# 3 Deep Learning Architecture

UNIT - III

**Syllabus**

Width and Depth of Neural Networks, Different Activation Functions, Batch-normalization, Overfitting and generalization., Dropout, regularization Unsupervised Training of Neural Networks, Restricted Boltzmann Machines, Auto Encoders, Deep Learning Applications.

## 3.1 Neural Networks

As already discussed, a paradigm for information processing that draws inspiration from the brain is called an Artificial Neural Network (ANN). ANNs learn via imitation just like people do. Through a learning process, an ANN is tailored for a particular purpose, such as pattern recognition or data classification. The synaptic connections that exist between the neurons change as a result of learning.

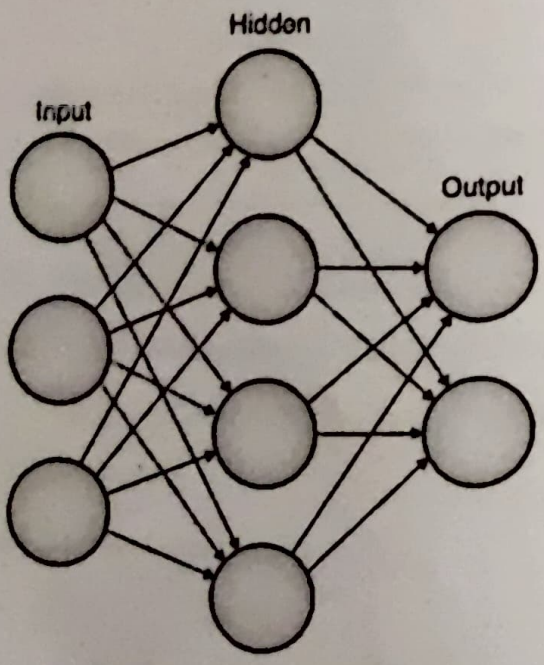



**Fig. 3.1.1 : Artificial Neural Network**

### 3.1.1 Width and Depth of Neural Network

In a Neural Network, the depth is its number of layers including output layer but not input layer. The width is the maximum number of nodes in a layer.

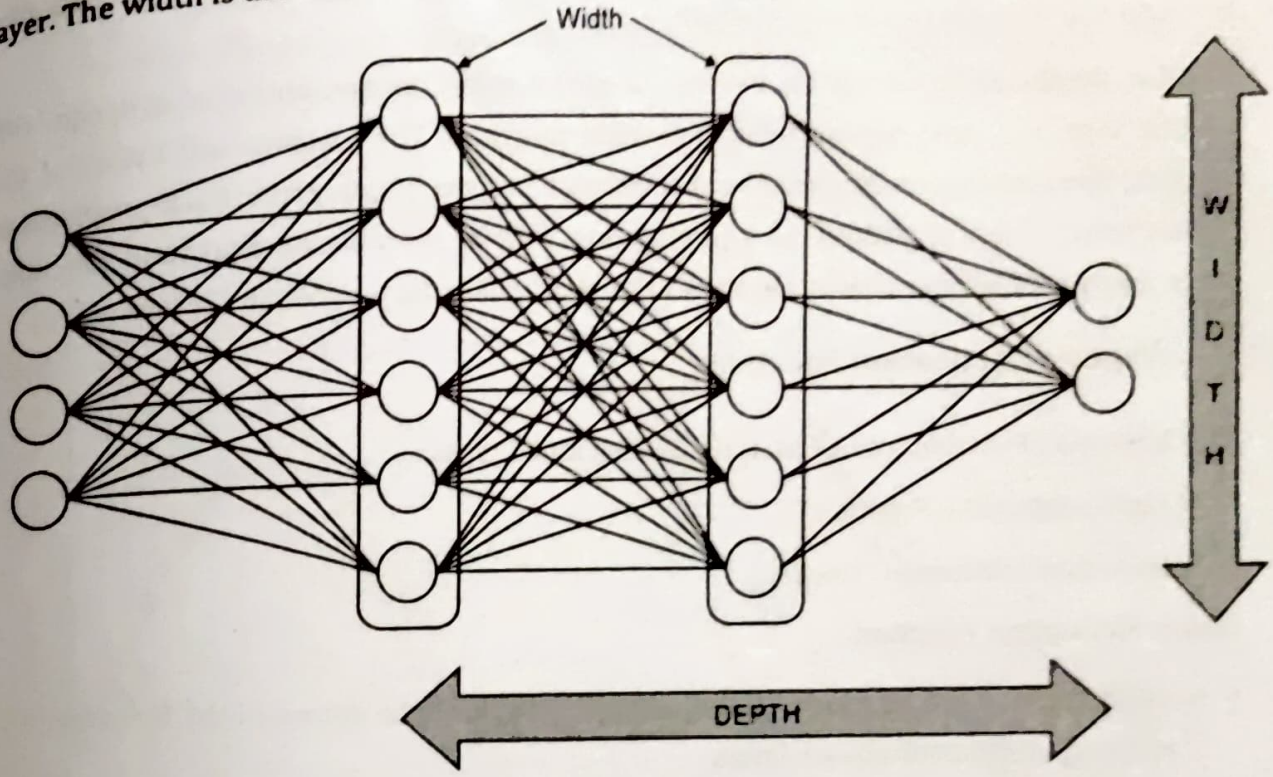



**Fig. 3.1.2 : Width and Depth in Neural Network**

## 3.2 Activation Functions

- What activation function to employ in the hidden layer and at the output layer of the network is one of the decisions you get to make while creating a neural network.
- Parts of a neural network include :
  - **Input Layer :** The input layer is the layer that takes features as input. No computation is done at this layer; instead, nodes simply send information (features) to the hidden layer. It gives the network information from the outside world.
  - **Hidden Layer :** Nodes of the hidden layer, which is a component of the abstraction offered by any neural network, are not visible to the outside world. The features entered through the input layer are processed by the hidden layer in any way, with the results being sent to the output layer.
  - **Output Layer :** This layer communicates the knowledge that the network has acquired to the outside world.

**What is an activation function and why to use them?**

- By generating a weighted total and then including bias with it, the activation function determines whether or not a neuron should be turned on. The activation function's objective is to add non-linearity to a neuron's output.
- In other words, as we are aware, neurons in neural networks operate in accordance with weight, bias, and their corresponding activation functions. The weights and biases of the neurons in a neural network would be updated based on the output error. Back-propagation is the name of this procedure. Back-propagation is made possible by activation functions since they provide the gradients and error needed to update the weights and biases.

## 3.2.1 Types of Activation Functions

The Activation Functions can be basically divided into 2 types :

1. Linear Activation Function
2. Non-linear Activation Functions

**1. Linear Activation Function :**

- As you can see the function is a line or linear. Therefore, the output of the functions will not be confined between any range.

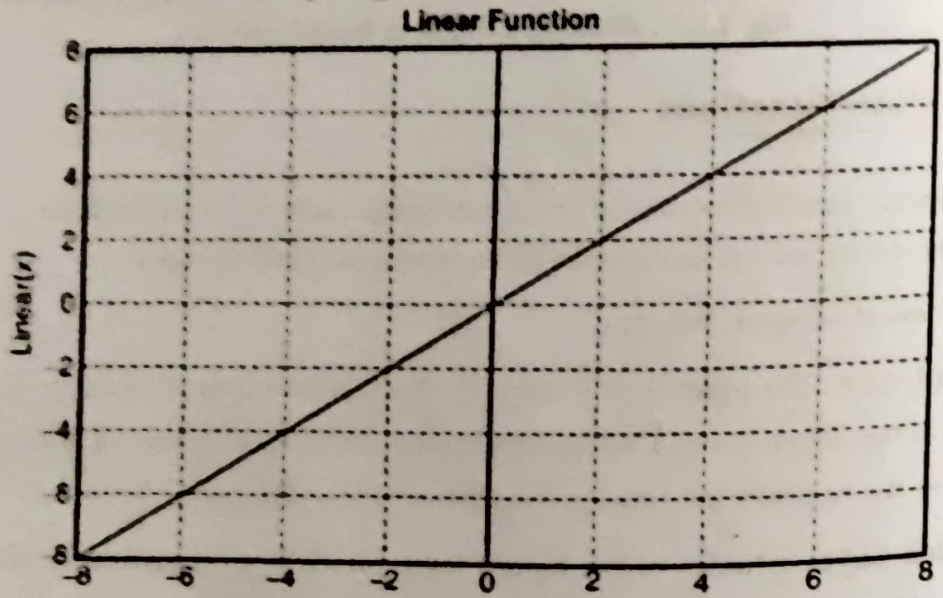



**Fig. 3.2.1 : Linear Activation Function**

- Equation : f(x) = x
- Range : (– infinity to infinity)

- The complexity or other parameters of the typical data that is input to the neural networks are unaffected.

**2. Non-linear Activation Functions :**

- The most often utilised activation functions are nonlinear activation functions. The graph appears like this thanks to nonlinearity.

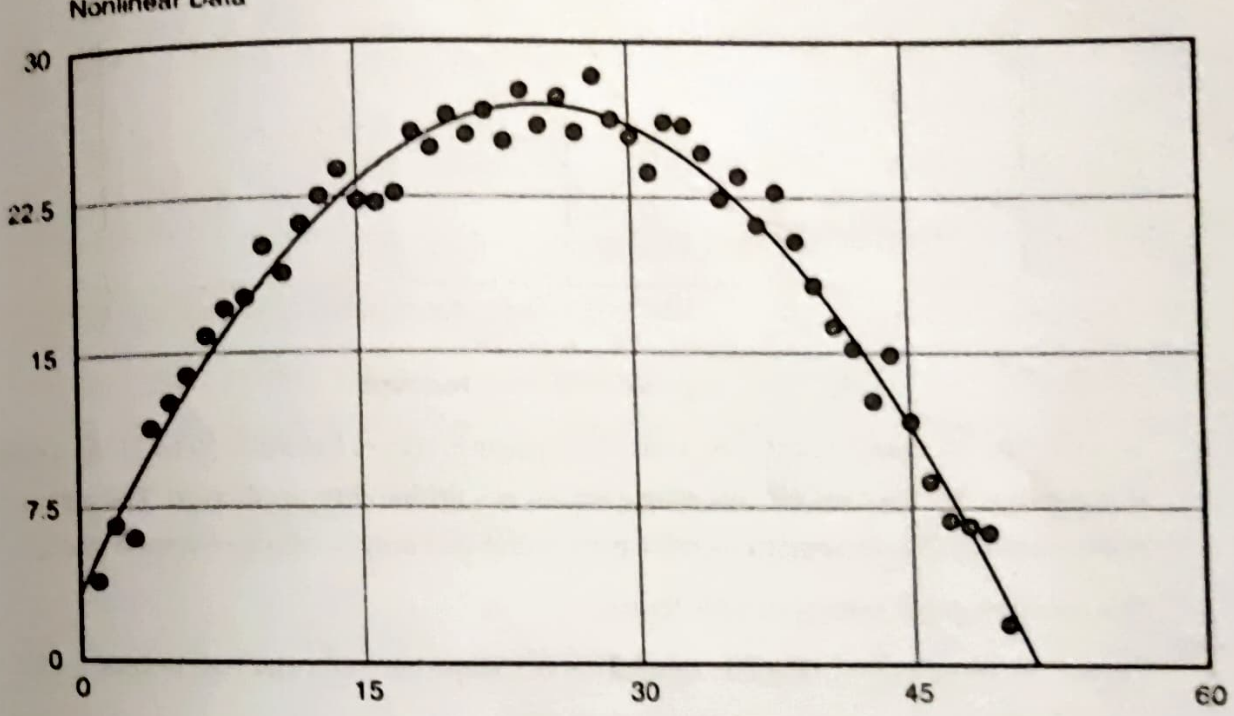



**Fig. 3.2.2 : Non Linear Activation Function**

- It facilitates the model's generalisation or adaptation to a variety of data and facilitates output differentiation.
- The key terms for nonlinear functions that you need to learn are :
  - **Derivative or Differential** : Y-axis change relative to X-axis change. Slope is another name for it.
  - **Monotonic** : A function that is totally non-increasing or non-decreasing is referred to as monotonic.
- An activation that is nonlinear Functions are typically categorised according to their range or curvature.

**1. Sigmoid or Logistic Activation Function**

- Sigmoid is an 'S' based curve.

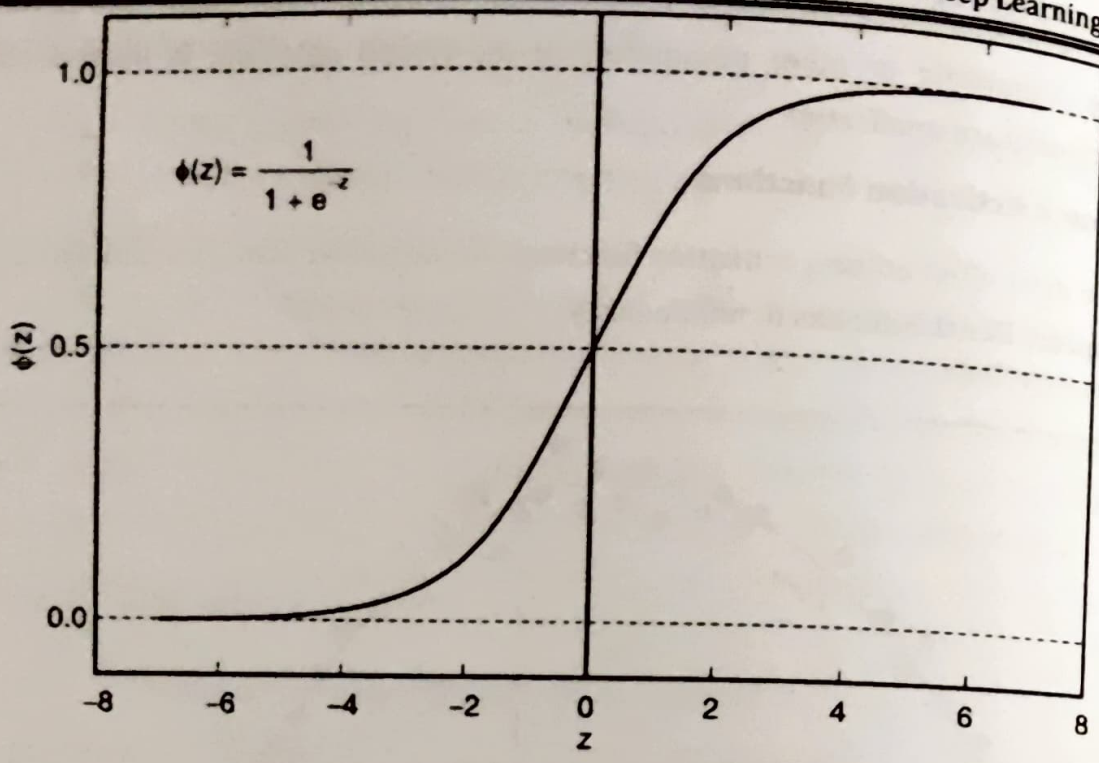



**Fig. 3.2.3 : Sigmoid Activation Function**

- We employ the sigmoid function primarily because it occurs between (0 to 1). As a result, it is particularly used for models whose output is a probability prediction. The sigmoid is the best option because anything has a probability that only occurs between 0 and 1.
- The function might take numerous forms.
- Therefore, we can determine the sigmoid curve's slope between any two points.
- Although the function is monotonic, its derivative is not.
- A neural network may become stuck during training as a result of the logistic sigmoid function.
- For multiclass classification, the softmax function, a more generalised logistic activation function, is utilised.

**ii. Tanh or hyperbolic tangent Activation Function**

- Tanh is also like logistic sigmoid but better. The range of the tanh function is from (−1 to 1). tanh is also sigmoidal (s - shaped).
- The positive aspect of this is that the zero inputs will be mapped near zero and the negative inputs will be highly negative in the tanh graph.
- The function might take numerous forms.
- While the derivative of the function is not monotonic, the function itself is.
- The tanh function is mostly used to classify data into two groups.

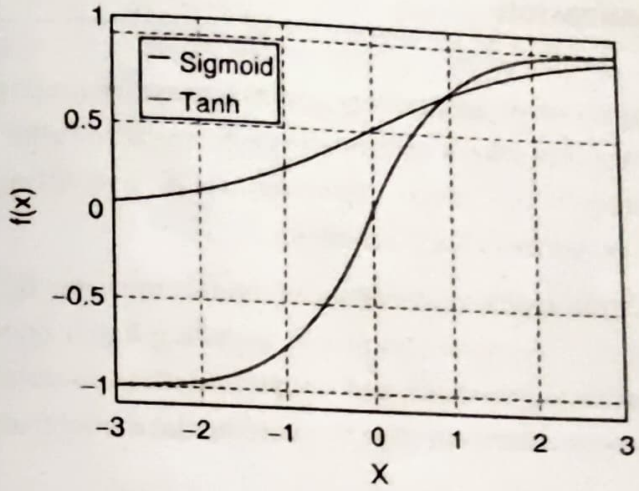



**Fig. 3.2.4 : tanh Activation Function**

### iii. ReLU (Rectified Linear Unit) Activation Function :

- Currently, the ReLU is the activation function that is employed the most globally. Since practically all convolutional neural networks and deep learning systems employ it.

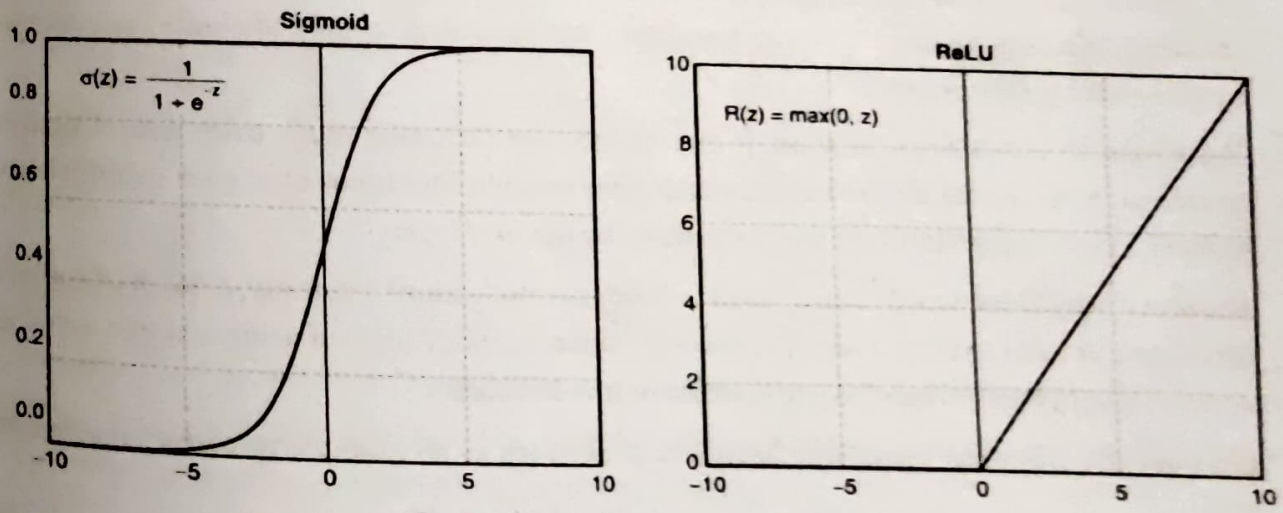



**Fig. 3.2.5 : ReLU Activation Function**

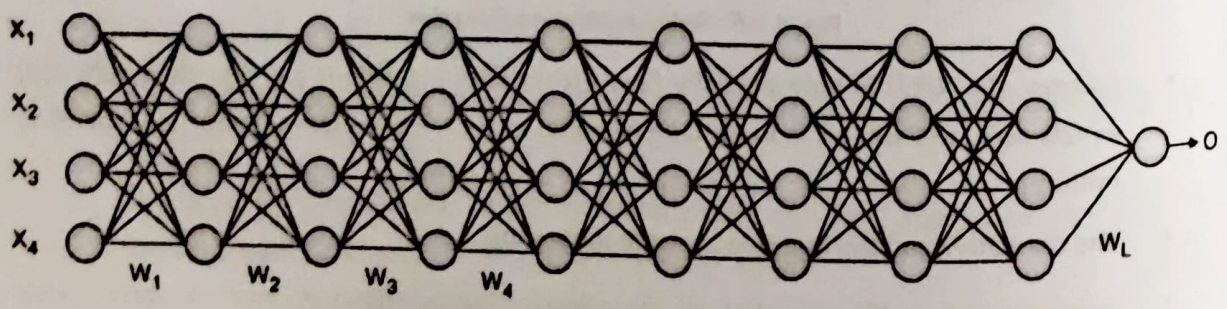
- As seen, the ReLU is only partially fixed (from bottom). When z is less than zero, f(z) equals zero, and when z is more than or equal to zero, f(z) equals z. Range is [From 0 to infinity]
- The derivative and the function are both monotonic.
- However, the problem is that all the negative values instantly become zero, which reduces the model's capacity to effectively fit or train from the data. This means that any negative input to the ReLU activation function immediately becomes zero in the graph, which has an impact on the final graph by improperly mapping the negative values.

## 3.3 Batch Normalization

- Avoiding over-fitting is one of data science experts' most frequent problems. Have you ever encountered a circumstance where your model performs admirably on training data but fails to appropriately predict test data? Your model is overfitting, which is the cause. Regularization is the response to such a challenge.
- The use of regularisation techniques enhances models and speeds up convergence. At our end, we have a number of regularisation tools, including batch normalisation, early halting, dropout, weight initialization methods, and weight initialization strategies. The regularisation makes the learning process more effective by avoiding the model from becoming overfit.

### What is Batch Normalization?

- Let's first define the term "Normalization" before moving on to batch normalisation.
- A data pre-processing technique called normalisation is used to scale down numerical data without changing its structure.
- In general, we tend to modify the numbers to a balanced scale when we submit the data to a machine learning or deep learning algorithm. We normalise in part to make sure that our model generalises correctly.
- Returning to Batch normalisation, it is a method for including more deep neural network layers to make neural networks faster and more reliable. The input of a layer coming from a previous layer is standardised and normalised by the new layer.
- However, why is the word "Batch" used in batch normalisation? Typically, a batch of gathered input data is used to train a neural network. Similar to this, rather of using a single input, the normalising process in batch normalisation occurs in batches.
- Let's use the following image of a deep neural network as an example to better comprehend this.




L = Number of layers
Bias = 0
Activation function : sigmoid

**Fig. 3.3.1 : A : Batch Normalisation**

- Our initial inputs $X_1$, $X_2$, $X_3$, and $X_4$ come from the pre-processing stage and are in normalised form. The input transforms as a sigmoid function is applied over the dot product of the input X and the weight matrix W as it moves through the first layer.

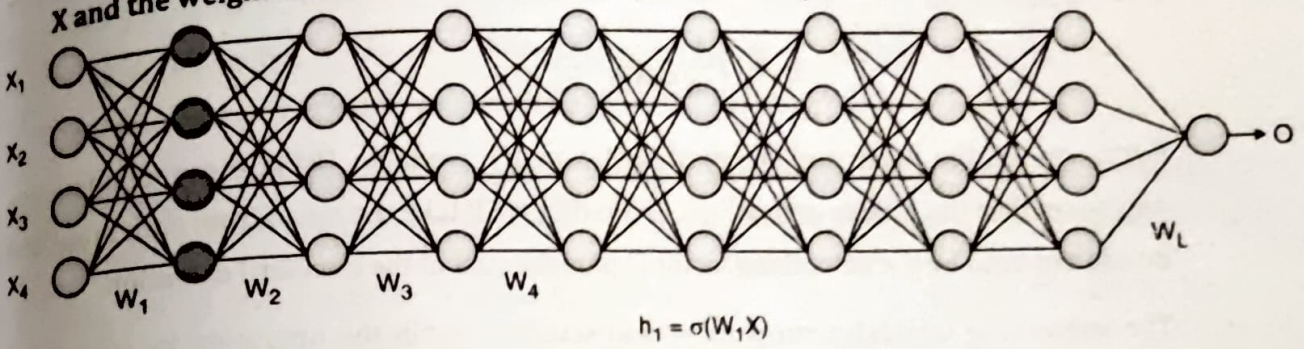



**Fig. 3.3.2 : B : Batch Normalisation**

- As seen in the next graphic, this transformation will also occur for the second layer and continue through layer L.

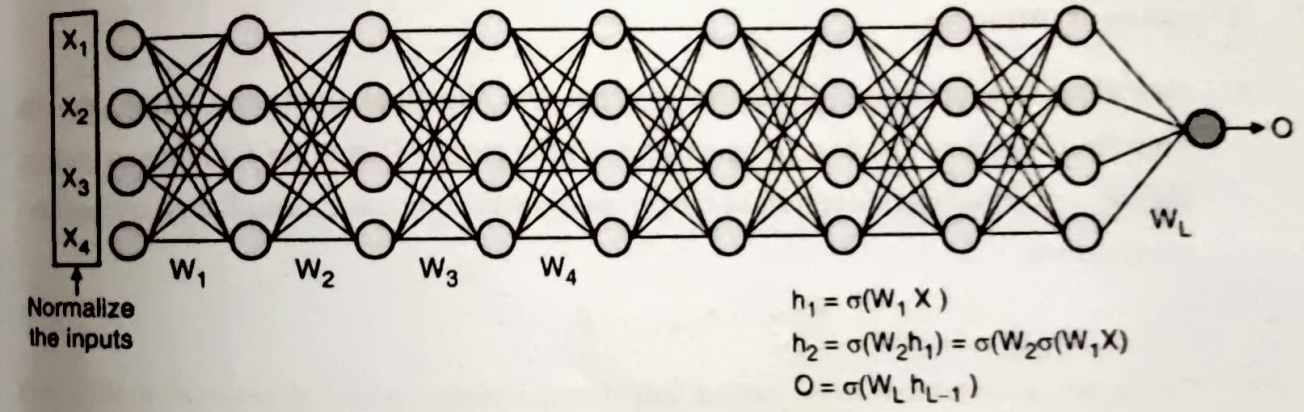



**Fig. 3.3.3 : C : Batch Normalisation**

- Despite the fact that our input X was normalised, the scale of the output will change with time. Data undergoes an internal covariate shift as it passes through the neural network's various levels and is subjected to L activation functions.

**How does Batch Normalization work?**

This is a first of the two steps in batch normalisation. Batch normalisation involves two steps. The input is first normalised, and then rescaling and offsetting are carried out.

1. **Normalization of the Input**

   - The act of changing data so that the mean and standard deviation are both 0 is known as normalisation. We have our batch input from layer h in this phase, thus we must first determine the mean of this hidden activation.

$$\mu = \frac{1}{m}\sum h_i$$

- In this case, m represents the quantity of layer h neurons.
- The next step is to determine the standard deviation of the hidden activations after we have meant at our end.

$$\sigma = \left[\frac{1}{m}\sum(h_i - \mu)^2\right]^{1/2}$$

- Furthermore, the mean and standard deviation are available. Using these numbers, we will normalise the hidden activations. To do this, we'll take the mean from each input and divide the total by the smoothing factor 0 plus the sum of the standard deviation.
- The smoothing term(ε) assures numerical stability within the operation by stopping a division by a zero value.

$$h_{i(norm)} = \frac{(h_i - \mu)}{\sigma + \varepsilon}$$

2. **Rescaling of Offsetting**

- Rescaling and offsetting of the input happen in the final operation. Here, the BN algorithm's gamma and beta components enter the picture (beta). Re-scaling and shifting the vector containing the results of the previous operations are accomplished using these parameters.

$$h_i = \gamma h_{i(norm)} + \beta$$

- These two parameters can be learned, and during training, a neural network makes sure that the best values for and are employed. This will make it possible to accurately normalise each batch.

**Advantages of Batch Normalization**

1. Speed Up the Training
2. Handles internal covariate shift
3. Smoothens the Loss Function

## 3.4 Overfitting and Generalization

- Prior to going farther on generalisation and overfitting, it's critical to comprehend supervised learning. Only while learning under supervision is overfitting a concern. One way for a machine learning model to learn and comprehend data is through supervised learning

Unsupervised and reinforcement learning are two further types of learning, but those are topics for another day and another blog post. A model is given a set of labelled training data when using supervised learning. The more training data the model has access to, the better it gets at producing predictions since the model learns to do so from this training data. With training data, the result is predetermined.

- The model's parameters are adjusted until the model's predictions and observed results are in line. The goal of training is to strengthen the model's capacity for accurate generalisation.
- The ability of a model to respond to fresh data is referred to as generalisation. That is, a model can process fresh data and produce reliable predictions after being trained on a training set. A model's capacity for generalisation is essential to its effectiveness. A model won't be able to generalise if it has been trained too successfully on training data.
- Even while the model is capable of making accurate predictions for the training data, it will produce inaccurate predictions when presented with new data, rendering the model useless. Overfitting is the term for this. The opposite is also accurate. When a model has not been sufficiently trained on the data, underfitting occurs. Underfitting renders the model equally useless and prevents it from producing precise predictions, even using training data.

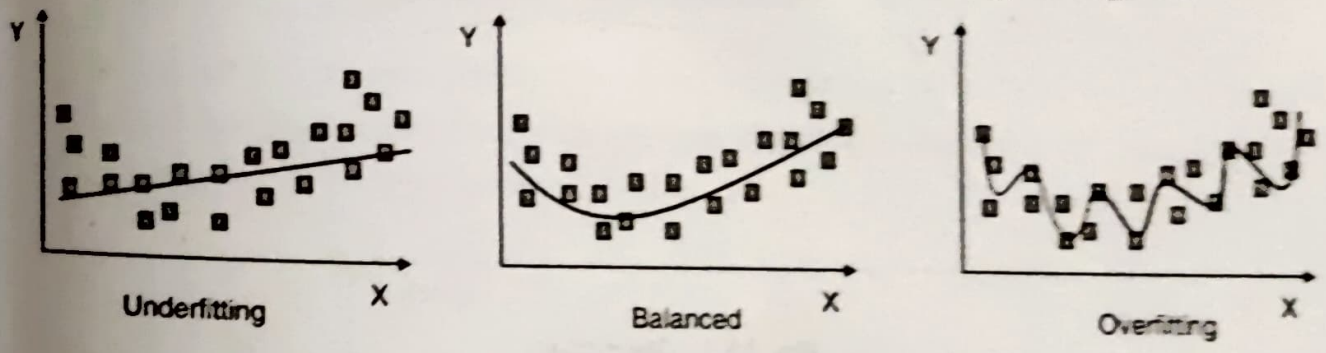



**Fig. 3.4.1 : Overfitting vs Underfitting**

- The graphic illustrates the three ideas that were previously mentioned. The blue line on the left depicts an underfitting model. Although the model detects a trend in the data, it is not detailed enough to include all of the essential details. It is unable to correctly forecast training or fresh data. The blue line in the centre denotes a balanced model. This model recognises a trend in the data and faithfully reproduces it. This middle model will successfully generalise.
- The blue line on the right depicts an overfitting model. The model accurately models the training data and detects a trend in the data, but it is overly precise. Because it learned the training data too well, it will be unable to generate reliable predictions with new data.

## 3.5 Dropout

### 3.5.1 Overfitting in Deep Learning

- In deep learning Overfitting of the model is one of most practical and real problem faced. The accuracy of the training data would be very high and that of the validation data would be very low when the model is overfitting because the model has learned all the patterns in the training data but is unable to generalise and make good predictions on data it has not seen before, defeating the entire purpose of training the model, which is to make predictions in the future on data.
- Our goal is to lessen overfitting, which occurs when our training and validation accuracies are near and high, suggesting that the model can do better predictions on data it has never seen before.

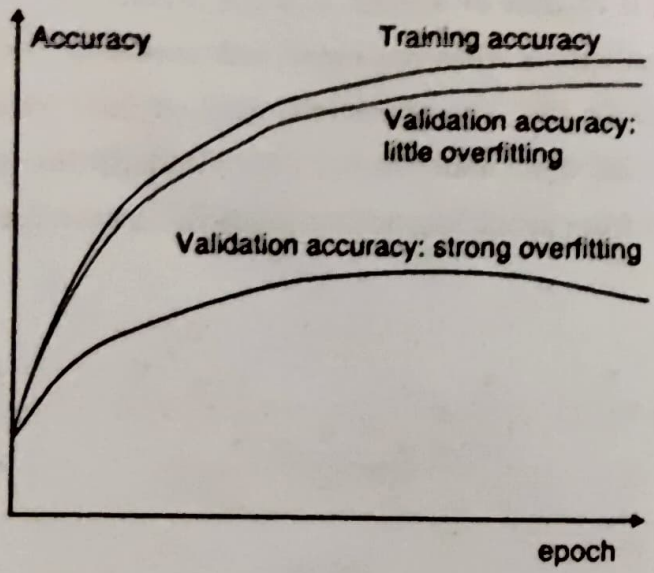



**Fig. 3.5.1 : Overfitting**

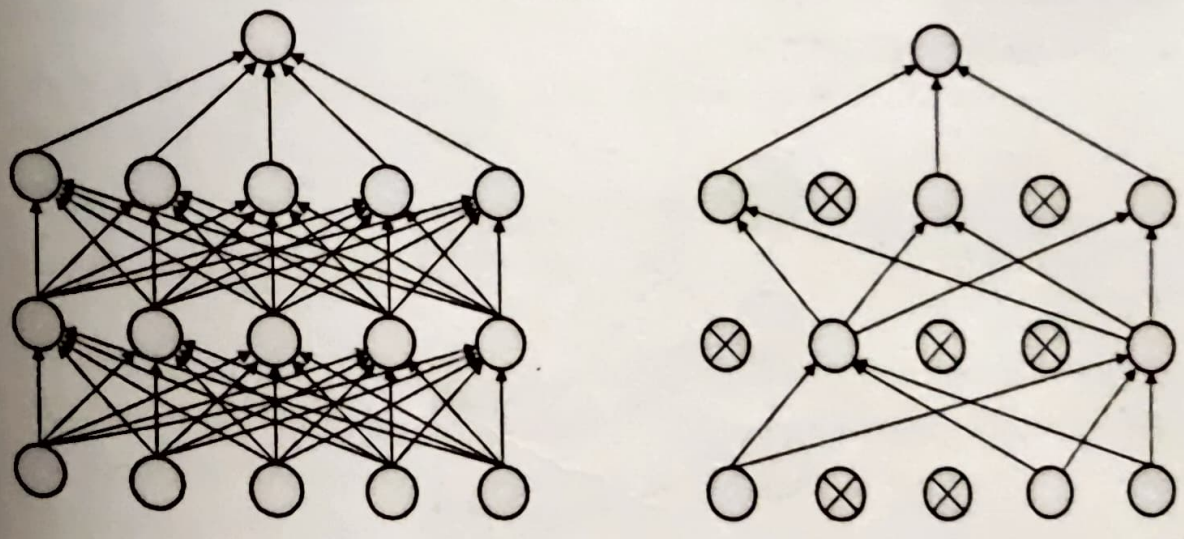
### 3.5.2 Overfitting Strategies

- There are several things we can do to reduce this overfitting whiles training our neural network models;
  1. We can reduce the complexity of our model (that is reducing the number of hidden layers),
  2. We could augment the data (that is increase the number of samples present in the dataset),
  3. We could also apply regularization, that is applying a penalty factor to the loss function (L1 or L2 regularization).

- These are all practical methods we can use to lessen overfitting in deep learning models, but they do not completely resolve the problem because, as we train the model iteratively, all the weights are learned together and some of the neurons adapt and make better predictions than others, leading to those neurons only taking part in training as the iteration moves forward. The stronger neurons learn more when the network is trained across numerous epochs, while the weaker ones are disregarded. After numerous iterations, just a portion of the neurons get trained at this point, and the weaker ones stay out of the training. As a result, we must discover a better solution, which is when the idea of dropout emerged.

### 3.5.3 Dropout Function

- During training, neurons are dropped from the neural network or "ignored" using the dropout approach, which means that various neurons are temporarily removed from the network.
- Dropout during training changes the plan of learning every weight in the network to learning just a portion of them. According to the image above, all neurons are active during the typical training phase, however after dropout, only a small subset of neurons are active while the rest are "shut off." In order to avoid one set of neurons from controlling the process, new sets of neurons are engaged after each cycle.
- As a result, we are able to lessen the threat of overfitting and promote the development of deeper and larger network topologies that are capable of making accurate predictions using data that the network has never seen before.



(a) Standard Neural Net (b) After applying dropout

**Fig. 3.5.2 : Dropout**

- The key benefit of this approach is that it stops all neurons in a layer from improving their weights simultaneously. This random adaptation keeps the neurons from convergent toward the same objective, decor relating the weights.

## 3.6 Unsupervised Training of Neural Network

Neural Networks can also be trained in an unsupervised way. Two of the most popular and widely used unsupervised deep neural network are Restricted Boltzmann Machines and Auto Encoders.

### 3.6.1 Restricted Boltzmann Machines

- The Restricted Boltzmann Machine, developed by Geoffrey Hinton in 1985, is a network of symmetrically connected units that functions like neurons and makes stochastic judgments.
- It is a type of unsupervised learning technique. It is helpful for collaborative filtering, feature learning, dimensionality reduction, regression, classification, and feature learning.
- Restricted Boltzmann Machines are stochastic two-layered neural networks that can automatically identify underlying patterns in data by reconstructing input. They are a subset of energy-based models. They have two layers, one of which is hidden. The hidden layer is made up of nodes that create the visible layer and collect feature information from the data. The output at the hidden layer is a weighted sum of the input layers. They don't have any output nodes, which might appear weird, and they don't have the typical binary output that allows for the learning of patterns.

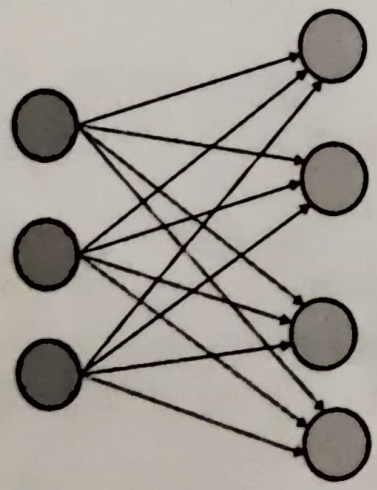



**Fig. 3.6.1 : Restricted Boltzmann Machines**

- They vary because learning cannot take place without that capacity. We don't concerned about hidden nodes; we only care about input nodes. RBMs automatically capture all the patterns, parameters, and relationships among the data once the input is given.
- A restricted term means that we are not permitted to connect two type layers that are of the same type to one another. In other words, the two input layer or hidden layer neurons are unable to form connections with one another. Although there may be connections between the apparent and hidden layers.
- Since there is no output layer in our machine, it is unclear how we will detect, modify the weights, and determine whether or not our prediction was right. One response fits all the questions: Restricted Boltzmann Machine.

**Benefits and Drawbacks of RBM Benefits**

**Advantages :**

1. Efficiently computed and expressive enough to encode any distribution.
2. Due to the limitations on connections between nodes, it is faster than a standard Boltzmann machine.
3. The hidden layer's activations can be included into other models as valuable features to boost performance.

**Disadvantages :**

1. Training is more challenging because it is challenging to calculate the Energy gradient function.
2. The back propagation algorithm is more well known than the CD-k algorithm, which is utilised in RBMs.
3. Weight Modification

### 3.6.2 Auto Encoders

- The input and output of feedforward neural networks that use autoencoders are identical. They reduce the input's dimension before using this representation to recreate the output. The code, also known as the latent-space representation, is an efficient "summary" or "compression" of the input.
- Encoder, code, and decoder are the three parts of an autoencoder. The input is compressed by the encoder, which also creates a code. The decoder then reconstructs the input exclusively using the code.

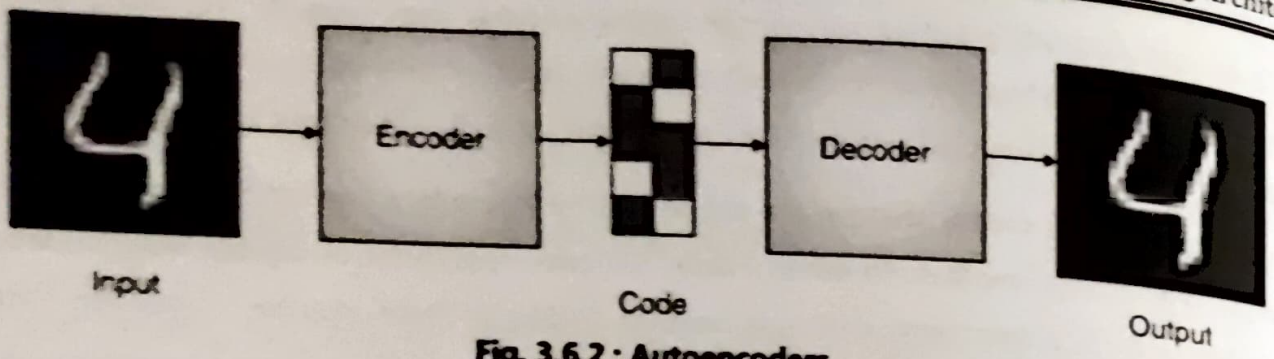



**Fig. 3.6.2 : Autoencoders**

- An encoding method, a decoding method, and a loss function to compare the output with the target are required in order to construct an autoencoder.
- Autoencoders primarily function as dimensionality reduction (or compression) algorithms and have the following key characteristics :

1. **Data Specific :** Autoencoders can only effectively compress data that is identical to the data they were trained on. They differ from a typical data compression technique like gzip because they learn features relevant to the provided training data. Therefore, it is unrealistic to expect a handwritten digit autoencoder to compress landscape photographs.
2. **Lossy :** The autoencoder's output won't be an exact match to the input; rather, it will be a somewhat degraded representation. They are not the best option if you desire lossless compression.
3. **Unsupervised :** We only need to feed the autoencoder the raw input data in order to train it. Since autoencoders don't require explicit labels to train on, they are regarded as an unsupervised learning technique. But since they create their own labels from the training data, they can actually be considered self-supervised.

- Let's examine the encoder, code, and decoder in more depth. The encoder and decoder, which are essentially ANNs, are both fully linked feedforward neural networks. A single layer of an ANN with the dimension of our choice is called code. A hyperparameter that we specify prior to training the autoencoder is the number of nodes in the code layer (also known as the code size).

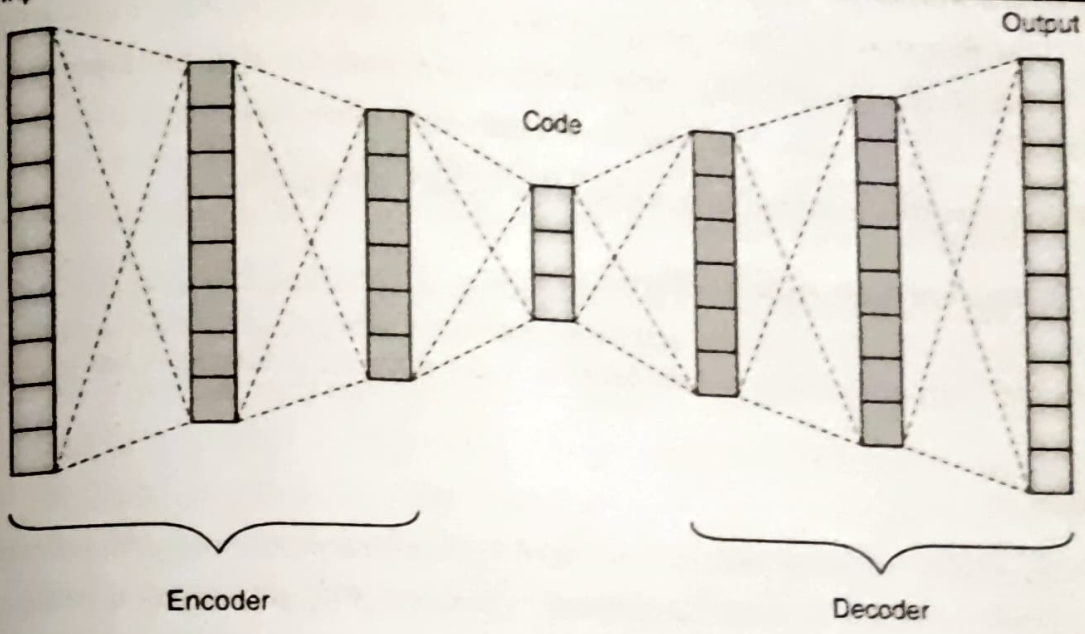



**Fig. 3.6.3 : Autoencoder Architecture**

- This illustrates an autoencoder in more depth. To create the code, the input first goes via the encoder, which is a fully-connected ANN. The output is subsequently produced solely using the code by the decoder, which has a similar ANN structure. Getting a result that matches the input is the aim. Keep in mind that the architecture of the encoder and decoder are identical. Although not required, this is frequently the case. The only prerequisite is that the input and output dimensions must match. Any object in the centre is playable.
- Before training an autoencoder, we must establish the following 4 hyperparameters :
  1. **Code Size :** Number of middle layer nodes is a measure of code size. More compression is achieved with smaller dimensions.
  2. **Number of layers :** We are free to choose the depth of the autoencoder. Without taking into account the input and output, the encoder and decoder in the aforementioned diagram both have two layers.
  3. **Number of nodes per layer :** The autoencoder design we're working on is referred to as a stacked autoencoder since the layers are placed one on top of the other. Stacks of autoencoders typically resemble switches. Each additional layer of the encoder results in fewer nodes per layer, which rises in the decoder. In terms of layer structure, the decoder and the encoder are also symmetric. Since we have complete control over these factors, as was previously mentioned, this is not essential.

4. **Loss Function :** We have two options for the loss function : binary cross entropy or mean squared error (mse). Cross entropy is commonly used if the input values fall within the [0, 1] range; otherwise, mean square error is employed.

- Autoencoders are trained the same way as ANNs via back-propagation.

## 3.7 Deep Learning Applications

There are many applications of Deep Learning some of the important ones have been listed below.

1. **Healthcare :**
   - One of the industries that has embraced contemporary technology most widely to transform itself is the healthcare industry. The fact that Deep Learning is being used to analyse medical data for prescription of medications, analysis of MRIs, CT scans, ECGs, and X-rays to identify and alert about medical anomalies, personalization of treatment, keeping track of patients' health, and more.
   - To rank various cancer cell types, medical professionals employ a CNN, or Convolutional Neural Network, a deep learning technique. They 20X or 40X-magnify high-res histopathology pictures before exposing them to deep CNN models. The deep CNN models then distinguish different cellular properties present in the sample and identify materials that are carcinogenic.
2. **Personalized Marketing :**
   - The idea of personalised marketing has been widely used in recent years. Marketers are now focusing their advertising campaigns on the needs of specific consumers and providing them with the solutions to their problems. And Deep Learning is crucially important in this.
   - Thanks to their use of social media platforms, Internet of Things (IoT) devices, online browsers, wearables, and other similar technologies, consumers today generate a lot of data. However, the majority of the data produced by these sources are inconsistent (text, audio, video, location data, etc.).
   - Businesses utilise adaptable Deep Learning models to evaluate data from many sources and distil it in order to derive insightful customer information. They then employ this data to forecast consumer behaviour and more effectively focus their marketing efforts.

### 3. Financial Fraud Detection :

- Almost no industry is immune to the evil known as "fraudulent transactions" or "financial fraud." However, the financial institutions (banks, insurance companies, etc.) are the ones who must deal with this threat's worst effects. Criminals target financial institutions every single day. There are numerous ways to hijack their financial resources.
- Therefore, identifying and anticipating financial fraud is, to put it mildly, crucial for these businesses. Deep Learning enters the scene in this situation.
- The idea of anomaly detection is currently being used by financial companies to indicate improper activities. To examine the patterns shared by valid transactions, they use deep learning techniques including decision trees, random forests, logistic regression, and decision trees (credit card fraud detection is one of their main use cases). These models are then used to detect financial transactions that appear to have a chance of being fraudulent.
- The following are some instances of fraud detection that deep learning has prevented :
  - Theft of identity and insurance fraud
  - Investment theft and appropriation of funds

### 4. Natural Language Processing :

- Another significant area where Deep Learning is demonstrating promising results is in NLP, or Natural Language Processing.
- The goal of natural language processing, as the name suggests, is to make it possible for computers to comprehend and analyse human language. The idea seems straightforward, right? But the truth is, machines have a terrible time understanding human language. The context, accents, handwriting, and other factors, in addition to the alphabet and words, prevent machines from accurately processing or producing human language.
- By teaching machines (Autoencoders and Distributed Representation) to respond appropriately to linguistic inputs, Deep Learning-based NLP eliminates many of the problems associated with interpreting human language.
- The personal assistants that we utilise on our smartphones are one such example. These applications include Deep Learning-infused Natural Language Processing (NLP) models to recognise human speech and produce the intended results. Therefore, it makes sense that Siri and Alexa sound so much like real people.

- The automatic translation of websites from one human language to another using Deep Learning-based NLP is another example.

**5. Autonomous Vehicles :**

- Since the first semi-automatic car was introduced by the Tsukuba Mechanical Engineering Laboratory 45 years ago, the idea of creating automated or self-governing vehicles has been around. The car, which at the time was a technological marvel, had two cameras and an analogue computer to drive itself down a specially constructed street.
- But it wasn't until 1989 when an altered military ambulance called ALVINN (Autonomous Land Vehicle in a Neural Network) used neural networks to find its way on roadways on its own.
- Since then, deep learning and autonomous vehicles have forged a close relationship, with the former significantly improving the performance of the latter.
- Cameras, sensors, including LiDARs, RADARs, and motion sensors, as well as outside data like geo-mapping, are used by autonomous cars to perceive their environment and gather pertinent information. They employ this gear both singly and collectively to record the data.
- After being provided with this information, deep learning algorithms lead the vehicle to take the proper actions, like steering, braking, and acceleration
- locating or arranging ways through the traffic, spotting other vehicles and pedestrians from a distance as well as up close, and recognising traffic signs The perceived goals of self-driving cars, such as minimising traffic accidents, assisting the disabled in driving, and eradicating traffic bottlenecks, are being realised in large part because to deep learning.
- Although still in their infancy, deep learning-powered vehicles will soon make up the majority of the traffic on the roads.

**6. Fake News Detection :**

- The idea of disseminating false information to sway public opinion is not new. Fake news has, however, become pervasive as a result of the internet's rapid popularity and social media platforms in particular.
- In addition to misleading the public, fake news can be used to sway political campaigns, demonise certain events and people, and engage in other immoral behaviours. Therefore, stopping all bogus news becomes a top concern.

- Deep Learning suggests a solution to the problem of false news by classifying phoney news sources using sophisticated language detection methods. This technique essentially involves obtaining data from reliable sources and comparing it to a piece of news to determine its veracity.
- Deep Learning algorithms like CNN and RNN have been effectively used for fake news detection.

**7. Facial Recognition :**

- With the aid of technology, people can be recognised from pictures and movies by having their faces captured. It records a person's face using cutting-edge biometric technology and compares it to a database to determine who they are.
- An old technology, facial recognition was first proposed in the 1960s. However, the inclusion of neural networks has significantly improved the detection accuracy of facial recognition.
- Face embeddings are recorded for Deep Learning enforced Facial Recognition, which then maps them against a massive database of millions of photos using a trained model.
- For instance, DeepFace, a facial recognition technique that uses deep learning (thus the name), has been shown to accurately identify people 97% of the time. It has been trained using four million photos of roughly 4000 people and a nine-layer neural network.

**8. Recommendation Systems :**

- Have you ever paused to consider how services like Spotify or Netflix can accurately propose content based on your preferences? Deep Learning, in brief, is the answer. The lengthy response is still deep learning but includes more details.
- As was previously described, deep learning models assemble user data gathered from various sources and process it to extract consumer information. Deep learning-based recommender systems employ this data to create choices for users that are appropriate.
- Despite being frequently employed by audio/video streaming services, deep learning-enabled suggestions are not solely confined to them. Similar methods are used by social media platforms to suggest posts, videos, accounts, and other content to people in their feeds.

**9. Smart Agriculture :**

- Agriculture is one of the businesses and fields that artificial intelligence and its subcategories are strengthening.

- A growing agricultural effort to enhance different facets of conventional agriculture is known as "smart farming." Nowadays, farmers monitor and improve their farming practises utilising IoT devices, satellite-based soil-composition detection, GPS, remote sensing, etc.
- To enhance crop and soil health, forecast the weather, find disease outbreaks, and other purposes, deep learning algorithms collect and analyse agricultural data from the aforementioned sources.
- Crop genomics is another area in which deep learning is useful. To ascertain the genetic make-up of various crop plants, experts employ neural networks, and they use it for things like improving resistance to diseases and natural disasters, and raising crop yield per acre by creating superior hybrids.

**10. Space Travel :**

- The majority of us equate space travel with the most cutting-edge technology at our disposal. We envision humanoid robots, highly intelligent artificial intelligences (AIs), cutting-edge machinery, etc., toiling tirelessly in space to support the astronauts in their arduous endeavours.
- Even though most of this is ridiculous, it does highlight one element of space travel: its high technological demands.
- To assure the security, reliability, and accomplishment of space missions, scientists and engineers must use the most up-to-date and effective technology, including both hardware and software.
- It follows that AI, Machine Learning, and Deep Learning are essential elements of anything related to astronomy.
- ESA, for example, claims that Deep Learning can be utilised (and, to some extent, is) in automating the rocket's landing, developing autonomous, intelligent space flight systems that do not require human intervention.
- Deep learning will also actively assist future Mars rovers in better and more autonomously navigating and determining their environment.

## REVIEW QUESTIONS

**Q. 1** Write Short note on Neural Network.

**Q. 2** What are Activation Functions?

Q. 3 List and explain different types of Activation Functions

Q. 4 What is Batch Normalization and how does it work?

Q. 5 Explain what is Underfitting and Overfitting?

Q. 6 Explain the importance of Dropout function.

Q. 7 Write Short note on RBM.

Q. 8 Write Short note on Auto Encoders.

Q. 9 List and Explain different Applications of Deep Learning.

❑❑❑

# 4 Computer Vision

UNIT - IV

**Syllabus**

Architectural Overview, Motivation, Layers, Filters, Parameter sharing, Regularization, Convolution neural networks (CNNs), convolution, pooling and its variations, different deep CNN architectures - LeNet, AlexNet, VGG, PlacesNet, DenseNet, Training a CNNs: weights initialization, batch normalization, hyperparameter tuning. Popular CNN Architectures : ResNet, AlexNet - Applications.

## 4.1 Architectural Overview

- Computer Vision is defined as a field of study that seeks to develop techniques to help computers "see" and understand the content of digital images such as photographs and videos".
- In other words, CV is a branch of Artificial Intelligence that enables computers to comprehend visual assets. Further, digital images and videos use deep learning models to build machine accuracy to classify objects. It also develops the ability of the computer to "see" and react to the visuals.
- Deep Learning Architectures for Computer Vision offer multi-layer components to execute tasks. Moreover, it enables neural networks to prioritize any image's pivotal features and aspects. Further, Deep Learning architectures are the ideal solutions for Computer Vision as it helps resolve complex problems.
- In this chapter, we will see different types of Convolution Neural Network (CNN) architectures, convolution, pooling and how to trained a CNNs.

## 4.2 Motivation

- In today's world, Computer Vision technologies are everywhere. They are embedded within many of the tools and applications that we use on a daily basis. However, we often pay little attention to those underlaying Computer Vision technologies because they tend to run in the background. As a result, only a small fraction of those outside the tech industries know about the importance of those technologies.

- Computer vision needs lots of data. It runs analyses of data over and over until it discerns distinctions and ultimately recognize images. For example, to train a computer to recognize automobile tires, it needs to be fed vast quantities of tire images and tire-related items to learn the differences and recognize a tire, especially one with no defects.
- Two essential technologies are used to accomplish this : a type of machine learning called deep learning and a Convolutional Neural Network (CNN).
- Convolutional networks (LeCun, 1989), also known as convolutional neural networks or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology. Examples include time-series data, which can be thought of as a 1D grid taking samples at regular time intervals, and image data, which can be thought of as a 2D grid of pixels. Convolutional networks have been tremendously successful in practical applications. The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

## 4.3 Layers

**The different layers of a CNN**

There are four types of layers for a convolutional neural network : the convolutional layer, the pooling layer, the ReLU correction layer and the fully-connected layer.

### 4.3.1 The Convolutional Layer

- The convolutional layer is the most important component of convolutional neural networks, and it is always at least the first layer.
- Its objective is to find a certain collection of features in the photographs supplied as input. Convolution filtering is used to do this. The basic idea is to "drag" a window representing the feature onto the image, calculate the convolution product between the feature and each section of the scanned image, and then apply the result to the entire image. The two concepts are equal in this context and a feature is then considered as a filter.
- As a result, the convolutional layer processes several images and computes the convolution of each image with each filter. The features we're looking for in the photographs are precisely what the filters match.

- We get for each pair (image, filter) a feature map, which tells us where the features are in the image : the higher the value, the more the corresponding place in the image resembles the feature.

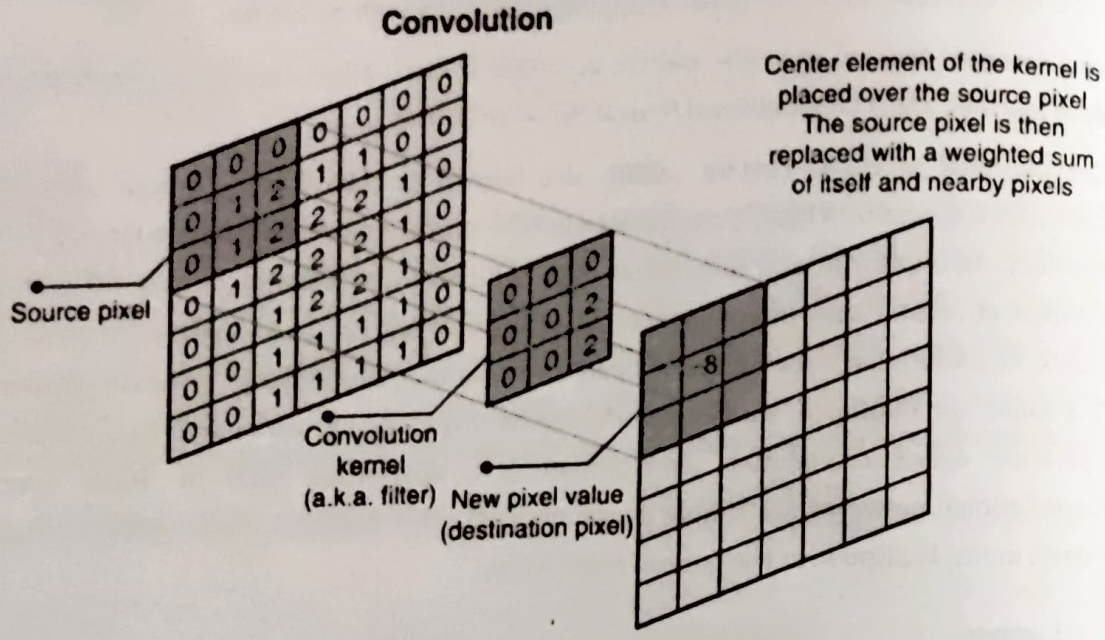



**Fig. 4.3.1 : Convolutional Layer**

- Unlike traditional methods, features are not pre-defined according to a particular formalism (for example SIFT), but learned by the network during the training phase! Filter kernels refer to the convolution layer weights. They are initialized and then updated by backpropagation using gradient descent.

## 4.3.2 The Pooling Layer

- This type of layer is often placed between two layers of convolution : it receives several feature maps and applies the pooling operation to each of them.
- The pooling operation consists in reducing the size of the images while preserving their important characteristics.
- To do this, we cut the image into regular cells, then we keep the maximum value within each cell. In practice, small square cells are often used to avoid losing too much information. The most common choices are 2 × 2 adjacent cells that don't overlap, or 3 × 3 cells, separated from each other by a step of 2 pixels (thus overlapping).
- We get in output the same number of feature maps as input, but these are much smaller.

- The pooling layer reduces the number of parameters and calculations in the network. This improves the efficiency of the network and avoids over-learning.
- The maximum values are spotted less accurately in the feature maps obtained after pooling than in those received in input this is a big advantage! For example, when you want to recognize a dog, its ears do not need to be located as precisely as possible: knowing that they are located almost next to the head is enough!

### 4.3.3 The ReLU Correction Layer

- ReLU (Rectified Linear Units) refers to the real non-linear function defined by ReLU(x)=max(0, x). Visually, it looks like the following :

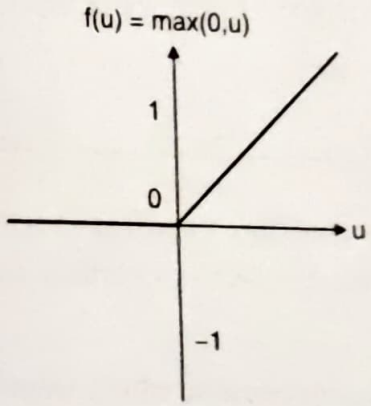



**Fig. 4.3.2 : non-linear function**

- The ReLU correction layer replaces all negative values received as inputs by zeros. It acts as an activation function.

### 4.3.4 The Fully-connected Layer

- The fully-connected layer is always the last layer of a neural network, convolutional or not so it is not characteristic of a CNN.
- This type of layer receives an input vector and produces a new output vector. To do this, it applies a linear combination and then possibly an activation function to the input values received.
- The last fully-connected layer classifies the image as an input to the network: it returns a vector of size N, where N is the number of classes in our image classification problem. Each element of the vector indicates the probability for the input image to belong to a class.

- To calculate the probabilities, the fully-connected layer, therefore, multiplies each input element by weight, makes the sum, and then applies an activation function (logistic if N=2, softmax if N>2). This is equivalent to multiplying the input vector by the matrix containing the weights. The fact that each input value is connected with all output values explains the term fully-connected.
- The convolutional neural network learns weight values in the same way as it learns the convolution layer filters: during the training phase, by backpropagation of the gradient.
- The fully connected layer determines the relationship between the position of features in the image and a class. Indeed, the input table being the result of the previous layer, it corresponds to a feature map for a given feature: the high values indicate the location (more or less precise depending on the pooling) of this feature in the image. If the location of a feature at a certain point in the image is characteristic of a certain class, then the corresponding value in the table is given significant weight.

## 4.4 Filters

- A filter provides a measure for how close a patch or a region of the input resembles a feature. A feature may be any prominent aspect – a vertical edge, a horizontal edge, an arch, a diagonal, etc.
- A filter acts as a single template or pattern, which, when convolved across the input, finds similarities between the stored template & different locations/regions in the input image.
- Let us consider an example of detecting a vertical edge in the input image.
- Each column of the 4 × 4 output matrix looks at exactly three columns & three rows (the coloured boxes show the output of the filter as it moves over the input image). The values in the output matrix represent the change in the intensity along the horizontal direction w.r.t the columns in the input image.
- The output image has the value 0 in the 1st & last column. It means there is no change in intensity in the first three columns & the previous three columns of the input image. On the other hand, the output is 30 in the 2nd & 3rd column, indicating a change in the intensity of the corresponding columns of the input image.

## 4.5 Parameter Sharing

- Parameter sharing refers to using the same parameter for more than one function in a model. In a traditional neural net, each element of the weight matrix is used exactly once when

computing the output of a layer. It is multiplied by one element of the input and then never revisited. As a synonym for parameter sharing, one can say that a network has tied weights, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere. In a convolutional neural net, each member of the kernel is used at every position of the input (except perhaps some of the boundary pixels, depending on the design decisions regarding the boundary).

- The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set. This does not affect the runtime of forward propagation it is still $O(k \times n)$ but it does further reduce the storage requirements of the model to k parameters. Recall that k is usually several orders of magnitude less than m. Since m and n are usually roughly the same size, k is practically insignificant compared to $m \times n$. Convolution is thus dramatically more efficient than dense matrix multiplication in terms of the memory requirements and statistical efficiency. For a graphical depiction of how parameter sharing works, see Fig. 4.5.1.

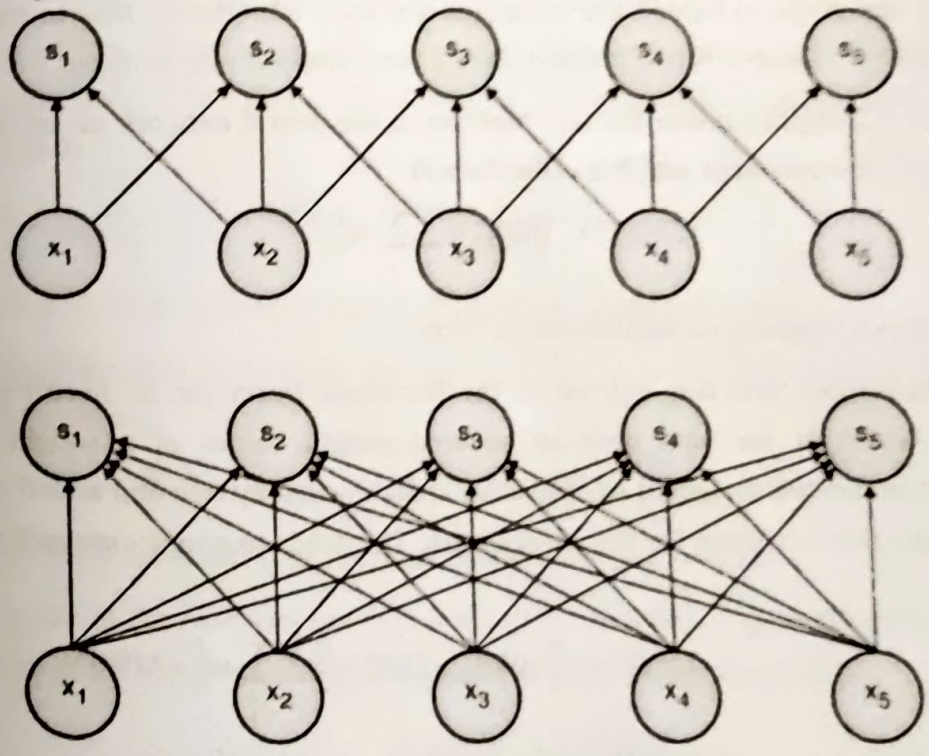



**Fig. 4.5.1 : Parameter sharing**

- In above Fig. 4.5.1, black arrows indicate the connections that use a particular parameter in two different models. (Top)The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Due to parameter sharing, this single parameter is used at all input locations. (Bottom) The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing so the parameter is used only once.

## 4.6 Regularization

- Regularization refers to a set of different techniques that lower the complexity of a neural network model during training, and thus prevent the overfitting.
- There are three very popular and efficient regularization techniques called L1, L2, and dropout which we are going to discuss in the following.

### 4.6.1 L2 Regularization

- The L2 regularization is the most common type of all regularization techniques and is also commonly known as weight decay or Ride Regression.
- The mathematical derivation of this regularization, as well as the mathematical explanation of why this method works at reducing overfitting, is quite long and complex. Since this is a very practical article I don't want to focus on mathematics more than it is required. Instead, I want to convey the intuition behind this technique and most importantly how to implement it so you can address the overfitting problem during your deep learning projects.
- During the L2 regularization the loss function of the neural network as extended by a so-called regularization term, which is called here Ω.

$$\Omega(W) = ||W||_2^2 = \sum_i \sum_j w_{ij}^2 \qquad ...(4.6.1)$$

- Equation (4.6.1) represents Regularization Term.
- The regularization term Ω is defined as the Euclidean Norm (or L2 norm) of the weight matrices, which is the sum over all squared weight values of a weight matrix. The regularization term is weighted by the scalar alpha divided by two and added to the regular loss function that is chosen for the current task. This leads to a new expression for the loss function :

$$\hat{L}(W) = \frac{\alpha}{2}||W||_2^2 + L(W) = \frac{\alpha}{2}\sum_i \sum_j w_{ij}^2 + L(W) \qquad ...(4.6.2)$$

- Equation (4.6.2) represents Regularization loss during L2 regularization.
- Alpha is sometimes called as the regularization rate and is an additional hyperparameter we introduce into the neural network. Simply speaking alpha determines how much we regularize our model.
- In the next step we can compute the gradient of the new loss function and put the gradient into the update rule for the weights :

$$\nabla_w \hat{L}(W) = \alpha W + \nabla_w L(W)$$

$$W_{new} = W_{old} - \epsilon(\alpha W_{old} + \nabla_w L(W_{old})) \quad ...(4.6.3)$$

- Equation (4.6.3) represents Gradient Descent during L2 Regularization.
- Some reformulations of the update rule lead to the expression which very much looks like the update rule for the weights during regular gradient descent :

$$W_{new} = (1 - \epsilon\alpha) W_{old} - \epsilon \nabla_W L(W_{old}) \quad ...(4.6.4)$$

- Equation (4.6.4) represents Gradient Descent during L2 Regularization.
- The only difference is that by adding the regularization term we introduce an additional subtraction from the current weights (first term in the equation).In other words independent of the gradient of the loss function we are making our weights a little bit smaller each time an update is performed.

### 4.6.2 L1 Regularization

- In the case of L1 regularization (also known as Lasso regression), we simply use another regularization term Ω. This term is the sum of the absolute values of the weight parameters in a weight matrix :

$$\Omega(W) = ||W||_1 = \sum_i \sum_j |w_{ij}| \quad ...(4.6.5)$$

- Equation (4.6.5) represents Regularization Term for L1 Regularization.
- As in the previous case, we multiply the regularization term by alpha and add the entire thing to the loss function.

$$\hat{L}(W) = \alpha||W||_1 + L(W) \quad ...(4.6.6)$$

- Equation (4.6.6) represents Loss function during L1 Regularization.
- The derivative of the new loss function leads to the following expression, which the sum of the gradient of the old loss function and sign of a weight value times alpha.

$$\nabla_w \hat{L}(W) = \alpha \, sign(W) + \nabla_w L(W) \quad ...(4.6.7)$$

- Equation (4.6.7) represents Gradient of the loss function during L1 Regularization.

### 4.6.3 Dropout

- In addition to the L2 and L1 regularization, another famous and powerful regularization technique is called the dropout regularization. The procedure behind dropout regularization is quite simple.

- In a nutshell, dropout means that during training with some probability P a neuron of the neural network gets turned off during training. Let's look at a visual example.

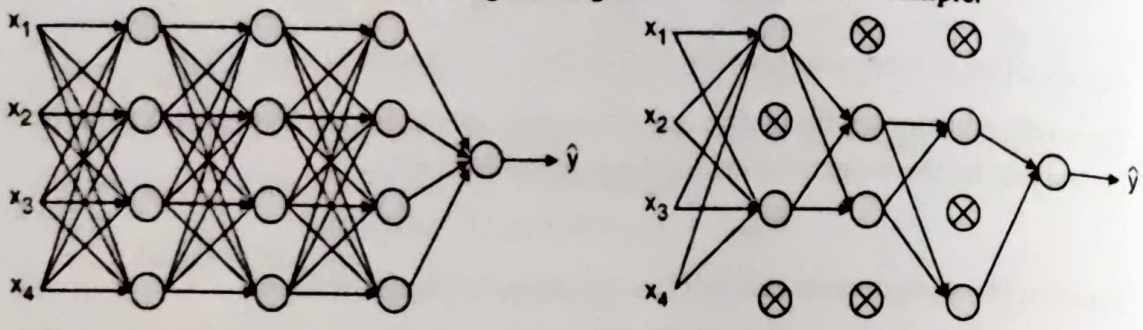



**Fig. 4.6.1 : Neural Network with dropout (right) and without (left).**

- Assume on the left side we have a feedforward neural network with no dropout. Using dropout with let's say a probability of P = 0.5 that a random neuron gets turned off during training would result in a neural network on the right side.
- In this case, you can observe that approximately half of the neurons are not active and are not considered as a part of the neural network. And as you can observe the neural network becomes simpler.
- A simpler version of the neural network results in less complexity that can reduce overfitting. The deactivation of neurons with a certain probability P is applied at each forward propagation and weight update step.

## 4.7 Convolution Neural Networks (CNNs)

- Convolution Neural Networks or covnets are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (as images generally have red, green, and blue channels).

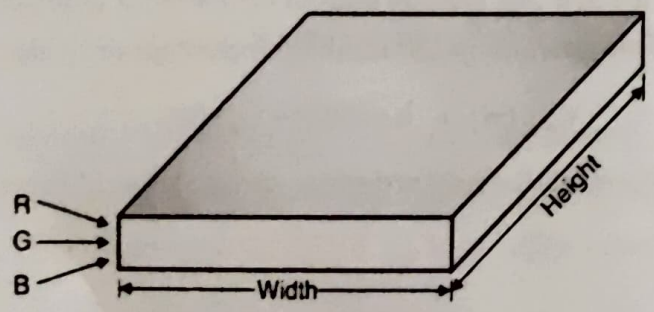



**Fig. 4.7.1 : Cuboid image**

- Now imagine taking a small patch of this image and running a small neural network on it, with say, k outputs and represent them vertically. Now slide that neural network across the

whole image, as a result, we will get another image with different width, height, and depth. Instead of just R, G, and B channels now we have more channels but lesser width and height. This operation is called Convolution. If the patch size is the same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.

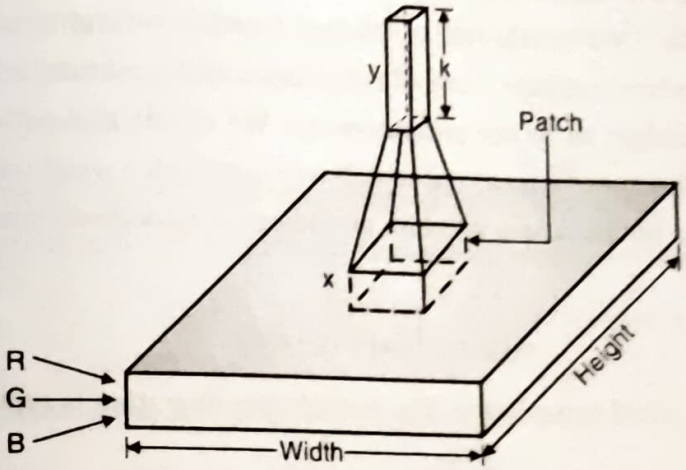



**Fig. 4.7.2**

- Now let's talk about a bit of mathematics that is involved in the whole convolution process.
  - Convolution layers consist of a set of learnable filters (a patch in the above image). Every filter has small width and height and the same depth as that of input volume (3 if the input layer is image input).
  - For example, if we have to run convolution on an image with dimension 34 × 34 × 3. The possible size of filters can be a × a × 3, where 'a' can be 3, 5, 7, etc but small as compared to image dimension.
  - During forward pass, we slide each filter across the whole input volume step by step where each step is called stride (which can have value 2 or 3 or even 4 for high dimensional images) and compute the dot product between the weights of filters and patch from input volume.
  - As we slide our filters we'll get a 2-D output for each filter and we'll stack them together and as a result, we'll get output volume having a depth equal to the number of filters. The network will learn all the filters.

## 4.8 Convolution

- In its most general form, convolution is an operation on two functions of a real valued argument. To motivate the definition of convolution, we start with examples of two functions we might use.

- Suppose we are tracking the location of a spaceship with a laser sensor. Our laser sensor provides a single output x(t), the position of the spaceship at time t. Both x and t are real-valued, i.e., we can get a different reading from the laser sensor at any instant in time.
- Now suppose that our laser sensor is somewhat noisy. To obtain a less noisy estimate of the spaceship's position, we would like to average together several measurements. Of course, more recent measurements are more relevant, so we will want this to be a weighted average that gives more weight to recent measurements. We can do this with a weighting function w(a), where a is the age of a measurement. If we apply such a weighted average operation at every moment, we obtain a new function providing s a smoothed estimate of the position of the spaceship :

$$s(t) = \int x(a)\, w(t-a)\, da$$

- This operation is called convolution. The convolution operation is typically denoted with an asterisk :

$$S(t) = (x * w)(t)$$

- In our example, w needs to be a valid probability density function, or the output is not a weighted average. Also, w needs to be 0 for all negative arguments, or it will look into the future, which is presumably beyond our capabilities.
- These limitations are particular to our example though. In general, convolution is defined for any functions for which the above integral is defined and may be used for other purposes besides taking weighted averages.
- In convolutional network terminology, the first argument (in this example, the function x) to the convolution is often referred to as the input and the second argument (in this example, the function w) as the kernel. The output is sometimes referred to as the feature map.
- In our example, the idea of a laser sensor that can provide measurements at every instant in time is not realistic. Usually, when we work with data on a computer, time will be discretized, and our sensor will provide data at regular intervals. In our example, it might be more realistic to assume that our laser provides a measurement once per second. The time index t can then take on only integer values. If we now assume that x and w are defined only on integer t, we can define the discrete convolution :

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a) w(t-a)$$

- In machine learning applications, the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm. We will refer to these multidimensional arrays as tensors. Because each element of the input and kernel must be explicitly stored separately, we usually assume that these functions are zero everywhere but the finite set of points for which we store the values. This means that in practice we can implement the infinite summation as a summation over a finite number of array elements.
- Finally, we often use convolutions over more than one axis at a time. For example, if we use a two-dimensional image I as our input, we probably also want to use a two-dimensional kernel K :

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n)$$

- Convolution is commutative, meaning we can equivalently write :

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n)$$

- Usually the latter formula is more straightforward to implement in a machine learning library, because there is less variation in the range of valid values of m and n.
- The commutative property of convolution arises because we have flipped the kernel relative to the input, in the sense that as m increases, the index into the input increases, but the index into the kernel decreases. The only reason to flip the kernel is to obtain the commutative property.

## 4.9 Pooling and It's Variations

- Pooling is a layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.
- The pooling layer summarises the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarised features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

**Types of Pooling Layers :**

1. Max Pooling
2. Average Pooling
3. Global Pooling

### 4.9.1 Max Pooling

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

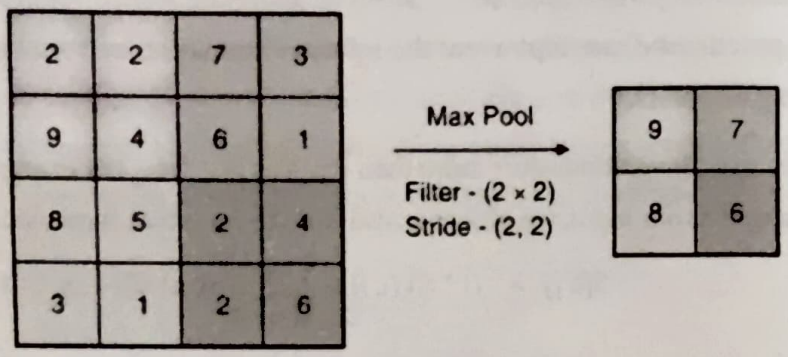



**Fig. 4.9.1 : Max Pooling**

### 4.9.2 Average Pooling

Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.

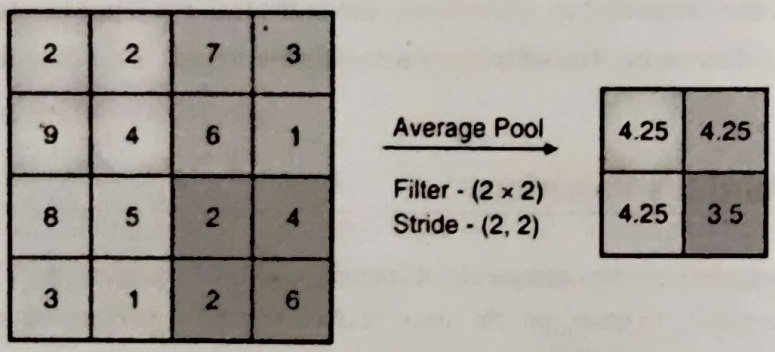



**Fig. 4.9.2 : Average Pooling**

### 4.9.3 Global Pooling

- Global pooling reduces each channel in the feature map to a single value. Thus, an $n_h \times n_w \times n_c$ feature map is reduced to $1 \times 1 \times n_c$ feature map. This is equivalent to using a filter of dimensions $n_h \times n_w$ i.e. the dimensions of the feature map.
- Further, it can be either global max pooling or global average pooling.

## 4.10 Different Deep CNN Architectures

The CNN architectures are the most popular deep learning framework. CNNs are used for a variety of applications, ranging from computer vision to natural language processing. The following is a list of different types of CNN architectures :

### 4.10.1 LeNet

- LeNet is the first CNN architecture. It was developed in 1998 by Yann LeCun, Corinna Cortes, and Christopher Burges for handwritten digit recognition problems. LeNet was one of the first successful CNNs and is often considered the "Hello World" of deep learning. It is one of the earliest and most widely-used CNN architectures and has been successfully applied to tasks such as handwritten digit recognition. The LeNet architecture consists of multiple convolutional and pooling layers, followed by a fully-connected layer.
- The model has five convolution layers followed by two fully connected layers. LeNet was the beginning of CNNs in deep learning for computer vision problems. However, LeNet could not train well due to the vanishing gradients problem. To solve this issue, a shortcut connection layer known as max-pooling is used between convolutional layers to reduce the spatial size of images which helps prevent overfitting and allows CNNs to train more effectively. The diagram below represents LeNet-5 architecture.

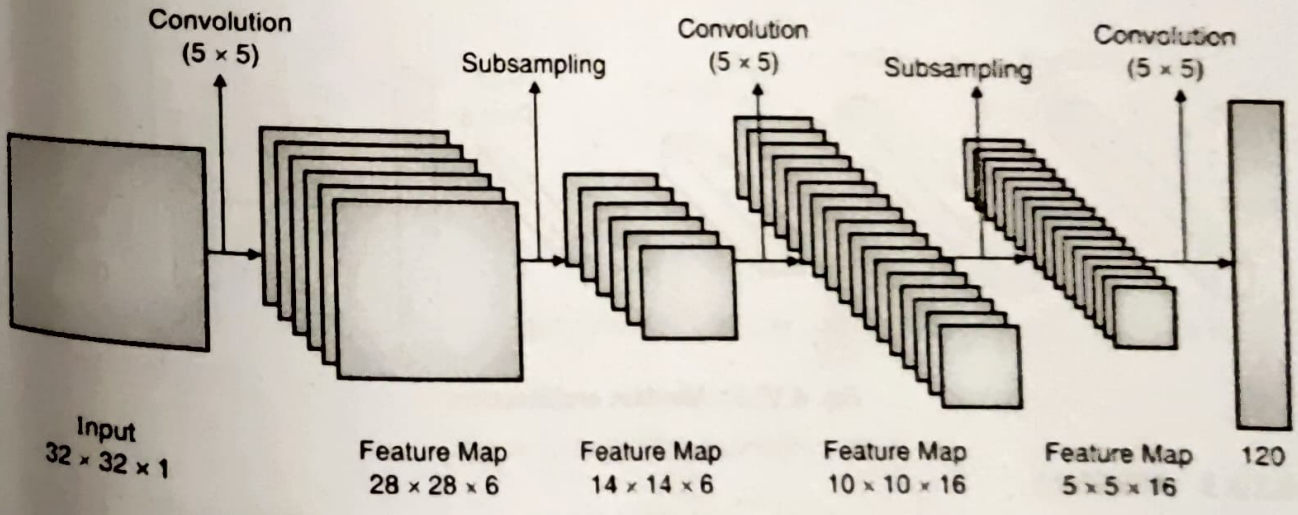



**Fig. 4.10.1 : LeNet-5 architecture**

- The LeNet CNN is a simple yet powerful model that has been used for various tasks such as handwritten digit recognition, traffic sign recognition, and face detection. Although LeNet was developed more than 20 years ago, its architecture is still relevant today and continues to be used.

### 4.10.2 AlexNet

- AlexNet is the deep learning architecture that popularized CNN. It was developed by Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. AlexNet network had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other. AlexNet was the first large-scale CNN and was used to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012.
- The AlexNet architecture was designed to be used with large-scale image datasets and it achieved state-of-the-art results at the time of its publication. AlexNet is composed of 5 convolutional layers with a combination of max-pooling layers, 3 fully connected layers, and 2 dropout layers. The activation function used in all layers is Relu. The activation function used in the output layer is Softmax. The total number of parameters in this architecture is around 60 million.

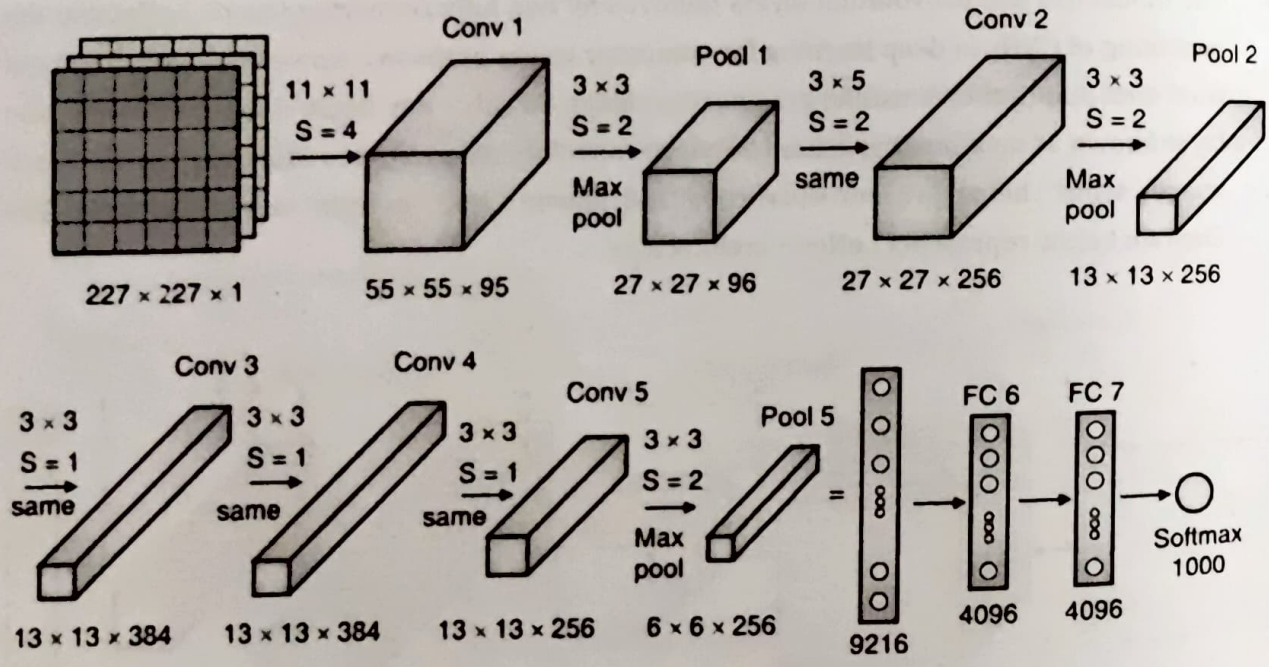



**Fig. 4.10.2 : AlexNet architecture**

### 4.10.3 VGGNet

- VGGNet is the CNN architecture that was developed by Karen Simonyan, Andrew Zisserman et al. at Oxford University. VGGNet is a 16-layer CNN with up to 95 million parameters and trained on over one billion images (1000 classes). It can take large input images of 224 x 224-pixel size for which it has 4096 convolutional features. CNNs with such large filters are

expensive to train and require a lot of data, which is the main reason why CNN architectures like GoogLeNet (AlexNet architecture) work better than VGGNet for most image classification tasks where input images have a size between 100 x 100-pixel and 350 x 350 pixels.

- Real-world applications/examples of VGGNet CNN architecture include the ILSVRC 2014 classification task, which was also won by GoogleNet CNN architecture. The VGG CNN model is computationally efficient and serves as a strong baseline for many applications in computer vision due to its applicability for numerous tasks including object detection. Its deep feature representations are used across multiple neural network architectures like YOLO, SSD, etc. The diagram below represents the standard VGG16 network architecture diagram :

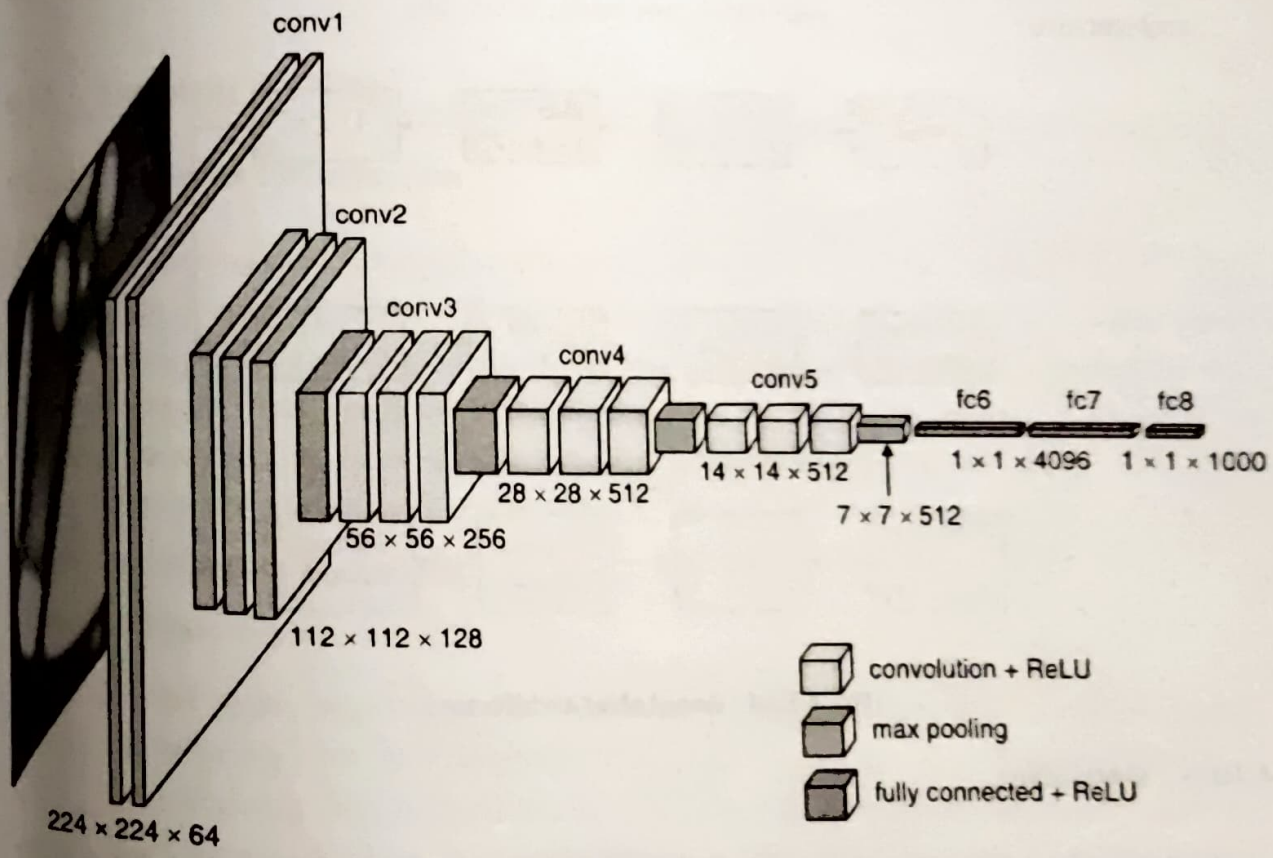



**Fig. 4.10.3 : VGGNet architecture**

## 4.10.4 PlacesNet

- PlacesNet is also called as GoogLeNet. GoogLeNet is the CNN architecture used by Google to win ILSVRC 2014 classification task. It was developed by Jeff Dean, Christian Szegedy, Alexandro Szegedyet al. It has been shown to have a notably reduced error rate in

comparison with previous winners AlexNet (Ilsvrc 2012 winner) and ZF-Net (Ilsvrc 2013 winner). In terms of error rate, the error is significantly lesser than VGG (2014 runner up).

- It achieves deeper architecture by employing a number of distinct techniques, including 1 × 1 convolution and global average pooling. GoogleNet CNN architecture is computationally expensive. To reduce the parameters that must be learned, it uses heavy unpooling layers on top of CNNs to remove spatial redundancy during training and also features shortcut connections between the first two convolutional layers before adding new filters in later CNN layers. Real-world applications/examples of GoogLeNet CNN architecture include Street View House Number (SVHN) digit recognition task, which is often used as a proxy for roadside object detection. Below is the simplified block diagram representing GoogLeNet CNN architecture :

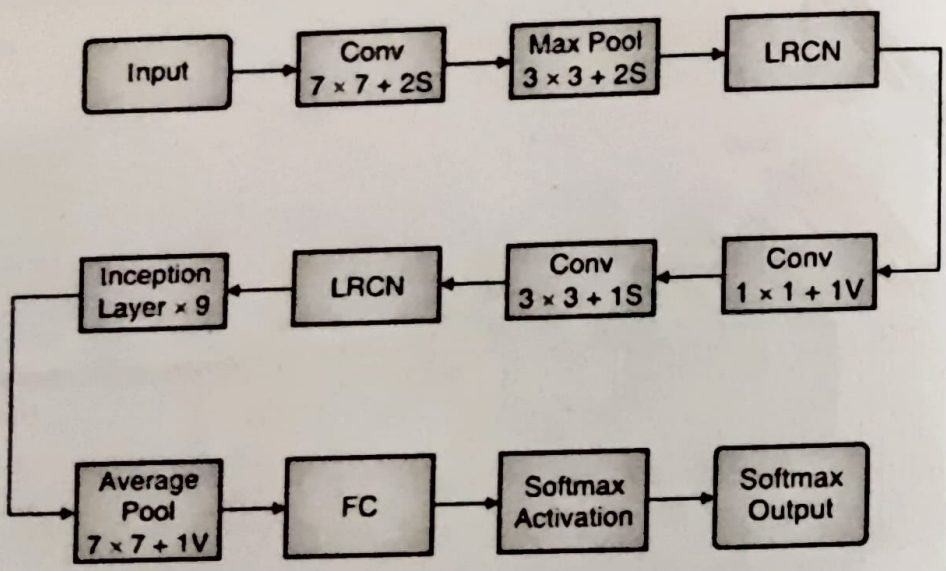



**Fig. 4.10.4 : GoogLeNet architecture**

## 4.10.5 DenseNet

In ResNet, we added the stacked layer along with its input layer. In DenseNet, for a given layer, all other layers preceding to it are concatenated and given as input to the current layer. With such an arrangement, we can use smaller filter counts and also, this will minimize the vanishing gradient problem as all layers are directly connected to the output, gradients can be calculated directly from the output for each layer.

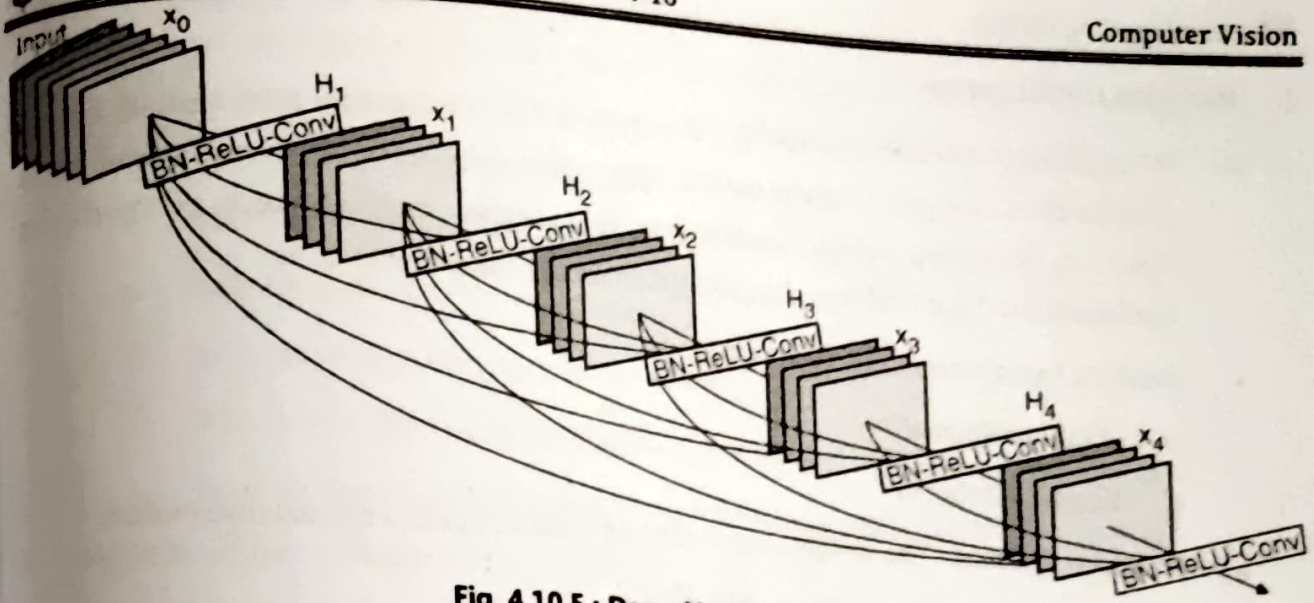



**Fig. 4.10.5 : DenseNet architecture**

## 4.11 Training a CNNs

### 4.11.1 Weights Initialization

While building and training neural networks, it is crucial to initialize the weights appropriately to ensure a model with high accuracy. If the weights are not correctly initialized, it may give rise to the Vanishing Gradient problem or the Exploding Gradient problem. Hence, selecting an appropriate weight initialization strategy is critical when training DL models. Here we will see some of the most common weight initialization techniques, along with their implementation in Python using Keras in TensorFlow.

**Weight Initialization Techniques**

1. **Zero Initialization**

- As the name suggests, all the weights are assigned zero as the initial value is zero initialization. This kind of initialization is highly ineffective as neurons learn the same feature during each iteration. Rather, during any kind of constant initialization, the same issue happens to occur. Thus, constant initializations are not preferred.
- Zero initialization can be implemented in Keras layers in Python as follows :
  - # Zero Initialization
  - from tensorflow.keras import layers
  - from tensorflow.keras import initializers
  - initializer = tf.keras.initializers.Zeros()
  - layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)

**2. Random Initialization**

- In an attempt to overcome the shortcomings of Zero or Constant Initialization, random initialization assigns random values except for zeros as weights to neuron paths. However, assigning values randomly to the weights, problems such as Overfitting, Vanishing Gradient Problem, Exploding Gradient Problem might occur.
- Random Initialization can be of two kinds:
  - Random Normal
  - Random Uniform

**a) Random Normal :** The weights are initialized from values in a normal distribution.

$$w_i \sim N(0, 1)$$

Random Normal initialization can be implemented in Keras layers in Python as follows :

```
# Random Normal Distribution
from tensorflow.keras import layers
from tensorflow.keras import initializers
initializer = tf.keras.initializers.RandomNormal(mean=0., stddev=1.)
layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)
```

**b) Random Uniform :** The weights are initialized from values in a uniform distribution.

$$w_i = N(0, 1)$$

Random Uniform initialization can be implemented in Keras layers in Python as follows :

```
# Random Uniform Initialization
from tensorflow.keras import layers
from tensorflow.keras import initializers
initializer = tf.keras.initializers.RandomUniform(minval=0.,maxval=1.)
layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)
```

**3. Xavier/Glorot Initialization**

- In Xavier/Glorot weight initialization, the weights are assigned from values of a uniform distribution as follows :

$$w_i \sim U\left[-\sqrt{\frac{\sigma}{fan\_in + fan\_out}}, \sqrt{\frac{\sigma}{fan\_in + fan\_out}}\right]$$

- Xavier/Glorot Initialization often termed as Xavier Uniform Initialization, is suitable for layers where the activation function used is Sigmoid. Xavier/Gorat initialization can be implemented in Keras layers in Python as follows :
  - # Xavier/Glorot Uniform Initialization
  - from tensorflow.keras import layers
  - from tensorflow.keras import initializers
  - initializer = tf.keras.initializers.GlorotUniform()
  - layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)

**4. Normalized Xavier/Glorot Initialization**

- In Normalized Xavier/Glorot weight initialization, the weights are assigned from values of a normal distribution as follows :

$$w_i \sim N(0, \sigma)$$

$$\sigma = \sqrt{\frac{6}{fan\_in + fan\_out}}$$

- Xavier/Glorot Initialization, too, is suitable for layers where the activation function used is Sigmoid. Normalized Xavier/Gorat initialization can be implemented in Keras layers in Python as follows :
  - # Normailzed Xavier/Glorot Uniform Initialization
  - from tensorflow.keras import layers
  - from tensorflow.keras import initializers
  - initializer = tf.keras.initializers.GlorotNormal()
  - layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)

**5. He Uniform Initialization**

- In He Uniform weight initialization, the weights are assigned from values of a uniform distribution as follows :

$$w_i \sim U\left[-\sqrt{\frac{6}{fan\_in}}, \sqrt{\frac{6}{fan\_out}}\right]$$

- He Uniform Initialization is suitable for layers where ReLU activation function is used. He Uniform Initialization can be implemented in Keras layers in Python as follows :
  - # He Uniform Initialization

- from tensorflow.keras import layers
- from tensorflow.keras import initializers
- initializer = tf.keras.initializers.HeUniform()
- layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)

**6. He Normal Initialization**

- In He Normal weight initialization, the weights are assigned from values of a normal distribution as follows :

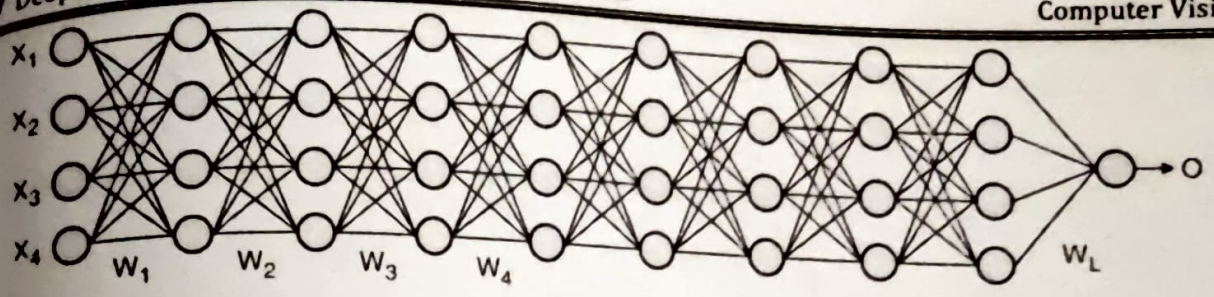
$$w_i \sim N(0, \sigma)$$

$$\sigma = \sqrt{\frac{2}{fan\_in}}$$

- He Uniform Initialization, too, is suitable for layers where ReLU activation function is used. He Uniform Initialization can be implemented in Keras layers in Python as follows:
  - # He Normal Initialization
  - from tensorflow.keras import layers
  - from tensorflow.keras import initializers
  - initializer = tf.keras.initializers.HeNormal()
  - layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)

## 4.11.2 Batch Normalization

- Normalization is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape.
- Generally, when we input the data to a machine or deep learning algorithm we tend to change the values to a balanced scale. The reason we normalize is partly to ensure that our model can generalize appropriately.
- A typical neural network is trained using a collected set of input data called batch. Similarly, the normalizing process in batch normalization takes place in batches, not as a single input, is called as Batch normalization.
- Let's understand this through an example, we have a deep neural network as shown in the Fig. 4.11.1.




L = Number of layers
Bias = 0
Activation Function : Sigmoid

**Fig. 4.11.1 : CNN Architecture**

- Initially, our inputs $X_1$, $X_2$, $X_3$, $X_4$ are in normalized form as they are coming from the pre-processing stage. When the input passes through the first layer, it transforms, as a sigmoid function applied over the dot product of input X and the weight matrix W.

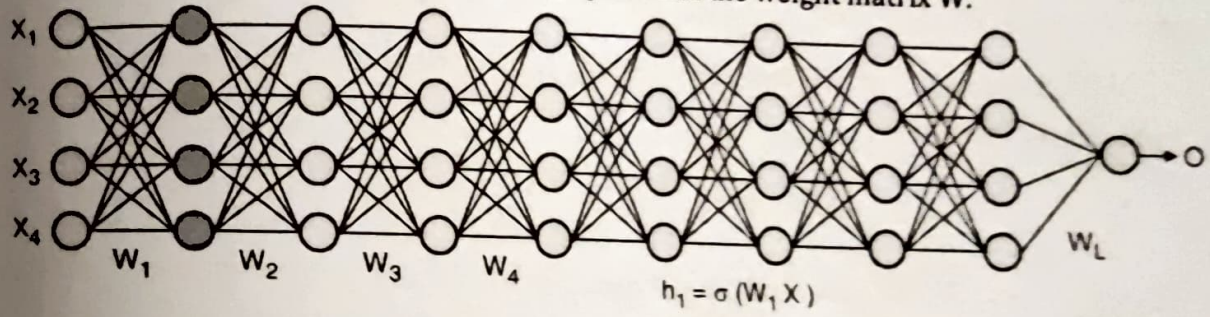



**Fig. 4.11.2 : Transformation on CNN Architecture**

- Similarly, this transformation will take place for the second layer and go till the last layer L as shown in the Fig. 4.11.3.

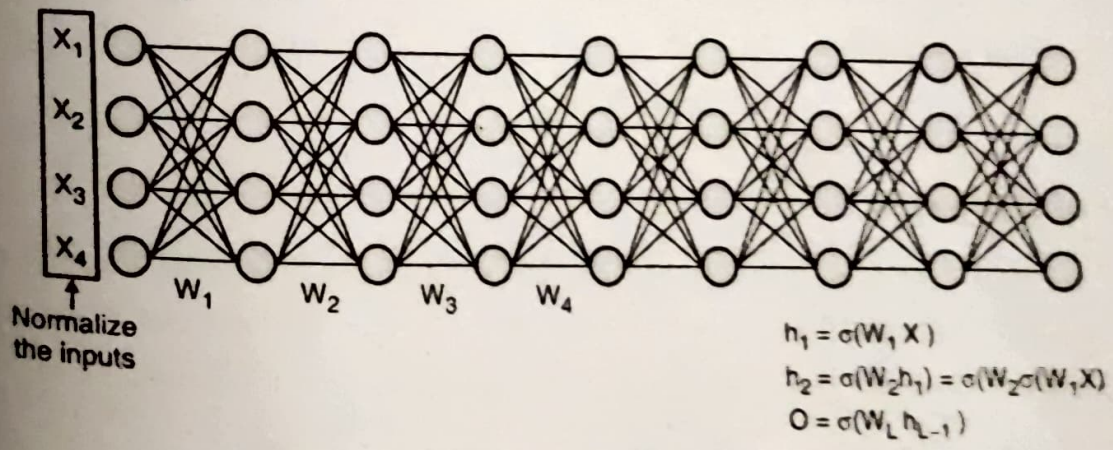



**Fig. 4.11.3 : Normalization on CNN Architecture**

- Although, our input X was normalized with time the output will no longer be on the same scale. As the data go through multiple layers of the neural network and L activation functions are applied, it leads to an internal co-variate shift in the data.

**How does Batch Normalization work?**

Since by now we have a clear idea of why we need Batch normalization, let's understand how it works. It is a two-step process. First, the input is normalized, and later rescaling and offsetting is performed.

**Normalization of the Input**

- Normalization is the process of transforming the data to have a mean zero and standard deviation one. In this step we have our batch input from layer h, first, we need to calculate the mean of this hidden activation.

$$\mu = \frac{1}{m}\sum h_i$$

- Here, m is the number of neurons at layer h.
- Once we have meant at our end, the next step is to calculate the standard deviation of the hidden activations.

$$\sigma = \left[\frac{1}{m}\sum (h_i - \mu)^2\right]^{1/2}$$

- Further, as we have the mean and the standard deviation ready. We will normalize the hidden activations using these values. For this, we will subtract the mean from each input and divide the whole value with the sum of standard deviation and the smoothing term (ε).
- The smoothing term (ε) assures numerical stability within the operation by stopping a division by a zero value.

$$h_{i(norm)} = \frac{(h_i - \mu)}{\sigma + \varepsilon}$$

**Rescaling of Offsetting**

- In the final operation, the re-scaling and offsetting of the input take place. Here two components of the BN algorithm come into the picture, γ (gamma) and β (beta). These parameters are used for re-scaling (γ) and shifting (β) of the vector containing values from the previous operations.

$$h_i = y\, h_{i(norm)} + \beta$$

- These two are learnable parameters, during the training neural network ensures the optimal values of γ and β are used. That will enable the accurate normalization of each batch.

### 4.11.3 Hyperparameter Tuning

- A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters.
- However, there is another kind of parameter, known as Hyperparameters, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.
- Some examples of model hyperparameters include :
  1. The penalty in Logistic Regression Classifier i.e. L1 or L2 regularization.
  2. The learning rate for training a neural network.
  3. The C and sigma hyperparameters for support vector machines.
  4. The k in k-nearest neighbors.
- Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. The two best strategies for Hyperparameter tuning are :
  - GridSearchCV
  - RandomizedSearchCV

### 4.11.3(A) GridSearchCV

- In GridSearchCV approach, the machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for the best set of hyperparameters from a grid of hyperparameters values.
- For example, if we want to set two hyperparameters C and Alpha of the Logistic Regression Classifier model, with different sets of values. The grid search technique will construct many versions of the model with all possible combinations of hyperparameters and will return the best one.
- As in the image, for C = [0.1, 0.2, 0.3, 0.4, 0.5] and Alpha = [0.1, 0.2, 0.3, 0.4]. For a combination of ***C = 0.3 and Alpha = 0.2***, the performance score comes out to be **0.726 (Highest)**, therefore it is selected.

| C | Alpha | | | |
|---|---|---|---|---|
| | 0.1 | 0.2 | 0.3 | 0.4 |
| 0.5 | 0.701 | 0.703 | 0.697 | 0.696 |
| 0.4 | 0.699 | 0.702 | 0.698 | 0.702 |
| 0.3 | 0.721 | 0.725 | 0.713 | 0.703 |
| 0.2 | 0.706 | 0.705 | 0.704 | 0.701 |
| 0.1 | 0.698 | 0.692 | 0.688 | 0.675 |

**Fig. 4.11.4 : GridSerachCV example**

```
# Necessary imports
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Creating the hyperparameter grid
c_space = np.logspace(-5, 8, 15)
param_grid = {'C': c_space}

# Instantiating logistic regression classifier
logreg = LogisticRegression()

# Instantiating the GridSearchCV object
logreg_cv = GridSearchCV(logreg, param_grid, cv = 5)

logreg_cv.fit(X, y)

# Print the tuned parameters and score
print("Tuned Logistic Regression Parameters: {}".format(logreg_cv.best_params_))
print("Best score is {}".format(logreg_cv.best_score_))
```

### 4.11.3(B) RandomizedSearchCV

- RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in a random fashion to find the best set of hyperparameters. This approach reduces unnecessary computation.
- The following code illustrates how to use RandomizedSearchCV

```
# Necessary imports
from scipy.stats import randint
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV

# Creating the hyperparameter grid
param_dist = {"max_depth": [3, None],
              "max_features": randint(1, 9),
              "min_samples_leaf": randint(1, 9),
              "criterion": ["gini", "entropy"]}

# Instantiating Decision Tree classifier
tree = DecisionTreeClassifier()

# Instantiating RandomizedSearchCV object
tree_cv = RandomizedSearchCV(tree, param_dist, cv = 5)

tree_cv.fit(X, y)
# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(tree_cv.best_params_))
print("Best score is {}".format(tree_cv.best_score_))
```

## 4.12 Popular CNN Architectures

### 4.12.1 ResNet

- ResNet is the CNN architecture that was developed by Kaiming He et al. to win the ILSVRC 2015 classification task with a top-five error of only 15.43%. The network has 152 layers and over one million parameters, which is considered deep even for CNNs because it would have taken more than 40 days on 32 GPUs to train the network on the ILSVRC 2015 dataset.
- CNNs are mostly used for image classification tasks with 1000 classes, but ResNet proves that CNNs can also be used successfully to solve natural language processing problems like sentence completion or machine comprehension, where it was used by the Microsoft Research Asia team in 2016 and 2017 respectively.
- Real-life applications/examples of ResNet CNN architecture include Microsoft's machine comprehension system, which has used CNNs to generate the answers for more than 100k questions in over 20 categories. The CNN architecture ResNet is computationally efficient and can be scaled up or down to match the computational power of GPUs.

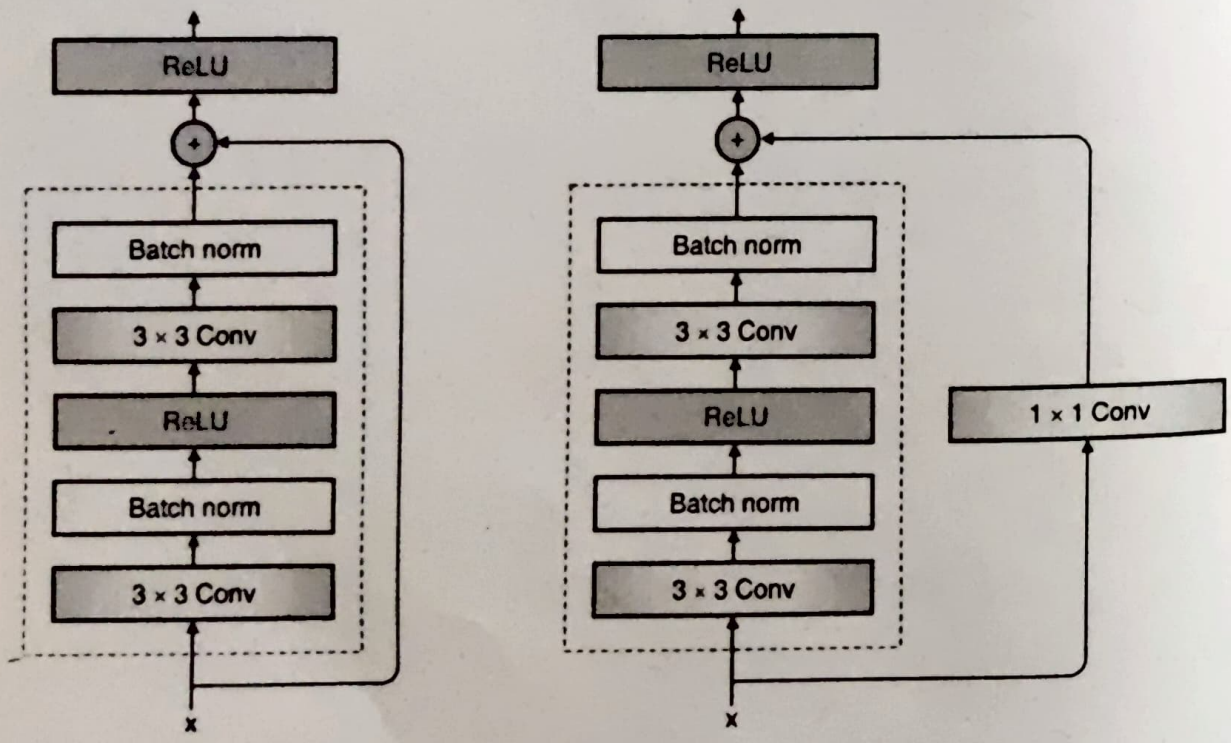



**Fig. 4.12.1 : ResNet Architecture**

## REVIEW QUESTIONS

Q. 1 Explain different types of CNN.

Q. 2 What is Filters?

Q. 3 Write a Short note on Parameter Sharing.

Q. 4 Explain Regularization with its variants.

Q. 5 Explain Pooling and its types in details.

Q. 6 Explain different CNN Architectures.

Q. 7 Explain in details batch normalization.

Q. 8 Which parameters are considering while training the CNN network?

Q. 9 Explain applications for ResNet.

Q. 10 Explain two strategies for Hyperparameter tuning.

❑❑❑

# 5 Natural Language Processing

UNIT - V

**Syllabus**

Recurrent Neural Networks, Bidirectional RNNs, Encoder-decoder sequence to sequence architectures - BPTT for training RNN, Long Short Term Memory Networks. Advanced RNN : LSTM, GRU, introduction to Generative Adversarial Networks (GANs).

## 5.1 Recurrent Neural Networks

- Recurrent Neural Networks (RNNs) are a type of neural network in which the results of one step are fed into the next step's computations. Traditional neural networks have inputs and outputs that are independent of one another, but there is a need to remember the previous words in situations where it is necessary to anticipate the next word in a sentence. As a result, RNN was developed, which utilised a Hidden Layer to resolve this problem. The Hidden state, which retains some information about a sequence, is the primary and most significant characteristic of RNNs.


**Fig. 5.1.1 : General RNN**

- RNNs have a "memory" that retains all data related to calculations. It executes the same action on all of the inputs or hidden layers to produce the output, using the same settings for each input. In contrast to other neural networks, this minimises the complexity of the parameter set.

### 5.1.1 How RNN Works

The working of a RNN can be understood with the help of below example :

Consider a deeper network that has three hidden layers, one output layer, one input layer, and three hidden levels. Each hidden layer will thus, like other neural networks, have its own set of weights and biases. For example, let's suppose that the weights and biases for hidden layer 1 are $(w_1, b_1)$, $(w_2, b_2)$ for the second hidden layer, and $(w_3, b_3)$ for the third hidden layer. This indicates that each of these layers is independent of the others and does not retain information from earlier outputs.

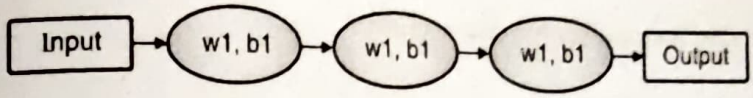



**Fig. 5.1.2 : RNN as Deep Neural Network**

The RNN will now perform the following :

- By giving all of the layers the same weights and biases, RNN transforms independent activations into dependent activations, decreasing the complexity of raising parameters and memorising each previous output by using each output as an input to the following hidden layer.
- Thus, all three layers can be combined into a single recurrent layer so that the weights and bias of all the hidden levels are the identical.

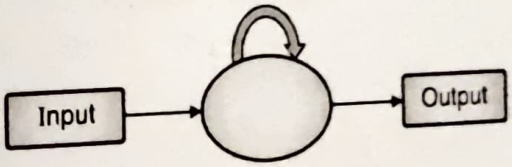



**Fig. 5.1.3(a) : RNN Structure**

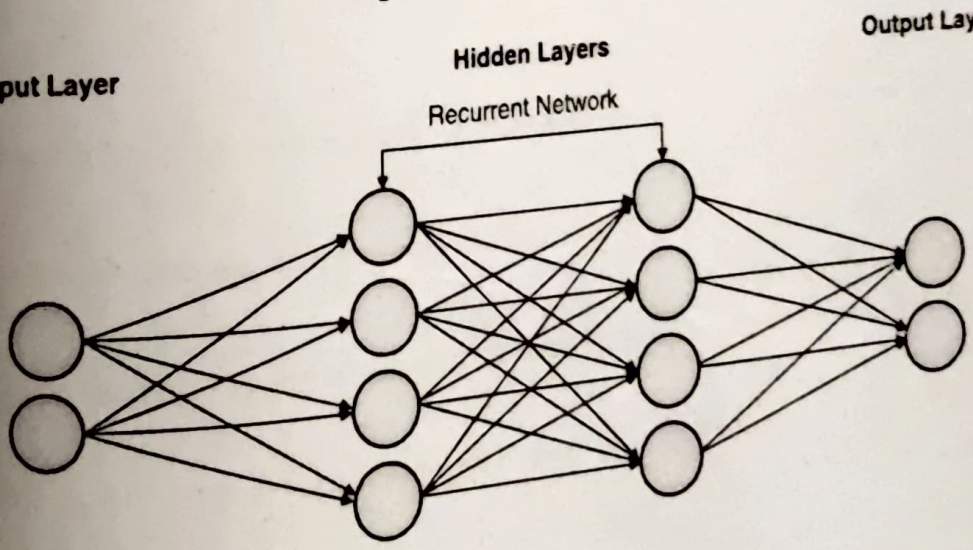



**Fig. 5.1.3(b) : Recurrent Network**

**Formula for Calculating current state**

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

where :

$h_t \rightarrow$ Current state

$h_{t-1} \rightarrow$ Previous state

$x_t \rightarrow$ Input state

**Formula for applying Activation function (tanh) :**

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

where :

$w_{hh} \rightarrow$ weight at recurrent neuron

$w_{xh} \rightarrow$ weight at input neuron

**Formula for calculating output :**

$$y_t = w_{hy} h_t$$

$Y_t \rightarrow$ output

$W_{hy} \rightarrow$ weight at output layer

### 5.1.2 Training

- The input is given to the network in a single time step.
- Then, using the set of current input and the prior state, calculate the present state of the system.
- For the following time step, the current time becomes time-1.
- Depending on the issue, one can travel back as many time steps and combine the data from all the prior states.
- The final current state is used to determine the output after all the time steps have been finished.
- The error is then generated once the output is compared to the goal output, which is the actual output.
- The network (RNN) is trained after the error is back-propagated to it in order to update the weights.

### 5.1.3 Recurrent Neural Networks : Why use them?

- There were a few problems with the feed-forward neural network, which led to the development of RNN :
  - can't deal with consecutive data
  - merely takes into account current input
  - unable to remember earlier inputs
- The RNN offers a remedy for these problems. An RNN can handle sequential data, accepting both the input data being used at the moment and inputs from the past. RNNs' internal memory allows them to remember prior inputs.

### 5.1.4 Working of each Recurrent Unit

- Take the current input vector and the previously hidden state vector as input. Keep in mind that each element of the vector is positioned in a different dimension that is orthogonal to the other dimensions because the hidden state and current input are both considered as vectors. As a result, when one element is multiplied by another, only non-zero elements in the same dimension and that element's own dimension provide non-zero values.
- The hidden state vector is multiplied by the hidden state weights element-wise, while the current input vector and current input weights are also multiplied element-wise. This produces the current input vector and the parameterized hidden state vector. Note that the trainable weight matrix contains weights for several vectors.
- To create the new hidden state vector, perform the vector addition of the two parameterized vectors and then compute the element-wise hyperbolic tangent.

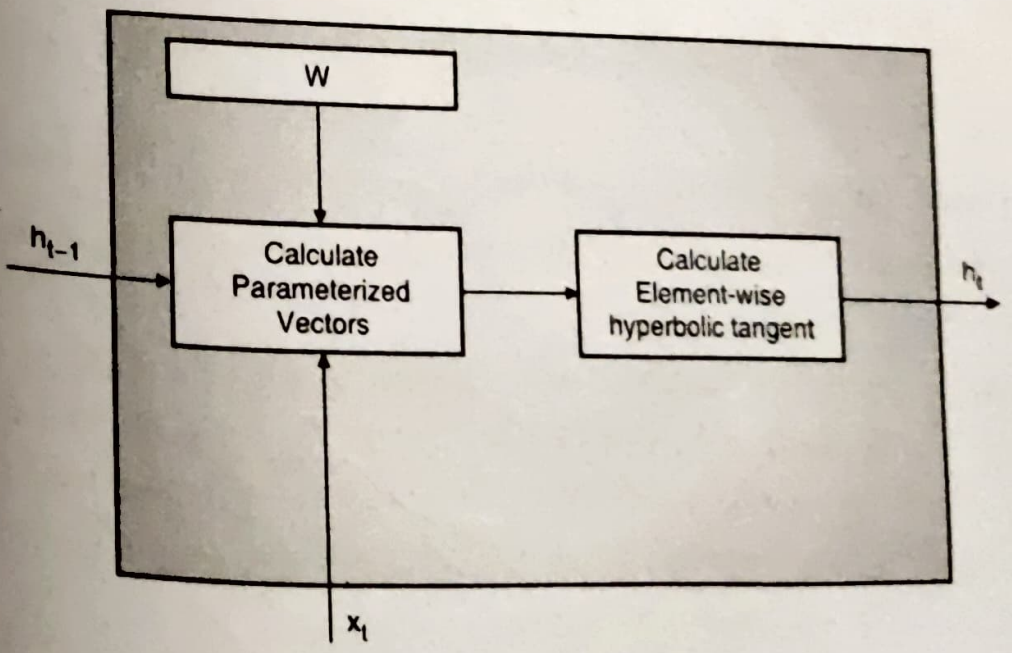



**Fig. 5.1.4 : Training of Recurrent Unit**

- The recurrent network produces an output at each time step while it is being trained. Gradient descent is used to train the network using this output.

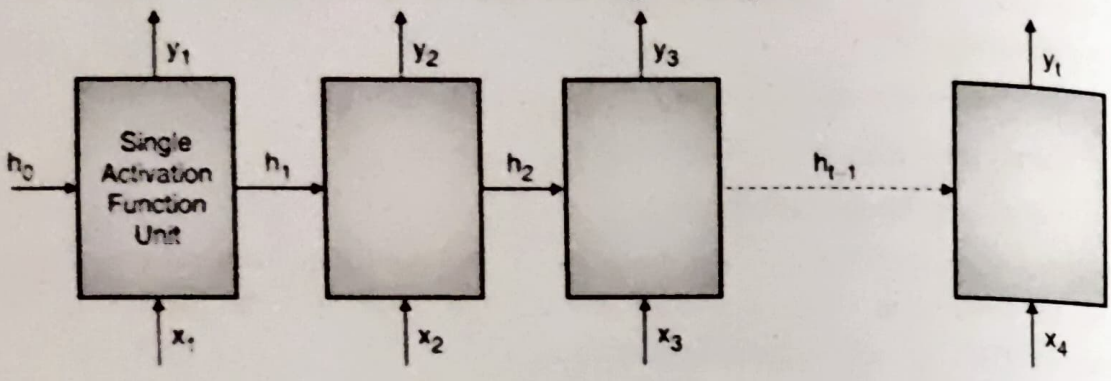



**Fig. 5.1.5 : Training RNN**

- With a few slight modifications, the Back-Propagation used here is comparable to the one used in a conventional Artificial Neural Network.

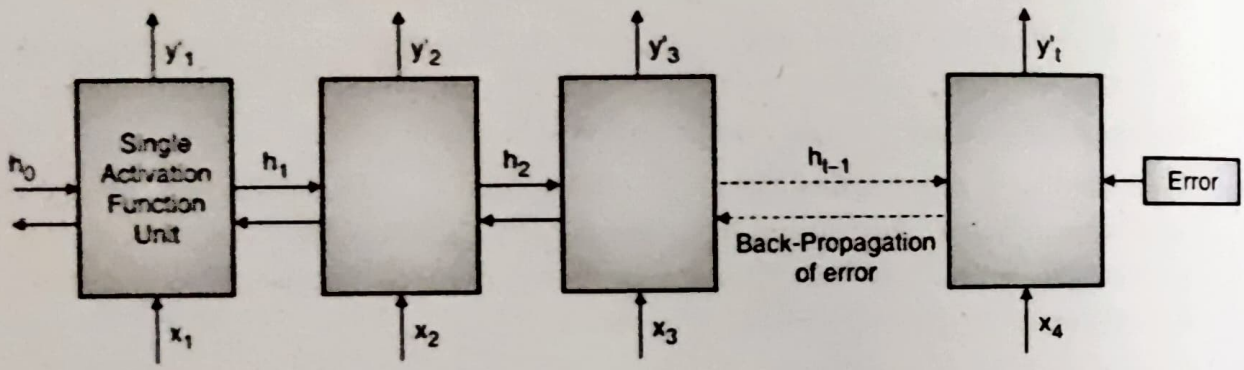



**Fig. 5.1.6 : Back Propagation in RNN**

- Although the fundamental Recurrent Neural Network is reasonably effective, it can have a serious issue. The back-propagation technique for deep networks can result in the following problems :
  - **Vanishing gradients :** These are gradients that are so small that they gravitate to zero.
  - **Exploding Gradients :** These happen when back-propagation causes the gradients to grow excessively large.
- By placing a threshold on the gradients being transported back in time, it may be possible to use a hack to overcome the Exploding Gradients problem. However, this technique is not regarded as resolving the issue and could harm the network's effectiveness. Long Short Term Memory Networks and Gated Recurrent Unit Networks, two key versions of Recurrent Neural Networks, were created to address these issues.

### 5.1.5 Advantages and Disadvantages

**Advantages**

- An RNN retains every piece of knowledge throughout time. Only the ability to remember past inputs makes it helpful for time series prediction. Long Short Term Memory is the term for this.
- Convolutional layers and recurrent neural networks are even combined to increase the effective pixel neighbourhood.

**Disadvantages**

- Problems with gradient disappearing and explosions.
- It is exceedingly tough to train an RNN.
- If tanh or relu are used as the activation function, it cannot process very long sequences.

## 5.2 Bidirectional Neural Networks

- Recurrent neural networks that are bidirectional are basically just two separate RNNs combined. For one network, the input sequence is fed in regular time order; for a different network, it is fed in reverse time order. At each time step, the outputs of the two networks are typically concatenated, however there are other choices, such as summation.
- The networks may access both forward and backward information about the sequence at each time step because to this structure. The idea sounds simple enough.
- But there remains uncertainty when it comes to actually putting a neural network with a bidirectional topology into practise.

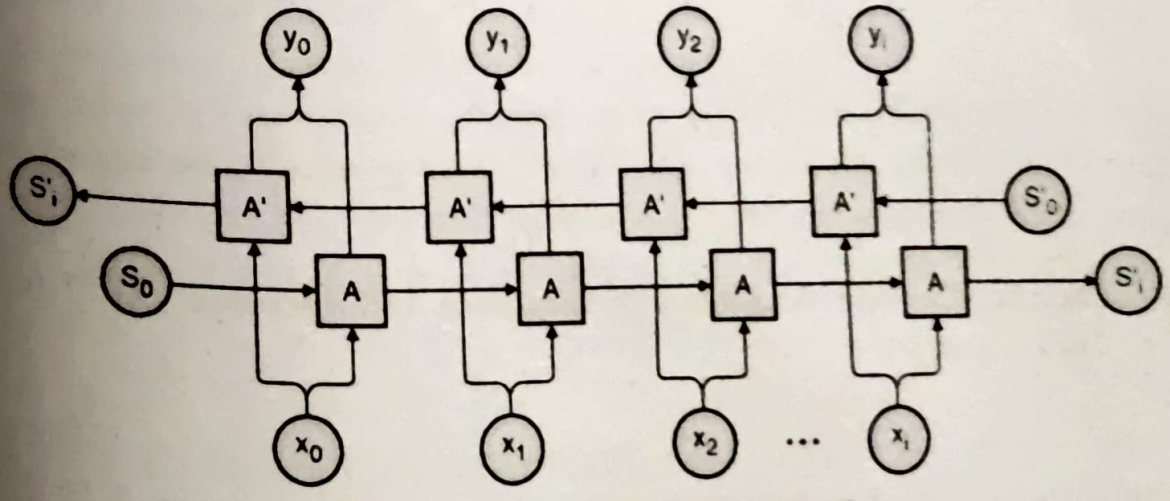



**Fig. 5.2.1 : Bidirectional Neural Network**

### 5.2.1 The Confusion

- The first area of uncertainty is how to send a bidirectional RNN's outputs to a dense neural network. The next image, which I obtained using Google, depicts a similar strategy on a bidirectional RNN, demonstrates how we might simply forward the outputs at the last time step for conventional RNNs.

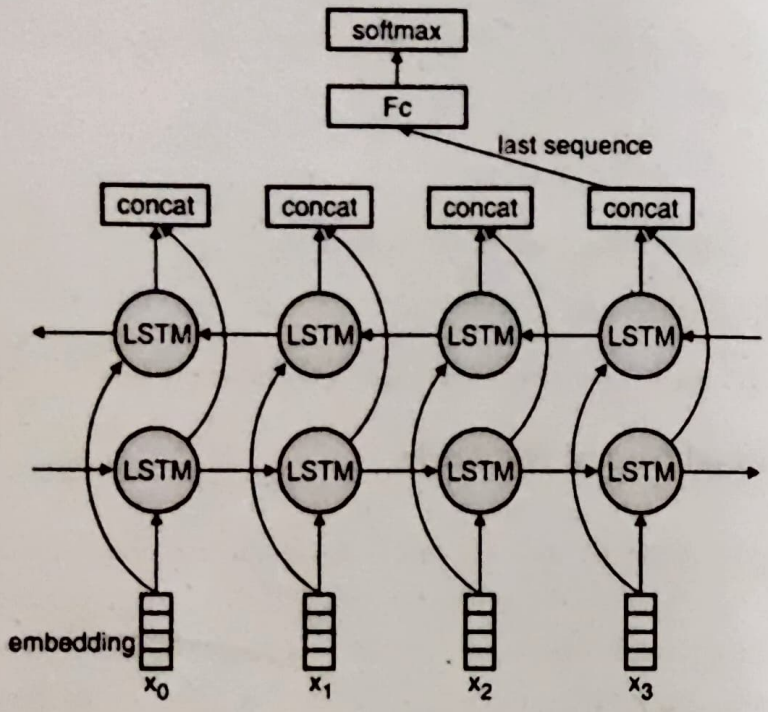



**Fig. 5.2.2 : Formulation of Bidirectional Neural Network**

- The reverse RNN will only have viewed the most recent input (x 3 in the illustration) if we choose the output at the final time step. It won't have much predictive power.
- The returning hidden states are the second area of uncertainty. We need the encoder's hidden states to initialise the decoder's hidden states in seq2seq models. It seems sense that, if we can only select hidden states at a single time step, we would prefer the state where the RNN has just finished consuming the most recent input in the sequence. We will, however, get the hidden states of the reversed RNN with only one step of inputs seen if the hidden states of time step n (the last one) are returned, as previously.

## 5.3 Encoder-Decoder Sequence to Sequence Architectures - BPTT for Training RNN

- Gradient descent is simply known under the fancy moniker of back propagation. Although it has some intriguing characteristics, the process is the same: all that needs to be done is calculate the gradient and move in that direction.

- Similar to gradient descent, BPTT (Back Propagation through Time), sometimes abbreviated as BPTT, is just another fancy acronym for back propagation. We refer to this as BPTT, or back propagation through time, because we are essentially changing the weights by travelling back in time (BPTT).
- In feed-forward when a neural network moves forward, the weights are updated via an error signal from any influencing nodes. The upward arrows coming into the output Y only relevant, if you're treating that as a part of output, as I have here indicated the influence of in purple. Whether you have a target for each time step or only a target at the conclusion of the sequence will affect how these are different. Given that these are the hidden to hidden weights, the arrows pointing from left to right must always be back-propagated.

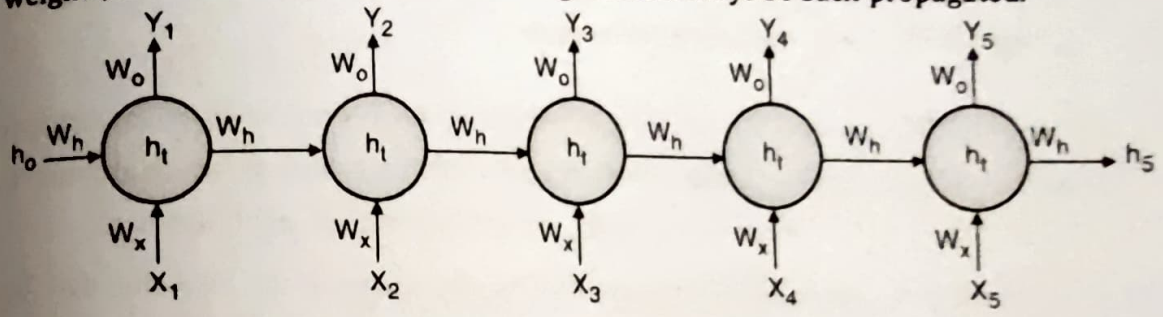



**Fig. 5.3.1 : Back Propagation through time (BPTT )**

- The Backpropagation Through Time (BPTT) technique applies the Backpropagation training method to recurrent neural networks that are trained on sequence data, such as time series.
- Each timestep, one input is presented to a recurrent neural network, which then predicts one output.
- The way BPTT operates conceptually is by unrolling each input timestep. One copy of the network, one output, and one input timestep are all present in each timestep. Then, errors are computed and totaled for each timestep. The weights are adjusted and the network is rolled back up.
- Given the order dependence of the problem and the fact that the internal state from the previous timestep is accepted as an input on the upcoming timestep, each timestep of the unrolled recurrent neural network may be spatially viewed as an additional layer.
- The algorithm can be summed up as follows :
  - Give the network a series of input and output pair timesteps.
  - Calculate the network after which you acquire errors over each timestep.
  - Update weights and roll-up the network.
  - Repeat.

- As the number of timesteps rises, BPTT may become computationally expensive.
- This is the number of derivatives needed for a single update weight update if input sequences are made up of input sequences with thousands of timesteps. Weights may disappear or explode (go to zero or overflow), making sluggish learning and model skill noisy as a result.

## 5.4 Advanced RNN

As mentioned earlier Vanishing gradients and Exploding Gradients are two major issues in recurrent neural network and to solve these issues Long Short Term Memory Networks and Gated Recurrent Unit have been put forward.

### 5.4.1 Long Short Term Memory Networks

- An enhanced RNN, or sequential network, called a long short-term memory network, permits information to endure. It is capable of resolving the RNN's vanishing gradient issue. RNNs, also referred to as recurrent neural networks, are utilised for persistent memory.
- RNN retain the prior knowledge and apply it to the processing of the incoming data. Due to diminishing gradient, RNN have the flaw of being unable to recall long-term dependencies. Long-term dependency issues are specifically avoided when designing LSTMs.

**LSTM Architecture :**

- LSTM functions on a high level very similarly to an RNN cell. The LSTM network's internal operation is seen below. As seen in the image below, the LSTM is composed of three sections, each of which has a distinct function.

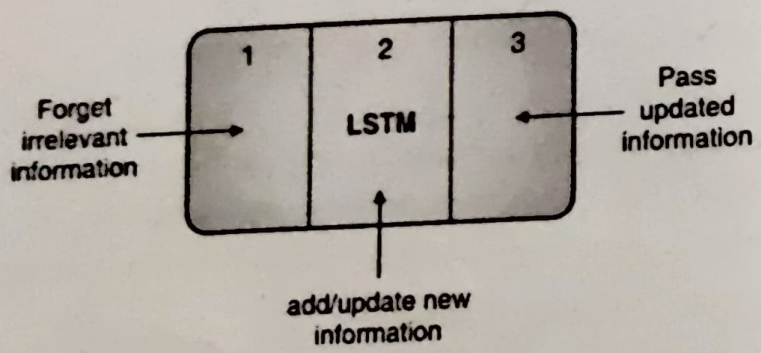



**Fig. 5.4.1 : LSTM Components**

- The first section determines whether the information from the preceding timestamp needs to be remembered or can be ignored. The cell attempts to learn new information from the input to this cell in the second section. The cell finally transmits the revised data from the current timestamp to the next timestamp in the third section.

- Gates refer to these three LSTM cell components. The Forget gate, Input gate, and Output gate are the names of the three components, respectively.

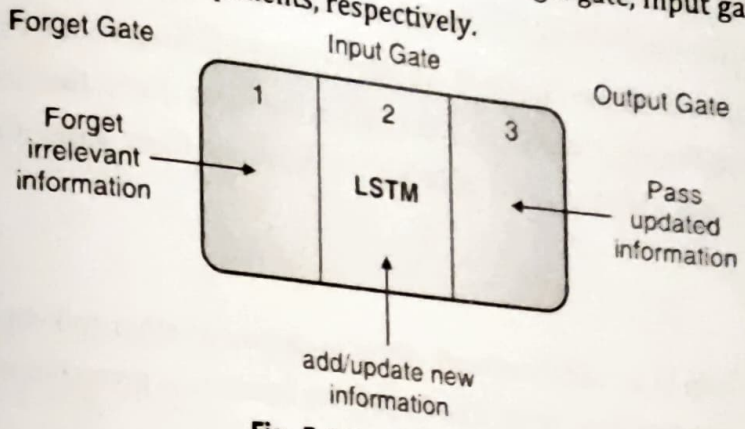



**Fig. 5.4.2 : LSTM Gates**

- An LSTM has a hidden state, just like a straightforward RNN, with H(t-1) standing for the hidden state of the prior timestamp and Ht for the hidden state of the present timestamp. Additionally, LSTMs have a cell state that is denoted by the timestamps C(t-1) and C(t), which stand for the prior and current timestamps, respectively.
- In this case, the cell state is referred to as the long-term memory and the hidden state as the short-term memory. See the illustration below.

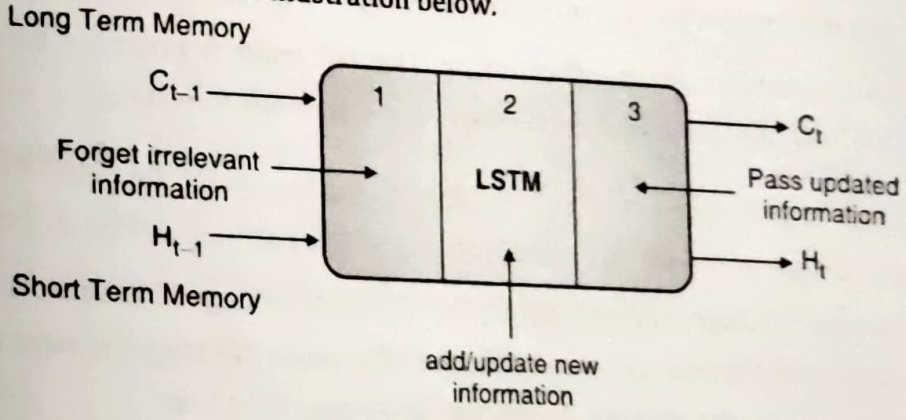



**Fig. 5.4.3 : LSTM Cell State**

- It's interesting to note that all of the timestamps are carried by the cell state along with the information.

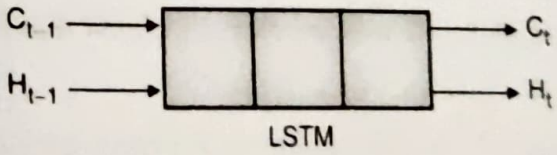



**Fig. 5.4.4 : LSTM Gate**

- Bob is a nice person. Dan on the other hand is evil.
- To comprehend how LSTM functions, let's look at an example. Here, a full stop separates two phrases. Dan, on the other hand, is nasty, and the first sentence reads, "Bob is a lovely

person." It is crystal evident that we are talking about Bob in the first phrase and switching to Dan as soon as we reach the full stop (.).

- Our network should understand that we have stopped talking about Bob as soon as we go from the first to the second phrase. Now Dan is the focus. Here, the network's Forget gate enables it to forget about it. Let's examine the functions these gates perform in the LSTM architecture.

**1. Forget Gate :**

- The initial step in an LSTM network cell is to choose whether to keep or discard the data from the preceding timestamp. The forget gate equation is given below.

$$f_t = \sigma(X_t * U_f + H_{t-1} * W_f)$$

Here,

$X_t$ : input to the current timestamp.

$U_f$ : weight associated with the input

$H_{t-1}$ : The hidden state of the previous timestamp

$W_f$ : It is the weight matrix associated with hidden state.

- Later, a sigmoid function is applied over it. That will make ft a number between 0 and 1. This ft is later multiplied with the cell state of the previous timestamp as shown below.

$$C_{t-1} * f_t = 0 \quad \text{... if } f_t = 0 \text{ (forget everything)}$$

$$C_{t-1} * f_t = C_{t-1} \quad \text{... if } f_t = 1 \text{ (forget nothing)}$$

- The network will forget everything if ft is set to 0, but nothing if ft is set to 1. Returning to our example, the first line discussed Bob, and following a full stop, the network will come across Dan. Ideally, the network should have forgotten about Bob.

**2. Input Gate :**

- Take yet another illustration

"Bob has swimming skills. He told me over the phone that he had spent four arduous years in the navy."

- So, Bob is the subject of both of these phrases. However, each offers a unique perspective on Bob. He knows how to swim, as indicated in the first sentence. The second sentence, on the other hand, mentions that he uses a phone and spent four years in the navy.
- Just consider which details in the second line are crucial in light of the context provided in the first. He first mentioned his service in the navy over the phone. It makes no

difference in this situation wh;ether he sent the information via phone or another method of contact. We want our model to keep in mind that it is significant information that he served in the navy. The Input gate's job is to perform this.

- The value of the fresh information carried by the input is measured by the input gate. The input gate's equation is shown below.

$$i_t = \sigma(X_t * U_i + H_{t-1} * W_i)$$

Here,

$X_t$ : Input at the current timestamp t

$U_i$ : weight matrix of input

$H_{t-1}$ : A hidden state at the previous timestamp

$W_i$ : Weight matrix of input associated with hidden state

- We once more applied the sigmoid function to it. As a result, I will have a value between 0 and 1 at timestamp t.

**New Information :**

$$N_t = \tanh (X_t * U_c + H_{t-1} * W_c) \text{ (new information)}$$

- The new data that had to be sent to the cell state now depends on a concealed state at timestamp t-1 in the past and input x at timestamp t. Tanh is the activation function in this case. The tanh function causes the value of fresh information to range from -1 to 1. The information is deducted from the cell state if the value of $N_t$ is negative, and added to the cell state at the current timestamp if the value is positive.
- The $N_t$ won't, however, be added to the cell state immediately. This is the revised equation.

$$C_t = f_t * C_{t-1} + i_t * N_t \text{ (updating cell state)}$$

- In this case, Ct-1 represents the cell state at the current timestamp, and the other variables are those we previously calculated.

3. **Output Gate :**

- Consider the following sentence.

"Bob faced the enemy alone and gave his life for his nation. brave, for his contributions."

- We have to finish the second sentence in this task. Today, we immediately recognise a person when we hear the word brave. Bob is the only one being brave in the phrase; we cannot say that the nation or the enemy is brave. We must therefore provide a pertinent

word to fill in the blank depending on the existing expectation. This is the purpose of our Output gate, and that term is our output.

- The Output gate's equation, which is quite similar to the equations for the two earlier gates, is shown below.

$$O_t = \sigma(X_t * U_o + H_{t-1} * W_o)$$

- Due to this sigmoid function, it will also have a value between 0 and 1. We will now use $O_t$ and tanh of the updated cell state to determine the current hidden state. As displayed below.

$$H_t = O_t * \tanh(C_t)$$

- It turns out that the concealed state depends on both the present output and long-term memory ($C_t$). Simply activate SoftMax on hidden state $H_t$ if you need to take the output of the current timestamp.

$$\text{Output} = \text{Softmax}(H_t)$$

- Prediction is the token in this case having the highest score in the output.
- This is the LSTM network's more understandable diagram.

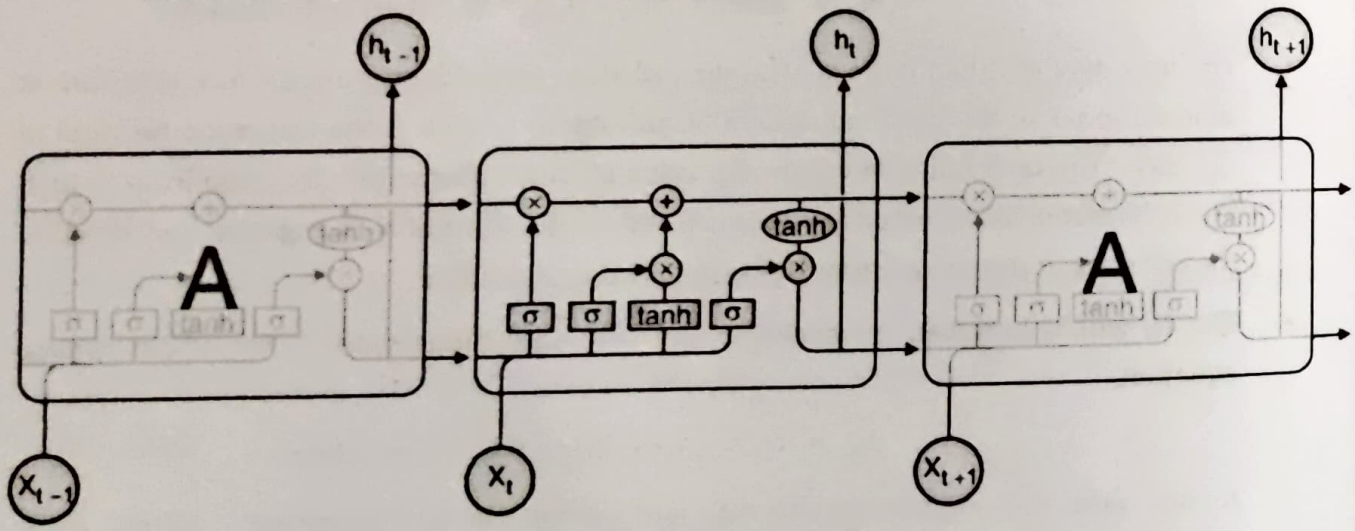



**Fig. 5.4.5 : LSTM Network**

## 5.4.2 Gated Recurrent Unit

- A development of the traditional RNN, or recurrent neural network, is the GRU, or gated recurrent unit. In the year 2014, Kyunghyun Cho and others introduced it.
- GRUs and Long Short Term Memory are quite similar (LSTM). GRU use gates to regulate the information flow, just like LSTM. When compared to LSTM, they are quite new. They have a simpler architecture and provide certain improvements over LSTM because of this.

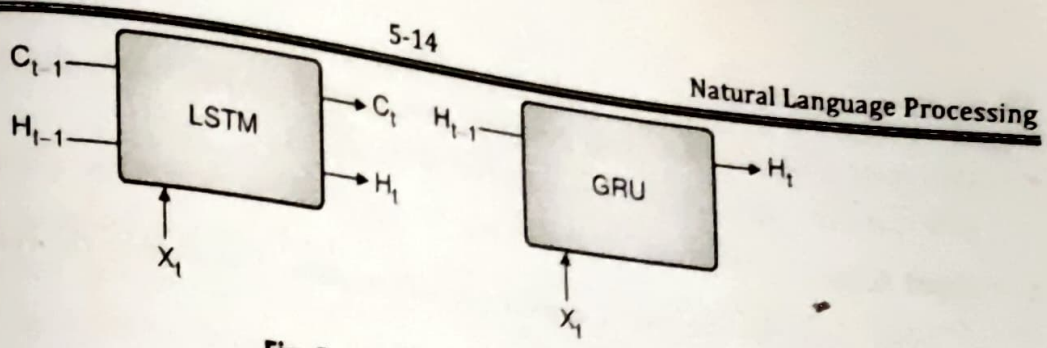
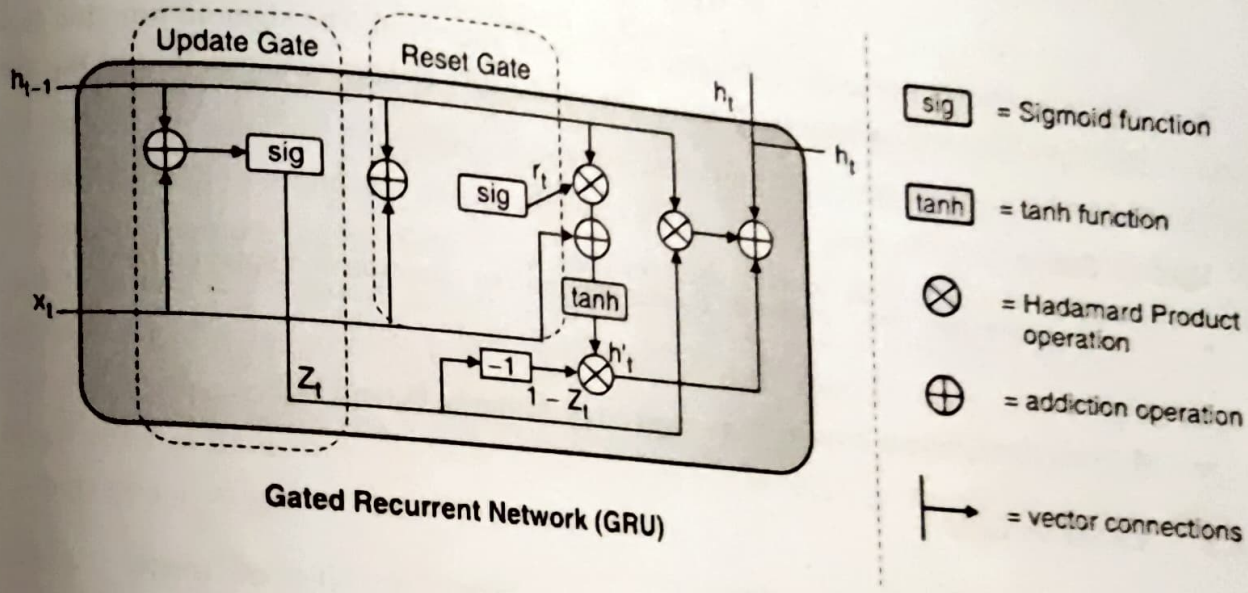



**Fig. 5.4.6 : LSTM and GRU**




**Gated Recurrent Network (GRU)**

**Fig. 5.4.7 : GRU Architecture**

- Another intriguing feature of GRU is that, in contrast to LSTM, it lacks a distinct cell state ($C_t$). There is just a hidden state ($H_t$). GRUs are quicker to train because of the architecture's simplicity.

**The architecture of Gated Recurrent Unit**

- Let's now see how GRU functions. Here, we have a GRU cell that resembles an LSTM or RNN cell more or less.

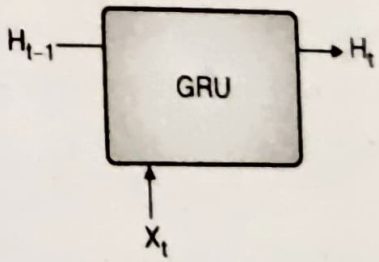



**Fig. 5.4.8 : GRU Gates**

- It accepts an input $X_t$ and the hidden state $H_{t-1}$ from timestamp t-1 for each timestamp t. Later, a new hidden state, Ht, is output and again given to the following timestamp.

- Currently, a GRU has primarily two gates as opposed to an LSTM cell's three gates. The Reset gate is the first gate, while the Update gate is the second.

**1. Reset Gate :**

- The network's short-term memory, or hidden state, is handled by the reset gate ($H_t$). The Reset gate's equation is given below.

$$r_t = \sigma (X_t * U_r + H_{t-1} * W_r)$$

- This is pretty similar to the LSTM gate equation, if you recall. The sigmoid function will lead $r_t$ to have a value between 0 and 1. The reset gate's weight matrices, $U_r$ and $W_r$, are shown here.

**2. Update Gate :**

- For long-term memory, we have an Update gate, and its equation is displayed below.

$$U_t = \sigma (X_t * U_u + H_{t-1} * W_u)$$

- The only distinction is between the weight measurements, $U_u$ and $W_u$.

### 5.4.3 LSTM vs GRU

- The few differencing points are as follows :
  - The GRU has two gates, LSTM has three gates
  - GRU does not possess any internal memory, they don't have an output gate that is present in LSTM
- In LSTM the input gate and target gate are coupled by an update gate and in GRU reset gate is applied directly to the previous hidden state. In LSTM the responsibility of reset gate is taken by the two gates i.e., input and target.

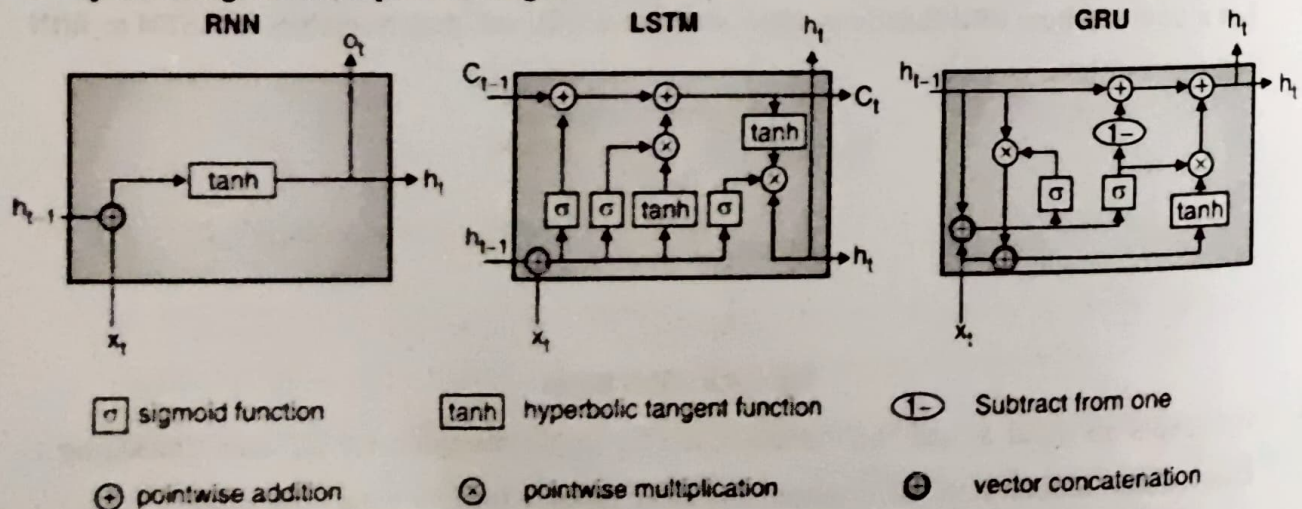



**Fig. 5.4.9 : RNN vs LSTM vs GRU**

## 5.5 Generative Adversarial Networks (GANs)

A potent family of neural networks called generative adversarial networks (GANs) is employed in unsupervised learning. Ian J. Goodfellow created and unveiled it in 2014. GANs are essentially a system of two competing neural network models that can assess, capture, and duplicate the changes within a dataset.

### 5.5.1 Why were GANs developed in the first place?

By introducing a negligible amount of noise into the original data, it has been observed that the majority of standard neural nets are easily tricked into misclassifying objects. Surprisingly, the model has higher confidence in the incorrect forecast after noise addition than it does in the accurate prediction. Due to the fact that most machine learning models learn from little amounts of data, which is a major disadvantage as it increases the risk of overfitting, this enemy exists. Additionally, there is a nearly linear mapping between the input and the output. Despite the fact that it may appear that the boundaries separating the various classes are linear, they are actually constructed of linearities, and even a little change in a point in the feature space could result in incorrect data categorization.

### 5.5.2 How does GANs work?

- The three components of Generative Adversarial Networks (GANs) are as follows :
  - **Generative :** Learning a generative model, which explains how data is produced in terms of a probabilistic model, is referred to as generative modelling.
  - **Adversarial :** A model is trained in an adversarial environment.
  - **Network :** Deep neural networks are a type kind network that can be used as Artificial Intelligence (AI) training methods.
- A generator and a discriminator are both present in GANs. The Generator attempts to trick the Discriminator by creating phoney samples of data (such as an image, audio, etc.). On the other hand, the Discriminator tries to tell the difference between the genuine and fraudulent samples. Both the Generator and the Discriminator are neural networks, and throughout the training phase, they compete with one another. The procedures are repeated multiple times, and each time, the Generator and Discriminator become better at what they are doing. The diagram below helps you understand how it works.

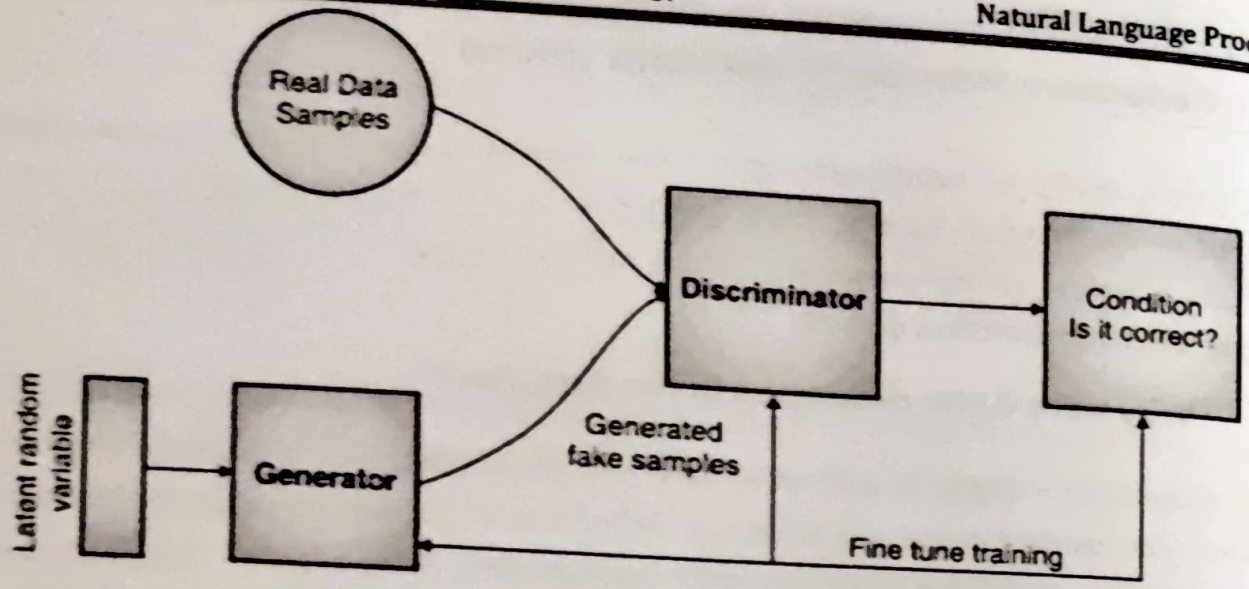



**Fig. 5.5.1 : Generative Adversarial Networks**

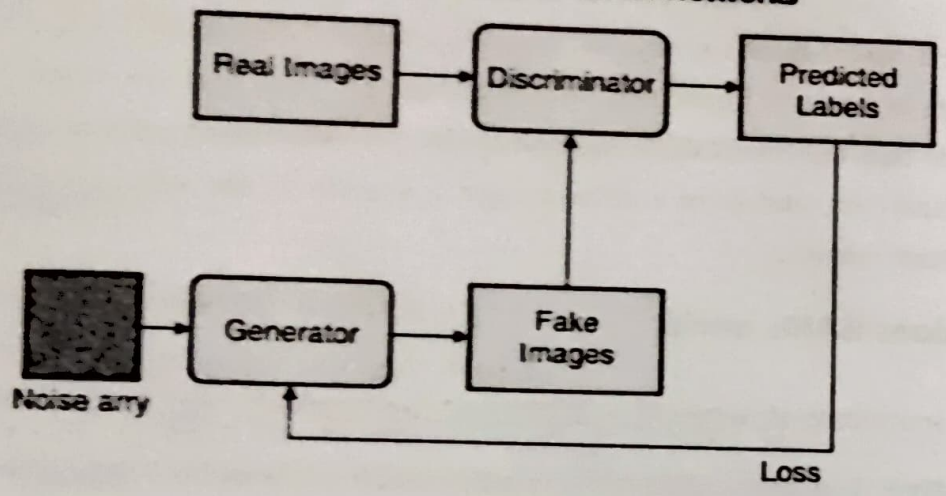



**Fig. 5.5.2 : General Structure of Generative Adversarial Networks**

- Here, the generative model captures the data distribution and is trained to try to increase the likelihood that the Discriminator will make a mistake. On the other hand, the Discriminator is based on a model that calculates the likelihood that the sample it received came from the training data and not the generator.
- The Discriminator is seeking to reduce its reward V(D, G) in the minimax game that the GANs are designed as, while the Generator is trying to maximise its loss by minimising the Discriminator's reward. The following equation can be used to mathematically describe it :

$$\min_{g} \max_{D} V(D, G)$$

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

Here,

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

### 5.5.3 Training a GAN has Two Parts

- **Part 1 :** While the Generator is not in use, the discriminator is trained. The network is only forward propagated during this phase; no back propagation is carried out. The Discriminator is tested to determine if it can accurately identify them as real after being trained on real data for n epochs. Additionally, the Discriminator is trained on the fictitious data produced by the Generator at this phase to see if it can correctly identify them as such.
- **Part 2 :** While the Discriminator is not in use, the Generator is being taught. After the Discriminator has been trained using the Generator's fabricated data, we may utilise the predictions to train the Generator and improve from the Discriminator's prior state.
- Following a few epochs of the aforementioned procedure, the bogus data is personally verified if it appears to be real. The training is ended if it appears to be acceptable; else, it is permitted to continue for a few more epochs.

### 5.5.4 Different Types of GANs

GAN implementation has taken many various forms, and the field of GAN research is now quite active. Following is a list of some of the significant ones that are now in use :

1. **Vanilla GAN :** The most basic GAN type is called a vanilla GAN. The Generator and Discriminator in this scenario are straightforward multi-layer perceptrons. Simple stochastic gradient descent is used in vanilla GAN's approach to try and optimise the mathematical problem.

2. **Conditional GAN (CGAN) :** CGAN is a deep learning technique that employs a number of conditional parameters. In CGAN, the Generator is given an extra parameter, "y," to produce the necessary data. In order for the discriminator to aid distinguish between the real data and the phoney generated data, labels are additionally added to the input.

3. **Deep Convolutional GAN (DCGAN) :** DCGAN is among the most well-liked and effective GAN implementations. ConvNets are used instead of multi-layer perceptronsin its construction. Convolutional stride actually replaces max pooling in the ConvNets implementation. The layers are also not entirely connected.

4. **Laplacian Pyramid GAN (LAPGAN) :** A set of band-pass pictures that are separated by an octave and a low-frequency residual make up the Laplacian pyramid, a linear invertible image representation. This method makes use of several Generator and Discriminator networks as well as various Laplacian Pyramid levels. The major reason this method is employed is that it results in photographs of extremely high quality. The image is initially downscaled at each pyramidal layer before being upscaled at each layer once again in a backward pass. At each of these layers, the image picks up noise from the Conditional GAN until it reaches its original size.

5. **Super Resolution GAN (SRGAN) :** As the name implies, SRGAN is a method for creating a GAN that uses both a deep neural network and an adversarial network to produce higher resolution images. This particular sort of GAN is very helpful in enhancing details in native low-resolution photos while minimising mistakes.

### 5.5.5 Advantages and Disadvantages

**Advantages :**

- GANs produce data that resembles the original data in appearance. If you provide GAN with an image, it will create a new version of the image that closely resembles the original. It can also produce several text, video, and audio versions.
- GANs dig into the specifics of the data and are simple to parse into several versions, making them useful for machine learning tasks.
- We can quickly identify trees, streets, bicyclists, people, and parked cars using GANs and machine learning. We can even determine the distance between various items.

**Disadvantages :**

- **Harder to train :** To determine whether it operates accurately or not, you must regularly submit new types of data.
- It's incredibly difficult to produce results from speech or text.

## REVIEW QUESTIONS

Q. 1 Write short note on Recurrent Neural Network.

Q. 2 Write short note of Bidirectional RNN.

Q. 3 Distinguish between RNN, LSTM and GRU.

Q. 4 Explain gates in LSTM.

Q. 5 Explain gates in GRU.

Q. 6 Short note on GAN.

□□□

# 6

UNIT - VI

# Case Study and Applications

**Syllabus**

**Computer Vision :** Image Classification, Image net- Detection-Audio Wave Net.

**Natural Language Processing :** Sentimental Analysis, Text preprocessing and ChatBot.

## 6.1 Computer Vision

- Computer vision has traditionally been one of the most active research areas for deep learning applications, because vision is a task that is effortless for humans and many animals but challenging for computers (Ballarcd et al., 1983). Many of the most popular standard benchmark tasks for deep learning algorithms are forms of object recognition or optical character recognition.
- Computer vision is a very broad field encompassing a wide variety of ways of processing images, and an amazing diversity of applications. Applications of computer vision range from reproducing human visual abilities, such as recognizing faces, to creating entirely new categories of visual abilities. As an example of the latter category, one recent computer vision application is to recognize sound waves from the vibrations they induce in objects visible in a video (Davis et al., 2014). Most deep learning research on computer vision has not focused on such exotic applications that expand the realm of what is possible with imagery but rather a small core of AI goals aimed at replicating human abilities.
- Most deep learning for computer vision is used for object recognition or detection of some form, whether this means reporting which object is present in an image, annotating an image with bounding boxes around each object, transcribing a sequence of symbols from an image, or labelling each pixel in an image with the identity of the object it belongs to. Because generative modelling has been a guiding principle of deep learning research, there is also a large body of work on image synthesis using deep models.

## 6.1.1 Image Classification

- Image classification is where a computer can analyse an image and identify the 'class' the image falls under. (Or a probability of the image being part of a 'class'.) A class is essentially a label, for instance, 'car', 'animal', 'building' and so on. For example, you input an image of a sheep. Image classification is the process of the computer analysing the image and telling you it's a sheep. (Or the probability that it's a sheep.)
- Early image classification relied on raw pixel data. This meant that computers would break down images into individual pixels. The problem is that two pictures of the same thing can look very different. They can have different backgrounds, angles, poses, etcetera. This made it quite the challenge for computers to correctly 'see' and categorize images.
- Deep learning is a type of machine learning; a subset of Artificial Intelligence (AI) that allows machines to learn from data. Deep learning involves the use of computer systems known as neural networks.
- In neural networks, the input filters through hidden layers of nodes. These nodes each process the input and communicate their results to the next layer of nodes. This repeats until it reaches an output layer, and the machine provides its answer.
- There are different types of neural networks based on how the hidden layers work. Image classification with deep learning most often involves convolutional neural networks, or CNNs. In CNNs, the nodes in the hidden layers don't always share their output with every node in the next layer (known as convolutional layers).
- Deep learning allows machines to identify and extract features from images. This means they can learn the features to look for in images by analysing lots of pictures. So, programmers don't need to enter these filters by hand.

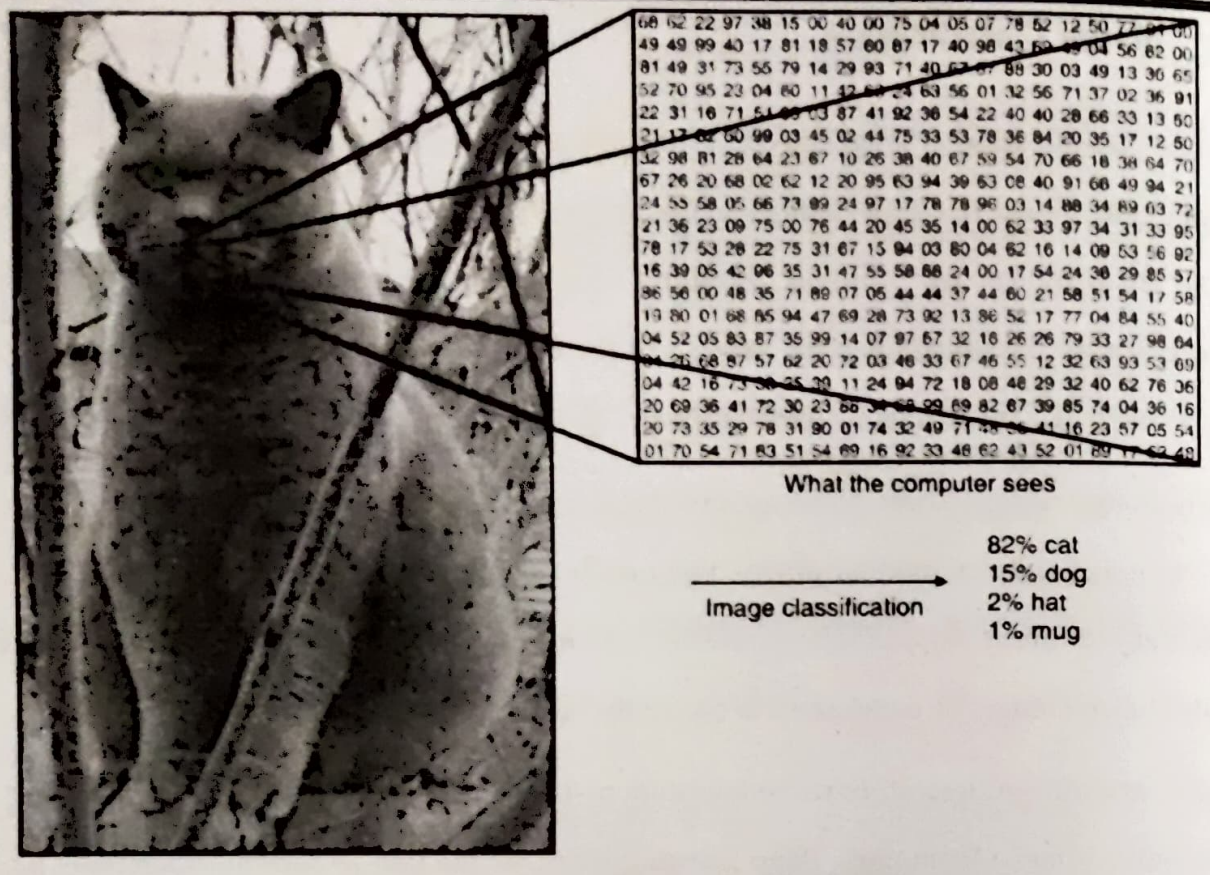



**Fig. 6.1.1 : Computer "vision" via data.**

**Why is image classification useful?**

- Image classification has a few uses and vast potential as it grows in reliability. Here are just a few examples of what makes it useful.
- Self-driving cars use image classification to identify what's around them i.e. trees, people, traffic lights and so on.
- Image classification can also help in healthcare. For instance, it could analyse medical images and suggest whether they classify as depicting a symptom of illness.
- Or, for example, image classification could help people organise their photo collections.

## 6.1.2 Image Net- Detection

- ImageNet is a large database or dataset of over 14 million images. It was designed by academics intended for computer vision research. It was the first of its kind in terms of scale. Images are organized and labelled in a hierarchy.
- In Machine Learning and Deep Neural Networks, machines are trained on a vast dataset of various images. Machines are required to learn useful features from these training images. Once learned, they can use these features to classify images and perform many other tasks

associated with computer vision. ImageNet gives researchers a common set of images to benchmark their models and algorithms. It's fair to say that ImageNet has played an important role in the advancement of computer vision.

- The diversity and size of ImageNet meant that a computer looked at and learned from many variations of the same object. These variations could include camera angles, lighting conditions, and so on. Models built from such extensive training were better at many computer vision tasks. ImageNet convinced researchers that large datasets were important for algorithms and models to work well. In fact, their algorithms performed better after they were trained with ImageNet dataset.
- ImageNet consists of 14,197,122 images organized into 21,841 subcategories. These subcategories can be considered as sub-trees of 27 high-level categories. Thus, ImageNet is a well-organized hierarchy that makes it useful for supervised machine learning tasks.On average, there are over 500 images per subcategory. The category "animal" is most widely covered with 3822 subcategories and 2799K images. The "appliance" category has on average 1164 images per subcategory, which is the most for any category. Among the categories with least number of images are "amphibian", "appliance", and "utensil".
- As many as 1,034,908 images have been annotated with bounding boxes. For example, if an image contains a cat as its main subject, the coordinates of a rectangle that bounds the cat are also published on ImageNet. This makes it useful for computer vision tasks such as object localization and detection.
- Then there's Scale-Invariant Feature Transform (SIFT) used in computer vision. SIFT helps in detecting local features in an image. ImageNet gives researchers 1000 subcategories with SIFT features covering about 1.2 million images. Images vary in resolution but it's common practice to train deep learning models on sub-sampled images of 256 × 256 pixels.

## ImageNet Challenge

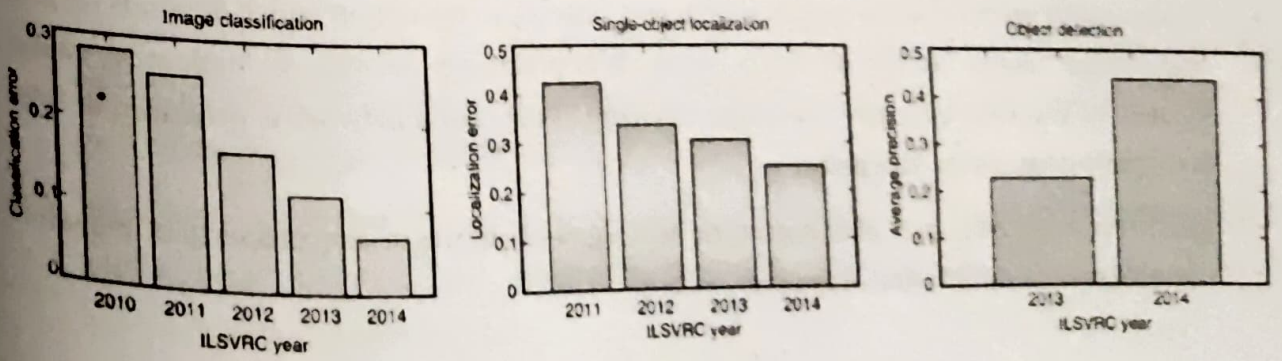



**Fig. 6.1.2 : ImageNet**

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was an annual computer vision contest held between 2010 and 2017. It's also called ImageNet Challenge.
- For this challenge, the training data is a subset of ImageNet: 1000 synsets, 1.2 million images. Images for validation and test are not part of ImageNet and are taken from Flickr and via image search engines. There are 50K images for validation and 150K images for testing. These are hand-labeled with the presence or absence of 1000 synsets.
- The Challenge included three tasks : image classification, single-object localization (since ILSVRC 2011), and object detection (since ILSVRC 2013). More difficult tasks are based upon these tasks. In particular, image classification is the common denominator for many other computer vision tasks. Tasks related to video processing, but not part of the main competition, were added in ILSVRC 2015. These were object detection in video and scene classification.

### 6.1.3 Audio Wave Net

- WaveNet is deep autoregressive, generative model, which produces human-like voice, where raw audio is feeded as input to the model, taking speech synthesis to another level.
- WaveNet is combination of two different ideas wavelet and Neural networks. Raw audio is generally represented as a sequence of 16 bits. 16 bits samples produces $^{216}$ (65536) quantization values, which are processed through SoftMax, making it computationally expensive. Hence the sequences of samples is reduced to 8 bits, using μ-law transformation, $F(x) = \text{sign}(x)\ \ln(1+\mu|x|)/\ln(1+\mu)$, $-1 \le x \le 1$, where μ takes value from 0 to 255 and x denotes input samples and then quantize to 256 values
- The first step is an audio preprocessing step, after the input waveform is quantized to a fixed integer range. The integer amplitudes are then one-hot encoded. These one-hot encoded samples are passed through causal convolution.

**Causal Convolution Layer**

- In signal and system, Causal system is system is referred as the output which depends on past and current inputs but not on future inputs. It is practically possible to implement causal system. In WaveNet the current acoustic intensity of the neural network is produced at time step t only depends on data before t.
- This layer is the main part of architecture, as it signifies autoregressive property of WaveNet and also maintains ordering of samples.

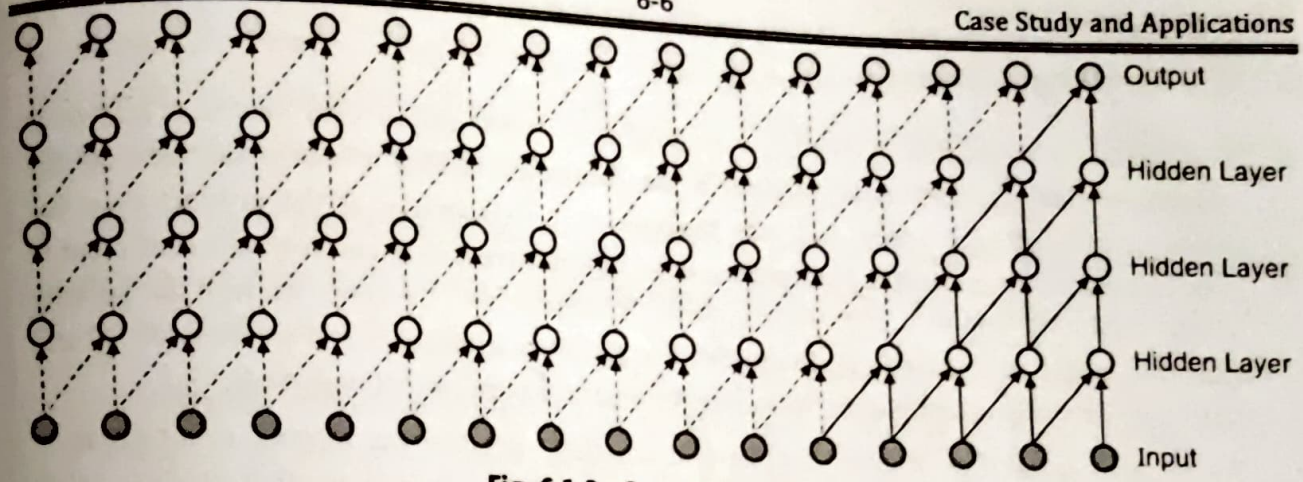



**Fig. 6.1.3 : Causal Convolution**

- For training of 1 output sample, 5 input samples are used. Receptive field of this network is 5.

$$P(x) = \prod_{n=1}^{N} p(X_N \mid X_1, X_2 \ldots, X_{N-1})$$

samples are denoted as $x = (x_1, x_2 \ldots, x_N)$, $p(.)$ is the probability.

- The following equation is used for generation of new samples by predicting probability of next samples, given the probabilities of previous and current samples. Problem with causal convolution is that they require many layers, or large filters to increase the receptive field.

## 6.2 Natural Language Processing

- Undoubtedly, ML and DL have made substantial contributions to the creation of intelligent computers, but it can be challenging to take advantage of these systems. A limited group of machine learning engineers had access to even basic communication or use of these complex ML systems. By enabling them to communicate with them in human languages, Natural Language Processing (NLP) may be seen as having contributed to the public's introduction of these intelligent systems.
- Another aspect of NLP's popularity is the broad range of important applications it has. Popular natural language processing applications include sentiment analysis, conversational agents, question answering systems, machine translation, and information retrieval. This calls for a closer investigation of NLP and its use.
- It has been noted that ML and DL approaches have proved quite important in the many NLP applications. Understanding the crucial role that learning approaches play in NLP applications is therefore crucial.

### 6.2.1 Sentiment Analysis

- The method of determining whether a block of text is good, negative, or neutral is known as sentiment analysis. Sentiment analysis is the contextual mining of words that reveals the social sentiment of a brand and aids businesses in determining if the product they are producing will find a market. Sentiment analysis's objective is to examine public sentiment in a way that will support corporate growth. It emphasises emotions as well as polarity (positive, negative, and neutral) (happy, sad, angry, etc.). It makes use of a variety of Natural Language Processing techniques, including Automatic, Hybrid, and Rule-based.
- For instance, what if we wanted to determine whether a product was meeting client needs or whether there was a market need for it? We can track the reviews for that product using sentiment analysis. When there is a big amount of unstructured data and we wish to classify that data by automatically tagging it, sentiment analysis is very effective to apply. Surveys of the Net Promoter Score (NPS) are frequently used to learn how customers view a product or service. The capability of sentiment analysis to swiftly and consistently produce consistent results while processing massive amounts of NPS answers contributed to its popularity.

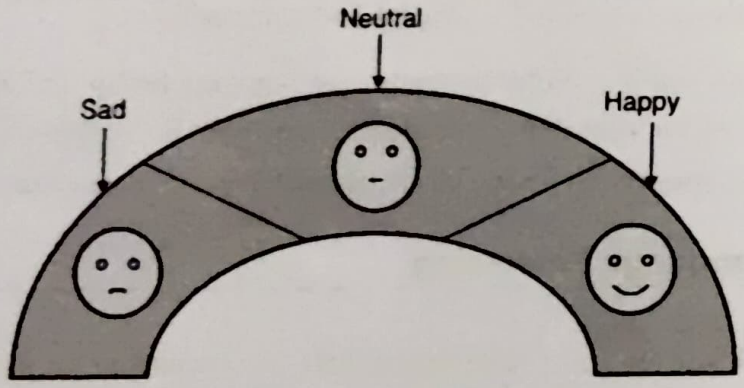



**Fig. 6.2.1 : Sentiments**

**Why Perform Sentiment Analysis?**

- 80% of the data in the globe is unstructured, according to the survey. Regardless of whether the data is in the form of emails, texts, documents, articles, or anything else, it needs to be examined and organised.
- Sentiment Analysis is necessary since it stores data in a productive, economical manner.
- Sentiment analysis helps you resolve all real-world problems and real-world circumstances.

**Types of Sentiment Analysis**

1. **Sentiment analysis with finer resolution :** This is based on polarity. A very positive, positive, neutral, negative, or extremely negative category could be created for this. Ratings are given on a scale of 1 to 5. If the rating is 5, it is highly positive; if it is 2, it is negative; and if it is 3, it is neutral.

2. **Emotion detection :** Emotion detection refers to the recognition of feelings such as happiness, sadness, rage, sorrow, merriment, and so forth. The lexical approach of sentiment analysis is another name for it.

3. **Aspect-based sentiment analysis :** Aspect-based sentiment analysis focuses on a certain aspect. For example, if a person wants to evaluate a cell phone's feature, they would check the aspect such as battery, screen, and camera quality.

4. **Multilingual sentiment analysis :** When analysing sentiment in many languages, it is necessary to categories the sentiment as positive, negative, or neutral. This is quite demanding and challenging in comparison.

**How does Sentiment Analysis work?**

There are three approaches used :

1. **Rule-based Approach :** In this case, tokenization, parsing, and the lexicon method are rule-based. The strategy counts how many positive and negative terms are present in the sample. If there are more positive words than negative words, the emotion is positive; otherwise, it is the opposite.

2. **Automatic Approach :** This strategy relies on machine learning and deep learning. Predictive analysis is first performed once the datasets have been trained. Word extraction from the text is the subsequent procedure. Different methods machine learning can be used, including Naive Bayes, Linear Regression, Support Vector, and Deep Learning techniques like Recurrent Neural Networks and Convolution Neural Network, can be used to extract emotions and sentiments from the given texts.

3. **Hybrid Approach :** This method combines the rule-based and automatic procedures described above. The advantage is that, in comparison to the other two procedures, accuracy is great.

**Applications :**

Sentiment Analysis has a wide range of applications as :

1. **Social media :** Take comments on Instagram as an example. Here, all reviews are examined and divided into three categories : favourable, negative, and neutral.

2. **Customer service :** Sentiment analysis techniques are used to complete all comments in the Play Store that are rated from 1 to 5.

3. **Marketing Sector :** In the marketing field, a certain product must be evaluated to determine whether it is good or terrible.

4. **Reviewer side :** Each reviewer will read the comments, check their accuracy, and provide an overall assessment of the product.

**Challenges of Sentiment Analysis**

There are major challenges in sentiment analysis approach :

- It is very challenging to determine whether a comment is optimistic or pessimistic when the data is presented in the form of a tone.
- You must determine if the data is beneficial or negative if it is shown as an emoji.
- Even detecting ironic, caustic, or comparative comments is difficult.
- A neutral statement is difficult to compare.

### 6.2.2 Text Preprocessing

- Building a machine learning and deep learning model requires the preprocessing of data, and the quality of the preprocessing determines the model's performance.
- Preprocessing text is the initial stage in NLP's model-building process.
- The various text preprocessing steps are :
  - i) Tokenization
  - ii) Lower casing
  - iii) Stop words removal
  - iv) Stemming
  - v) Lemmatization

**1. Tokenization :**

- Any NLP pipeline starts with tokenization. It significantly affects the remainder of your pipeline. Tokenization is the process of dividing unstructured data and natural language text into units of data that can be regarded as discrete pieces. One can directly utilise a document's token occurrences as a vector to represent the document.
- This instantly converts a text document or unstructured string into a numerical data format appropriate for machine learning. They can also be directly employed by a computer to initiate helpful answers and actions. They could also be employed as features in a machine learning pipeline to initiate more complicated actions or judgments.
- Sentences, words, letters, and subwords can all be broken apart using tokenization. Sentence tokenization refers to the process of dividing a text document into sentences.

**Challenges in Tokenization :**

- This task is typically applied to text corpora written in English or French, since these languages use white spaces or punctuation marks to demarcate the beginning and end of sentences. Unfortunately, other languages such as Chinese, Japanese, Korean, Thai, Hindi, Urdu, Tamil, and others could not be used with this method. The necessity to design a universal tokenization tool that integrates all languages is brought on by this issue.

**2. Lower casing :**

- Lowercase word conversion (NLP to nlp).
- Even if terms like "book" and "book" have the same meaning when written in lower case, the vector space model represents them as two different words (resulting in more dimensions).

**3. Stop words removal :**

- Pre-processing is the process of transforming data into a form that a computer can comprehend. Filtering away pointless data is one of the main types of pre-processing. Stop words are worthless words (data) that are used in natural language processing.
- A stop word is a frequently used term that a search engine has been configured to ignore, both while indexing entries for searching and when retrieving them as the result of a search query. Examples of stop words include "the," "a," "an," and "in."
- We don't want these terms to take up any unnecessary storage space or processing time in our database. By keeping a record of the terms you believe to be stop words, we may easily eliminate them for this reason. A list of stopwords is stored in NLTK (Natural Language Toolkit) in Python and is available in 16 different languages.

**When to remove stop words?**

- Stop words should be eliminated from corpora when performing text classification or sentiment analysis because they don't add any useful information to our models, however stop words are helpful when performing language translation because they must be translated alongside other words.
- When to eliminate stop words is not governed by any strict guidelines. However, if our goal involves language classification, spam filtering, caption generation, auto-tag generation, sentiment analysis, or another task that is connected to text classification, it would be advised for eliminating stop words.

- On the other hand, it's wiser to leave the stop words in if our assignment involves machine translation, question-answering problems, text summarization, or language modelling as they are an essential component of these applications.

**4. Stemming :**

The process of stemming entails creating morphological variations of a root or base word. Stemming algorithms or stemmers are other names for stemming programmes. The terms "chocolates," "chocolatey," and "choco" are reduced by a stemming algorithm to the word "chocolate," and the words "retrieval," "retrieved," and "retrieves" are reduced to the word "retrieve." In the pipeline lining stage of natural language processing, stemming plays a significant role. Tokenized words are used as the stemmer's input.

**Errors in Stemming :**

There are mainly two errors in stemming :

- **Over-stemming :** When two words with different stems that have the same root are over-stemmed. False-positive results can also be attributed to over-stemming.
- **Under-stemming :** When two words with the same root but with the same stem are stemmed, this is known as under-stemming. False-negatives can be understood as under-stemming.

**Applications of stemming :**

- Stemming is used in information retrieval systems like search engines.
- It is used to determine domain vocabularies in domain analysis.

**Popular Stemming Algorithms :**

**a. Porter Stemmer :**

- It was one of the most well-liked stemming strategies put forth in 1980. It is predicated on the notion that the suffixes in English are composites of smaller and simpler suffixes. This stemmer is renowned for its efficiency and ease of use. Data mining and information retrieval are two of Porter Stemmer's most common applications. However, only English terms can be used in its applications. Additionally, the output stem is not always a valid term because the collection of stems is mapped onto the same stem. The algorithms are known to be the oldest stemmer and are fairly lengthy in nature.
- Example : EED -> EE means "if the word has at least one vowel and consonant plus EED ending, change the ending to EE" as 'agreed' becomes 'agree'.

**b. Lovins Stemmer :**

- In order to turn a word's stem into a legitimate word, Lovins advocated in 1968 that the word's longest suffix be removed.
- As an illustration, sit ->sitt -> sit

**c. Dawson Stemmer :**

The suffixes are stored in a Lovins stemmer extension that is indexed by their length and last letter and stores them in reverse order.

**d. Krovetz Stemmer :**

Robert Krovetz made the suggestion in 1993. The steps are as follows :

1) Change a word's plural form to its singular form.
2) Change a word's past tense to its present tense and eliminate the prefix "ing."

   For instance : "children" -> "child"

**e. Xerox Stemmer :**

Example :

'children' → 'child'

'understood' → 'understand'

'whom' → 'who'

'best' → 'good'

**f. N-Gram Stemmer :**

- An n-gram is a collection of n consecutive characters taken from a word, and comparable words tend to share a lot of n-grams.
- Example : I, IN, NT, TR, RO, OD, DU, UC, CT, TI, IO, ON, NS, S are the new values for "INTRODUCTIONS" when n = 2.

**g. Snowball Stemmer :**

In contrast to the Porter Stemmer, the Snowball Stemmer can map words that are not in the English language. The Snowball Stemmers can be referred to as a multi-lingual stemmer because it supports other languages. Additionally imported from the NLTK package are the Snowball stemmers. This stemmer, which processes short strings and is the most popular stemmer, is based on the computer language "Snowball." The Porter2 Stemmer, commonly known as the Snowball Stemmer, is far more aggressive than the Porter Stemmer. The Snowball stemmer has faster computation because of the enhancements made in comparison to the Porter stemmer.

**h. Lancaster Stemmer :**

Comparatively speaking, the Lancaster stemmers are more aggressive and active. Although the stemmer is much faster than the algorithm, it is very difficult to understand when dealing with short words. They are less effective than Snowball Stemmers, though. The Lancaster stemmers essentially employ an iterative method and save the rules externally.

**5. Lemmatization :**

- One of the most popular text pre-processing methods in Natural Language Processing (NLP) and machine learning in general is lemmatization. If you've already read my piece on word stemming in natural language processing, you already know that lemmatization isn't all that dissimilar. In both stemming and lemmatization, we aim to distil a word down to its most basic component.
- In the stemming process, the root word is referred to as a stem, and in the lemmatization process, it is referred to as a lemma. But there are some more distinctions between the two than that. Let's find out what they are.

**How Lemmatization is different than Stemming?**

- At get to the stem of the word, a portion of the word is simply sliced off at the end. While several algorithms are employed to determine how many letters must be removed, none of them genuinely understand the meaning of the word in the language to which it belongs. On the other hand, the algorithms in lemmatization are aware of this. In fact, you could argue that these algorithms use dictionaries to comprehend the meaning of the word before distilling it to its lemma, or fundamental term.
- The word better is thus generated from the word good, and as a result, the phrase is good, according to a lemmatization algorithm. However, a stemming algorithm couldn't accomplish the same thing. The word better may be over- or under-stemmed, and it may be changed to bet or bett or just left as better. But there is no possible way to stem it down to the word good at its core. This essentially distinguishes stemming from lemmatization.

**Advantages and Disadvantages of Lemmatization :**

- The obvious benefit of lemmatization is that it is more accurate, as you can surely tell by now. Lemmatization would therefore be helpful if you're working with an NLP application like a chat bot or virtual assistant where comprehending the context of the discussion is essential. But this accuracy comes at a cost.

- Lemmatization takes a lot of time because it requires determining a word's meaning using a tool like a dictionary. Consequently, most lemmatization techniques are slower than their stemming equivalents. Lemmatization has a computational overhead as well, but in MLproblems, computational resources are rarely a problem.

### 6.2.3 Chatbot

**What is a chatbot?**

- A chatbot is a piece of software or a computer programme that mimics human conversation through voice or text exchanges.
- More users are using chatbot virtual assistants to complete simple tasks in business-to-business (B2B) and business-to-consumer (B2C) situations. Chatbot assistants allow businesses to provide customer care when live agents aren't accessible, cut down on overhead costs, and make better use of support staff time.

**How do chatbots work?**

- Chatbots can be stateless or stateful, and their levels of complexity vary. Stateless chatbots approach every interaction as though it were with a different user. Stateful chatbots, on the other hand, may look back on previous conversations and contextualise new responses.
- Low or no coding is needed to integrate a chatbot into a service or sales department. Developers may create conversational user interfaces for third-party business applications thanks to the availability of several chatbot service providers.
- The correct Natural Language Processing (NLP) engine must be chosen before a chatbot can be implemented. The chatbot needs a speech recognition engine, for instance, if the user communicates with the bot by voice.
- Furthermore, business owners must choose between having scheduled and unstructured talks. The scripted nature of chatbots designed for organised discussions makes programming easier but limits the types of questions that may be posed by users. Chatbots are usually programmed to answer frequently requested queries or carry out easy, repetitive activities in B2B contexts. For instance, chatbots can help sales representatives swiftly obtain phone numbers.

**Why are chatbots important?**

- Because chatbots can speak with customers and provide common answers, Artificial Intelligence (AI) chatbots are often adopted by businesses trying to boost sales or service productivity.

- Many experts anticipate that chat-based communication methods will become more popular as customers turn away from conventional modes of communication. Businesses are increasingly using chatbot-based virtual assistants to complete straightforward tasks, freeing up human agents to concentrate on other duties.

**How have chatbots evolved?**

- Early attempts to develop software that might at least momentarily fool a real human into thinking they were speaking with someone else included chatbots like ELIZA and PARRY. A Turing test was used to measure the effectiveness of PARRY in the early 1970s; testers could distinguish between a person and a chatbot at a level consistent with guesswork.
- Since then, chatbots have advanced significantly. Using AI tools like deep learning, Natural Language Processing (NLP), and Machine Learning (ML) algorithms, developers create contemporary chatbots. These chatbots need a tonne of information. The more the bot's voice recognition predicts acceptable responses, the more a user engages with it.
- Both the commercial and consumer segments are increasingly using chatbots. Consumers have fewer complaints to make while interacting with chatbots as they get better. Chatbots assist fill a void left by phone conversations as a result of cutting-edge technology and a cultural shift toward more passive, text-based communication.

**Types of Chatbots**

There is controversy about the number of distinct types of chatbots that exist and what the industry should label them because chatbots are still a relatively new business technology.

The following are a few examples of popular chatbot types :

i. **Quick or scripted chatbots :** They serve as a hierarchical decision tree and are the simplest kind of chatbots. These chatbots communicate with users by asking predetermined questions that continue until the chatbot responds to the user's query.

The menu-based chatbot is comparable to this one in that it asks users to choose options from a predetermined list or menu in order to better understand the user's wants.

ii. **Keyword recognition-based chatbots :** These chatbots are a little more sophisticated; they try to listen to the user as they type and then answer using words from customer comments. To answer effectively, this bot blends user-customizable keywords and AI. Unfortunately, these chatbots have trouble with overused keywords or repeated inquiries.

iii. **Hybrid chatbots :** These chatbots integrate features from bots that use keyword recognition and menus. If keyword identification is unsuccessful, users can select from options on the chatbot's menu or have their inquiries answered directly.

iv. **Voice-enabled chatbots :** The future of this technology lies in chatbots like this one. Voice-enabled chatbots employ user speech as input to inspire creative activities or responses. These chatbots can be made by developers utilising voice recognition and text-to-speech APIs. Apple's Siri and Amazon Alexa are two examples.

**How do businesses use chatbots?**

Chatbots have long been a feature of online games and instant messaging apps, but they have only lately begun to be employed for B2C and B2B sales and services.

Chatbots can be applied by businesses in the following ways :

- **Online shopping :** In these settings, sales staff can make use of chatbots to respond to simple product queries or offer useful details that customers could look up later, such as shipping costs and availability.
- **Customer support :** Chatbots can be used by service departments to assist service personnel with routine queries. A customer support agent might provide the chatbot with the order number and ask when the order was dispatched, for instance. When a conversation becomes too complicated, a chatbot will typically transfer the call or text to a human support person.
- **Virtual assistant :** Virtual assistants can also be used as chatbots. All four tech giants—Apple, Amazon, Google, and Microsoft—offer various virtual assistants. A personal chatbot can be an application, like Apple's Siri or Microsoft's Cortana, or a product, like Amazon's Echo with Alexa or Google Home.

**Benefits of Chatbots**

- Can hold multiple conversations at once.
- Cost-effective.
- Saves time.
- Proactive customer interaction.
- Monitors and analyzes consumer data.
- Improves customer engagement.
- Eases scalability to global markets.
- Expands the customer base.
- Measures lead qualifications.

**What are the challenges of using chatbots?**

- **New barriers and new technology :** Organizations might not know how to deal with the challenges that chatbot technology encounters as it is still in its infancy. While AI-enabled

bots can learn from every encounter and alter their behaviour, this process can be quite expensive for businesses if the first few interactions make customers uninterested and unresponsive.

- **Security :** Users must feel comfortable sharing personal information with the chatbot. Therefore, businesses must take care to design their chatbots so that they only ask for pertinent data and send that data safely over the internet. Hackers should not be able to access chat interfaces thanks to chatbots' secure designs.
- **Various keyboard styles used by message writers :** This may result in unclear intentions. Long and short statements, several brief entries, and chat bubbles with lengthy material must all be handled by chatbots.
- **The various ways that people speak :** These variances can be challenging for chatbots to comprehend. The user might, for instance, utilise acronyms, slang, or misspell terms. Unfortunately, due to its limitations, NLP is unable to fully address this issue.
- **Unpredictable human moods, feelings, and behaviour :** Users may swiftly change their thoughts since people are unpredictable and emotions and moods frequently influence user behaviour. They could want to issue a command after first requesting a suggestion. Chatbots must be able to accommodate and comprehend this irrationality and spontaneity.
- **User satisfaction :** Users always want the best experiences but are rarely satisfied. They are constantly hoping for improvements to the chatbot. As a result, businesses that utilise chatbots must constantly update and enhance them to give users the impression that they are conversing with knowledgeable, trusted sources.

## REVIEW QUESTIONS

Q.1 Write a short note on : Image Classification.

Q.2 Explain Audio Wave Net.

Q.3 Describe natural language processing in detail.

Q.4 Explain Sentiment Analysis and where is it used.

Q.5 What are the applications of Sentiment Analysis?

Q.6 Write a short note on : Text Preprocessing.

Q.7 Write a short note on : Lemmatization.

Q.8 What are the types of chatbots?

ooo