann Machines" means that connections between nodes of the same layer are prohibited (e.g., there are no visible-visible or hidden connections along which signal passes).



Network layout There are five main parts of a basic RBM:

- Visible units
- Hidden units
- Weights
- Visible bias units
- Hidden Bias Units

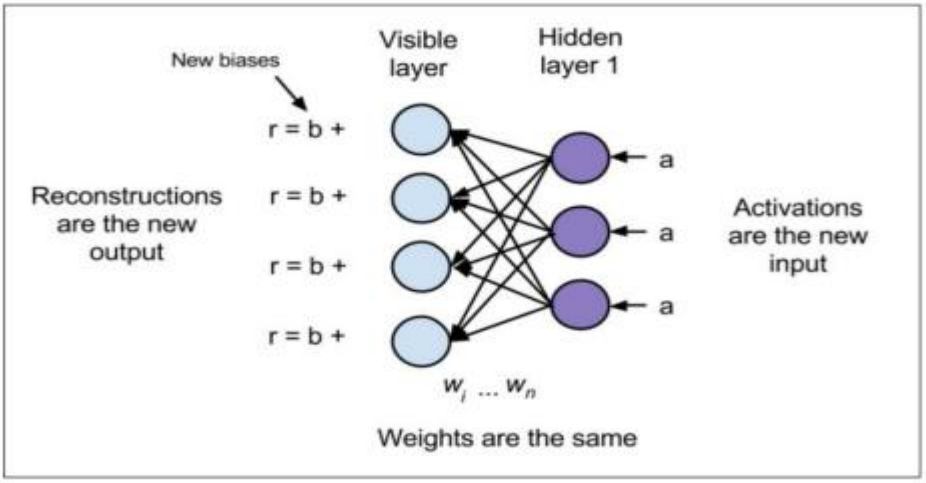
1. Layers: Restricted Boltzmann Machines are shallow, two-layer neural nets that constitute the building blocks of deep-belief networks. The first layer of the RBM is called the visible, or input layer, and the second is the hidden layer. Each circle represents a neuron-like unit called a node. The nodes are connected to each other across layers, but no two nodes of the same layer are linked.



2. Visible and hidden layers: In an RBM, every single node of the input (visible) layer is connected by weights to every single node of the hidden layer, but no two nodes of the same layer are connected. The second layer is known as the "hidden" layer. Hid- den units are feature detectors, learning features from the input data. Each node performs computation based on the input to the node and outputs a result based on a stochastic decision whether or not to transmit data through an activation.

3. Connections and Weights: All connections are visible-hidden; none are visible-visible or hidden-hidden. The edges represent connections along which signals are passed. Loosely speaking, those circles, or nodes, act like human neurons. They make decisions about whether to be on or off through acts of computation. "On" means that they pass a signal further through the net; "off" means that they don't. Usually, being "on" means the data passing through the node is valuable; it contains information that will help the network make a decision. Being "off" means the net- work thinks that particular input is irrelevant noise.

1. **Biases:** There is a set of bias weights ("parameters") connecting the bias unit for each layer to every unit in the layer. Bias nodes help the network better triage and model cases in which an input node is always on or always off.
2. **Training:** The technique known as pre training using RBMs means teaching it to reconstruct the original data from a limited sample of that data. That is, given a chin, a trained net- work could approximate (or "reconstruct") a face. RBMs learn to reconstruct the input dataset.
3. **Reconstruction:** Deep neural networks with unsupervised pretraining (RBMs, autoencoders) perform feature engineering from unlabeled data through reconstruction.



Example: We can visually explain reconstruction in RBMs by looking at the MNIST dataset. The MNIST dataset is a collection of images representing the handwritten numerals 0 through 9.



The training dataset in MNIST has 60,000 records and the test dataset has 10,000 records. If we use a RBM to learn the MNIST dataset, we can sample from the trained network to see how well it can reconstruct the digits. If the training data has a normal distribution, most of them cluster around a central mean, or average, and become scarcer the further you stray from that average.

(OR)

4 A) Draw a neat diagram to illustrate Variational Auto encoder. Explain the purpose of each component in that architecture and how it will work [CO2-L2] [7M]

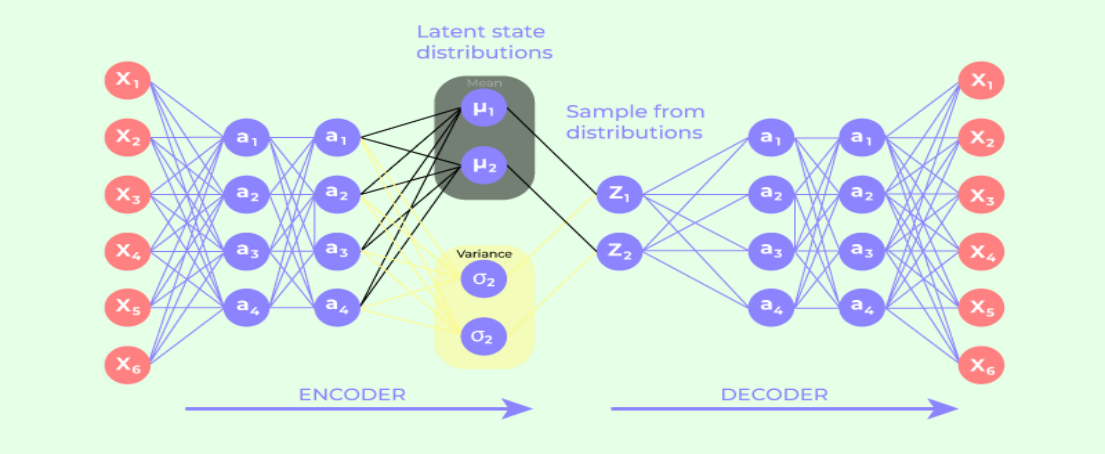
Ans: Variational Autoencoders (VAEs) are generative models in deep learning that extend the concept of traditional autoencoders by incorporating probabilistic techniques. They aim to learn the underlying probability distribution of the input data, enabling the generation of new, similar data samples. The VAE is similar to **compression and Denoising autoencoders** in that they are all trained in an **unsupervised manner** to **reconstruct inputs.**

Architecture: We can fix these issues of Autoencoders, we can make two changes to the autoencoder.

- They memorize data rather than generalize.
- They lack a structured latent space, making generation of new data difficult.

- To overcome these issues, we make two key changes to the autoencoder model, resulting in a new architecture called a Variational Autoencoder (VAE).

In a compression/denoising autoencoder, activations are mapped to activations throughout the layers, as in a standard neural network; comparatively, a **VAE uses a probabilistic approach for the forward pass**. First, **we map each point x in our dataset to a low-dimensional vector of means μ(x) and variances σ(x)2 for a diagonal multivariate Gaussian distribution.**



Components:

1. Encoder (Inference Network):

The encoder network takes raw input data and transforms it into a probability distribution within the latent space. Maps input x to a distribution over latent variables z:

$$q_\phi(z|x) \sim \mathcal{N}(\mu(x), \sigma^2(x))$$

Outputs mean (μ) and log-variance (log σ2).

2. Latent Space: A latent vector (z) is sampled from this learned distribution using the parameterization trick. The encoder doesn't produce a fixed z, but a distribution:

- Sample $z \sim \mathcal{N}(\mu, \sigma^2)$
- Use **reparameterization trick**:

$$z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$

Sampling means picking random values from a probability distribution.

- **For example:** From a coin flip distribution (Bernoulli), sampling gives either 0 (tails) or 1 (heads). From a normal distribution N(0,1), sampling might give values like 0.1, -0.5, 1.3, etc.

3. Decoder (Generative Network): The decoder takes the sampled latent vector (z) and reconstructs the original data. **In reconstruction Phase**, instead of directly sampling from **z_mean & Z_log_var** , we use re-parameterized trick by the use of **Back propagation Algorithm.**In our example, you **approximate z using the decoder parameters** and **another parameter ε** as follows:

$$z_i = \mu_i + \sigma_i.\epsilon$$
$$\epsilon \sim \mathcal{N}(0,1)$$

We start by **Sampling epsilon** from a standard normal distribution. where **μ and σ represent** the **mean and Variance of a Gaussian distribution** respectively. They can be **derived from the decoder output.**

---

4 B) Define GAN? Explain each component with neat Labeled Diagram [CO2-L4] [7M]

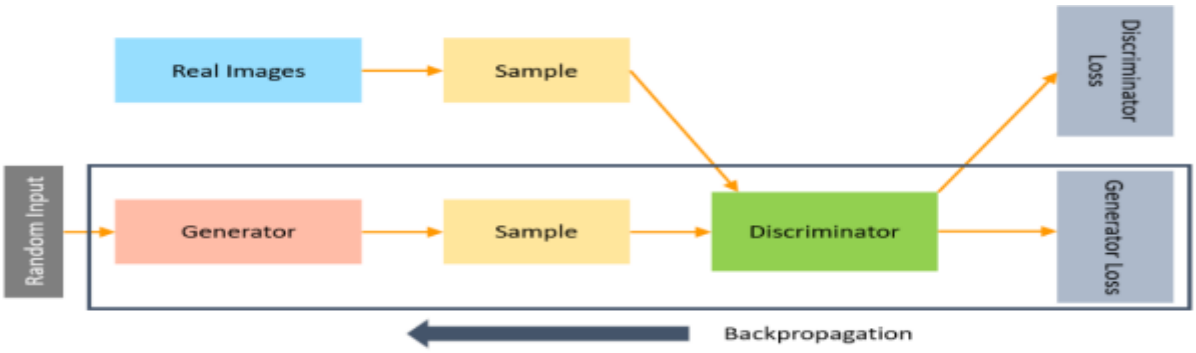
Ans: **Generative Adversarial Networks**: GANs are basically made up of a system of two competing neural network models which compete with each other and are able to analyze, capture and copy the variations within a dataset. To understand the term GAN let's break it into separate three parts.

- **Generative –** To learn a generative model, which describes how data is generated in terms of a probabilistic model. In simple words, it explains how data is generated visually.
- **Adversarial –** The training of the model is done in an adversarial setting.
- **Networks –** use deep neural networks for training purposes. GAN consists of 2 models that automatically discover and learn the patterns in input data.
- **The two models are known as Generator and Discriminator.** They compete with each other to **scrutinize, capture, and replicate the variations within a dataset.**

What is a Generator? A Generator in GANs is a neural network that creates fake data to be trained on the discriminator. It learns to generate plausible data. The generated examples/instances become negative training examples for the discriminator. It takes a fixed-length random vector carrying noise as input and generates a sample.

The main aim of the Generator is to make the **discriminator classify its output as real.** The part of the GAN that trains the Generator includes:

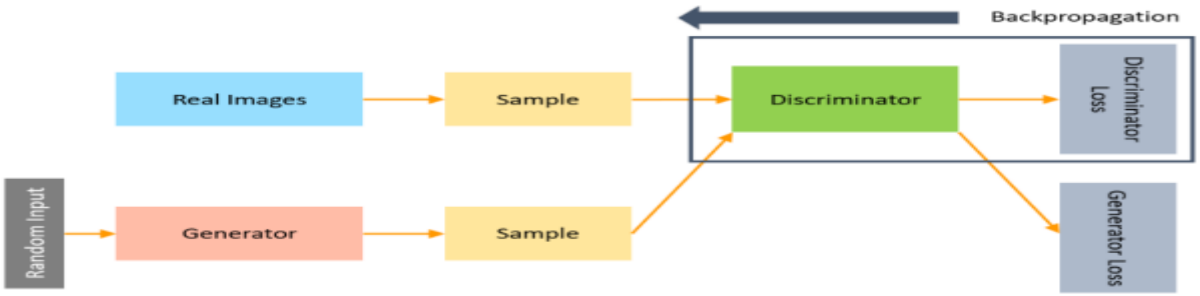
- noisy input vector
- generator network, which transforms the random input into a data instance
- discriminator network, which classifies the generated data
- generator loss, which penalizes the Generator for failing to dolt the discriminator



What is a Discriminator? The Discriminator is a neural network that identifies real data from the fake data created by the Generator. The discriminator's training data comes from **different two sources**:

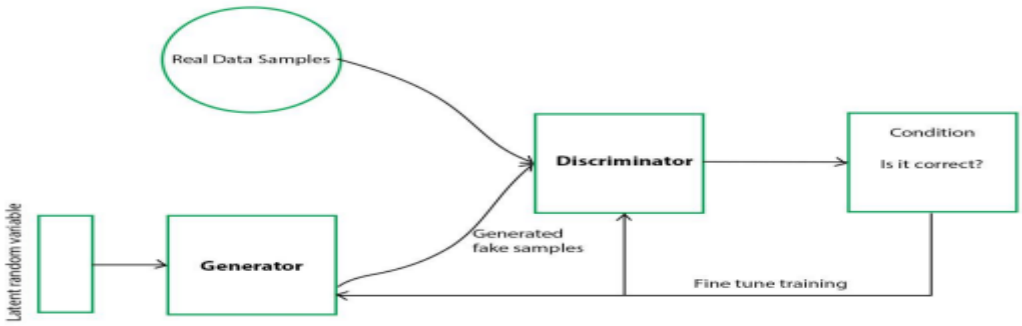
1. The real data instances, such as real pictures of birds, humans, currency notes, etc., are used by the Discriminator as positive samples during training.

2. The fake data instances created by the Generator are used as negative examples during the training process.



While training the discriminator, it connects to two loss functions. During discriminator training, the discriminator ignores the generator loss and just uses the discriminator loss. In the process of training the discriminator, the discriminator classifies both real data and fake data from the generator. The discriminator loss penalizes the discriminator for misclassifying a real data instance as fake or a fake data instance as real. The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

Working of GAN:



In GANs, there is a generator and a discriminator. The Generator generates fake samples of data (be it an image, audio, etc.) and tries to fool the Discriminator.

- The Discriminator, on the other hand, tries to distinguish between the real and fake samples. The Generator and the Discriminator are both Neural Networks and they both run in competition with each other in the training phase.
- The GANs are formulated as a minimax game, where the Discriminator is trying to minimize its reward V(D, G) and the Generator is trying to minimize the Discriminator's reward or in other words, maximize its loss. It can be mathematically described by the formula below:

It can be mathematically described by the formula below:

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))]$$

where,
G = Generator
D = Discriminator
Pdata(x) = distribution of real data
P(z) = distribution of generator
x = sample from Pdata(x)
z = sample from P(z)
D(x) = Discriminator network
G(z) = Generator network

UNIT-III

5 A) Illustrate the Convolutional Operation with a neat diagram. Also illustrate the different effects of convolutional or matrix multiplication operations between the input image and kernel map with neat diagrams [CO3-L3] [7M]

Ans: **1. The Convolution Layer:** Convolution operation focuses on extracting/preserving important features from the input. It means the network will learn specific patterns within the picture and will be able to recognize it everywhere in the picture. Convolution operation allows the network to detect horizontal and vertical edges of an image and then based on those edges build high-level features in the following layers of neural network.

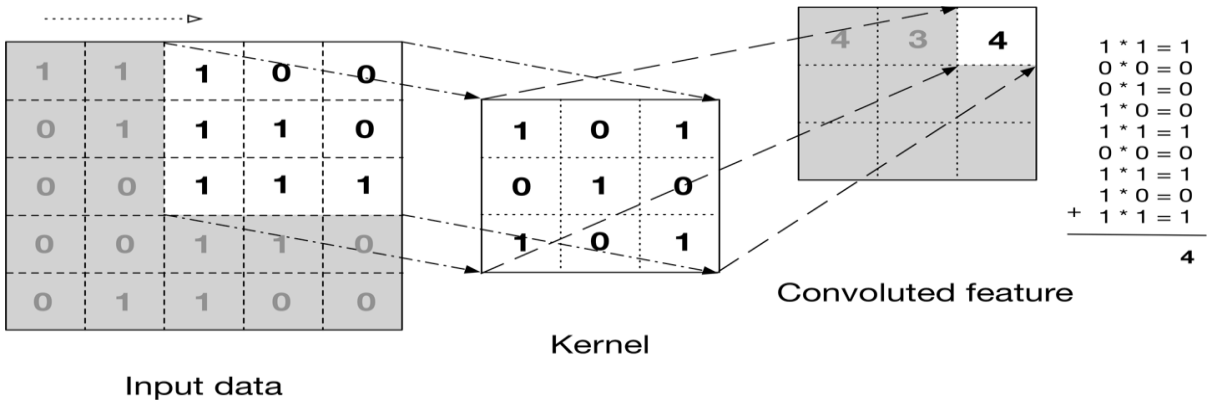
Convolution operation uses **three parameters:**

- **Input image,**
- **Feature detector and**
- **Feature map.**

- It uses the mathematical operation which takes two inputs such as image matrix and a kernel or filter. These filters slide over the input data, performing element-wise multiplications and summations to produce feature maps. Each filter specializes in detecting a specific pattern or feature within the input.
- Input matrix can be pixel values of a gray scale image whereas a filter is a relatively small matrix that detects edges by darkening areas of input image .There can be different types of filters depending upon what type of features we want to detect.

  Eg: Vertical, Horizontal, or Diagonal, etc.
- Input image is converted into binary 1 and 0. In this Image, how CNN recognizes an image: The convolution operation shown in the figure is known as the feature detector of a CNN.
- A feature detector, also known as a kernel or a filter, will traverse over the image's receptive fields, checking for the presence of the feature.

- For simplicity, let's stick with grayscale images as we try to understand how CNNs work**.**



- The **given image shows what a convolution is.** We take a **filter/kernel(3×3 matrix)** and **apply it to the input image** to get the **convolved feature**. This convolved feature is passed on to the next layer.

- Generally, an image can be considered as **a matrix** whose elements are numbers **between 0 and 255.** The size of image matrix is:

image height * image width * number of image channels.

- A **gray scale image has 1 channel**, where a **colour image has 3 channels.**Let's pretend the input is a **color picture**, which is made up of a **3D matrix of pixels**. This implies the **input will have three dimensions**: **height, width, and depth**, which match the **RGB color space of a picture.** Here we try to decompose RGB to a multidimensional layer and apply a filter to each layer. The dimension of the image matrix **is h×w×d.** The dimension of the filter is **fh×fw×d.**

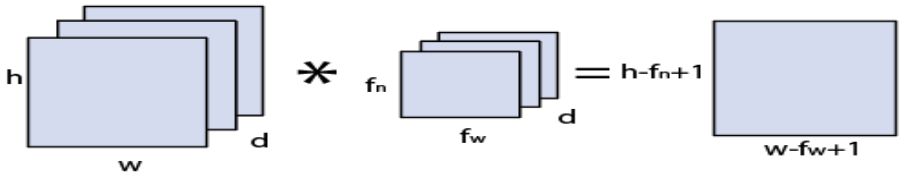
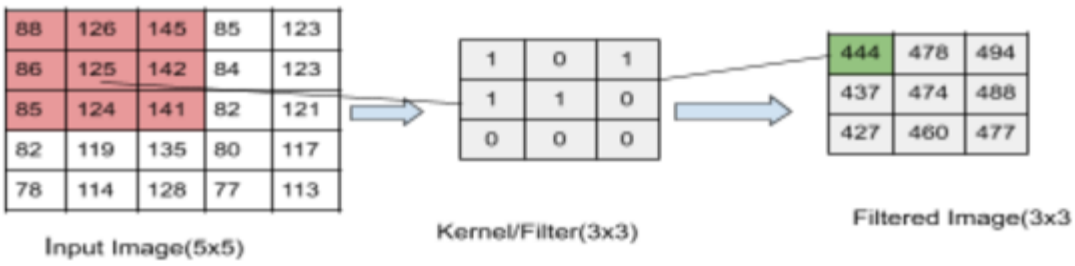


Image matrix multiplies kernl or filter matrix

Example:



(88*1 + 126*0 + 145*1) + (86*1 + 125*1 + 142*0) +
(85*0 + 124*0 + 141*0)
= (88 + 145) + (86 + 125)
= 233 + 211
= 444

Local Receptive Field

Filtered pixel value

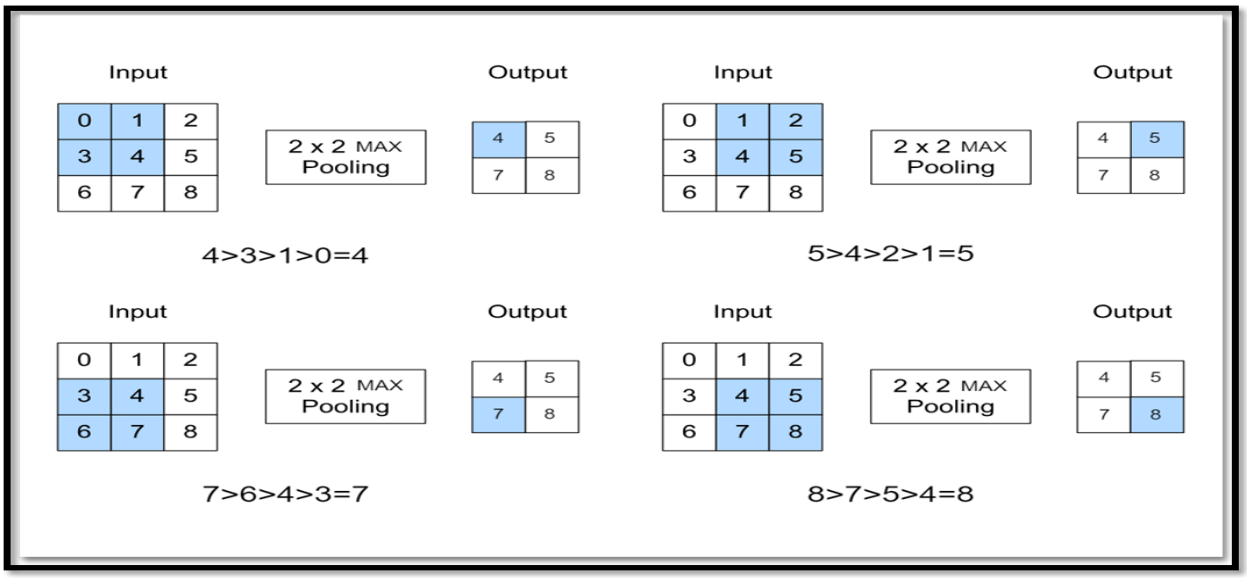
5 B) Demonstrate various pooling operations in Convolutional Neural Network. Also explain the need of these pooling operations in CNN with examples [CO3-L3] [7M]

Ans: Pooling layers are one of the building blocks of Convolutional Neural Networks. Where Convolutional layers extract features from images, Pooling layers consolidate the features learned by CNNs.

Types of Pooling Layer:

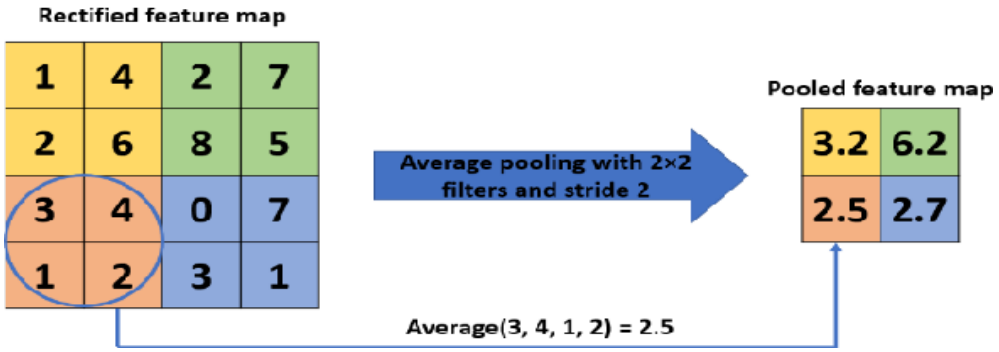
- 1. **Max Pooling**
- **2. Average Pooling**
- **3. Min Pooling**

1. **Max Pooling**: Finally, we can say it **selects the maximum valued element** from the **region captured by the filter** in **any feature map.** This works by selecting the **maximum value from every pool.** Max Pooling **retains the most prominent features** of the **feature map,** and the **returned image is sharper than the original image.** i.e, if we talk about **image processing** the **max-pooling** helps to **extract the sharpest features on the image.**. Max Pooling is more suitable for images with a **dark background** as it will select **brighter pixels in a region of the input image. Kernel/filter is the size of a matrix** that is **applied over the complete input data.** As the **filter moves across the input**, it **selects the pixel** with the **maximum value to send to the output array**.

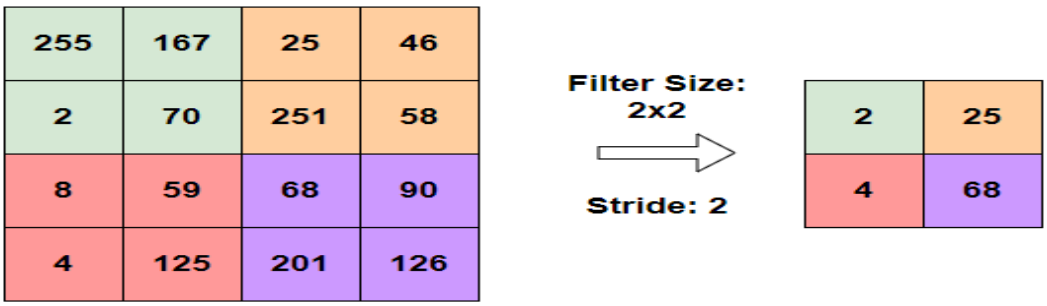


2. **Average Pooling**: As the filter moves across the input, it calculates the average value within the receptive field to send to the output array. This pooling layer works by getting the average of the pool. Average pooling retains the average values of features of the feature map. Average pooling method smooth's out the image and hence the sharp features may not be identified when this pooling method is used.

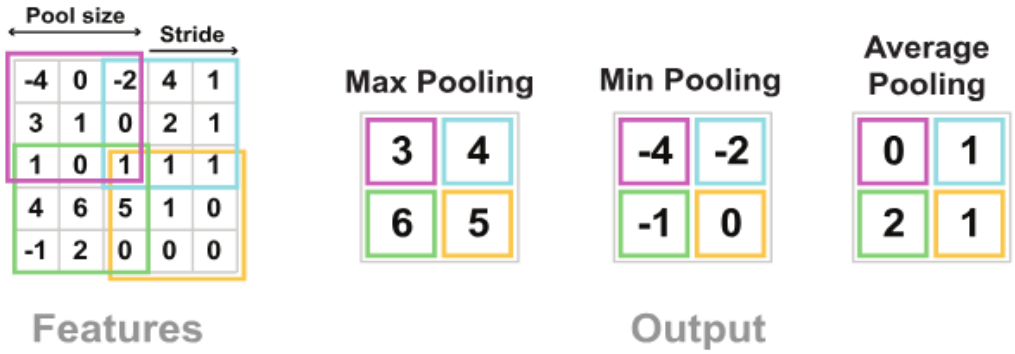
With average pooling, the **harsh edges of a picture** are **smoothened,** and this type of pooling layer can used **when harsh edges can be ignored.**



3. **Min Pooling**: In **this type of pooling,** the **summary of the features in a region** is represented by the **minimum value in that region.** It is mostly used when the image has **a light background** .



Max Pooling, Min Pooling, Average Pooling:



Need for Pooling Operations

1. Dimensionality Reduction: Pooling reduces the size of feature maps, which decreases the number of parameters and computations in the network. This leads to faster training and reduced memory usage.

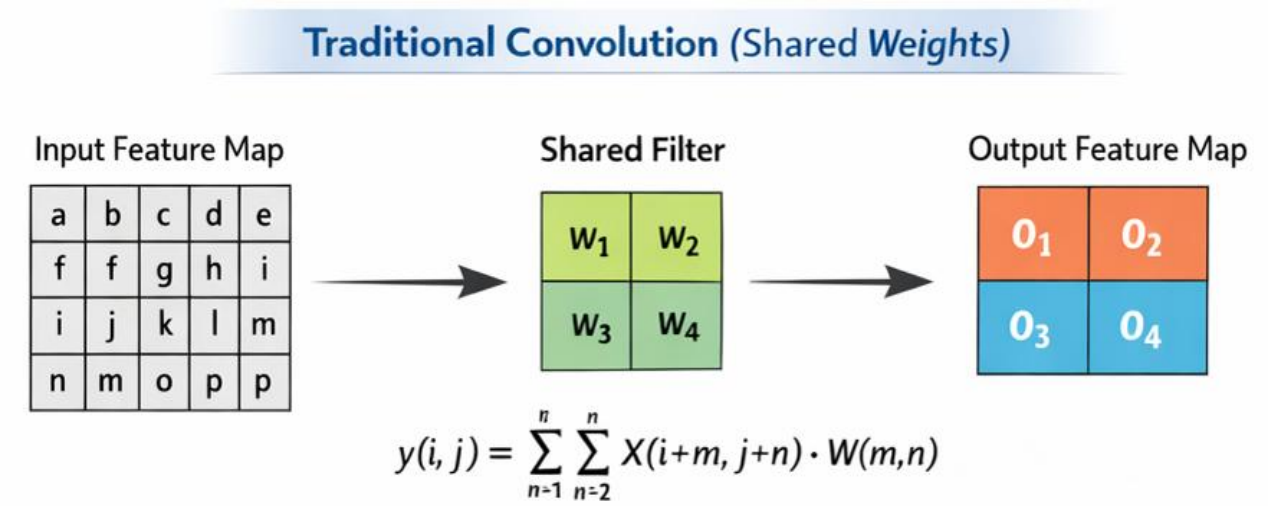
Example: A 28×28 feature map after 2×2 pooling becomes 14×14.

2. Control Overfitting: By reducing the number of parameters, pooling lowers the chance of overfitting, especially when training data is limited. It acts as a form of regularization.

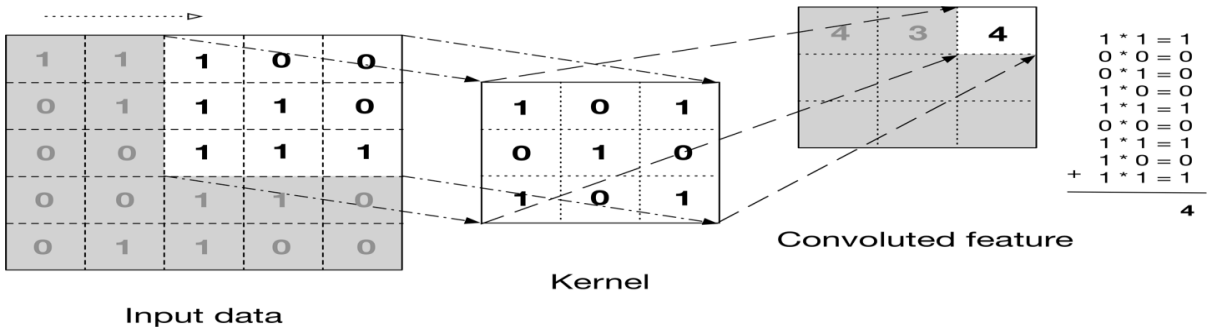
(OR)

6 A) Explain the differences between the effects of traditional convolution, unshaded convolution, and tiled convolution operations with neat diagrams and mathematical operations [CO3-L3] [7M]

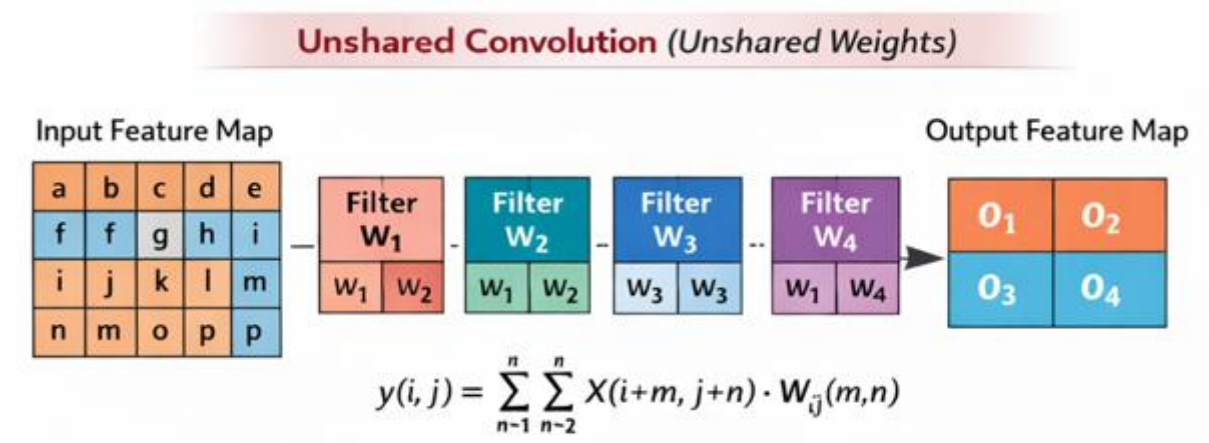
Ans: **Traditional convolution**: Equivalent to tiled convolution with t=1. i.e, there is only one kernel and it is applied everywhere. It looks likes Same connectivity as Unshared Convolution. But here we have to use Padding if Needed.



Example:



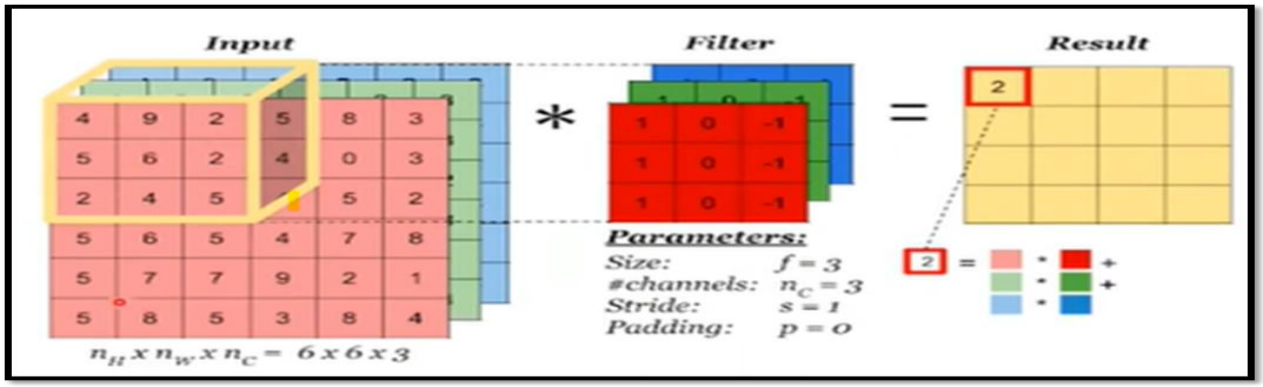
Unshaded convolution: The connectivity graph of convolution operation and locally connected layer is the same The only difference is that parameter sharing is not performed. i.e, Each output unit performs a linear operation on its neighbourhood but parameters are not shared across output Units. This allows models to capture local connectivity while allowing different features to be computed at different spatial locations. This however requires much more parameters than the convolution operation.



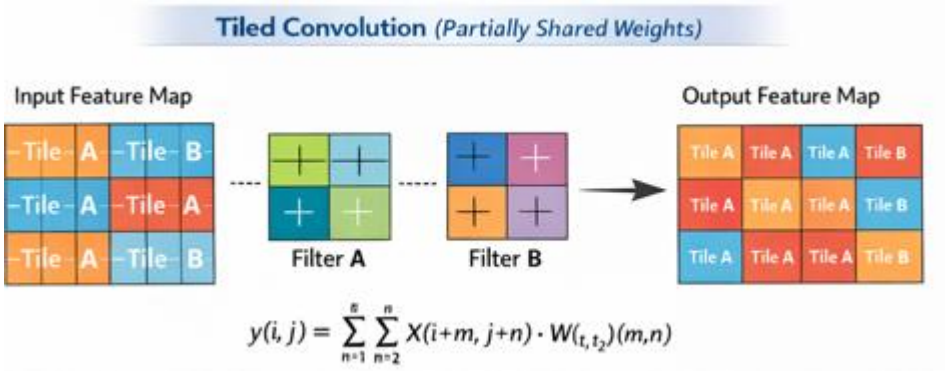
In the Unshared Convolution, every connection has its own weight

i, the output channel,
j, the output row,
k, the output column,
l, the input channel,
m, the row offset within the input, and
n, the column offset within the input.

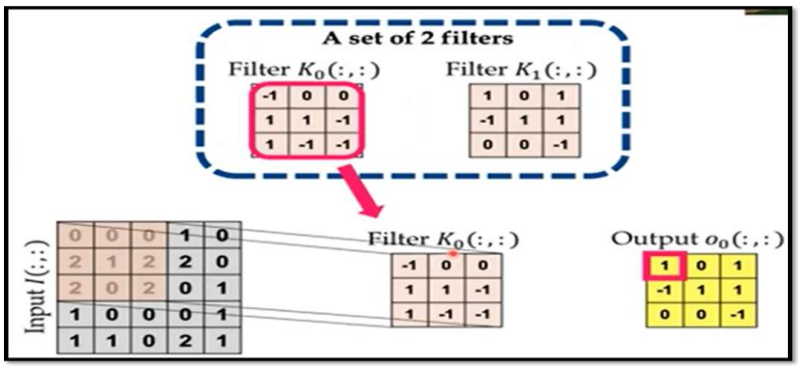
Example:



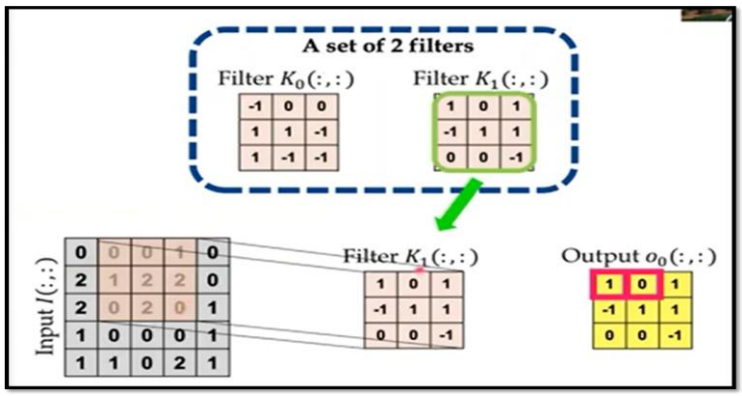
Tiled Convolution: It Offers a compromise b/w unshared and traditional convolution. i.e, Rather than learning a separate set of weights at every spatial location, we learn a set of kernels that we rotate through as we move through space. This means that immediately neighboring locations will have different filters, like in a locally connected layer, makes use of parameter sharing.



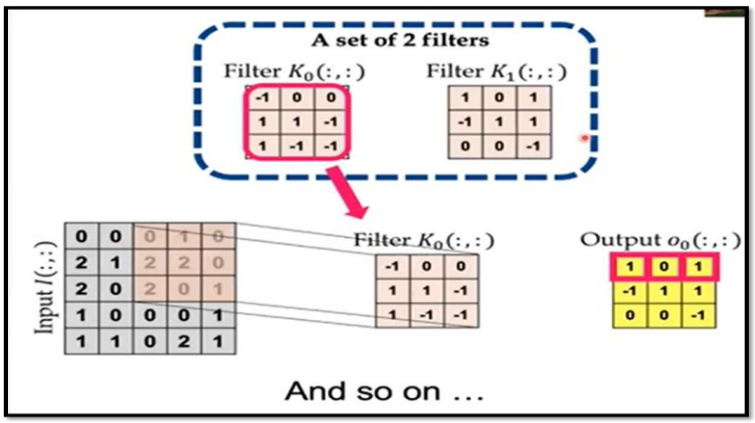
K0 Filter with 1st Portion of the Image:



K0 Filter with 2nd Portion of the Image:



K0 Filter with 3rd Portion of the Image:



---

6 B) What properties of primary visual cortex (it means a small region in the brain) will be captured by CNN. Explain them [CO3-L4] [7M]

Ans:

The **primary visual cortex (V1)** in the human brain processes basic visual patterns such as edges, orientation, and motion. **Convolutional Neural Networks (CNNs)** are inspired by these biological mechanisms. Several important **V1 properties are captured by CNNs**, especially in the early convolutional layers.

Properties of the Primary Visual Cortex Captured by CNNs:

1. Local Receptive Fields:

- **In V1:**

  Each neuron responds only to a **small local region** of the visual field (called a receptive field).

- **In CNNs:**

  Convolutional filters operate on **small patches** of the input image (e.g., 3×3, 5×5).

- **Significance:**

  Helps detect **local features** such as edges, corners, and textures.

✓ **Captured by:** Convolution operation

2. Orientation Selectivity:

- **In V1:**Neurons respond selectively to edges at specific orientations (horizontal, vertical, diagonal).
- **In CNNs:**

  Learned filters in early layers detect **edges and gradients** at different orientations.

- **Example:**

  One filter may detect vertical edges, another horizontal edges.

✓ **Captured by:** Learned convolution kernels

3. Spatial Frequency Sensitivity:

- **In V1:** Some neurons respond to **fine details (high frequency)**, others to **coarse patterns (low frequency)**.
- **In CNNs:**
  - Small kernels capture fine details
  - Larger receptive fields (deeper layers) capture broader structures
- **Significance:**

  Enables detection of both textures and shapes.

✓ **Captured by:** Kernel size and depth of layers

4. Hierarchical Feature Processing:

- **In V1:**

  Simple cells detect edges → complex cells combine them into shapes.

- **In CNNs:**
  - Early layers: edges, lines
  - Middle layers: corners, textures
  - Deep layers: objects or parts

- **Significance:**

  Mimics the visual hierarchy of the brain.

✓ **Captured by:** Stacked convolution layers

5. Translation Invariance:

- **In V1:**

  Higher-level neurons respond to patterns **regardless of small shifts** in position.
- **In CNNs:**

  Pooling layers (max/average pooling) make features **robust to small translations**.
- **Example:**

  An edge detected anywhere in the image still activates the feature map.

✓ **Captured by:** Pooling layers

6. Weight Sharing:

- **In V1:**

  Similar neurons detect the same feature at different locations.
- **In CNNs:**

  The same filter is applied **across the entire image**.
- **Significance:**

  Reduces parameters and improves generalization.

✓ **Captured by:** Shared convolution filters

7. Simple and Complex Cells Analogy:

- **In V1:**
  - Simple cells: detect edges at specific positions
  - Complex cells: respond to edges regardless of exact position
- **In CNNs:**
  - Convolution layers ≈ simple cells
  - Pooling layers ≈ complex cells

✓ **Captured by:** Convolution + pooling structure

UNIT-IV

7 A) Design the Architecture of RNN Model and then explain its objective function and model equations [CO3-L3] [7M]

Ans: Recurrent neural network (RNN) is a type of artificial neural network that is used to process sequential data. In Recurrent Neural Network(RNN) where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other. Still, in cases when it is required to predict the next word of a sentence, the previous words are

required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer.

Architecture of RNN: A recurrent neural network appears very just like feed forward neural networks, except it also has connections pointing backwards.

It has 3 Layers. Those are

- **Input Layer**
- **Hidden Layer and**
- **Output Layer**

- **RNNs are a type of neural network** that has **hidden states and allows past outputs to be used as inputs**.

x (green circle): Input at a given time step (e.g., a word in a sentence, sensor reading, etc.).

h (blue rectangle): Hidden state (memory). It captures information from the **current input** and the **previous hidden state**.

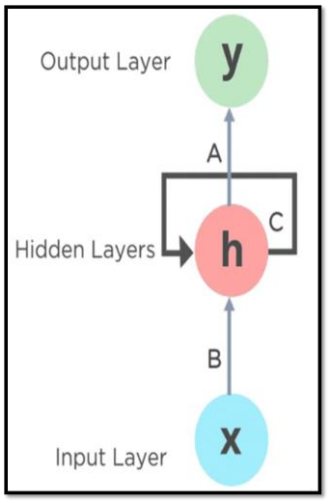
o (red circle): Output at that time step.

U, V, W (weights): Parameters of the RNN (learned during training).

U: Weight matrix connecting input **x to hidden state h**

V: Weight matrix connecting **hidden state h to the next hidden state** (recurrent connection).

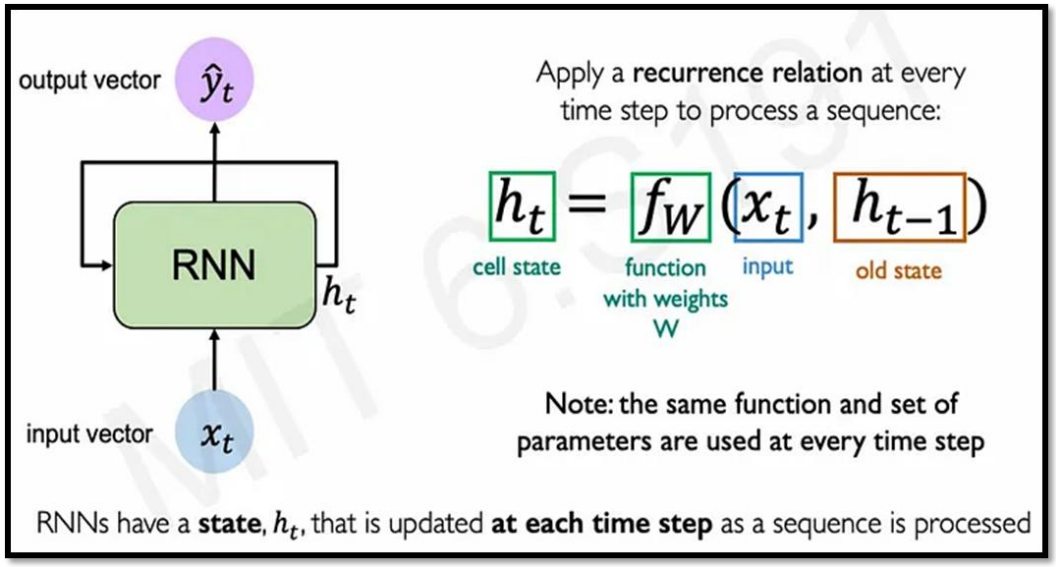
W: Weight matrix connecting **hidden state 'h' to output 'o'**



- The fundamental processing unit in a Recurrent Neural Network (RNN) is a Recurrent Unit, which is not explicitly called a "Recurrent Neuron."
- **Input Layer:** Here, "x" is the input layer, "h" is the hidden layer, and "y" is the output layer. A, B, and C are the network parameters used to improve the output of the model.
- **Recurrent Layer:** Normally a hidden layer or node has two parameters: bias and weight. But a recurrent node has three parameters: input, bias, and weight.

- At any given time t, the current input is a combination of input at x(t) and x(t-1). The output at any given time is fetched back to the network to improve on the output.

In RNNs, the information cycles through the loop to the middle hidden layer.



For calculating the current state –

$$h_t = f(h_{t-1}, x_t)$$

ht – current state
ht-1 – previous state
xt – input state

In order to apply the activation function tanh, we have –

$$h_t = \tanh (W_{hh}h_{t-1} + W_{xh}x_t)$$

where:

whh -> weight of recurrent neuron and,
wxh -> weight of the input neuron

The formula for calculating output:

$$y_t = W_{hy}h_t$$

Yt -> output
Why -> weight at output layer

These parameters are updated using Back propagation. However, since **RNN works on sequential data** here we use an **updated back propagation** which is known as **Back propagation through time**.

7 B) Design the Architecture of Encoder- decoder sequence- to – sequence architecture with example. [CO3-L3] [7M]

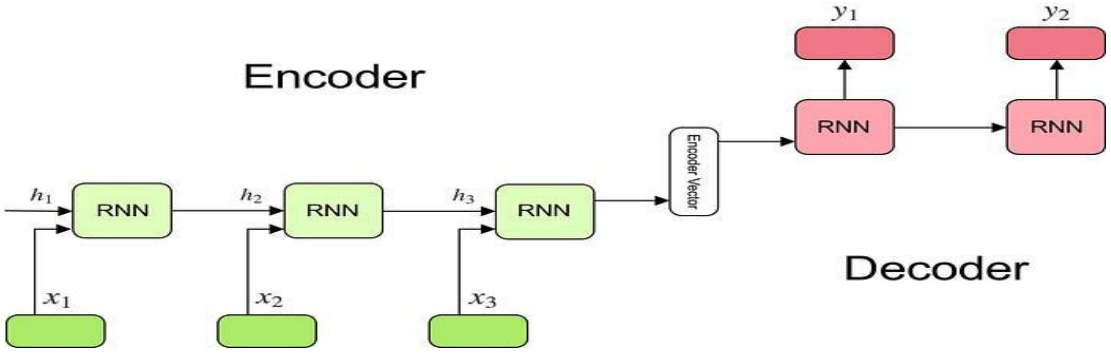
Ans: Encoder – Decoder Model:

There are three main blocks in the encoder-decoder model,

- Encoder
- Hidden Vector
- Decoder

- The Encoder will convert the input sequence into a single-dimensional vector (hidden vector). The decoder will convert the hidden vector into the output sequence.
- Encoder-Decoder models are jointly trained to maximize the conditional probabilities of the target sequence given the input sequence.

Encoder-Decoder Architecture in Sequence- to – Sequence tasks : In order to fully understand the model's underlying logic, we will go over the below illustration:



1. **Encoder :** Multiple RNN cells can be stacked together to form the encoder. RNN reads each inputs sequentially

- For every timestep (each input) t, the hidden state (hidden vector) h is updated according to the input at that timestep X[i].
- After all the inputs are read by encoder model, the final hidden state of the model represents the context/summary of the whole input sequence.
- Example: Consider the input sequence "I am a Student" to be encoded. There will be totally 4 timesteps (4 tokens) for the Encoder model. At each time step, the hidden state h will be updated using the previous hidden state and the current input.

2. **Encoder Vector:** This is the final hidden state produced from the encoder part of the model. It is calculated using the formula above.This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.it acts as the initial hidden state of the decoder part of the model.

3. Decoder : The Decoder generates the output sequence by predicting the next output Yt given the hidden state ht.

- The input for the decoder is the final hidden vector obtained at the end of encoder model.

- Each layer will have three inputs, hidden vector from previous layer ht-1 and the previous layer output yt-1, original hidden vector h.
- At the first layer, the output vector of encoder and the random symbol START, empty hidden state ht-1 will be given as input, the outputs obtained will be y1 and updated hidden state h1 (the information of the output will be subtracted from the hidden vector).
- The second layer will have the updated hidden state h1 and the previous output y1 and original hidden vector h as current inputs, produces the hidden vector h2 and output y2.
- The outputs occurred at each timestep of decoder is the actual output. The model will predict the output until the END symbol occurs.
- A stack of several recurrent units where each predicts an output y_t at a time step t.

(OR)

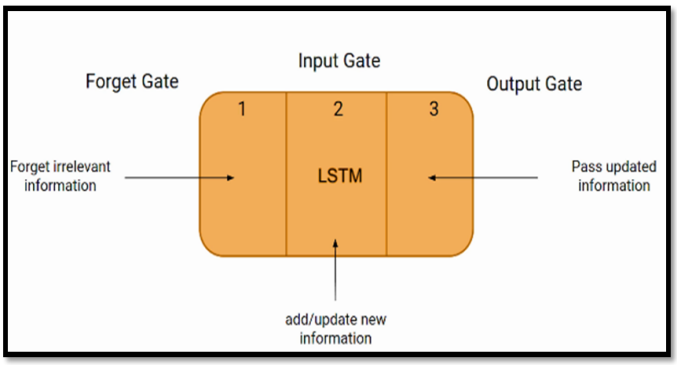
8 A) Distinguish between FFNN & RNN [CO3-L4] [7M]

Ans:

No.	Feed Forward Neural Network (FFNN)	Recurrent Neural Network (RNN)
1	Information flows **only in one direction** (input → hidden → output).	Information flows **forward as well as backward through time** using feedback loops.
2	**No memory** of previous inputs.	Has **memory** to store information from previous time steps.
3	Output depends **only on current input**.	Output depends on **current input and past inputs**.
4	Suitable for **static data**.	Suitable for **sequential or time-dependent data**.
5	Each input is processed **independently**.	Inputs are processed **as a sequence**.
6	Same input always gives the same output (no context).	Same input may give **different outputs depending on context**.
7	Training is done using **Backpropagation (BP)**.	Training is done using **Backpropagation Through Time (BPTT)**.
8	Cannot handle variable-length input sequences easily.	Can handle **variable-length sequences**.
9	Architecture is **simple and easy to train**.	Architecture is **complex and harder to train**.
10	Does **not suffer from vanishing gradient** problem severely.	Often suffers from **vanishing and exploding gradient** problems.
11	Number of parameters is generally **fixed**.	Parameters are **shared across time steps**.
12	Cannot model temporal dependencies.	Efficiently models **temporal dependencies**.
13	Used mainly for **classification and regression** problems.	Used mainly for **sequence modeling** problems.
14	Faster to train due to simpler structure.	Slower to train due to recurrent connections.
15	No feedback connections between neurons.	Contains **feedback (recurrent) connections**.

8 B) Demonstrate the Long Short Term Memory(LSTM) Model with a neat diagram and its mathematical equations [CO3-L3] [7M]

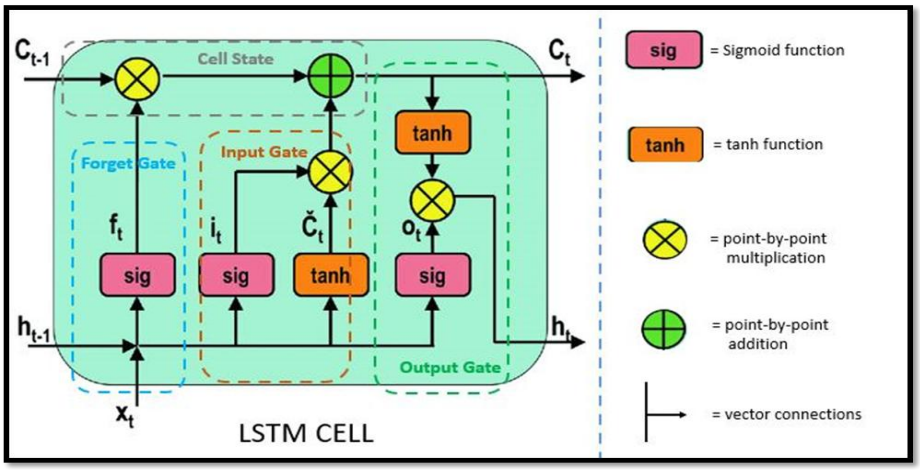
Ans: <u>The architecture of LSTM:</u> The LSTM architectures involves the memory cell which is controlled by three gates:



- **These gates decide** what **information to add to**, **remove from**, and **output from the memory cell**.
- **<u>The input gate controls</u> what information** is **added to the memory cell.**
- **<u>The forget gate controls</u>** what information is **removed from the memory cell.**
- **<u>The output gate controls</u>** what information is **output from the memory cell.**
- **<u>Cell (the memory part of LSTM):</u>** The **cell stores the state of a sequence**, so it **has the ability to either keep** or **forget certain information.**
- The **LSTM maintains a hidden state**, which acts as the **short-term memory of the network**. The **hidden state is updated based on the input,** the **previous hidden state**, and **the memory cell's current state**.
- Here **Information is retained by the cells** and **the memory manipulations** are **<u>done by the gates.</u>**
- Let's assume we have a **sequence of words (w1, w2, w3, ..., wn)** and we are **processing the sequence one word at a time**. Let's denote the **state of the LSTM at time step t** as **<u>(ht, ct),</u>**
- where **ht** is the **hidden state** and **ct** is the **cell state**.
- **ct-1** stands for the **input from a memory cell in time point t;**

Xt is an **input in time point t;**

- **ht** is an **output in time point t** that **goes to both the output layer** and the **hidden layer in the next time point.**



❖ Here Information is retained by the cells and the memory manipulations are done by the gates.

❖ Let's assume we have a sequence of words (w1, w2, w3, ..., wn) and we are processing the sequence one word at a time. Let's denote the state of the LSTM at time step t as (ht, ct),

❖ where ht is the hidden state and ct is the cell state.

❖ ct-1 stands for the input from a memory cell in time point t;
Xt is an input in time point t;

❖ ht is an output in time point t that goes to both the output layer and the hidden layer in the next time point.

The equation for the forget gate is:

$$f_t = \sigma\left(W_f \cdot \left[h_{(t-1)}, x_t\right] + b_f\right)$$

- where:
  - **Wf** represents the **weight matrix** associated with the **forget gate.**
  - **[ht-1, Xt]** denotes the **concatenation** of the **current input** and **the previous hidden state.**
  - **bf** is **the bias** with the **forget gate.**
  - **σ** is the **sigmoid activation function**.

Input Gate: The **input gate (it)** decides what **new information to store in the cell state**.

It has two parts. A **sigmoid Function** called the **"input gate layer"** decides **which values we'll update**, and **a tanh Function** creates a **vector of new candidate values (Ct~)** that **could be added to the state**. Ie, **First, the information is regulated** using the **sigmoid function** and **filter the values to be remembered similar to the forget gate** using **inputs h_{t-1} and x_t. .**

➢ Then, **a vector** is **created using *tanh* function** that **gives an output from -1 to +1,** which **contains all the possible** values **from h_{t-1} and x_t. At last, the values of the vector** and the **regulated values** are **multiplied to obtain the useful information.**

- **The equation for the input gate is:**

$$i_t = \sigma\left(W_i \cdot \left[h_{(t-1)}, x_t\right] + b_i\right)$$

❖ **Candidate Values(Cell State Update):**

$$\tilde{C_t} = \tanh\left(W_c \cdot \left[h_{(t-1)}, x_t\right] + b_c\right)$$

Output Gate: The task of extracting useful information from the **current cell state** to be **presented as output is done by the output gate.** First, **a vector is generated** by **applying tanh function** on **the cell**. Then, the information is regulated **using the sigmoid function** . At last, the **values of the vector** and the **regulated values are multiplied to be sent** as **an output** and **input to the next cell. The equation for the output gate is:**

$$o_t = \sigma\left(W_o \cdot \left[h_{(t-1)}, x_t\right] + b_o\right)$$

Hidden State:

$$h_t = o_t \times \tanh\left(C_t\right)$$

UNIT-V

9A) Explain how can apply local and global contrast normalization processing operations whenever CNN Model is used for Computer Vision application task?

[CO4-L4] [7M]

Ans:

1. **Global contrast normalization: Global Contrast Normalization** normalizes the **entire image** using global statistics such as mean and standard deviation.

Mathematical Formulation

For an image X:

$$X' = \frac{X - \mu}{\sigma + \epsilon}$$

where:

- μ = mean pixel intensity of the entire image
- σ = standard deviation of the image
- ϵ = small constant to avoid division by zero

How to Apply in CNN Workflow

1. Read input image.
2. Compute global mean and standard deviation.
3. Normalize all pixels using the formula.
4. Feed normalized image to CNN.

Role in CNN

- Ensures **zero mean and unit variance** input.
- Prevents domination of large pixel values.
- Improves gradient stability during backpropagation.

Advantages

- Simple and computationally efficient.
- Reduces lighting and exposure variations.
- Helps faster convergence of CNN training.

Limitations

- Does not handle **local lighting changes**.
- May suppress important local texture details.

2. **Local Contrast Normalization:** **Local Contrast Normalization** normalizes each pixel based on its **local neighborhood**, mimicking the behavior of neurons in the human visual system.

Mathematical Formulation

For pixel $x_{i,j}$:

$$x'_{i,j} = \frac{x_{i,j} - \mu_{i,j}}{\sqrt{\sigma^2_{i,j} + \epsilon}}$$

where:

- $\mu_{i,j}$ = local mean around pixel (i, j)
- $\sigma_{i,j}$ = local standard deviation
- Computed using a sliding window or Gaussian kernel

How to Apply in CNN Workflow

1. Define a local window (e.g., 5×5 or 7×7).
2. Compute local mean and variance for each pixel.
3. Normalize pixel using local statistics.
4. Input normalized feature maps into CNN layers.

Role in CNN

- Enhances **edges and textures.**
- Reduces effects of shadows and uneven illumination.
- Encourages competition among nearby neurons.

Advantages

- Preserves local structure.
- Improves robustness to illumination changes.
- Enhances discriminative local features.

9B) Model the Speech Recognition problem in mathematical notation and then write and explain its objective function. [CO4-L3] [7M]

Ans: Speech recognition is the ability of a machine or program to identify and understand human speech. It has a wide range of applications, from virtual assistants like Siri and Alexa, to transcription of audio tracks, like generating subtitles for YouTube videos. However, with the rise of deep learning, speech recognition systems have become far more accurate. Deep neural networks are able to learn speech representations directly from data, surpassing previous state-of-the-art models

- The primary goal of speech recognition systems is to accurately and efficiently transcribe spoken words into a format that can be processed, stored, or used for various applications. This technology relies on sophisticated algorithms and Deep Learning techniques to interpret and understand human speech patterns.

- Automatic speech recognition (ASR) refers to the task of recognizing human speech and translating it into text.

Mathematical Modeling of Speech Recognition

Problem Definition

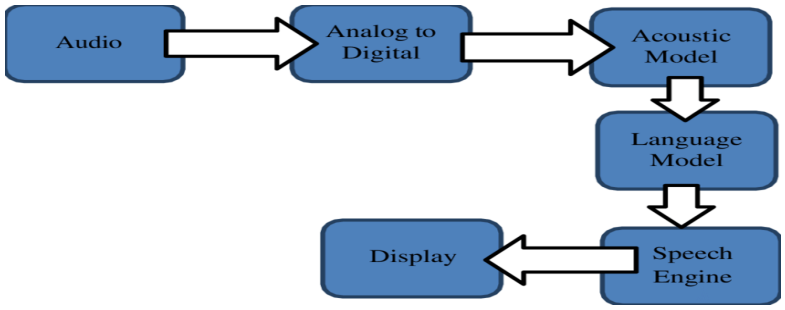
Let:

- $X = (x_1, x_2, \ldots, x_T)$ be the **acoustic observation sequence**, where each x_t represents a feature vector (e.g., MFCCs) at time t.
- $W = (w_1, w_2, \ldots, w_N)$ be the **word sequence** (sentence) to be recognized.

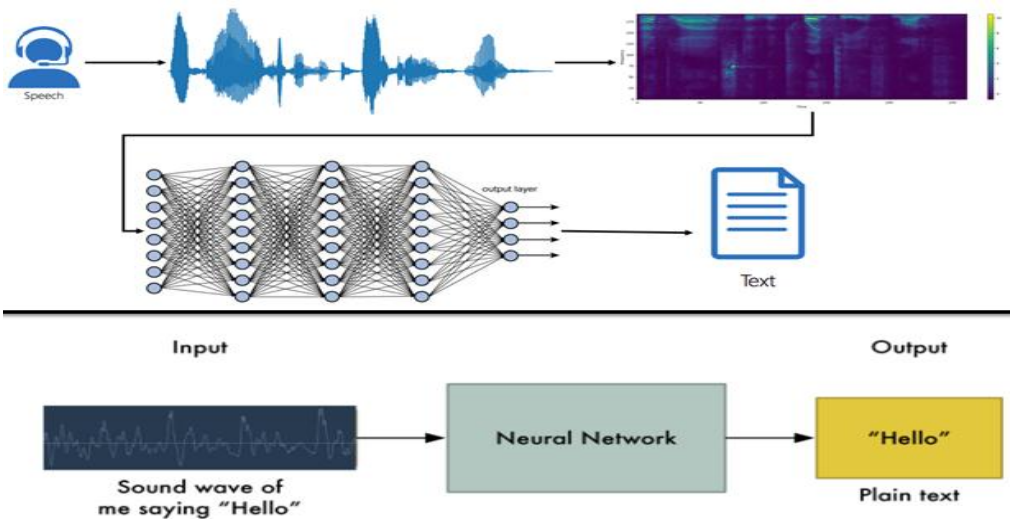
The speech recognition problem is to find the word sequence W^* that maximizes the posterior probability:

$$W^* = \arg\max_W P(W \mid X)$$

Architecture:



Working Procedure:



- The big problem is that **speech varies in speed.**
- One person might say **"hello!" very quickly** and another person might say **"heeeellllllllllllooooo!" very slowly**, producing a much **longer sound file with much more data.**
- Both both sound files should be recognized as exactly the **same text — "hello!"**
- Automatically aligning **audio files of various lengths** to a fixed-length piece of text turns out to be pretty hard.

Components of the Model

Acoustic Model $P(X \mid W)$

- Represents the probability of observing the acoustic signal X given the word sequence W.
- Models the relationship between **audio features and phonetic units.**
- Commonly implemented using:
  - HMM + GMM
  - HMM + DNN
  - End-to-end neural networks

$$P(X \mid W) = \prod_{t=1}^{T} P(x_t \mid s_t)$$

where s_t is the hidden speech state (phoneme).

Language Model $P(W)$

- Represents the probability of a word sequence occurring in a language.
- Captures grammatical structure and word order.
- Often modeled using **N-grams** or **neural language models.**

$$P(W) = \prod_{i=1}^{N} P(w_i \mid w_{i-1}, \ldots, w_{i-(n-1)})$$

Objective Function of Speech Recognition

Combined Objective Function

The final **objective function** is:

$$\boxed{W^* = \arg\max_W [P(X \mid W) \cdot P(W)]}$$

For numerical stability, logarithms are used:

$$\boxed{W^* = \arg\max_W [\log P(X \mid W) + \log P(W)]}$$

To balance acoustic and language contributions, a scaling factor λ is introduced:

$$\boxed{W^* = \arg\max_W [\log P(X \mid W) + \lambda \log P(W)]}$$

where:

- λ controls the influence of the language model.

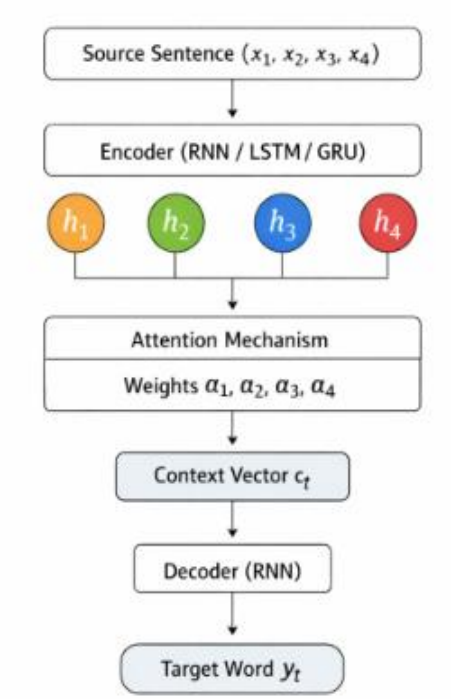
(OR)

10 A) Explain the Attention mechanism in machine translation with neat diagram and its mathematical equations [CO4-L4] [7M]

Ans: In **machine translation**, the task is to convert a source sentence (e.g., English) into a target sentence (e.g., Telugu). Traditional **Encoder–Decoder models** compress the entire source sentence into a **single fixed-length context vector**, which leads to information loss for long sentences.

The **Attention Mechanism** solves this problem by allowing the decoder to **focus on different parts of the source sentence at each decoding step**, similar to how humans translate by looking back and forth between words.

Encoder- Decoder Architecture:



Mathematical Modeling of Attention

Encoder Representation

Let the input sentence be:

$$X = (x_1, x_2, \ldots, x_T)$$

The encoder produces hidden states:

$$h_i = \text{Encoder}(x_i), \quad i = 1, 2, \ldots, T$$

. Decoder Hidden State

At decoding time step t:

$$s_t = \text{Decoder}(y_{t-1}, s_{t-1}, c_t)$$

where:

- s_t = decoder hidden state
- c_t = context vector from attention

Alignment Score (Attention Score)

The **alignment score** measures how well the decoder state matches each encoder state.

$$e_{t,i} = \text{score}(s_{t-1}, h_i)$$

Common scoring functions:

1. **Dot Product**

$$e_{t,i} = s_{t-1}^T h_i$$

2. **General**

$$e_{t,i} = s_{t-1}^T W h_i$$

3. **Additive (Bahdanau Attention)**

$$e_{t,i} = v^T \tanh(W_s s_{t-1} + W_h h_i)$$

Attention Weights (Softmax)

The alignment scores are normalized using softmax:

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{k=1}^{T} \exp(e_{t,k})}$$

where:

- $\alpha_{t,i}$ represents **how much attention** the decoder pays to word i.

Context Vector

The context vector is a **weighted sum of encoder hidden states**:

$$\boxed{c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i}$$

This vector captures the **relevant source information** for predicting the next target word.

Output Prediction

The probability of the next target word:

$$P(y_t \mid y_{<t}, X) = \text{softmax}(W_o[s_t; c_t])$$

where:

- $[s_t; c_t]$ = concatenation of decoder state and context vector

10 B) Define the problem statement of Recommender System and then explain how recommender system use collaborative filtering concept. [CO3-L4] [7M]

Ans:

Definition

A **Recommender System (RS)** is an intelligent system that predicts the **most relevant items** (products, movies, books, videos, etc.) for a user based on their **preferences, behavior, or interactions.**

Mathematical Problem Formulation

Let:

- $U = \{u_1, u_2, \ldots, u_m\}$ be the set of users
- $I = \{i_1, i_2, \ldots, i_n\}$ be the set of items
- R be a **user–item rating matrix**, where

$$r_{u,i} = \text{rating given by user } u \text{ to item } i$$

The recommender system aims to learn a function:

$$\boxed{f : U \times I \to \mathbb{R}}$$

such that $f(u, i)$ predicts the **unknown rating or preference** of user u for item i.

Objective

$$\boxed{\text{Recommend top-}K \text{ items } i \in I \text{ that maximize } f(u, i)}$$

Challenges (Mention for Extra Marks)

- Data sparsity
- Cold start problem (new user/item)
- Scalability
- Changing user preferences

Collaborative Filtering (CF): Concept

Definition

Collaborative Filtering is a recommendation technique that makes predictions based on the **collective behavior of users**, assuming that:

> *Users who agreed in the past will agree in the future.*

It **does not require item content**, only **user–item interactions** (ratings, clicks, likes).

3. Types of Collaborative Filtering

3.1 User-Based Collaborative Filtering

Idea

- Find users **similar to the target user**
- Recommend items liked by similar users

Steps

1. Compute similarity between users
2. Select top-k similar users (neighbors)
3. Predict rating using neighbors' ratings

Similarity Measures

- **Cosine Similarity**

$$\text{sim}(u, v) = \frac{\vec{r}_u \cdot \vec{r}_v}{\|\vec{r}_u\| \|\vec{r}_v\|}$$

- **Pearson Correlation**

$$\text{sim}(u, v) = \frac{\sum (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum (r_{v,i} - \bar{r}_v)^2}}$$

Rating Prediction

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{v \in N(} \text{sim}(u, v)(r_{v,i} - \bar{r}_v)}{\sum_{v \in N(u)} |\text{sim}(u, v)|}$$

4. Model-Based Collaborative Filtering (Matrix Factorization)

Idea

Factorize the rating matrix into **latent features:**

$$R \approx U \times V^T$$

where:

- U = user feature matrix
- V = item feature matrix

Prediction

$$\boxed{\hat{r}_{u,i} = \vec{p}_u^T \vec{q}_i}$$

Used in:

- Netflix Prize
- Amazon recommendations

3.2 Item-Based Collaborative Filtering

Idea

- Find items **similar to the item of interest**
- Recommend items similar to what the user already liked

Steps

1. Compute similarity between items
2. Select similar items
3. Predict rating using item similarities

Prediction Formula

$$\boxed{\hat{r}_{u,i} = \frac{\sum_{j \in S(i)} \text{sim}(i,j)\, r_{u,j}}{\sum_{j \in S(i)} |\text{sim}(i,j)|}}$$

Example

If a user liked *Movie A*, recommend *Movie B* that is similar to *Movie A*.

********** THE END **********