DEPARTMENT OF INFORMATION TECHNOLOGY
YEAR/SEM: III/VI

23CSX502 – DEEP LEARNING AND NEURAL NETWORKS MATERIAL

UNIT I
INTRODUCTION TO NEURAL NETWORK

Neural Networks Basics - Functions in Neural networks – Classification and Clustering problems - Deep networks basics - Shallow neural networks – Deep Neural Networks – Forward and Back Propagation. Deep Learning Frameworks – Data Augmentation - Underfitting- Overfitting.

1.1 NEURAL NETWORKS BASICS

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms.

- The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain.
- Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks.
- These neurons are known as nodes.

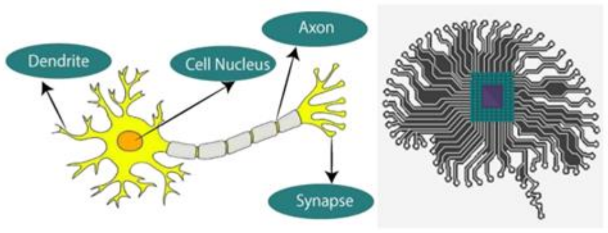


Figure 1.1: BNN

The given figure illustrates the typical diagram of Biological Neural Network.

- The typical Artificial Neural Network looks something like the given figure.
- Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

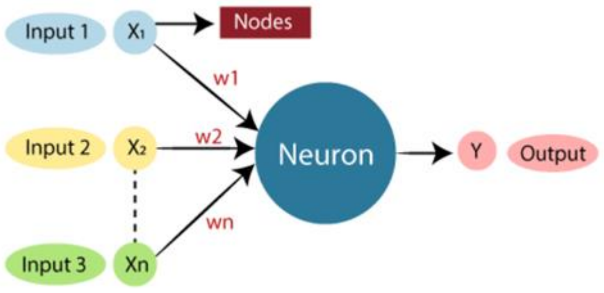


Figure 1.2: Neural Networks

1.1.1 Key Concepts of Neural Networks:

1. Neurons (Nodes)
   - The basic units of a neural network, similar to biological neurons.
   - Each neuron receives input, applies a mathematical function, and passes the output to the next layer.
2. Layers in Neural Networks
   - Input Layer: Receives the initial data (e.g., pixels of an image).

- Hidden Layers: Intermediate layers that extract and transform features.
- Output Layer: Produces the final prediction or classification.

3. Weights & Biases
   - Weights determine the importance of inputs.
   - Biases help adjust the output to improve learning.
4. Activation Functions
   - Apply a transformation to the input data to introduce non-linearity.
   - Common functions: ReLU, Sigmoid, Tanh, Softmax.
5. Training a Neural Network
   - Uses a dataset to learn patterns.
   - Forward Propagation: Passes input through layers to get an output.
   - Loss Function: Measures error between predicted and actual values.
   - Back propagation: Adjusts weights to reduce error using optimization techniques like Gradient Descent.
6. Types of Neural Networks
   - Feedforward Neural Networks (FNNs): Information moves in one direction.
   - Convolutional Neural Networks (CNNs): Used for image processing.
   - Recurrent Neural Networks (RNNs): Used for sequential data like speech and text.
   - Generative Adversarial Networks (GANs): Used for generating new data.

Applications of Neural Networks:

- Computer Vision: Face recognition, object detection.
- Natural Language Processing: Chatbots, sentiment analysis.
- Healthcare: Disease prediction, medical image analysis.
- Finance: Fraud detection, stock market prediction.
- Autonomous Systems: Self-driving cars, robotics.

1.1.2 Relationship between Biological Neural Network and Artificial Neural Network

Artificial Neural Network:

- ➢ Artificial Neural Network (ANN) is a type of neural network that is based on a Feed-Forward strategy.
- ➢ It is called this because they pass information through the nodes continuously till it reaches the output node. This is also known as the simplest type of neural network.

Biological Neural Network:

- ➢ Biological Neural Network (BNN) is a structure that consists of Synapse, dendrites, cell body, and axon. In this neural network, the processing is carried out by neurons.

- ➢ Dendrites receive signals from other neurons, Soma sums all the incoming signals and axon transmits the signals to other cells.

1.1.3 Comparison of ANN and BNN

Table 1.1 Comparison of ANN and BNN

Parameters	**ANN**	**BNN**
Structure	Input weight output hidden	Dendrites synapse axon cell body
Learning	very precise structures and formatted data	they can tolerate ambiguity
Processor	Complex high speed one or a few	Simple low speed large number
Memory	Separate from a processor localized non- content addressable.	Integrated into content processor distributed addressable
Computing	centralized sequential stored programs	distributed parallel self-learning
Reliability	very vulnerable	robust
Expertise	numerical and symbolic manipulations	perceptual problems
Operating Environment	well-defined well-constrained	poorly defined un-constrained
Fault Tolerance	The potential of fault tolerance	performance degraded even on partial damage

1.1.4 Artificial neural networks (ANNs):

- ➢ Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer.
- ➢ Each node, or artificial neuron, connects to another and has an associated weight and threshold.
- ➢ If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network.
- ➢ Otherwise, no data is passed along to the next layer of the network.

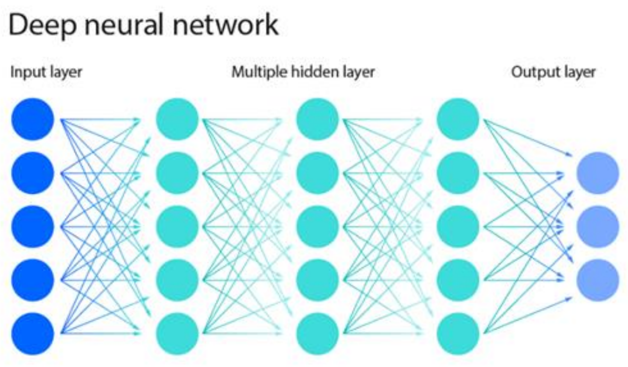


Figure 1.3: DNN

- ➢ Neural networks rely on training data to learn and improve their accuracy over time.

- However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity.
- Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts.
- One of the most well-known neural networks is Google's search algorithm.

1.1.5 Application Scope of Neural Networks

- ANNs have a wide range of applications because of their unique properties.

A few of the important applications of ANNs include:

1. ***Image Processing and Character recognition:***

- ANNs play a significant part in picture and character recognition because of their capacity to take in many inputs, process them, and infer hidden and complicated, non-linear correlations.
- Character recognition, such as handwriting recognition, has many applications in fraud detection (for example, bank fraud) and even national security assessments.

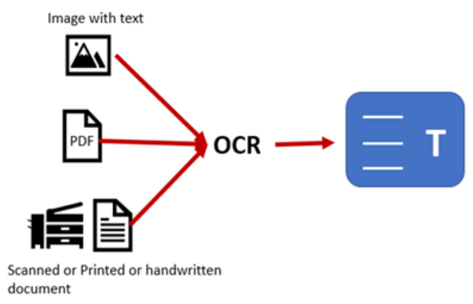


Figure 1.4: Image Processing and Character recognition

- Image recognition is a rapidly evolving discipline with several applications ranging from social media facial identification to cancer detection in medicine to satellite image processing for agricultural and defence purposes.
- Deep neural networks, which form the core of "deep learning," have now opened up all of the new and transformative advances in computer vision, speech recognition, and natural language processing notable examples being self-driving vehicles, thanks to ANN research.

2. Forecasting:

- Forecasting is widely used in everyday company decisions (sales, the financial allocation between goods, and capacity utilization), economic and monetary policy, finance, and the stock market.
- Forecasting issues are frequently complex; for example, predicting stock prices is complicated with many underlying variables (some known, some unseen).
- Traditional forecasting models have flaws when it comes to accounting for these complicated, non-linear interactions.
- Given its capacity to model and extract previously unknown characteristics and correlations, ANNs can provide a reliable alternative when used correctly.

- ANN also has no restrictions on the input and residual distributions, unlike conventional models.

Figure 1.5: Forecasting

3. Other Applications of Artificial Neural Networks:

1. *Image and speech recognition:*
   - ANNs, particularly CNNs, have revolutionized image and speech recognition systems, enabling applications such as facial recognition, object detection, and voice assistants.
2. It can be used to for Fraud Detection regarding credit cards, insurance or taxes by analysing the past records.
3. *Natural language processing:*
   - ANNs, including RNNs and transformer-based models, have significantly advanced tasks like machine translation, sentiment analysis, and language generation.
4. *Recommender systems*:
   - ANNs are used in recommendation engines to analyse user preferences and provide personalized recommendations for products, movies, music, and more.
5. *Financial analysis:*
   - ANNs can be employed for tasks like stock market prediction, fraud detection, credit risk assessment, and algorithmic trading.
6. *Medical diagnosis:*
   - ANNs have been applied to various healthcare domains, including disease diagnosis, medical imaging analysis, drug discovery, and personalized medicine.
7. *Autonomous vehicles:*
   - ANNs are crucial for self-driving cars, enabling perception, object recognition, decision-making, and control.
8. *Robotics:*
   - ANNs play a vital role in robotics applications, such as robot motion planning, object manipulation, and human-robot interaction.

8. Every new technology need assistance from the previous one i.e. data from previous ones and these data are analysed so that every pros and cons should be studied correctly. All of these things are possible only through the help of neural network.

9. Neural network is suitable for the research on Animal behaviour, predator/prey relationships and population cycles.

10. It would be easier to do proper valuation of property, buildings, automobiles, machinery etc. with the help of neural network.

11. Neural Network can be used in betting on horse races, sporting events, and most importantly in stock market.

12. It can be used to predict the correct judgment for any crime by using a large data of crime details as input and the resulting sentences as output.

13. By analyzing data and determining which of the data has any fault (files diverging from peers) called as Data mining, cleaning and validation can be achieved through neural network.

14. Neural Network can be used to predict targets with the help of echo patterns we get from sonar, radar, seismic and magnetic instruments.

15. It can be used efficiently in Employee hiring so that any company can hire the right employee depending upon the skills the employee has and what should be its productivity in future.

16. It has a large application in Medical Research.

1.1.6 Evolution of Neural Networks

- The evolution of neural networks is a fascinating journey that spans several decades.
- Neural networks have undergone significant developments and advancements, driven by both theoretical insights and practical applications.
- Here's a brief overview of the key milestones in the evolution of neural networks:

1. 1940s-1950s: The Birth of Neural Networks:

- The concept of neural networks was inspired by the structure and function of the human brain.
- Warren McCulloch and Walter Pitts introduced the first mathematical model of a neuron in 1943.
- In the 1950s, Frank Rosenblatt developed the perceptron, a single-layer neural network capable of binary classification.

2. 1960s-1980s: Perceptrons and Limitations:

- The perceptron's limitations were exposed, particularly its inability to solve problems that were not linearly separable.
- The perceptron fell out of favour, and research in neural networks waned during the AI winter.

3. 1980s-1990s: Back propagation and Multilayer Perceptrons:

- The rediscovery of the back propagation algorithm in the 1980s allowed training of multi-layer perceptrons (MLPs).
- The development of back propagation marked a resurgence of interest in neural networks, overcoming some of the limitations faced in the past.

4. 1990s-2000s: Support Vector Machines and Neural Network Winter:

- Support Vector Machines (SVMs) gained popularity for their success in classification tasks, overshadowing neural networks for a while.

- Lack of computational power and difficulties in training deep networks led to another decline in interest, known as the "neural network winter."

5. 2000s-2010s: Deep Learning Renaissance:

- The advent of powerful GPUs and the availability of large datasets enabled the training of deeper neural networks.
- Breakthroughs in deep learning, such as convolutional neural networks (CNNs) for image recognition and recurrent neural networks (RNNs) for sequential data, revitalized the field.
- Key contributions from researchers like Geoffrey Hinton, Yann LeCun, and Yoshua Bengio played a crucial role in the resurgence of neural networks.

6. 2010s-Present: Rise of Deep Learning in Applications:

- Deep learning has become the dominant paradigm in machine learning, achieving remarkable success in various applications such as image recognition, natural language processing, and speech recognition.
- Architectures like Long Short-Term Memory (LSTM) networks and Transformers have been instrumental in handling sequential and attention-based tasks.
- Transfer learning and pre-trained models have become common, allowing for efficient training on smaller datasets.

7. Current Trends:

- Neural architecture search (NAS) and AutoML techniques aim to automate the design of neural network architectures.
- Continued efforts to improve training stability, interpretability, and generalization of deep learning models.
- Exploration of neuromorphic computing and spiking neural networks inspired by the brain's biological structure.

➢ The evolution of neural networks reflects the continuous interplay between theoretical advancements, algorithmic innovations, and the availability of computational resources.

1.1.7 Perceptron

The perceptron was first proposed by Rosenblatt (1958)is a simple neuron that is used to classify its input into one of two categories. A perceptron is a single processing unit of a neural network. This is a good learning tool. This model follows perceptron training rule and it could operate well with linearly separable patterns.

Linearseparabilityistheseparationoftheinputspaceintoregionsisbased on whether the network response is positive or negative.

A perceptron uses a step function that returns+1if the weighted sum of its input(v) is greater than or equal to 0 else it returns -1.

$$\Phi(v) = \begin{cases} +1 \text{ if } v \geq \\ -1 \text{ if } v < \end{cases})$$

Working of perceptron

Neuron

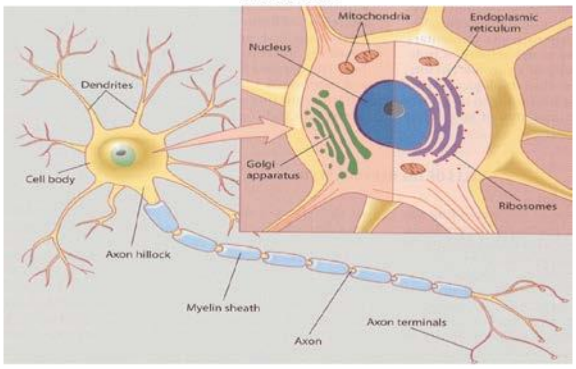


Figure 1.6: Representation of a biological neuron

- In the biological neurons, the dendrite receives the electrical signals from the axons of other neurons. The signals are modulated in various amounts before further transmission.
- The signals are transmitted to other neurons only if the modulated signal exceeds the threshold value. The same principle is applied in perceptron model.
- In the perceptron, the input received is always represented as numerical values. These values are multiplied by the weights.
- The total strength of the input is calculated as the weighted sum of the inputs. A step function (activation function) is applied to determine its output.
- This output is fed to the other perceptrons if it exceeds the threshold value.

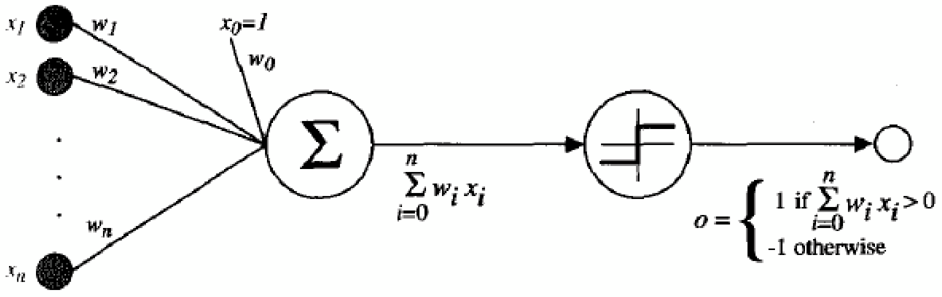

x_1
w_1
$x_0=1$
w_0
x_2
w_2
w_n
x_n
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

Figure 1.7: Perceptron Model (In this model x_0=1, which is the bias)

Basic Components of Perceptron

Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components. These are as follows:

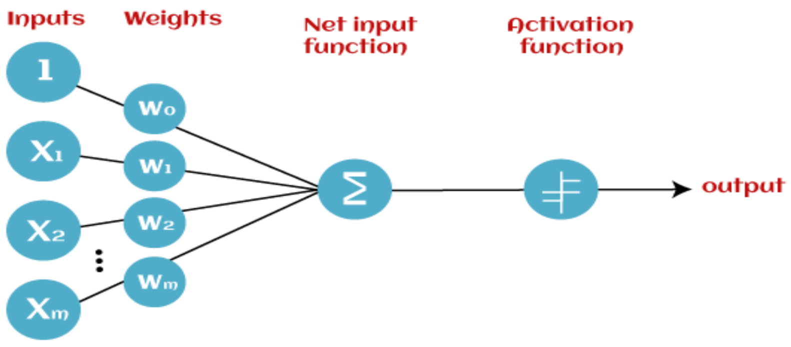


Figure 1.8: Basic Components of Perceptron

Input Nodes or Input Layer:

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

Weight and Bias:

Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

Activation Function:

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Types of Activation functions:

- Sign function
- Step function, and
- Sigmoid function

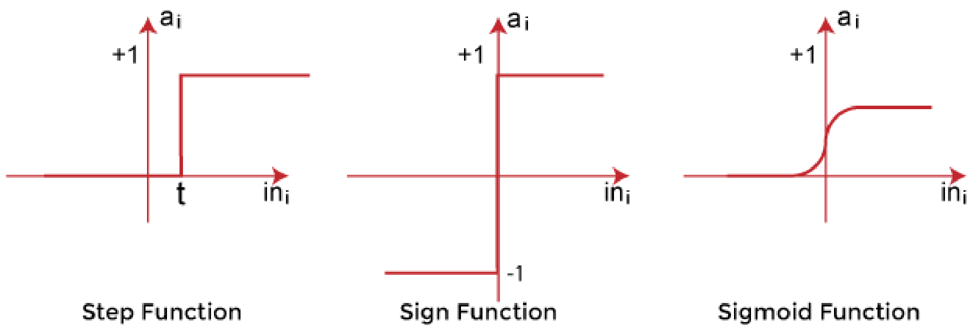


Figure 1.9: Types of Activation functions

The data scientist uses the activation function to take a subjective decision based on various problem statements and forms the desired outputs. Activation function may differ (e.g., Sign, Step, and Sigmoid) in perceptron models by checking whether the learning process is slow or has vanishing or exploding gradients.

How does Perceptron work?

In Machine Learning, Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and Bias, net sum, and an activation function. The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum. Then this weighted sum is applied to the activation function 'f' to obtain the desired output. This activation function is also known as the step function and is represented by 'f'.

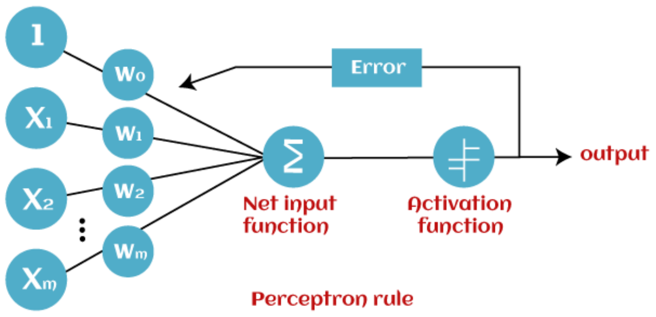


Figure 1.10: Working Process of Perceptron

This step function or Activation function plays a vital role in ensuring that output is mapped between required values (0,1) or (-1,1). It is important to note that the weight of input is indicative of the strength of a node. Similarly, an input's bias value gives the ability to shift the activation function curve up or down.

Perceptron model works in two important steps as follows:

Step-1

In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

$$\sum wi * xi = x1 * w1 + x2 * w2 + \cdots wn * xn$$

Add a special term called bias 'b' to this weighted sum to improve the model's performance.

$$\sum wi * xi + b$$

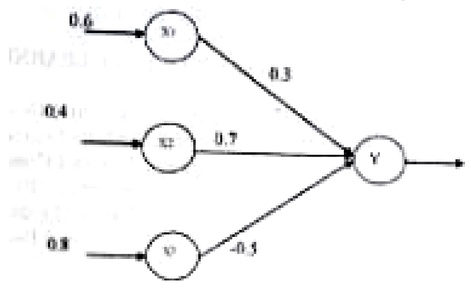
Step-2

In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

$$Y = f\left(\sum wi * xi + b\right)$$

Example:

For the given network shown in figure, Calculate the net input to the output neuron Y. (Or) Model Problem No Value



Let's calculate the output of a perceptron given the inputs and weights:

Inputs: $x1 = 0.6, x2 = 0.4, x3 = 0.8$,

Weight: $w1 = 0.3, w2 = 0.7, w3 = -0.5$

Bias=0

Perceptron Calculation

Step 1: Calculate the Weighted Sum

$$z = w1 \cdot x1 + w2 \cdot x2 + w3 \cdot x3$$

Calculate the weighted sum:

$$\boldsymbol{z = 0.3 \cdot 0.6 + 0.7 \cdot 0.4 + (-0.5) \cdot 0.8}$$

z=0.18+0.28−0.4

z=0.46−0.4

z=0.06

Step 2: Activation Function

For a simple perceptron, the step activation function is commonly used:

The output of the perceptron is Z=1

Types of Perceptron Models

Based on the layers, Perceptron models are divided into two types. These are as follows:

1. Single-layer Perceptron Model
2. Multi-layer Perceptron model

Single Layer Perceptron Model

This is one of the easiest artificial neural networks (ANN) types. A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.

In a single layer perceptron model, its algorithms do not contain recorded data, so it begins with inconstantly allocated input for weight parameters. Further, it sums up all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.

If the outcome is same as pre-determined or threshold value, then the performance of this model is stated as satisfied, and weight demand does not change. However, this model consists of a few discrepancies triggered when multiple weight inputs values are fed into the model. Hence, to find desired output and minimize errors, some changes should be necessary for the weights input.

"Single-layer perceptron can learn only linearly separable patterns."

Multi-Layered Perceptron Model

Like a single-layer perceptron model, a multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.

The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:

Forward Stage: Activation functions start from the input layer in the forward stage and terminate on the output layer.

Backward Stage: In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.

Hence, a multi-layered perceptron model has considered as multiple artificial neural networks having various layers in which activation function does not remain linear, similar to a single layer perceptron model. Instead of linear, activation function can be executed as sigmoid, TanH, ReLU, etc., for deployment.

A multi-layer perceptron model has greater processing power and can process linear and non-linear patterns. Further, it can also implement logic gates such as AND, OR, XOR, NAND, NOT, XNOR, NOR.

Advantages of Multi-Layer Perceptron:

- A multi-layered perceptron model can be used to solve complex non-linear problems.
- It works well with both small and large input data.
- It helps us to obtain quick predictions after the training.
- It helps to obtain the same accuracy ratio with large as well as small data.

Disadvantages of Multi-Layer Perceptron:

- In Multi-layer perceptron, computations are difficult and time-consuming.
- In multi-layer Perceptron, it is difficult to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of the training.

Table 1.2 Differences between SLP and MLP

Feature	**Single-Layer Perceptron (SLP)**	**Multi-Layer Perceptron (MLP)**
Layers	One layer (input to output)	Multiple layers (input, hidden, output)
Activation Function	Step function	Sigmoid, ReLU, Tanh, etc.
Solves Linearly Separable Problems?	Yes	Yes

Solves Non-Linearly Separable Problems?	No	Yes (e.g., XOR)
Learning Ability	Limited	Higher learning capability
Training Algorithm	Perceptron Learning Rule	Backpropagation

Perceptron Function

Perceptron function "f(x)" can be achieved as output by multiplying the input 'x' with the learned weight coefficient 'w'.

Mathematically, we can express it as follows:

$$f(x) = 1; if w.x + b > 0 otherwise, f(x) = 0$$

- 'w' represents real-valued weights vector
- 'b' represents the bias
- 'x' represents a vector of input x values.

Characteristics of Perceptron

The perceptron model has the following characteristics.

- Perceptron is a machine learning algorithm for supervised learning of binary classifiers.
- In Perceptron, the weight coefficient is automatically learned.
- Initially, weights are multiplied with input features, and the decision is made whether the neuron is fired or not.
- The activation function applies a step rule to check whether the weight function is greater than zero.
- The linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.
- If the added sum of all input values is more than the threshold value, it must have an output signal; otherwise, no output will be shown.

Limitations of Perceptron Model

A perceptron model has limitations as follows:

The output of a perceptron can only be a binary number (0 or 1) due to the hard limit transfer function.

Perceptron can only be used to classify the linearly separable sets of input vectors. If input vectors are non-linear, it is not easy to classify them properly.

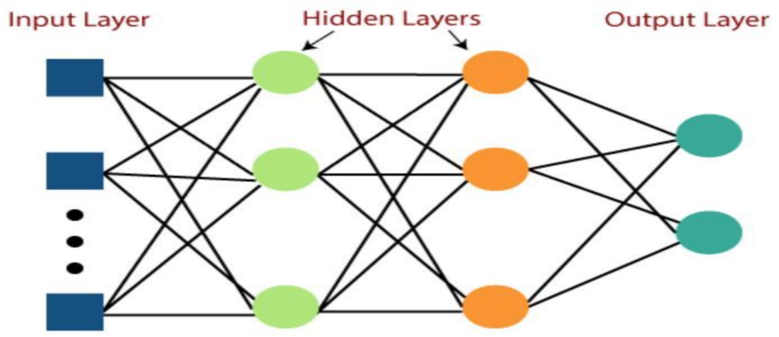


Figure 1.11: MLP

MLP networks are used for supervised learning format. A typical learning algorithm for MLP networks is also called backpropagation's algorithm.

A multilayer perceptron (MLP) is a feedforward artificial neural network that generates a set of outputs from a set of inputs. An MLP is characterized by several layers of input nodes connected as a directed graph between the input nodes connected as a directed graph between the input and output layers. MLP uses backpropagation for training the network. MLP is a deep learning method.

1.2 Functions in Neural networks

In neural networks, functions play a crucial role in processing inputs, learning patterns, and producing outputs. The most important functions used in neural networks are explained below.

Activation Function

An **activation function** is a mathematical function applied to a neuron's output in a neural network. It determines whether a neuron should be activated or not, helping the network learn **complex patterns**.

Why Are Activation Functions Important?

- Introduce **non-linearity**, allowing the neural network to learn **complex relationships**.
- Help in **backpropagation** by controlling gradients.
- Prevent problems like **vanishing gradients** in deep networks.

Linear Activation Function

$$f(x)=x$$

Output is **directly proportional** to the input.

- **Used in regression models** but **not suitable for deep learning** as it does not introduce non-linearity.

Example:

If x=2, then f(2)=2

Problem: Cannot model **complex patterns** in deep networks.

Non-Linear Activation Functions

Sigmoid Function (Logistic Function)

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

- Outputs a value between **0 and 1**.

- Used in **binary classification** (e.g., Yes/No, Spam/Not Spam).

Example Calculation:

If x=0, then

$$f(0) = \frac{1}{1+e^0} = 0.5$$

Pros:

✓ Used in **binary classification** problems.

✓ Converts inputs into probabilities.

Cons:

✗ Causes **vanishing gradient** problem in deep networks.

✗ Not ideal for **very large or very small values** (outputs saturate at 0 or 1).

Tanh (Hyperbolic Tangent) Function:

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

- Outputs values between **-1 and 1**.
- Used in **hidden layers** to center data around zero.

Example Calculation:

If x=0, then

$$f(0) = \frac{e^0 - e^{-0}}{e^0 + e^{-0}} = 0$$

Pros:

✓ Helps in **zero-centered data** (better learning).

✓ Works better than **Sigmoid** for hidden layers.

Cons:

✗ Still suffers from **vanishing gradient** problem.

ReLU (Rectified Linear Unit) Function:

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

- Outputs **0** if xxx is negative and **x** if xis positive.
- **Most widely used** in deep learning.

Example Calculation:

If x=-3 then f(-3) =0

If x=4 then f(4)=4

Pros:

✓ Solves the **vanishing gradient problem**.

✓**Computationally efficient**.

✓ Works well in deep networks.

Cons:

Dying ReLU problem: Neurons can get stuck at 0 for all inputs if weights are poorly initialized.

Leaky ReLU Function:

$$f(x) = max(0.01x, x)$$

- Like **ReLU**, but with a small slope (0.01) for negative inputs.
- Helps fix the **Dying ReLU** problem.

Example Calculation:

If x=−3 then f(−3)=−0.03.

If x=4, then f(4)=4.

Pros:

✓ Solves **Dying ReLU problem**.

✓ Works well for **negative values**.

Cons:

✗ Still **not entirely zero-centered**.

Softmax Function (for Multi-Class Classification)

$$f(x_i) = \frac{e^{x_i}}{\sum_j \quad e^{x_j}}$$

- Converts outputs into **probabilities** for multiple classes.
- Used in the **output layer of classification models** (e.g., image classification).

Example Calculation:

For scores **[2, 1, 0]**,

$$f(2) = \frac{e^2}{e^2 + e^1 + e^0} = 0.66$$

$$f(1) = \frac{e^1}{e^2 + e^1 + e^0} = 0.24$$

$$f(0) = \frac{e^0}{e^2 + e^1 + e^0} = 0.10$$

Pros:

✓ Converts raw outputs into **probabilities**.

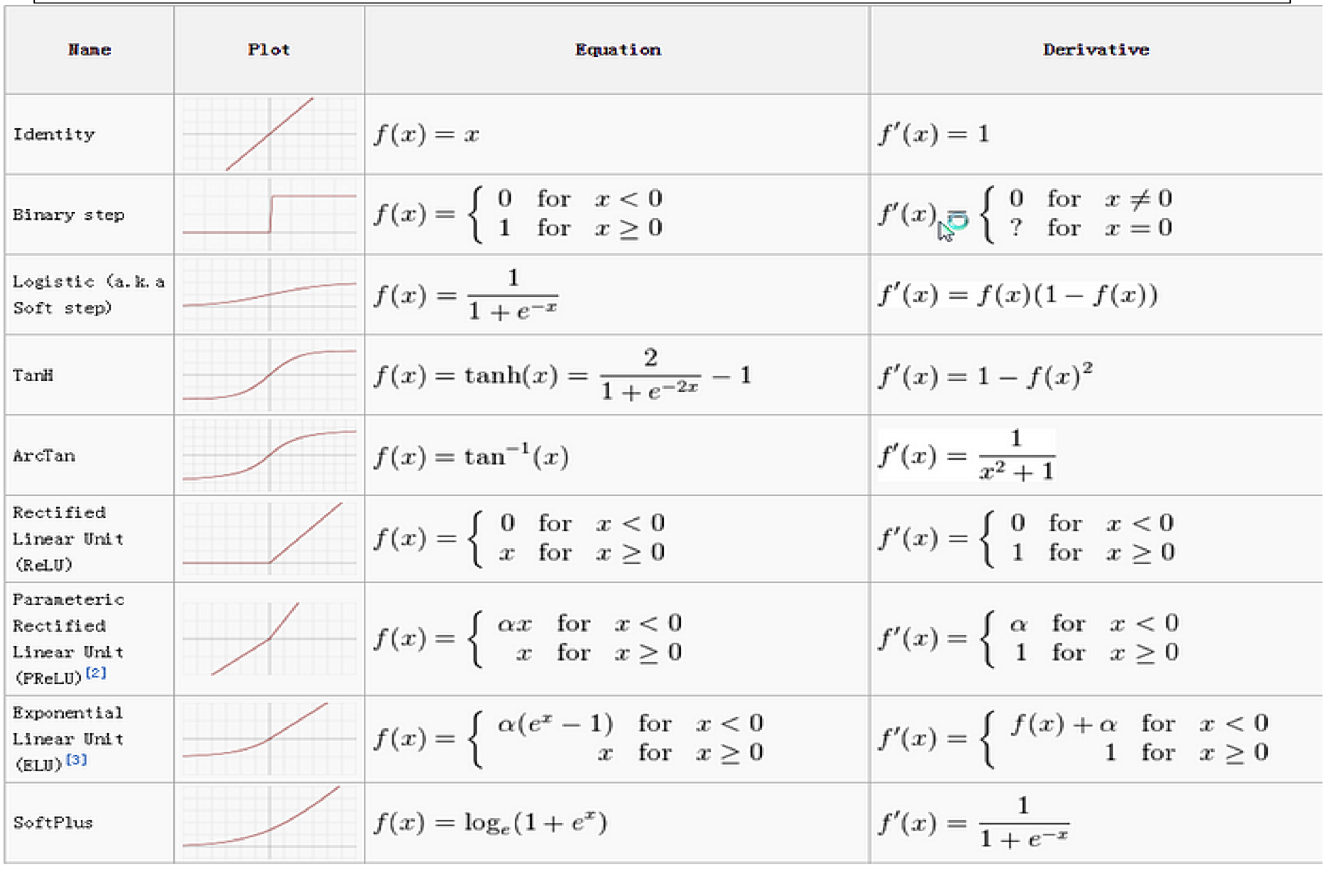
✓ Ensures the sum of outputs = **1**.

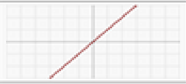
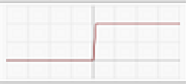
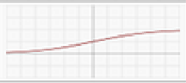
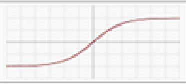
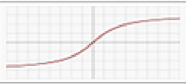
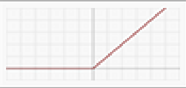
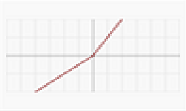
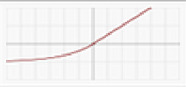
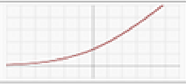
Cons:

✗**Computationally expensive** due to exponentiation.

Table 1.3 Choosing the Right Activation Function

Activation Function	**Range**	**Use Case**
Linear	$+\infty$	Regression models
Sigmoid	0 to 1	Binary classification (output layer)
Tanh	-1 to 1	Hidden layers for zero-centered data
ReLU	0 to $+\infty$	Most common for hidden layers
Leaky ReLU	$-\infty\ to + \infty$	Fixes dying ReLU issue
Softmax	0 to 1	Multi-class classification (output layer)



Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} & x < 0 \\ 1 & \text{for} & x \ge 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} & x \ne 0 \\ ? & \text{for} & x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for} & x < 0 \\ x & \text{for} & x \ge 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} & x < 0 \\ 1 & \text{for} & x \ge 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for} & x < 0 \\ x & \text{for} & x \ge 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for} & x < 0 \\ 1 & \text{for} & x \ge 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for} & x < 0 \\ x & \text{for} & x \ge 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for} & x < 0 \\ 1 & \text{for} & x \ge 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

1.3 CLASSIFICATION AND CLUSTERING PROBLEMS

In the field of Deep Learning and Neural Networks, ANN plays a crucial role in solving both classification and clustering problems by learning patterns from data. In classification tasks, ANN learns a mapping between input features and output class labels using labelled training data. Classification is a supervised learning process in which data is assigned to predefined classes or labels based on learned patterns.

The process involves

- Input data is fed into the **input layer.**
- Data is multiplied by **weights** and added with **bias.**
- The **net input** is calculated.
- An **activation function** is applied to produce output.
- The output is compared with the actual label.
- Error is calculated and minimized using **backpropagation**.
- Weights are updated iteratively to improve accuracy.

The architecture of Artificial Neural Networks (ANN)

- The **architecture of Artificial Neural Networks (ANNs)** refers to the structure and organization of the various layers, nodes (neurons), and connections that make up the network. The architecture is designed to simulate how the human brain processes information, allowing ANNs to learn from data and make predictions or decisions.

Here's a breakdown of the key components and architecture of an Artificial Neural Network:

1. Neurons (Nodes)

- **Basic Unit**: A neuron is the fundamental processing unit of an ANN. Each neuron receives one or more inputs, processes them, and produces an output.
- **Structure**: Neurons are organized in layers, where each neuron is connected to neurons in adjacent layers.
- **Activation Function**: Neurons use an activation function to determine the output. Common activation functions include:
    - **Sigmoid**: Outputs values between 0 and 1.
    - **ReLU (Rectified Linear Unit)**: Outputs values between 0 and infinity, making it popular in deep learning due to its simplicity and effectiveness.
    - **Tanh**: Outputs values between -1 and 1.
    - **Softmax**: Often used in the output layer for multi-class classification problems.

2. Layers in ANN Architecture

ANNs are structured as a series of layers. The basic architecture consists of three main types of layers:

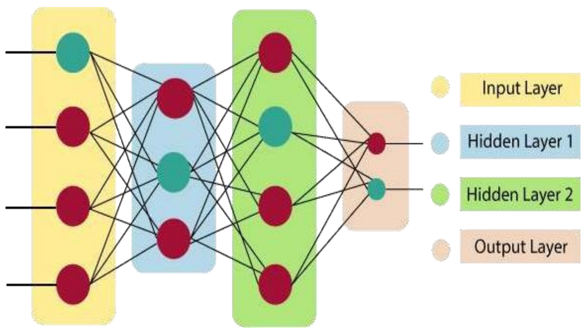


Figure 1.12: ANN

a. Input Layer

- **Role**: This is where the network receives external data. Each neuron in the input layer corresponds to one feature of the input data.
- **Example**: In image recognition, each pixel in the image might correspond to a neuron in the input layer.

b. Hidden Layers

- **Role**: These layers process information from the input layer and pass it forward. An ANN can have one or more hidden layers. More hidden layers allow the network to model more complex relationships in the data, a feature that is crucial in deep learning (with deep neural networks).
- **Function**: Neurons in the hidden layers apply transformations and use activation functions to generate outputs that will be passed to the next layer.

c. Output Layer

- **Role**: The output layer produces the final result of the network's computation. It usually contains one neuron for each class in classification tasks (e.g., one for each possible label) or a single neuron for regression tasks (e.g., to predict a numerical value).
- **Activation Function**: The activation function in the output layer depends on the task:
  - **Sigmoid** or **Softmax** for classification tasks.
  - **Linear** for regression tasks.

$$\sum_{i=1}^{n} \mathrm{Wi} * \mathrm{Xi} + \mathrm{b}$$

3. Weights and Biases

- **Weights**: Each connection between neurons has an associated weight that determines the strength of the connection. The network adjusts these weights during training to minimize the error in predictions.
- **Biases**: Each neuron, except for those in the input layer, has a bias term that allows the network to make better predictions, enabling it to shift the activation function to better fit the data.

4. Connections between Neurons

- **Fully Connected**: In a **fully connected neural network**, each neuron in one layer is connected to every neuron in the next layer.
- **Feedforward**: The data flows in one direction from the input to the output without any cycles or loops. This is called a **feedforward neural network**.
- **Recurrent Connections**: In networks like **Recurrent Neural Networks (RNNs)**, neurons can have connections that loop back on themselves, allowing the network to maintain a form of memory or context.

1.3.1 Types of Artificial Neural Network:

- There are various types of Artificial Neural Networks (ANN) depending upon the human brain neuron and a network function, an artificial neural network similarly performs tasks.
- The majority of the artificial neural networks will have some similarity with a more complex biological partner and are very effective at their expected tasks.
- For example, segmentation or classification.

1. **Feedback ANN:**

- In this type of ANN, the output returns into the network to accomplish the best-evolved results internally.
- As per the University of Massachusetts, Lowell Centre for Atmospheric Research.
- The feedback networks feed information back into itself and are well suited to solve optimization issues. The internal system error corrections utilize feedback ANNs.

2. **Feed-Forward ANN:**

   - A feed-forward network is a basic neural network comprising of an input layer, an output layer, and at least one layer of a neuron.
   - Through assessment of its output by reviewing its input, the intensity of the network can be noticed based on group behavior of the associated neurons, and the output is decided.
   - The primary advantage of this network is that it figures out how to evaluate and recognize input patterns.

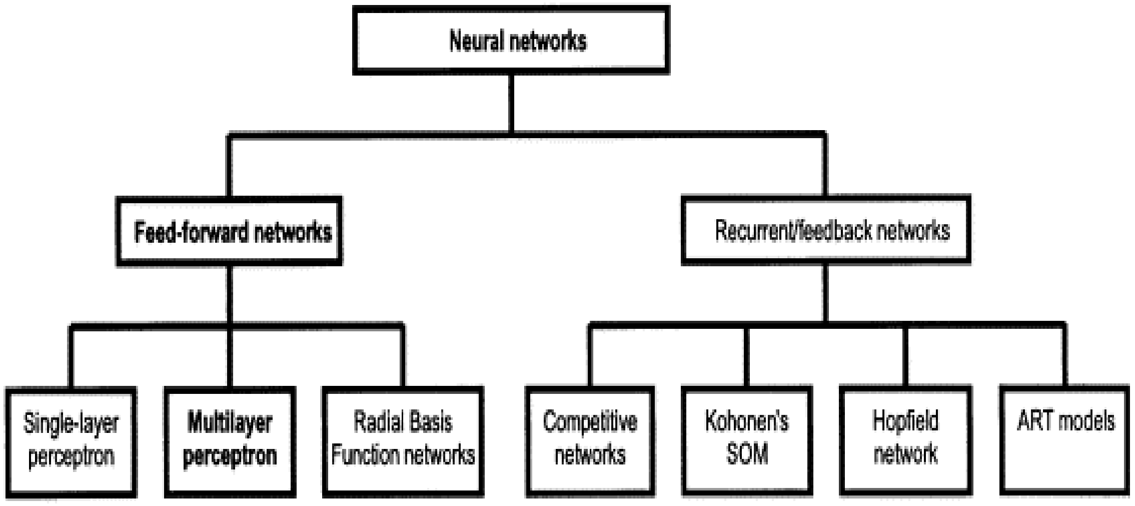


Figure 1.13: Basic Model of ANN

3. **Types of Modular Neural Networks (MNNs)**

It is one of the fastest-growing areas of Artificial Intelligence.

1. Feedforward Neural Network – Artificial Neuron
2. Radial Basis Function Neural Network
3. Kohonen Self Organizing Neural Network
4. Recurrent Neural Network (RNN)

5. Convolutional Neural Network (CNN)
6. Long / Short Term Memory

Feedforward Neural Network(FNN)-Artificial Neuron:

- A Feedforward Neural Network, also known as an Artificial Neural Network, is the most basic form of neural networks.
- It consists of input, hidden, and output layers of artificial neurons.
- The information flows only in one direction, from the input layer through the hidden layers to the output layer.
- Each neuron in the network processes the input data and passes the output to the next layer without any feedback loop.
- FNNs are commonly used for tasks such as classification and regression.

Radial Basis Function Neural Network (RBFNN):

- The Radial Basis Function Neural Network is a type of feedforward neural network that uses radial basis functions as activation functions.
- These functions evaluate the distance between the input data and a set of learned centers in a multidimensional space.
- RBFNNs are often employed for tasks like function approximation, interpolation, and pattern recognition.

Kohonen Self-Organizing Neural Network (SOM or Kohonen Network):

- The Kohonen Self Organizing Neural Network, named after its inventor Teuvo Kohonen, is an unsupervised learning network.
- It is used for clustering and dimensionality reduction.
- The network organizes its neurons into a grid, and during the learning process, similar input patterns lead to the activation of nearby neurons, causing the network to self-organize and learn the underlying data distribution.

Recurrent Neural Network(RNN):

- Recurrent Neural Networks are designed to handle sequential data by introducing feedback loops in the network architecture.
- These loops allow information to persist, making RNNs capable of processing variable-length sequences.
- RNNs have applications in natural language processing, speech recognition, time series analysis, and more.

Convolutional Neural Network (CNN):

- Convolutional Neural Networks are specialized for processing grid-like data, such as images and videos.

- They utilize convolutional layers to automatically learn and extract meaningful features from the input data.
- CNNs have revolutionized computer vision tasks, achieving state-of-the-art results in image classification, object detection, and image generation.

a) CNNs are primarily used for image and video analysis. They are designed to automatically and adaptively learn spatial hierarchies of features from input data.

b) CNNs consist of convolutional layers that apply filters to input data, followed by pooling layers to reduce dimensionality, and fully connected layers for classification or regression.

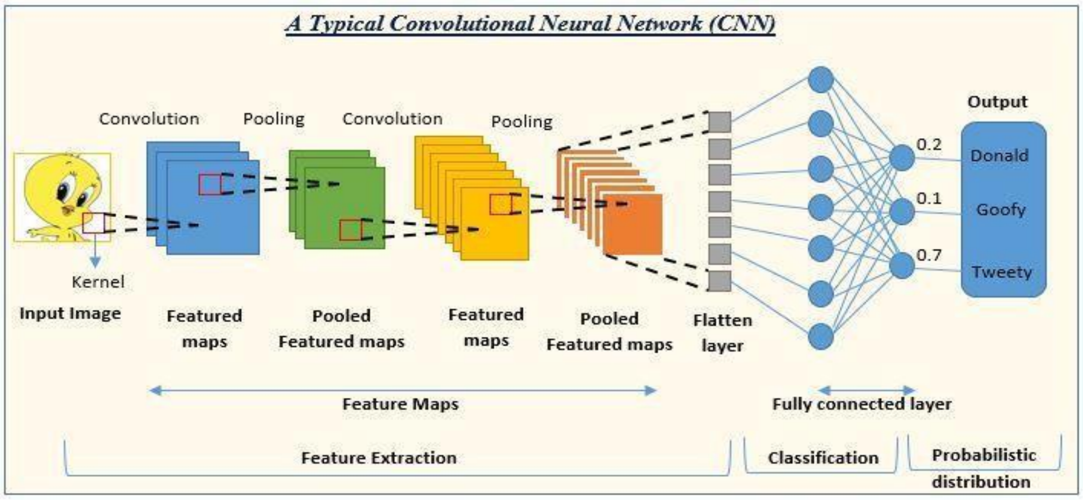


Figure 1.14: CNN

Long/Short Term Memory (LSTM):

- Long Short-Term Memory is a variant of Recurrent Neural Networks.
- LSTMs are designed to address the vanishing gradient problem, which can occur in traditional RNNs when learning long-term dependencies.
- LSTMs use memory cells with gating mechanisms that allow them to remember or forget information over extended sequences.
- They are widely used in tasks involving sequential data, such as language modeling, machine translation, and speech recognition.
- Each of these neural network architectures has its strengths and applications, contributing to the diverse landscape of Artificial Intelligence.

1.3.2 Important Terminologies in Artificial Neural Networks (ANNs)

1. Neuron (Node or Perceptron)

- The **basic unit** of an artificial neural network.
- Inspired by **biological neurons** in the human brain.
- Each neuron receives inputs, applies **weights and bias**, processes them through an **activation function**, and produces an output.

Mathematical Formula: $y = f(\sum \quad w_i x_i + b)$

where:

- W_i = weight of input x_i
- b = bias
- f(x) = activation function

2. Layers of ANN

- **Input Layer**: Receives the raw input data.
- **Hidden Layers**: Perform feature extraction and computation.
- **Output Layer**: Produces the final result (e.g., classification label or regression value).

3. Weights (w)

- Coefficients assigned to each input to determine **importance**.
- Higher weight = **stronger influence** on the output.

4. Bias (b)

- A **constant value** added to the weighted sum of inputs.
- Helps shift the activation function to make the model more flexible.

Example:

For a single neuron:

$$y = w_1x_1 + w_2x_2 + b$$

5. Activation Function

- Introduces **non-linearity** in the neural network.
- Determines whether a neuron should be activated.

Common Activation Functions:

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

ReLu: $f(x) = max(0, x)$

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

6. Forward Propagation

- Process where inputs are passed **forward** through the network.
- Computes weighted sums, applies activation functions, and produces an output.

7. Loss Function (Cost Function)

- Measures **error** between predicted and actual outputs.
- Common loss functions:
  - **Mean Squared Error (MSE)** (for regression): $MSE = \frac{1}{n}\sum(y_{true} - y_{pred})^2$
  - **Cross-Entropy Loss** (for classification): $L = -\sum y \log(\hat{y})$

8. Backpropagation

- An optimization algorithm that updates **weights** using **gradient descent**.

- Steps:
    1. Compute the **error** (loss function).
    2. Calculate **gradients** of the error with respect to weights.
    3. Adjust weights to minimize error.
    4.

9. Gradient Descent

- An algorithm that updates weights to minimize loss.
- Formula: $w = w - \eta \frac{\partial L}{\partial w}$
- where:
    - η= learning rate
    - L= loss function

Types of Gradient Descent:

- **Batch Gradient Descent** – Updates weights after all training samples.
- **Stochastic Gradient Descent (SGD)** – Updates weights after **each** training sample.
- **Mini-Batch Gradient Descent** – Updates weights after a **small batch** of samples.

10. Learning Rate (η)

- A small **positive number** that controls the step size in weight updates.
- **Too high** → Model may **not converge**.
- **Too low** → Training becomes **slow**.

11. Epochs, Batch Size, and Iterations

- **Epoch**: One complete pass through the entire dataset.
- **Batch Size**: Number of training samples used in **one forward and backward pass**.
- **Iteration**: Number of batches processed in one epoch.

Example: If dataset has **10,000 samples** and batch size is **100**, then:

$$\text{Iterations per epoch} = \frac{10{,}000}{100} = 100$$

12. Overfitting & Underfitting

- **Overfitting**: Model learns **too much detail**, including noise, leading to poor generalization.
- **Underfitting**: Model is **too simple**, failing to capture patterns in data.

Solutions for Overfitting:

- **Regularization (L1 & L2)**
- **Dropout**
- **More training data**

13. Regularization (L1 & L2)

- **L1 Regularization (Lasso Regression)**

$$\text{Loss} = \sum (y_{true} - y_{pred})^2 + \lambda \sum |w|$$

  - Encourages **sparsity** by making some weights **zero**.
- **L2 Regularization (Ridge Regression)**
  - Loss $= \sum(y_{true} - y_{pred})^2 + \lambda \sum w^2$

Reduces overfitting by **penalizing large weights**.

14. Dropout

- A technique to randomly **drop** neurons during training to prevent **overfitting**.
- Helps the model generalize better.

15. Optimizers

- Algorithms that adjust weights during training.
- **Common Optimizers:**
  - SGD (Stochastic Gradient Descent)
  - Adam (Adaptive Moment Estimation)
  - RMSprop (Root Mean Square Propagation)

16. Convergence

- When the model reaches a point where **loss stops decreasing** significantly.

17. Weight Initialization

- Assigning **initial** values to weights.
- Good initialization prevents vanishing/exploding gradients.

  Common Methods:
- Random Initialization
- Xavier/Glorot Initialization
- He Initialization

18. Hyper parameters

- **User-defined** parameters that control learning.
- Examples:
  - Learning rate
  - Number of hidden layers
  - Number of neurons per layer

19. Model Evaluation Metrics

- **Classification Metrics:**
  - **Accuracy** $= \frac{\text{correct predictions}}{\text{total predictions}}$

- **Precision, Recall, F1-Score**
- **Confusion Matrix**

- **Regression Metrics:**
    - **Mean Squared Error (MSE)**
    - **Mean Absolute Error (MAE)**
    - **R-Squared Score**

20. Transfer Learning

- Using a **pre-trained model** on a new task (e.g., using ResNet for image classification).

21. Auto encoders

- A type of **unsupervised neural network** for feature learning and data compression.

Consider the following five training example

X	Y
2	**8.8978**
3	**11.7586**
4	**15.3192**
5	**18.3129**
6	**20.1351**

We want to learn the function f(x) of the form (x)=ax+b which is parameterized (a,b). Using squared error as the loss function which of the following parameters would you use to model this function to get a solution with minimum loss? (4,3), (1,4), (4,1) (3,4). (Nov/Dec 2024) (8 marks)

It determine which parameters (a,b) best fit the function f(x)=ax+b with minimum squared error, we need to perform **linear regression** using the given data. The **least squares method** minimizes the sum of squared errors (SSE), given by:

$$SSE = \sum\left(Y_i - (aX_i + b)\right)^2$$

Step 1: Compute Mean of X and Y

Given data points:

X	Y
2	8.8978
3	11.7586
4	15.3192
5	18.3129
6	20.1351

Calculate the means:

$$\bar{X} = \frac{2+3+4+5+6}{5} = \frac{20}{5} = 4$$

$$\bar{Y} = \frac{8.8978 + 11.7586 + 15.3192 + 18.3129 + 20.1351}{5} = \frac{74.4236}{5} = 14.88472$$

Step 2: Compute Slope a

The formula for a (slope) in simple linear regression:

$$a = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sum(X_i - \bar{X})^2}$$

Step 3: Compute Intercept b

$$b = \bar{Y} - a\bar{X}$$

Now, let’s calculate these values to find the best-fitting parameters.

The computed best-fit parameters for f(x)=ax+bare:

- a≈2.9029
- b≈3.2732

1.3.3 Supervised Learning Network

- Supervised learning, also known as supervised machine learning, is a subcategory of machine learning and artificial intelligence.
- It is defined by its use of labelled datasets to train algorithms that to classify data or predict outcomes accurately.
- As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross validation process.
- Supervised learning helps organizations solve for a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox.

How supervised learning works?

- Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time.
- The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.
- Supervised learning can be separated into two types of problems when data mining—**classification and regression:**
- **Classification** uses an algorithm to accurately assign test data into specific categories.
- It recognizes specific entities within the dataset and attempts to draw some conclusions on how those entities should be labeled or defined.

- Common classification algorithms are linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbour, and random forest, which are described in more detail below.
- **Regression** is used to understand the relationship between dependent and independent variables.
- It is commonly used to make projections, such as for sales revenue for a given business. Linear regression, logistical regression, and polynomial regression are popular regression algorithms.

Supervised learning algorithms

- Various algorithms and computations techniques are used in supervised machine learning processes.
- Below are brief explanations of some of the most commonly used learning methods, typically calculated through use of programs like R or Python:

- **Neural networks:**
  - Primarily leveraged for deep learning algorithms, neural networks process training data by mimicking the interconnectivity of the human brain through layers of nodes.
  - Each node is made up of inputs, weights, a bias (or threshold), and an output.
  - If that output value exceeds a given threshold, it "fires" or activates the node, passing data to the next layer in the network.
  - Neural networks learn this mapping function through supervised learning, adjusting based on the loss function through the process of gradient descent.
  - When the cost function is at or near zero, we can be confident in the model's accuracy to yield the correct answer.
- **Naive bayes:**
  - Naive Bayes is classification approach that adopts the principle of class conditional independence from the Bayes Theorem.
  - This means that the presence of one feature does not impact the presence of another in the probability of a given outcome, and each predictor has an equal effect on that result.
  - There are three types of Naïve Bayes classifiers: Multinomial Naïve Bayes, Bernoulli Naïve Bayes, and Gaussian Naïve Bayes.
  - This technique is primarily used in text classification, spam identification, and recommendation systems.
- **Linear regression:**
  - Linear regression is used to identify the relationship between a dependent variable and one or more independent variables and is typically leveraged to make predictions about future outcomes.
  - When there is only one independent variable and one dependent variable, it is known as simple linear regression.
  - As the number of independent variables increases, it is referred to as multiple linear regression.
  - For each type of linear regression, it seeks to plot a line of best fit, which is calculated through the method of least squares.

- However, unlike other regression models, this line is straight when plotted on a graph.

- **Logistic regression:**
  - While linear regression is leveraged when dependent variables are continuous, logistic regression is selected when the dependent variable is categorical, meaning they have binary outputs, such as "true" and "false" or "yes" and "no."
  - While both regression models seek to understand relationships between data inputs, logistic regression is mainly used to solve binary classification problems, such as spam identification.
- **Support vector machines (SVM):**
  - A support vector machine is a popular supervised learning model developed by Vladimir Vapnik, used for both data classification and regression.
  - It is typically leveraged for classification problems, constructing a hyperplane where the distance between two classes of data points is at its maximum.
  - This hyperplane is known as the decision boundary, separating the classes of data points (e.g., oranges vs. apples) on either side of the plane.
- **K-nearest neighbor:**
  - K-nearest neighbor, also known as the KNN algorithm, is a non-parametric algorithm that classifies data points based on their proximity and association to other available data.
  - This algorithm assumes that similar data points can be found near each other.
  - As a result, it seeks to calculate the distance between data points, usually through Euclidean distance, and then it assigns a category based on the most frequent category or average.
  - Its ease of use and low calculation time make it a preferred algorithm by data scientists, but as the test dataset grows, the processing time lengthens, making it less appealing for classification tasks. KNN is typically used for recommendation engines and image recognition.

Supervised learning examples

Supervised learning models can be used to build and advance a number of business applications, including the following:

Image- and object-recognition:

- Supervised learning algorithms can be used to locate, isolate, and categorize objects out of videos or images, making them useful when applied to various computer vision techniques and imagery analysis.

- **Predictive analytics:**

- A widespread use case for supervised learning models is in creating predictive analytics systems to provide deep insights into various business data points.
- This allows enterprises to anticipate certain results based on a given output variable, helping business leaders justify decisions or pivot for the benefit of the organization.

Customer sentiment analysis:

- Using supervised machine learning algorithms, organizations can extract and classify important pieces of information from large volumes of data—including context, emotion, and intent—with very little human intervention.
- This can be incredibly useful when gaining a better understanding of customer interactions and can be used to improve brand engagement efforts.

Spam detection:

- Spam detection is another example of a supervised learning model.
- Using supervised classification algorithms, organizations can train databases to recognize patterns or anomalies in new data to organize spam and non-spam-related correspondences effectively.

Challenges of supervised learning

- Supervised learning models can require certain levels of expertise to structure accurately.
- Training supervised learning models can be very time intensive.
- Datasets can have a higher likelihood of human error, resulting in algorithms learning incorrectly.
- Unlike unsupervised learning models, supervised learning cannot cluster or classify data on its own.

Supervised, Unsupervised, and Reinforcement Learning

Machine Learning is a subset of Artificial Intelligence (AI) that enables computers to learn from data and make decisions without explicit programming. It is broadly categorized into **Supervised Learning, Unsupervised Learning, and Reinforcement Learning**.

1. Supervised Learning

Supervised learning is a type of machine learning where the model is trained using **labeled data**. Each input data has a corresponding correct output, allowing the model to learn a mapping function between input and output.

The goal is to make accurate predictions when given new, unseen data.

Examples:

1. **Classification** – Predicting if an email is spam or not.
2. **Regression** – Predicting house prices based on features like location and size.

Common Algorithms:

- **Linear Regression** – Used for predicting continuous values.
- **Decision Trees** – Used for both classification and regression.
- **Support Vector Machines (SVM)** – Used for separating data into different categories.
- **Neural Networks** – Used for complex pattern recognition.

Real-World Applications:

- Fraud detection in banking.
- Medical diagnosis (predicting diseases).

- Image recognition (identifying objects in photos).

Analogy: A student learning from a teacher with correct answers provided for reference.

2. Unsupervised Learning

Unsupervised learning is a type of machine learning where the model is trained on **unlabeled data**. There are no predefined outputs, and the algorithm identifies hidden patterns and relationships within the data.

The goal is to find structure **in the data, such as grouping similar data points or reducing complexity.**

Examples:

1. **Clustering** – Grouping customers based on purchasing behavior.
2. **Anomaly Detection** – Detecting fraudulent transactions in banking.
3. **Dimensionality Reduction** – Reducing data complexity for visualization.

Common Algorithms:

- **K-Means Clustering** – Groups data into clusters based on similarity.
- **Hierarchical Clustering** – Builds a tree-like structure of clustered data.
- **Principal Component Analysis (PCA)** – Reduces the number of variables in a dataset while preserving key information.

Real-World Applications:

- Market segmentation in business.
- Organizing news articles based on topic.
- Identifying criminal activities from transaction patterns.

Analogy: A student sorting books in a library without knowing the categories beforehand.

3. Reinforcement Learning (RL)

Reinforcement learning is a machine learning approach where an **agent learns by interacting with an environment** and receiving rewards or penalties based on its actions.

The goal is to maximize cumulative rewards by learning the best sequence of actions over time.

Key Components:

- **Agent** – The learner or decision-maker (e.g., a robot, AI in a game).
- **Environment** – The world in which the agent operates.
- **Actions** – The moves the agent can make.
- **Reward** – The feedback from the environment (positive or negative).

Examples:

1. **Game AI** – Training an AI to play chess or Go.
2. **Self-Driving Cars** – Learning how to navigate safely.
3. **Robotics** – Teaching robots to walk or complete tasks.

Common Algorithms:

- **Q-Learning** – A model-free algorithm for learning optimal policies.

- **Deep Q Networks (DQN)** – Uses deep learning to improve Q-Learning.
- **Policy Gradient Methods** – Directly optimize policies for better actions.

Real-World Applications:

- Autonomous robots in industries.
- Stock market trading algorithms.
- AI assistants improving responses over time.

Analogy: Training a pet using rewards (treats) and punishments to shape behavior.

Table 1.4 Difference Between Supervised, Unsupervised, and Reinforcement Learning

Feature	Supervised Learning	Unsupervised Learning	Reinforcement Learning
1. Definition	Learns from **labeled data** (input-output pairs).	Learns from **unlabeled data**, finding hidden patterns.	Learns through **trial and error**, receiving rewards or penalties.
2. Objective	Predict outcomes for new data.	Identify structures, clusters, or relationships in data.	Take optimal actions to maximize long-term rewards.
3. Training Data	Labeled (contains correct answers).	Unlabeled (no predefined correct answers).	No predefined dataset; learns from experience.
4. Learning Type	**Direct** – Mapping input to output.	**Indirect** – Identifies patterns and groups.	**Interactive** – Learns by interacting with an environment.
5. Example Tasks	Classification (spam detection), Regression (house price prediction).	Clustering (customer segmentation), Anomaly detection (fraud detection).	Game playing (chess AI), Robotics (self-driving cars).
6. Common Algorithms	Linear Regression, SVM, Neural Networks.	K-Means, Hierarchical Clustering, PCA.	Q-Learning, Deep Q Networks (DQN), Policy Gradients.
7. Output Type	A specific prediction (label or value).	A structure or grouping in data.	A sequence of actions (policy) for decision-making.
8. Dependency on Feedback	Uses direct feedback (correct labels).	No feedback; finds its own structure.	Receives delayed rewards or penalties.
9. Application Fields	Healthcare, finance, email filtering.	Market research, anomaly detection, recommendation systems.	Robotics, gaming, stock trading.
10. Human Involvement	Requires labeled datasets from humans.	Less human involvement; data patterns emerge naturally.	Requires designing a reward system but learns autonomously.
11. Computation Complexity	Moderate to high (depends on dataset size).	Lower than supervised (no need for labeling).	High (requires continuous learning and interaction).
12. Analogy	A student learning from a teacher with correct answers provided.	A student sorting books without predefined categories.	A child learning to ride a bicycle through trial and error.

1.4 DEEP NETWORKS BASICS

Deep learning is a subset of machine learning that uses deep neural networks to automatically learn hierarchical representations from large volumes of data, enabling high performance in complex real-world tasks such as image and speech recognition.

Machine Learning vs Deep Learning

Machine Learning (ML) is a subset of Artificial Intelligence in which systems learn patterns from data and make predictions without being explicitly programmed. Traditional machine learning algorithms rely heavily on **manual feature extraction**, where domain experts design features before training the model.

Deep Learning (DL) is a specialized subfield of machine learning that uses **deep neural networks with multiple hidden layers** to automatically learn hierarchical representations directly from raw data.

Table 1.5 Machine Learning vs Deep Learning

Aspect	Machine Learning	Deep Learning
Feature extraction	Manual	Automatic
Model depth	Shallow (1–2 layers)	Deep (many layers)
Data requirement	Small to medium	Large datasets
Performance	Limited for complex data	High accuracy for complex data
Examples	Linear regression, SVM	CNNs, RNNs, Transformers

A **deep neural network** contains multiple hidden layers between the input and output layers. Each layer learns increasingly **abstract features** from the data.

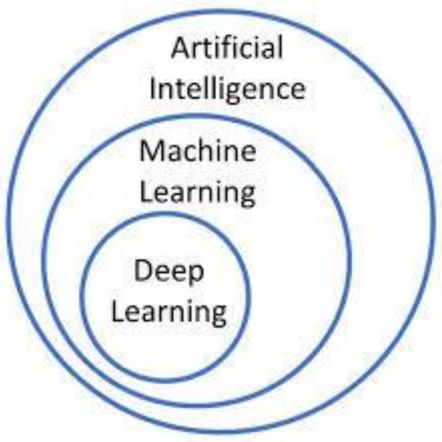


Figure 1.15: AI → ML → DL hierarchy

Hierarchical Feature Learning

- First layers learn simple patterns
- Middle layers learn combinations of patterns
- Deeper layers learn high-level concepts

Example: Image Recognition

- Layer 1 → edges and lines
- Layer 2 → shapes
- Layer 3 → object parts
- Output → full object (digit, face, car)

Deep learning models are powerful because they:

- Learn non-linear relationships
- Automatically extract features
- Scale well with large datasets
- Improve with more data and computation

Key enabling factors:

- Availability of large datasets
- Powerful GPUs
- Efficient algorithms like backpropagation

Deep learning excels in tasks where data is high-dimensional and unstructured.

Image Processing

- Handwritten digit recognition
- Face recognition
- Medical image analysis

(Explained in detail by Nielsen using the MNIST dataset)

Speech Recognition

- Voice assistants
- Speech-to-text systems

Text and Language

- Machine translation
- Sentiment analysis
- Chatbots

Handwritten digit recognition

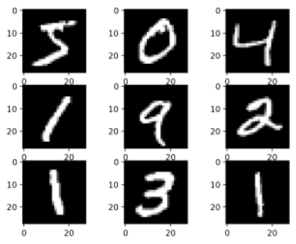


Figure 1.16: Handwritten digit recognition

- Input: pixel values of digit images.
- Hidden layers: detect patterns and shapes.
- Output: digit class (0–9).

This example clearly demonstrates:

- Feature learning without manual intervention
- Superiority of deep neural networks over simple classifiers

1.5 SHALLOW NEURAL NETWORKS

1.5.1 Introduction to Shallow Neural Networks

A **shallow neural network** is a type of **artificial neural network** that contains **only one hidden layer** between the input layer and the output layer. These networks represent the **earliest form of neural networks** and form the foundation for understanding deeper architectures.

Shallow networks are capable of learning **simple patterns and linear or mildly non-linear relationships** in data. However, their representational capacity is limited when compared to deep neural networks.

2. Structure of a Shallow Neural Network

A shallow neural network consists of **three main layers**:

1. **Input Layer**
   Receives raw input features and passes them to the hidden layer.
2. **Single Hidden Layer**
   Performs weighted summation of inputs followed by an activation function.
3. **Output Layer**
   Produces the final prediction or classification result.

Each neuron in the hidden layer computes a function of the form:

$$\sum wi * xi + b$$

followed by an activation function such as **sigmoid**, **tanh**, or **ReLU**.

Shallow neural networks give us basic idea about deep neural network which consist of only 1 or 2 hidden layers. Understanding a shallow neural network gives us an understanding into what exactly is going on inside a deep neural network A neural network is built using various hidden layers. Now that we know the computations that occur in a particular layer, let us understand how the whole neural network computes the output for a given input X. These can also be called the **forward-propagation** equations.

$$Z^{[1]} = W^{[1]T} X + b^{[1]}$$

$$A^{[1]} = \sigma \left(Z^{[1]} \right)$$

$$Z^{[2]} = W^{[2]T} A^{[1]} + b^{[2]}$$

$$\hat{y} = A^{[2]} = \sigma \left(Z^{[2]} \right)$$

1. The first equation calculates the intermediate output ***Z[1]*** of the first hidden layer.
2. The second equation calculates the final output ***A[1]*** of the first hidden layer.
3. The third equation calculates the intermediate output ***Z[2]*** of the output layer.
4. The fourth equation calculates the final output ***A[2]*** of the output layer which is also the final output of the whole neural network.

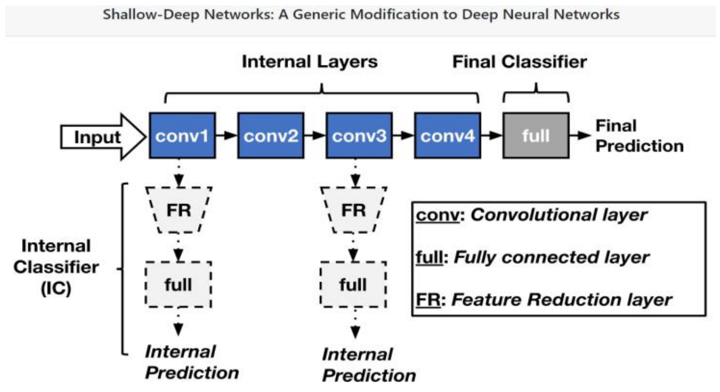


Figure 1.17:Shallow Networks – Generic Model

3. Mathematical Representation

In a shallow neural network, learning involves adjusting **weights and biases** to minimize a **loss function**.

- **Forward propagation** computes the output.
- **Loss function** (e.g., squared error or cross-entropy) measures error.
- **Gradient descent** updates parameters.

Shallow networks often use **gradient descent** or **stochastic gradient descent** for optimization.

4. Learning Capability and Limitations

Shallow neural networks can:

- Learn **linearly separable** and **simple non-linear functions**
- Perform **basic classification and regression tasks**

However, they suffer from several limitations:

- Limited representational power
- Cannot efficiently model **complex hierarchical features**
- Require a large number of neurons to approximate complex functions

Nielsen explains that while a single hidden layer network can theoretically approximate any function, in practice it becomes **computationally inefficient**.

5. Shallow Networks vs Deep Networks

Table 1.6: Difference between a Shallow Networks & Deep Learning Networks

Sl.No.	Shallow Net's	Deep Learning Net's
1	One Hidden layer(or very less no. of Hidden Layers)	Deep Net's has many layers of Hidden layers with more no. of neurons in each layers
2	Takes input only as VECTORS	DL can have raw data like image, text as inputs
3	Shallow net's needs more parameters to have better fit	DL can fit functions better with less parameters than a shallow network

4	Shallow networks with one Hidden layer (same no of neurons as DL) cannot place complex functions over the input space	DL can compactly express highly complex functions over input space
5	The number of units in a shallow network grows exponentially with task complexity.	DL don’t need to increase it size(neurons) for complex problems
6	Shallow network is more difficult to train with our current algorithms (e.g. it has issues of local minima etc)	Training in DL is easy and no issue of local minima in DL

6. Applications of Shallow Neural Networks

- Binary classification
- Linear and polynomial regression
- Simple pattern recognition
- Early handwriting recognition systems

1.6 DEEP NEURAL NETWORKS

1.6.1 Introduction

A Deep Neural Network (DNN) is a neural network containing more than one hidden layer between the input and output layers.

1.6.2 Architecture

Deep is a word we have all been reading for a long time. Let us understand why it is called deep neural network. The past rules developed for networks such as perceptron, HNN, etc., worked on the concept of one input layer and one output layer.But, if it's more than 3 layers, including the output and input layers, then it is called a deep neural network. Hence, deep in its rawest form is more than just one hidden layer. In DNNs, every layer of nodes trains on a different set of features based on the previous layer's output. So, the more you go into the net, the more you can advance into it, and the more complex features the nodes can recognize. This is because it recombines and learns from the features of the previous layer. This is what is called future hierarchy. These nets allow complex nonlinear relationships. The major advantage which DNNs hold is the ability of dealing with unstructured and unlabeled data which is actually most of the world's data. Especially in the medical field, most of the data is unstructured and unlabeled.

So DNNs are apt for this large amount of data and its processing. See Fig. 8.3.
A deep neural network consists of:
• Input layer
• Multiple hidden layers
• Output layer

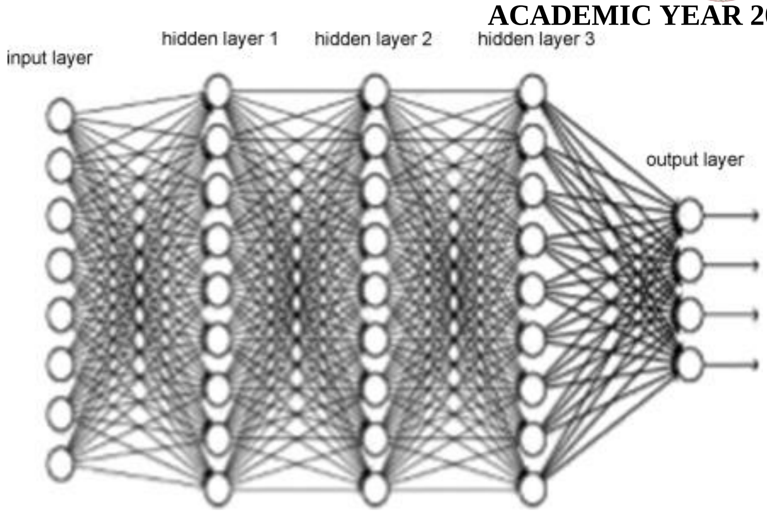


Figure 1.18: Sample DNN

So, basically, deep learning networks can take all the raw, unlabeled, unstructured data, and cluster it, as well as process, to similar forms. A simple example is that deep learning can take a billion images, and cluster them according to their similarities: dogs in one corner, chips in another, and in the ultimate one, all the pictures of a car. This is the basis of the so-called smart photo albums. Have you ever encountered a situation while browsing the internet wherein the website asks you to prove that you're not a bot by choosing a typical type of photos such as street signs, etc.? Yes, that's what the DNN can do on its own. And much more with extremely complicated data, analyzing which the human brain can take as long as a decade.

Each layer performs the operation:

$$a^{(l)} = \sigma(W^{(l)}a^{(l-1)} + b^{(l)})$$

where $W^{(l)}$ is the weight matrix, $b^{(l)}$ is the bias vector, and σ is the activation function.

1.6.3 Role of Depth

Depth allows hierarchical feature learning. Early layers capture low-level features, while deeper layers capture high-level abstractions.

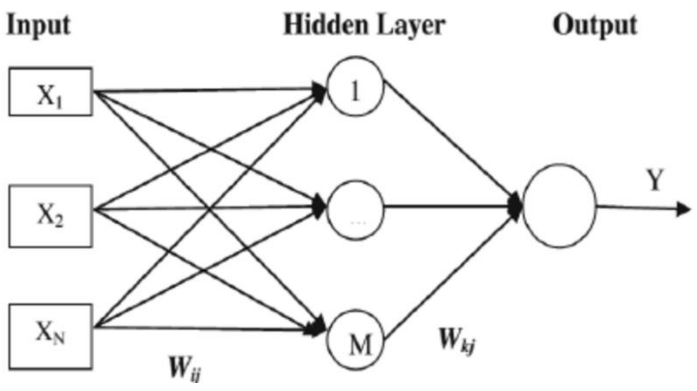


Figure 1.19: Multi-Layer Feed-Forward Neural Network

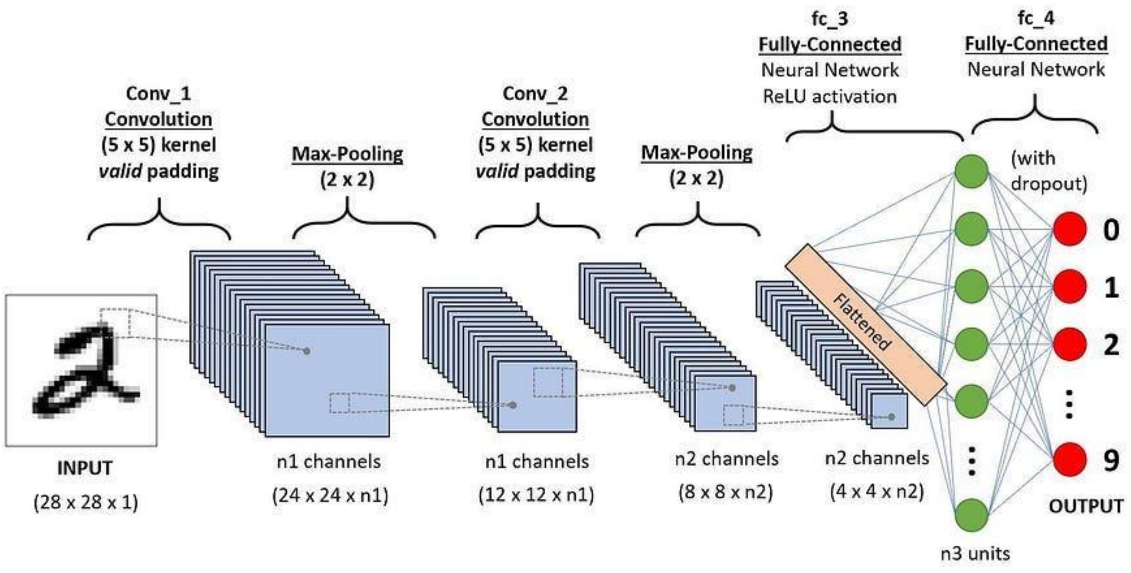


Figure 1.20: Hierarchical feature learning(Pixels → edges → shapes → objects)

1.6.4 Activation Functions

Common activation functions include:
• Sigmoid
• Hyperbolic tangent (tanh)
• Rectified Linear Unit (ReLU)
• Softmax (output layer)

1.6.5 Training Deep Neural Networks

Training involves forward propagation, loss computation, and backpropagation.

Forward propagation:

$z(l) = W(l)a(l-1) + b(l)$

$a(l) = \sigma(z(l))$

Loss functions:

• Mean Squared Error
• Cross Entropy Loss

Backpropagation updates weights using gradient descent:

$w \leftarrow w - \eta\, \partial C/\partial w$

ReLU

- Rectified Linear Unit (ReLU) solve the vanishing gradient problem. ReLU is a non-linear function or piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero.
- It is the most commonly used activation function in neural networks, especially Convolutional Neural Networks (CNNs) and Multilayer perceptron's.
- . Mathematically, it is expressed as

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

  where X : input to neuron

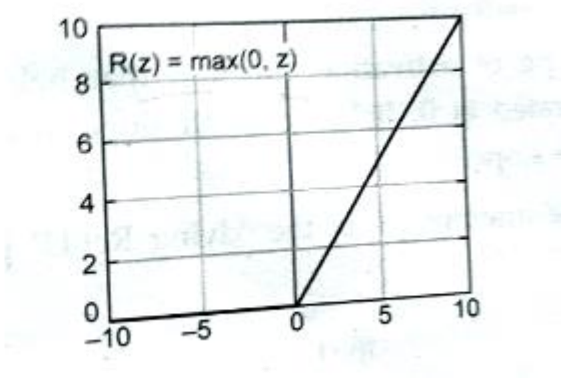


Figure 1.21: ReLU function

- The derivative of an activation function is required when updating the weights during the back-propagation of the error. The slope of ReLU is 1 for positive values and 0 for negative values. It becomes non-differentiable when the input x is zero, but it can be safely assumed to be zero and causes no problem in practice.
- ReLU is used in the hidden layers instead of Sigmoid or tanh. The ReLU function solves the problem of computational complexity of the Logistic Sigmoid and Tanh functions.
- A ReLU activation unit is known to be less likely to create a vanishing gradient problem because its derivative is always 1 for positive values of the argument.

Advantages of ReLU function:

- ReLU is simple to compute and has a predictable gradient for the backpropagation of the error.
- Easy to implement and very fast.
- The calculation speed is very fast. The ReLU function has only a direct relationship.
- It can be used for deep network training.

Disadvantages of ReLU function:

- When the input is negative, ReLU is not fully functional which means when it comes to the wrong number installed, ReLU will die. This problem is also known as the Dead Neurons problem.
- b) ReLU function can only be used within hidden layers of a Neural Network Model.

LReLU and ERELU

1. LReLU

- The Leaky ReLU is one of the most well-known activation function. It is the same as ReLU for positive numbers. But instead of being 0 for all negative values, it has a constant slope (less than 1.).
- Leaky ReLU is a type of activation function that helps to prevent the function from becoming saturated at 0. It has a small slope instead of the standard ReLU which has an infinite slope.
- Leaky ReLUs are one attempt to fix the "dying ReLU" problem. Fig. shows ReLU function.

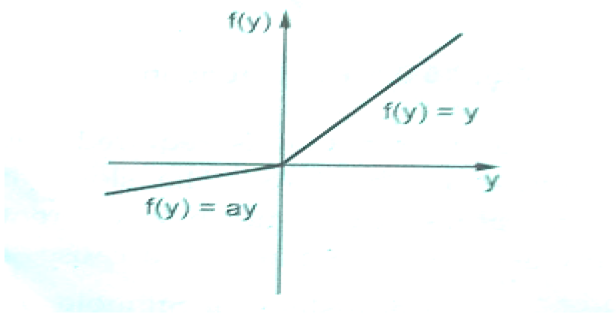


Figure 1.22: LReLU function

- The leak helps to increase the range of the ReLU function. Usually, the value of a is 0.01 or so.
- The n motivation foe using LReLU instead of ReLU is that constant zero gradients can also result in slow learning, as when a saturated neuron uses a sigmoid activation function.

EReLU:

- An Elastic ReLU (EReLU) Considers a slope randomly drawn from a uniform distribution during the training for the positive inputs to control the amount of non-Linearity.
- The EReLU 15 defined as :EReLU() max(RX: 0) in the output range of[0;1]where R is a random number
- At the test time, the EReLU becomes the identity function for positive inputs.

1.6.6 Challenges

Challenges include vanishing gradients, overfitting, slow convergence, and high computational cost.

1.6.7 Advantages

• Automatic feature extraction
• High representational power
• Better performance on complex tasks

1.6.8 Limitations

• Requires large datasets
• Computationally expensive
• Difficult to interpret

1.7 FORWARD AND BACK PROPAGATION

1.7.1 Forward Propagation

Definition

Forward propagation is the process by which input data is passed through the layers of a neural network to compute the output.

Purpose of Forward Propagation

• To compute the network’s prediction
• To evaluate the loss or error
• To supply intermediate values required for backpropagation

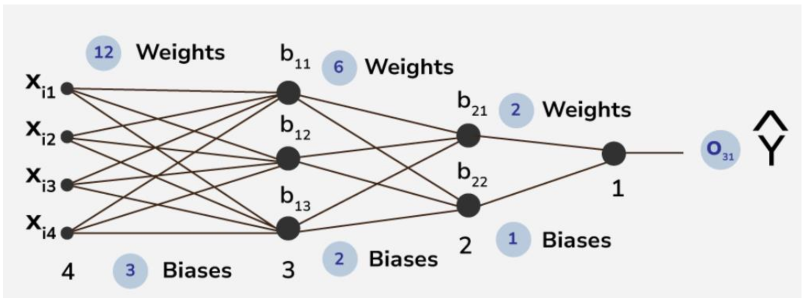


Figure 1.23: Forward Propagation in Neural Networks

Mathematical Representation

For each layer l, forward propagation consists of:

Linear combination:

$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$

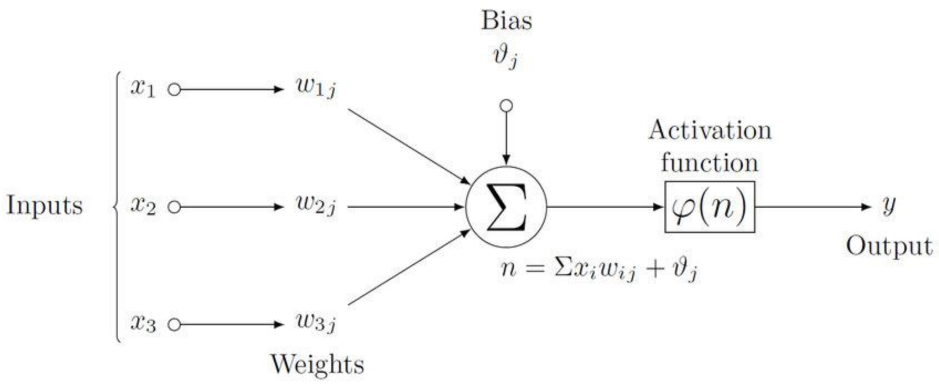


Figure 1.24: Computation Layer

Activation:

$a^{(l)} = \sigma(z^{(l)})$

where $W^{(l)}$ is the weight matrix, $b^{(l)}$ is the bias vector, $a^{(l-1)}$ represents activations from the previous layer, and σ is the activation function.

Forward Propagation in Shallow and Deep Networks

In a shallow neural network, forward propagation passes through one hidden layer.

In a deep neural network, forward propagation passes through multiple hidden layers, with each layer learning higher-level features.

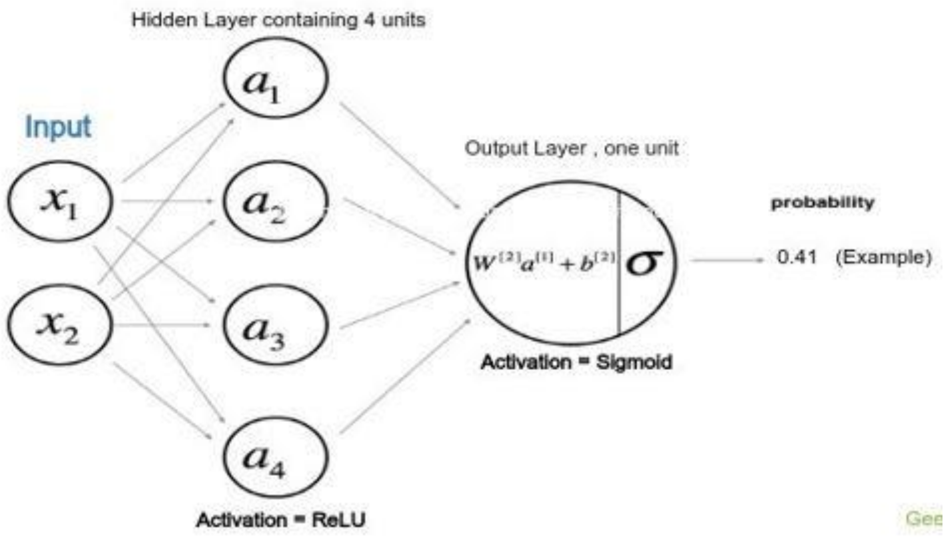


Figure 1.25: Forward Propagation in Deep Networks

Forward Propagation During Training and Testing

During training, forward propagation is followed by loss computation and backpropagation. During testing or inference, only forward propagation is performed using trained weights.

Output of Forward Propagation

The output layer produces:

• A probability value (sigmoid)

• A class probability distribution (softmax)

• A numerical output (linear activation)

The output is compared with the true label to compute the loss.

1.7.2 Backpropagation

Backpropagation is a training method used for a multi-layer neural network. It is also called the generalized delta rule. It is a gradient descent method which minimizes the total squared error of the output computed by the net.

The backpropagation algorithm looks for the minimum value of the error function in weight space using a technique called the delta rule or gradient descent. The weights that minimize the error function is then considered to be a solution to the learning problem.

Backpropagation is a systematic method for training multiple layer ANN. t is a generalization of Window-Hoff error correction rule. 80 % of ANN applications uses back propagation.

Consider a simple neuron:

a. Neuron has a summing junction and activation function.

b. Any non linear function which differentiable everywhere and increases everywhere with sum can be used as activation function.

C. Examples: Logistic function, Arc tangent function, hyperbolic tangent activation function, These activation functions make the multilayer network to have greater representational power than single layer network only when non-linearity is introduced.

Need of hidden layers

- A network with only two layers (input and output) can only represent the input with whatever representation already exists in the input data.
- If the data is discontinuous or non-linearly separable, the innate representation is inconsistent, and the mapping cannot be learned using two layers (Input and Output).
- Therefore, hidden layer(s) are used between input and output layers Weights connects unit (neuron) in one layer only to those in the next higher layer.
- The output of the unit is scaled by the value of the connecting weight, and it is fed forward to provide a portion of the activation for the units in the next higher layer.
- Backpropagation can be applied to an artificial neural network with any number of hidden layers. The training objective is to adjust the weights so that the application of a set of inputs produces the desired outputs.

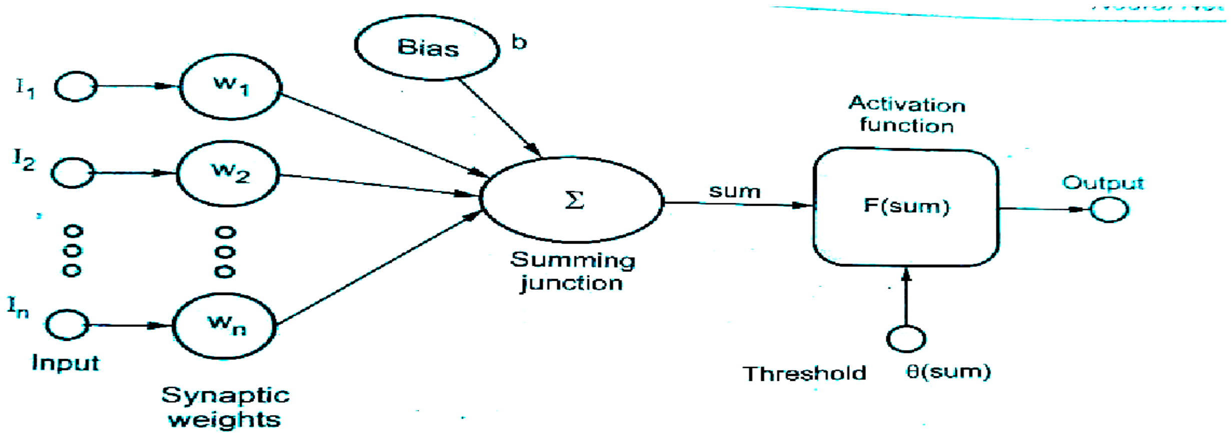


Figure 1.26: Back Propagation network

Training procedure: The network is usually trained with a large number of inputs - output pairs.

1. Generate weights randomly to small random values (both positive and negative) to ensure that the network is not saturated by large values of weights.
2. Choose a training pair from the training set.
3. Apply the input vector to network input.
4. Calculate the network output.
5. Calculate the error, the difference between the network output and the desired output.
6. Adjust the weights of the network in a way that minimizes this error.
7. Repeat steps 2 6 for each pair of input output in the training set until the error for the entire system is acceptably low.

Forward pass and backward pass:

• Backpropagation neural network training involves two passes.

1. In the forward pass, the input signals moves forward from the network input to the output.
2. In the backward pass, the calculated error signals propagate backward through the network, where they are used to adjust the weights.
3. In the forward pass, the calculation of the output is carried out, layer by layer in the forward direction. The output of one layer is the input to the next layer.

• In the reverse pass,

- The weights of the output neuron layer are adjusted first since the target value of each output neuron is available to guide the adjustment of the associated weights, using the delta rule.
- Next, we adjust the weights of the middle layers. As the middle layer neurons have no target values, it makes the problem complex.

Selection of number of hidden units: The number of hidden units depends on the number of input units.

1. Never choose h to be more than twice the number of input units.
2. You can load patterns of I elements into log, p hidden units.
3. Ensure that we must have at least 1/e times as many training examples.
4. Feature extraction requires fewer hidden units than inputs.
5. Learning many examples of disjointed inputs requires more hidden units than inputs.
6. The number of hidden units required for a classification task increases with the number of classes in the task. Large networks require longer training times.

Factor influencing Backpropagation training:

The training time can be reduced by using:

Bias : Networks with biases can represent relationships between outputs more easily than networks it out biases, Adding a bias to eat neuron 1s usually desirable to offset the origin of the activation function. The weight of the bias is trainable similar to weight except that the input is always +1.

1. **Momentum**: The use of momentum enhances the stability of the training process. Momentum is used to keep the training process going in the same general direction analogous to the way that momentum of a moving object behaves. In back propagation with momentum, the weight change is a combination of the current gradient and the previous gradient.

Advantages of backpropagation:

1. It is simple, fast and easy to program.
2. Only numbers of the input are tuned and not any other
3. No need to have prior knowledge about the network.
4. It is flexible.
5. A standard approach and works efficiently.

6. It does not require the user to learn special functions.

Disadvantages of backpropagation:

- ✓ Backpropagation possibly be sensitive to noisy data and irregularity.
- ✓ The performance of this is highly reliant on the input data.
- ✓ Needs excessive time for training.
- ✓ The need for a matrix-based method for backpropagation instead of mini - batch.

1.8 DEEP LEARNING FRAMEWORKS

Deep Learning Frameworks are software libraries and tools that provide a foundation for designing, training, testing, and deploying deep neural networks efficiently. They simplify complex mathematical computations and enable researchers and developers to focus on model architecture and experimentation.

Need for Deep Learning Frameworks

- Efficient handling of large-scale data and complex neural networks
- Automatic differentiation and optimized computation
- Support for GPU and distributed computing
- Faster development and experimentation
- Easy deployment of trained models

Popular Deep Learning Frameworks

1. TensorFlow

TensorFlow is an open-source deep learning framework developed by Google. It supports both low-level and high-level APIs, making it suitable for research as well as production environments. TensorFlow uses dataflow graphs to represent computations and supports distributed training and deployment on CPUs, GPUs, and TPUs.

2. PyTorch

PyTorch, developed by Facebook AI Research, is known for its dynamic computation graph and ease of use. It allows developers to build models intuitively and is widely used in academic research. PyTorch provides strong GPU acceleration and seamless integration with Python.

3. Keras

Keras is a high-level deep learning API that runs on top of TensorFlow. It is designed for fast experimentation and supports modular and user-friendly model building. Keras is ideal for beginners and rapid prototyping.

4. Caffe

Caffe is a deep learning framework developed by Berkeley AI Research. It is optimized for speed and is commonly used in image processing applications. Caffe uses a configuration-based approach for defining neural networks.

Comparison of Frameworks

TensorFlow is preferred for large-scale production systems, PyTorch is widely used for research due to its flexibility, Keras is suitable for beginners and quick model development, and Caffe is efficient for vision-based applications.

Deep learning frameworks play a crucial role in the development of modern AI systems. Choosing the right framework depends on the application requirements, ease of use, performance needs, and deployment considerations.

1.9 DATA AUGMENTATION

Data Augmentation is a technique used in deep learning to artificially increase the size and diversity of the training dataset by applying various transformations to existing data. It helps improve the generalization capability of deep learning models and reduces overfitting, especially when the available dataset is limited.

Techniques of Data Augmentation

Common data augmentation techniques include:

• Image rotation, flipping, scaling, and cropping
• Adding noise to input data
• Color jittering and brightness adjustment
• Random translation and zooming
• Time shifting and noise injection for audio data
• Synonym replacement and word shuffling for text data

Advantages of Data Augmentation

• Increases the effective size of the training dataset
• Improves model generalization
• Reduces overfitting
• Enhances robustness of the model
• Reduces dependency on large labeled datasets

1.10 UNDERFITTING

Underfitting occurs when a machine learning or deep learning model is too simple to capture the underlying patterns present in the training data. As a result, the model performs poorly on both training and testing datasets.

Causes of Underfitting

• Model is too simple
• Insufficient training time
• Inadequate feature representation
• Excessive regularization
• Poor choice of hyper parameters

Symptoms of Underfitting

• High training error
• High validation error
• Model fails to learn data patterns

Methods to Reduce Underfitting

• Increase model complexity
• Train the model for more epochs
• Reduce regularization

• Use better feature engineering
• Choose appropriate hyper parameters

1.11 OVERFITTING

Overfitting occurs when a model learns the training data too well, including noise and irrelevant details. Although the model achieves high accuracy on training data, its performance on unseen test data is poor.

Causes of Overfitting

• Model is excessively complex
• Limited training data
• Too many training epochs
• Lack of regularization
• Noisy data

Symptoms of Overfitting

• Very low training error
• High validation or test error
• Poor generalization to new data

Methods to Reduce Overfitting

• Data augmentation
• Regularization techniques (L1, L2)
• Dropout
• Early stopping
• Reducing model complexity

Comparison: Underfitting vs Overfitting

Underfitting results from overly simple models that fail to capture data patterns, while overfitting arises from overly complex models that memorize training data. A well-balanced model achieves good performance on both training and testing datasets.

1.11.1 Regularization Techniques

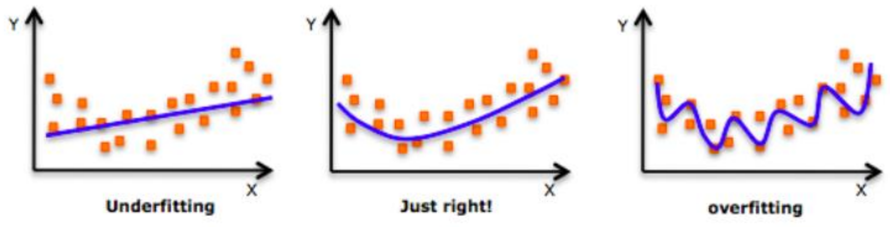


Figure 1.27: Regularization Techniques

- Just have a look at the above figure, and we can immediately predict that once we try to cover every minutest feature of the input data, there can be irregularities in the extracted features, which can introduce noise in the output. This is referred to as "Overfitting".
- This may also happen with the lesser number of features extract as some of the important details might be missed out. This will leave an effect on the accuracy of the outputs produced. This is referred to as "Underfitting".

- This also shows that the complexity for processing the input elements increases with overfitting. Also, neural networks being a complex interconnection of nodes the issue of overfitting may arise frequently.
- To eliminate this, regularization is used, in which we have to make the slightest modification in the design of the neural network, and we can get better outcomes.

Regularization in Machine Learning

- ✓ One of the most important factors. that affect the machine learning model is overfitting.
- ✓ The machine learning model may perform poorly if it tries to capture even the noise present in the dataset applied for training the system, which ultimately results in overfitting. In this context, noise doesn't mean the ambiguous or false data, but those inputs which do not acquire the required features to execute the machine learning model.

Let this be the simple relation for linear regression:

$$Y = bo + bX + b,X,t\,.bp$$

Y = Learned relation

B(beta) = Co-efficient estimators for different variables and/or predictors (X)

- ✓ Now, we shall introduce a loss function that implements the fitting procedure, which is referred to as "Residual Sum of Squares"' or RSS.
- ✓ The co-efficient in the function is chosen in such a way that it can minimize the loss function easily.

$$\text{RSS}=\sum_{i=1}^{n}(Yi-\beta 0-\sum_{j=1}^{p}\beta jXij)2$$

• In case noise is present in the training dataset, then the adjusted co-efficient won't be generalized when the future datasets will be introduced. Hence, at this point, regularization comes into picture and makes this adjusted co-efficient shrink towards zero.

One of the methods to implement this is the ridge regression, also known as L2 regularization. Let’s have a quick overview on this.

How does Regularization help reduce Overfitting?

Let’s consider a neural network which is overfitting on the training data as shown in the image below.

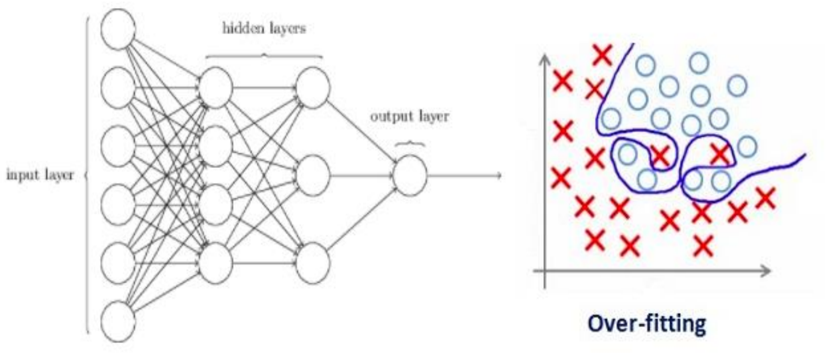


Figure 1.28: Regularization helps to reduce Overfitting

If you have studied the concept of regularization in machine learning, you will have a fair idea that regularization penalizes the coefficients. In deep learning, it actually penalizes the weight matrices of the nodes.

Assume that our regularization coefficient is so high that some of the weight matrices are nearly equal to zero.

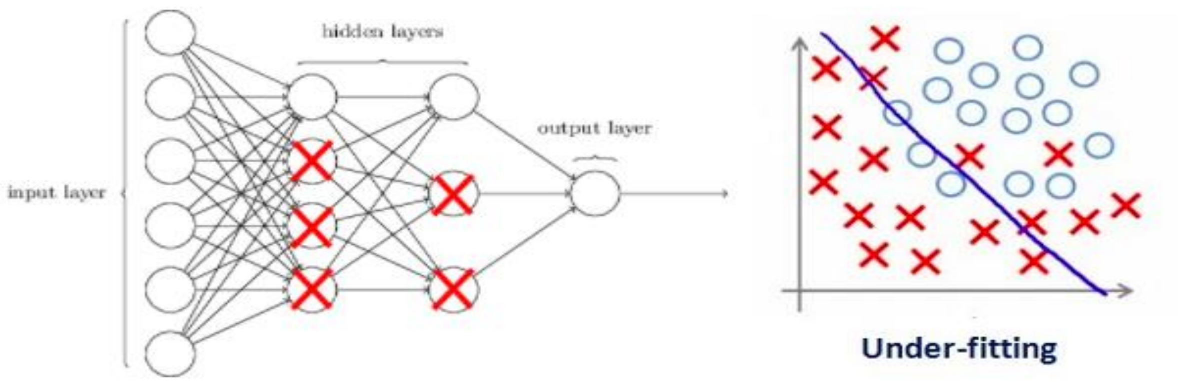


Figure 1.29: Regularization help reduce Under-fitting

This will result in a much simpler linear network and slight underfitting of the training data. Such a large value of the regularization coefficient is not that useful. We need to optimize the value of regularization coefficient in order to obtain a well-fitted model as shown in the image below.

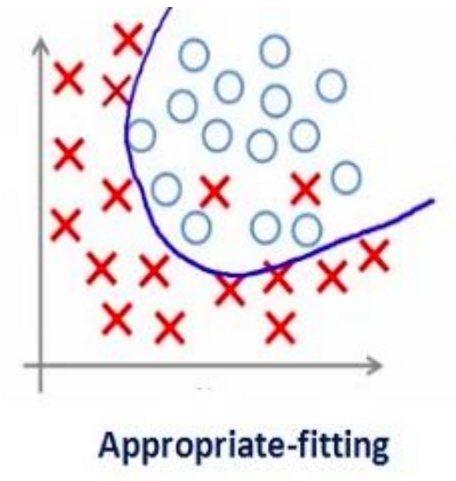


Figure 1.29: Appropriate-fitting

Ridge Regression (L2 Regularization)

- Ridge regression, also known as L2,regularization, is a technique of regularization to avoid the overfitting in training data set, which introduces a small bias in the training model, through which one can get long term predictions for that input.
- In this method, a penalty term is added to the cost function. This amount of bias altered to the cost function in the model is also known as ridge regression penalty hence, the equation for the cost

In L2 Regularization, the term we add to the cost function is the following:

$$\frac{\lambda}{2m}\sum_{l=1}^{L}||w^{[l]}||_2^2$$

In this case, the regularization term is the squared norm of the weights of each network's layer. **This matrix norm is called Frobenius norm and, explicitly, it's computed as follows:**

$$||w^{[l]}||_2^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (w_{ij}^{[l]})^2$$

Please note that the weight matrix relative to layer l has n^{[l]} rows and n^{[l-1]} columns.

Finally, the complete cost function under L2 Regularization becomes:

$$J(w^{[1]}, b^{[1]}, \cdots, w^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} ||w^{[l]}||_2^2$$

Again, λ is the regularization term and for λ=0 the effects of L2 Regularization are null.

L2 Regularization brings towards zero the values of the weights, resulting in a simpler model.

- ✓ It regularizes the co-efficient set for the model and hence the ridge regression term deduces the values of the coefficient, which ultimately helps in deducing the complexity of the machine learning model.
- ✓ From the above equation, we can observe that if the value of tends to zero, the last term on the right - hand side will tend to zero,.thus making the above equation a representation of a simple linear regression model:
- ✓ Hence, lower the value of, the model will tend to linear regression.
- ✓ This model is important to execute the neural networks for machine learning, as there would be risks of failure for generalized linear regression models, if there are dependencies found between its variables. Hence, ridge regression is used here.

Lasso Regression (L1 Regularization)

- ✓ One more technique to reduce the overfitting, and thus the complexity of the model is the lasso regression.
- ✓ Lasso regression stands for Least Absolute and Selection Operator and is also sometimes known as L1 regularization.
- ✓ The equation for the lasso regression is almost same as that of the ridge regression, except for a change that the value. of the penalty term is taken as the absolute weights.
- ✓ The advantage of taking the absolute values is that its slope can shrink to 0, as compared to the ridge regression, where the slope will shrink it near to 0.

The following equation gives the cost function defined in the Lasso regression:

In L1 Regularization we add the following term to the cost function J:

$$\frac{\lambda}{2m} \sum_{l=1}^{L} ||w^{[l]}||_1$$

where the matrix norm is the sum of the absolute value of the weights for each layer 1, …, L of the network:

$$||w^{[l]}||_1 = \sum_i \sum_j |w_{ij}^{[l]}|$$

λ is the regularization term. It's a hyperparameter that must be carefully tuned. λ directly controls the impact of the regularization: as λ increases, the effects on the weights shrinking are more severe.

The complete cost function under L1 Regularization becomes:

$$J(w^{[1]}, b^{[1]}, \cdots, w^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} ||w^{[l]}||_1$$

For λ=0, the effects of L1 Regularization are null. Instead, choosing a value of λ which is too big, will over-simplify the model, probably resulting in an underfitting network.

L1 Regularization can be considered as a sort of neuron selection process because it would bring to zero the weights of some hidden neurons.

- Due to the acceptance of absolute values for the cost function, some of the features of the input dataset can be ignored completely while evaluating the machine learning model, and hence the feature selection and overfitting can be reduced to much extent.
- On the other hand, ridge regression does not ignore any feature in the model and includes it all for model evaluation. The complexity of the model can be reduced using the shrinking of co-efficient in the ridge regression model.

Dropout

Dropout was introduced by "Hinton et al' and this method is now very popular. It Consists of setting to zero the output of each hidden neuron in chosen layer with some probability and is proven to be very effective in reducing overfitting,

Understand dropout, let's say our neural network structure is akin to the one shown below:

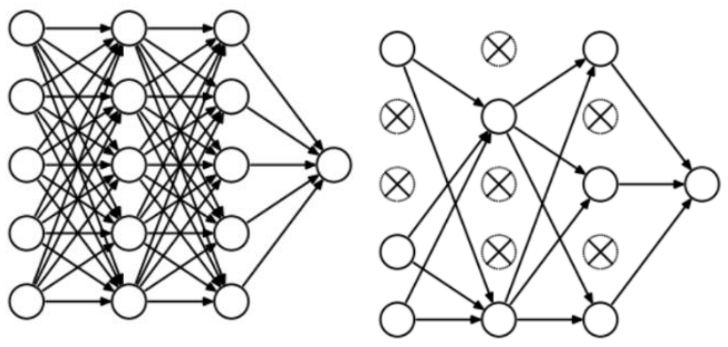


Figure1.30: Shows dropout regulations.

So what does dropout do? At every iteration, it randomly selects some nodes and removes them along with all of their incoming and outgoing connections as shown above.

❖ To achieve dropout regularization, some neurons in the artificial neural network randomly disabled. That prevents them from being too dependent another as they learn the correlations. Thus, the neurons work more independently, and the artificial neural network learns multiple independent correlations in the data based on different configurations of the neurons.

- ❖ It is used to improve the training of neural networks by omitting a hidden unit. It also speeds training.
- ❖ Dropout is driven by randomly dropping a neuron so that it will not contribute to the forward pass and back-propagation.
- ❖ Dropout is an inexpensive but powerful method of regularizing a broad family of models.

Drop Connect:

Drop Connect, known as the generalized version of Dropout, is the method used to regularizing deep neural networks.

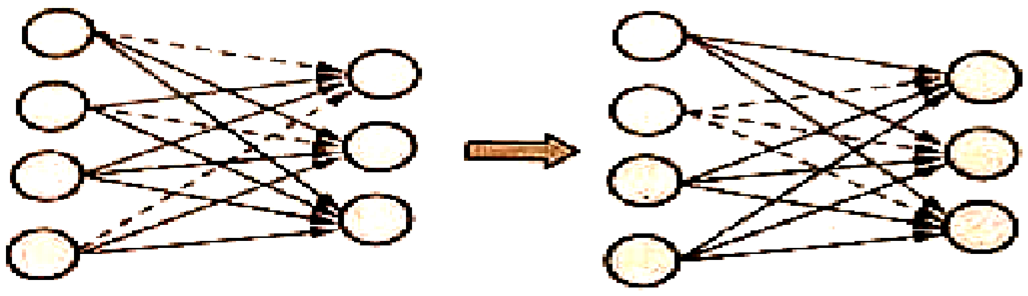


Figure 1.31: Dropconnect.

DropConnect has been proposed to add more noise to the network. The primary difference is that instead of randomly dropping the output of the neurons, we randomly drop the connection between neurons.

In other words, the fully connected layer with DropConnect becomes a sparsely connected layer in which the connections are chosen at random during the training stage.

Table 1.7 Difference between L1 and L2 Regularization

Feature	L1 Regularization (Lasso Regression)	L2 Regularization (Ridge Regression)
1. Definition	Adds **absolute values** of weights ∑	w
2. Effect on Weights	Some weights become **exactly zero**, leading to feature selection.	Reduces weights but **does not make them zero**.
3. Sparsity	Creates sparse models by removing less important features.	Retains all features but reduces their impact.
4. Complexity	Can lead to simpler models with fewer parameters.	Results in more complex models with all features considered.
5. Usage	Used when feature selection is important.	Used when all features contribute to predictions.
6. Mathematical Expression	Loss function: **L = Loss + λ ∑	w
7. Application	Used in high-dimensional data where irrelevant features should be removed.	Used when multicollinearity is an issue to prevent overfitting.