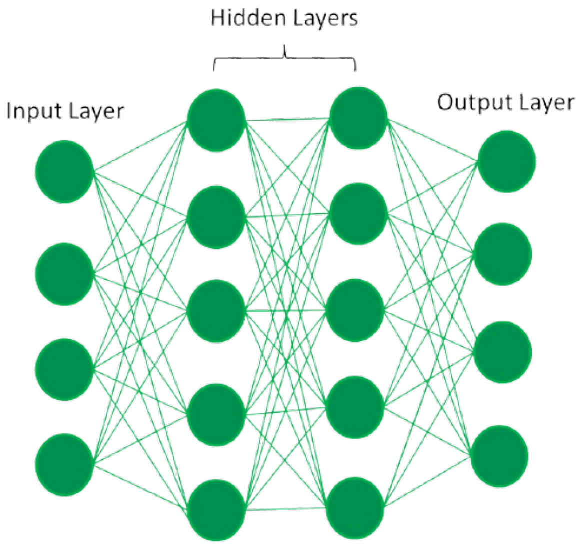
1. Fundamentals of Deep Learning

1.Define Deep Learning with suitable Architecture

- Deep Learning is transforming the way machines understand, learn and interact with complex data.
- **Deep Learning** is a type of machine learning where a computer learns from data using **neural networks with many layers**.
- It automatically learns features (patterns) from raw data like images, text, or sound, without needing manual feature extraction.

Architecture of Deep Learning



1. **Input Layer** – takes raw data (e.g., pixels of an image).
2. **Hidden Layers** – many layers that process the data and learn patterns.
    - Early layers learn simple things (edges, shapes).
    - Deeper layers learn complex things (faces, objects).
3. **Output Layer** – gives the final result (e.g., “Cat” or “Dog”).

Example

- For an image of a dog:
    - First layers detect edges.
    - Middle layers detect ears, eyes, nose.
    - Last layer says → **“Dog”**.

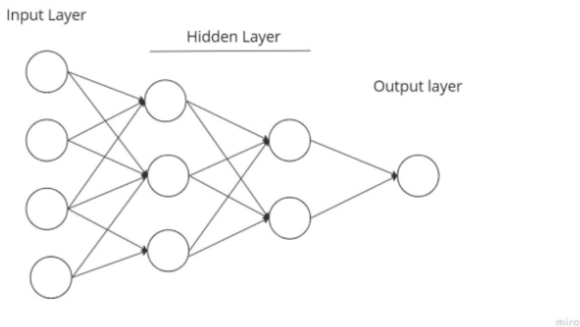
Application

1. **Computer Vision** – used for image classification, object detection, and face recognition.
2. **NLP (Natural Language Processing)** – used in machine translation, chatbots, and sentiment analysis.
3. **Speech & Audio Processing** – used for speech recognition and voice assistants.
4. **Healthcare** – used to detect diseases from medical images like X-rays and MRIs.
5. **Autonomous Vehicles** – used in self-driving cars for lane and obstacle detection.
6. **Finance** – used for fraud detection and stock market prediction.
7. **Recommendation Systems** – used by Netflix, Amazon for product/movie suggestions.

2.Draw and explain the architecture of Multilayered Feedforward Neural network.

- A **Multilayer Feedforward Neural Network** is a type of **artificial neural network** in which data flows in **one direction only** — from the **input layer → hidden layers → output layer**, without cycles or feedback connections.
- It is the most common architecture in **deep learning**.

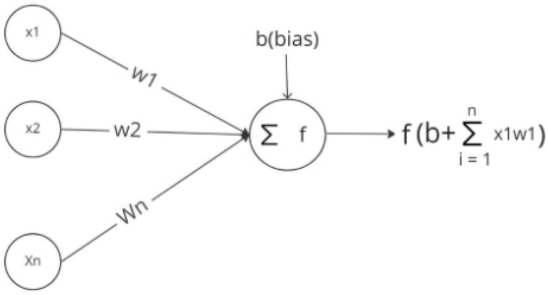
Architecture



1. **Input Layer**
   - Receives raw input data (e.g., image pixels, features).
   - Passes data to the next layer.
2. **Hidden Layers**
   - One or more layers between input and output.
   - Each neuron receives weighted inputs, applies an **activation function** (like ReLU, Sigmoid, Tanh), and passes output forward.
   - Responsible for learning complex patterns.
3. **Output Layer**
   - Produces the final prediction (e.g., class label, probability, numeric value).
   - Often uses **Softmax** (for classification) or **Linear** (for regression).
4. **Connections & Weights**
   - Every neuron in one layer is connected to every neuron in the next layer.
   - Weights are adjusted during **training (backpropagation)**.

Application of Multilayer Feed-Forward Neural Network:

1. Medical field
2. Speech regeneration
3. Data processing and compression
4. Image processing



i) Biases

- A **bias** is an extra parameter added to the weighted sum of inputs in a neuron.
- Formula: $z=\sum(x_i \cdot w_i)+b$
- It helps the activation function shift left or right, improving flexibility.
- Without bias, the output always passes through the origin (0,0).
- **Example:** In a line equation $y=mx+c$, the bias c acts like the intercept.

ii) Activation Functions

- An **activation function** decides whether a neuron should be activated (fired) or not.
- It introduces **non-linearity**, allowing networks to learn complex patterns.

---

1. ReLU (Rectified Linear Unit)

- **Formula:**

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

- **Working:** If $x > 0$, output = x; if $x \leq 0$, output = 0.
- **Advantages:** Simple, efficient, avoids vanishing gradient in positive region.
- **Limitation:** Neurons can "die" (output always 0).
- **Short Definition:** ReLU gives the input value if positive, else 0; widely used for speed and simplicity.

---

2. Leaky ReLU (LReLU)

- **Formula:**

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

  where α is small (e.g., 0.01).
- **Working:** Same as ReLU, but allows a small negative slope for negative inputs.
- **Advantages:** Prevents "dying ReLU" problem.
- **Limitation:** Small negative outputs may slow training.
- **Short Definition:** LReLU is like ReLU but gives a small negative output when input ≤ 0, keeping neurons active.

---

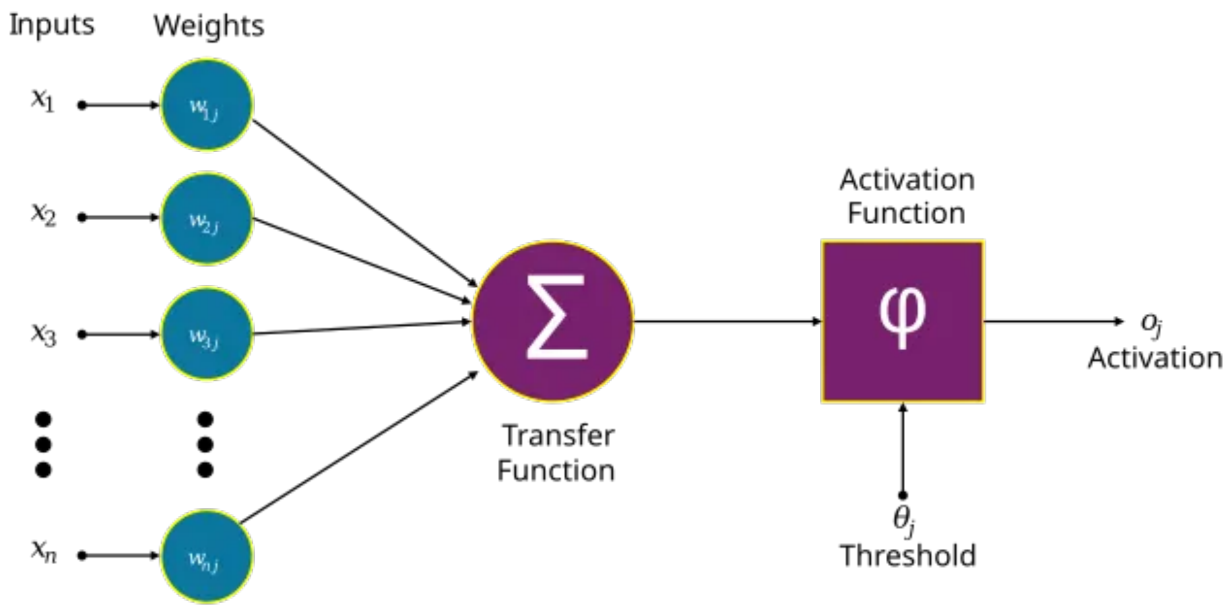
3. ELU (Exponential Linear Unit)

- **Formula:**

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

  where $\alpha > 0$.
- **Working:** Positive side behaves like ReLU; negative side uses a smooth exponential curve.
- **Advantages:** Fixes "dying ReLU" and keeps activations close to zero, improving learning.
- **Limitation:** More computation due to exponential function.
- **Short Definition:** ELU is like ReLU but uses a smooth exponential curve for negative inputs, improving training stability.

3.Explain the working of an Artificial neuron



Working of an Artificial Neuron (with Diagram)

1. **Inputs (x1 ,x2 ,…,xn)**
   - The neuron receives multiple inputs.
   - Each input represents a feature (like pixels in an image, words in text, etc.).
2. **Weights** (wij}
   - Each input is multiplied by a corresponding weight.
   - Weights decide the importance of each input.
   - Example: if w is high, that input has more influence on the output.
3. **Summation (Σ block)**
   - The neuron adds up all the weighted inputs and a bias b
   - Formula:- z=∑(xi ⋅wij)+b
4. **Activation Function (φ)**
   - The summation output z is passed through an activation function (like Sigmoid, ReLU, Tanh).
   - This introduces non-linearity and decides how “activated” the neuron should be.
5. **Output**
   - Finally, the neuron produces an output value.
   - This output is then passed to the next layer of neurons in the network.
   - Formula:- yj =φ(z)=φ(∑(xi ⋅wij)+b)

4.Explain loss function for regression operation

The **loss function** quantifies the disparity between the prediction value and the actual value. In the case of linear regression, the aim is to fit a linear equation to the observed data, the loss function evaluate the difference between the predicted value and true values

(a) Mean Squared Error (MSE)

The average of the squared difference between the real values and the forecasted values

MSE = 1/n ∑(y−y^)2

Takes the **square of errors** between actual value (y) and predicted value (y^).

- Large errors are penalized more.
- Most widely used in regression.

(b) Mean Absolute Error (MAE)

The average of the absolute differences between the real values and the forecasted values

MAE = 1/n ∑|y−y^ |

- Takes the absolute difference between actual and predicted values.
- Less sensitive to large errors (outliers) than MSE.

(c) Huber Loss (Combination of MSE & MAE)

The MSE and the MAE are combined to get the ***Huber Loss***. It is intended to maintain differentiation but be less susceptible to outliers than the MSE:

$$HuberLoss = (1/n) * \Sigma L_\delta(y_{pred} - y_{true})$$

where L_δ is the Huber loss function defined as:

$$L_\delta(x) = \begin{cases} 0.5 * x^2 & \text{if } |x| \leq \delta \\ \delta(|x| - 0.5 * \delta) & otherwise \end{cases}$$

- Uses MSE for small errors and MAE for large errors.
- More robust to outliers.

5. What is Gradient Descent?

- It is a method used to **train neural networks**.
- The main goal is to make the network's predictions **close to the correct output** by reducing error (loss).
- This is done by **adjusting weights and biases** using gradients.

2. How it Works (Step by Step):

1. **Forward Pass:**
   - Input data is passed through the network layer by layer.
   - The network produces an output (prediction).
2. **Loss Calculation:**
   - The error (loss) is calculated by comparing predicted output with the actual output.
   - Example: Mean Squared Error (for regression), Cross-Entropy (for classification).
3. **Backward Pass (Backpropagation):**
   - The gradient of the loss with respect to each weight is calculated.
   - A **gradient** tells us the direction and steepness of the slope of the loss function.
4. **Weight Update using Gradient Descent:**
   - We update weights in the **opposite direction of the gradient** to reduce the loss.

- Formula:

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w}$$

  - w = weight
  - η = learning rate (step size)
  - $\frac{\partial L}{\partial w}$ = gradient of the loss

3. Intuition (Easy to Remember):

- Imagine you are standing on a **hill in fog** and want to reach the bottom.
- You cannot see the bottom, but you can feel the slope.
- You take small steps in the **downward direction** until you reach the lowest point.
- Similarly, the network reduces its error step by step until learning improves.

4. Advantages of Gradient-Based Learning:

- Works well for large and complex networks.
- Automatically adjusts weights to reduce errors.
- Forms the backbone of **deep learning training**.

6. Vanishing Gradient Problem

- During training, neural networks use **back propagation** to update weights.
- In deep networks, when gradients are multiplied layer by layer, they can become **very small (close to zero)**.
- This makes the early layers (closer to input) learn very slowly or stop learning.
  This is called the **Vanishing Gradient Problem**.

2. Why does it happen

- Common with **sigmoid** or **tanh** activation functions, since their derivatives are small (between 0 and 1).
- In very deep networks, repeated multiplication of small values → gradient shrinks → almost zero.

3. Effects

- Slow or no learning in early layers.
- Poor performance in deep neural networks.
- Model stuck in suboptimal state.

Solutions to Avoid Vanishing Gradient:-

(a) Use Better Activation Functions

- Replace **sigmoid/tanh** with **ReLU, Leaky ReLU, or ELU**.
  ReLU has derivative = 1 for positive values, avoiding small gradients.

(b) Weight Initialization Techniques

- Properly initialize weights (e.g., **Xavier initialization, He initialization**) to prevent shrinking gradients.

(c) Batch Normalization

- Normalizes outputs of each layer → keeps gradients stable during training.

(d) Residual Connections (Skip Connections)

- Used in **ResNet** architecture.
- Allows gradients to flow directly to earlier layers, avoiding vanishing.

(e) Gradient Clipping

- Limit (clip) gradients during backpropagation to avoid too small or too large updates.

7.Regularization

Regularization is a technique used in deep learning to **reduce overfitting** and make the model **generalize better** to unseen data.

- Overfitting = when the model performs very well on training data but poorly on test data (because it memorizes instead of learning patterns).
- Regularization works by **controlling large weights** or **adding randomness** during training, so the model learns important patterns instead of noise.

Need for Regularization:-

Prevents Overfitting:

- Controls the complexity of the model so it doesn't memorize noise in training data.

Improves Generalization:

- Helps the model perform better on unseen/test data, not just training data.

Controls Large Weights:

- Penalizes very large weights (which make the model unstable).
- Keeps the weights small and balanced.

Introduces Robustness:

- By dropping neurons or connections (Dropout, DropConnect), the model becomes less dependent on specific parts and more reliable.

Ensures Efficient Training:

- Prevents the network from becoming too complex → reduces risk of vanishing/exploding gradients.

Types of Regularization

(i) L1 Regularization (Lasso)

- Adds the **absolute value of weights** as a penalty to the loss function.
- Formula:-

$$L = L_{original} + \lambda \sum |w|$$

- Effect: Forces some weights to become **zero**, leading to **sparse models** (useful for feature selection).

(ii) L2 Regularization (Ridge)

- Adds the **square of weights** as a penalty to the loss function.
- Formula:

$$L = L_{original} + \lambda \sum w^2$$

- Effect: Shrinks weights towards smaller values (but not zero). Prevents weights from growing too large, keeping the model simple.

(iii) Dropout Regularization

- During training, randomly **drops (turns off) some neurons** in each iteration.
- Example: If dropout rate = 0.5, then half of the neurons are ignored in one pass.
- Effect: Prevents the network from becoming dependent on particular neurons, improves robustness, and reduces overfitting.

(iv) DropConnect Regularization

- Similar to dropout, but instead of dropping entire neurons, it **randomly drops connections (weights)** between neurons.
- Effect: Prevents the model from depending too much on specific connections, further improving generalization.

8.Hyperparameter

A **hyperparameter** is a setting or configuration that is **decided before training a model** and controls how the learning process happens.

- They are **not learned by the model** (unlike weights and biases).
- They must be **manually set** or chosen using techniques like grid search, random search, or Bayesian optimization.

Categories of hyper parameters:-

1. Layer Size

- Refers to the **number of neurons in each layer** of the neural network.
- **Effect:**
    - Large layer size → model can learn complex patterns but may **overfit**.
    - Small layer size → model may **underfit** (fails to learn enough).
- Needs careful tuning for good performance.

2. Learning Rate

- It controls **how big the steps are** when updating weights during training.
- **Effect:**
    - High learning rate → model learns fast but may overshoot (unstable).
    - Low learning rate → stable but slow learning, may get stuck in local minima.
- Usually kept small (e.g., 0.001, 0.01).

3. Momentum

- A parameter that helps the model **move faster in the right direction** by remembering the previous updates.
- It reduces oscillations and speeds up training.
- **Effect:**
    - Prevents getting stuck in local minima.
    - Makes learning smoother and faster.

Aspect	Single Layer Feed Forward NN (SLFN)	Multi Layer Feed Forward NN (MLFN)
Number of Layers	Has only **input layer** and **one output layer**.	Has **input layer, one or more hidden layers, and output layer**.
Hidden Layer	No hidden layer.	Contains one or more hidden layers.
Complexity	Very **simple architecture**.	More **complex architecture**.
Problems it can Solve	Can solve only **linearly separable problems** (e.g., AND, OR).	Can solve **non-linear problems** (e.g., XOR, image classification).
Learning Capability	Limited learning capacity.	Higher learning capacity due to hidden layers.
Computation	Requires less computation (fast).	Requires more computation (slower).
Example	Simple Perceptron.	Multi-Layer Perceptron (MLP), Deep Neural Networks.

🔹 Multilayer Perceptron (MLP)

An **MLP** is one of the simplest forms of **artificial neural networks** used in deep learning.

1. Structure

- **Input Layer** → Takes the input features (e.g., pixels of an image, numerical values).
- **Hidden Layers** → Intermediate layers where neurons apply transformations using weights, biases, and activation functions.
- **Output Layer** → Produces the final prediction (e.g., class label or regression value).

2. Working

Each **neuron** in an MLP performs:

$$z = \sum (w_i \cdot x_i) + b$$

where

- w_i = weight
- x_i = input
- b = bias

Then passes through an **activation function** (like ReLU, sigmoid, or tanh):

$$a = f(z)$$

🔹 Backpropagation

Backpropagation is the **learning algorithm** used to train MLPs (and other neural networks).
It adjusts the weights and biases to minimize the **loss function** (error between prediction and true value).

1. Steps of Backpropagation

1. **Forward Pass**
   - Input → hidden layers → output.
   - Compute predicted output ($\hat{y}$) and loss (L).
2. **Backward Pass (Error Propagation)**
   - Calculate the gradient of the loss with respect to weights using the **chain rule of calculus**.
   - This tells us how much each weight contributed to the error.

   Example for a weight w:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

3. **Weight Update (Gradient Descent)**
   Update weights in the direction that reduces error:

$$w = w - \eta \cdot \frac{\partial L}{\partial w}$$

   where η is the **learning rate**.

🔹 Example

Suppose we want to predict if a student **passes** or **fails** based on hours studied.

1. **Input Layer:** Hours studied.
2. **Hidden Layer:** Processes using weighted sum + activation.
3. **Output Layer:** Probability of pass/fail (using sigmoid).
4. **Backpropagation:** If prediction ≠ actual, adjust weights to reduce future error.