

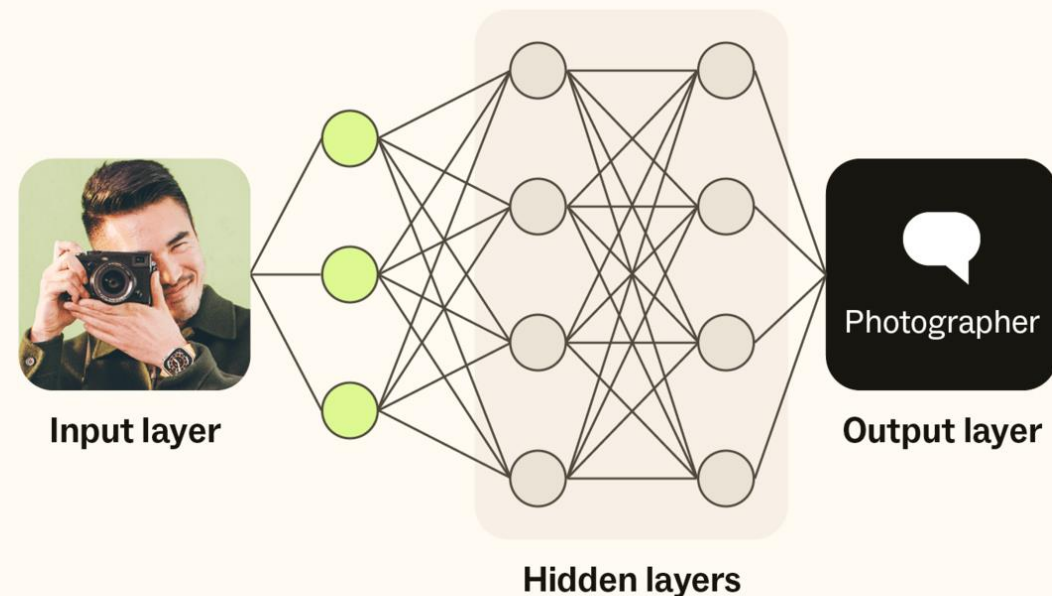
Deep Learning

Deep Learning is a subset of machine learning that uses artificial neural networks with multiple layers to analyze complex patterns in data. It's inspired by the structure and function of the human brain, where neurons connect and process information.

- **Input Layer:** This is where the raw data is fed into the network. For example, in image recognition, it could be a pixelated representation of an image.
- **Hidden Layers:** These layers process the input data and extract features. Each layer learns to identify different patterns.
- **Output Layer:** This layer produces the final result, such as a classification (e.g., "cat" or "dog").

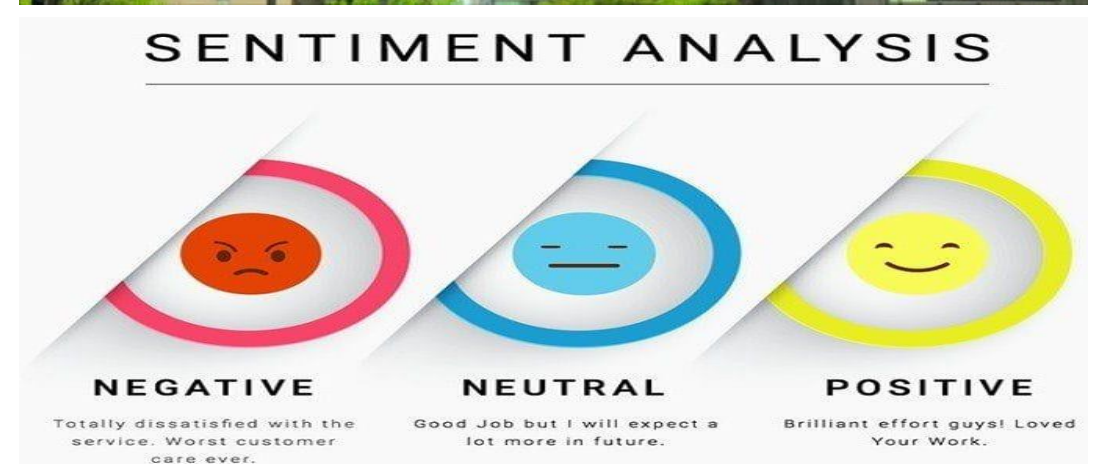
The deep learning process

The algorithm receives data, runs it through the input and hidden layers, and generates an output.



Applications

- **Image Recognition (classification):**
- Deep learning models, like CNNs, classify images into predefined categories.
- **Example:** Identifying whether an image contains a cat, dog, or car.
- **Object Detection:**
- Detects and identifies multiple objects within an image or video.
- **Example:** Self-driving cars identifying pedestrians, vehicles, and traffic lights.
- **Natural Language Processing (NLP):**
- Understanding and generating human language.
- Sentiment analysis, machine translation, and chatbots.

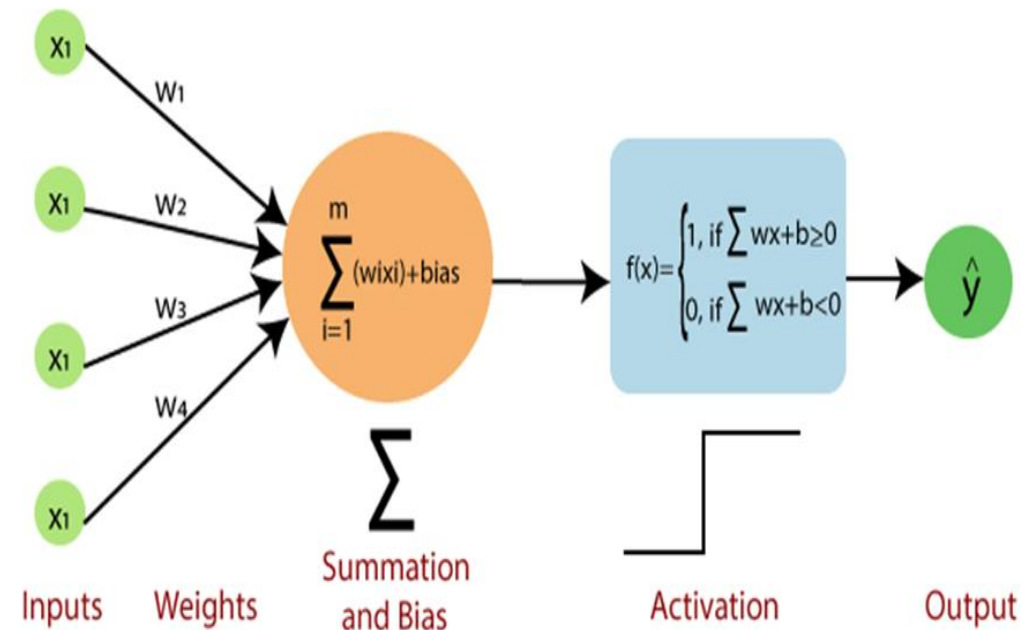


Deep Learning Algorithms

- Artificial Neural Network (ANN).
- Multilayer Perceptron (MLPs)
- Convolutional Neural Networks (CNNs).
- Recurrent Neural Networks (RNNs).
- Long Short Term Memory Networks (LSTMs).

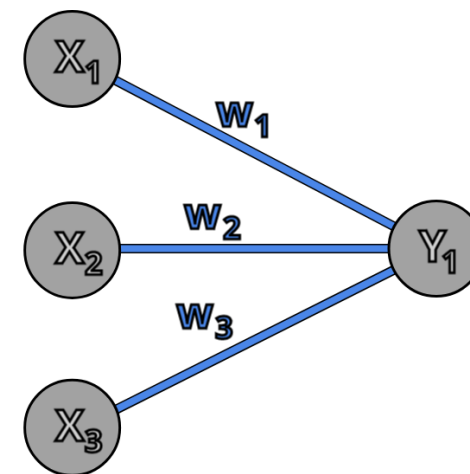
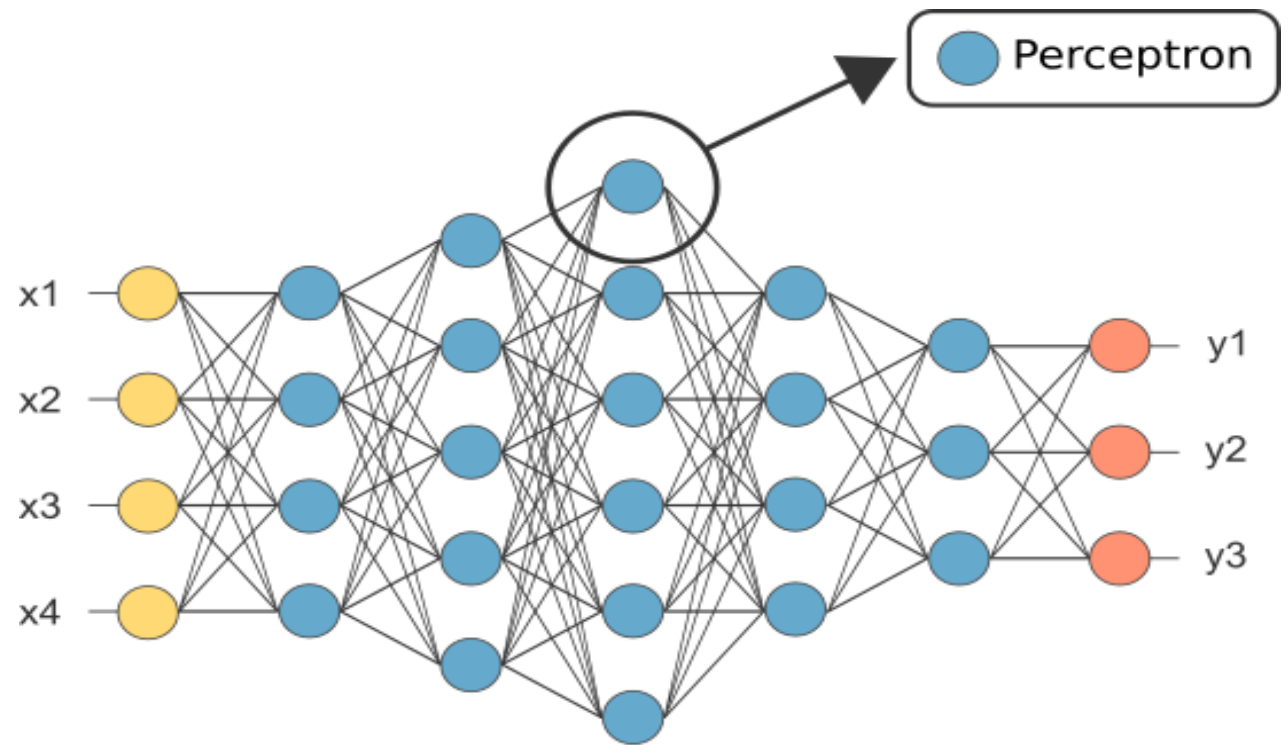
Perceptron

- A **perceptron** is a simple computational unit that forms the foundation of artificial neural networks. It is inspired by the biological neuron and is capable of performing binary classification tasks.
- **Input:** The perceptron receives multiple inputs, each with a corresponding weight.
- **Weighted Sum:** The inputs are multiplied by their respective weights and then summed.
- **Activation Function:** The sum is passed through an activation function, which introduces non-linearity. The most common activation function for perceptron is the step function.
- **Output:** The output of the activation function is the final output of the perceptron, typically 0 or 1.



Feed Forward Neural Network (FFNN)

- A **Feedforward Neural Network (FNN)** is one of the most basic types of artificial neural networks. In FNNs, information moves only in one direction — forward — from the input nodes, through the hidden layers, and to the output nodes.
- The **input layer** receives raw data (e.g., features from a dataset). Each neuron (node) in the input layer represents one feature of the input data. For example, in image classification, each pixel value could be an input node.
- Hidden layers are layers between the input and output layers. The hidden layers perform nonlinear transformations of the inputs to extract features and patterns. A feedforward network can have one or more hidden layers (a network with multiple hidden layers is called a **deep neural network**).
- The **output layer** gives the final result or prediction. For example, in classification, the output could be a probability distribution over different classes (e.g., cat, dog, car), and in regression, it could be a continuous value.



$$Y_1 = \text{Activation}(w_1 \times X_1 + w_2 \times X_2 + w_3 \times X_3)$$

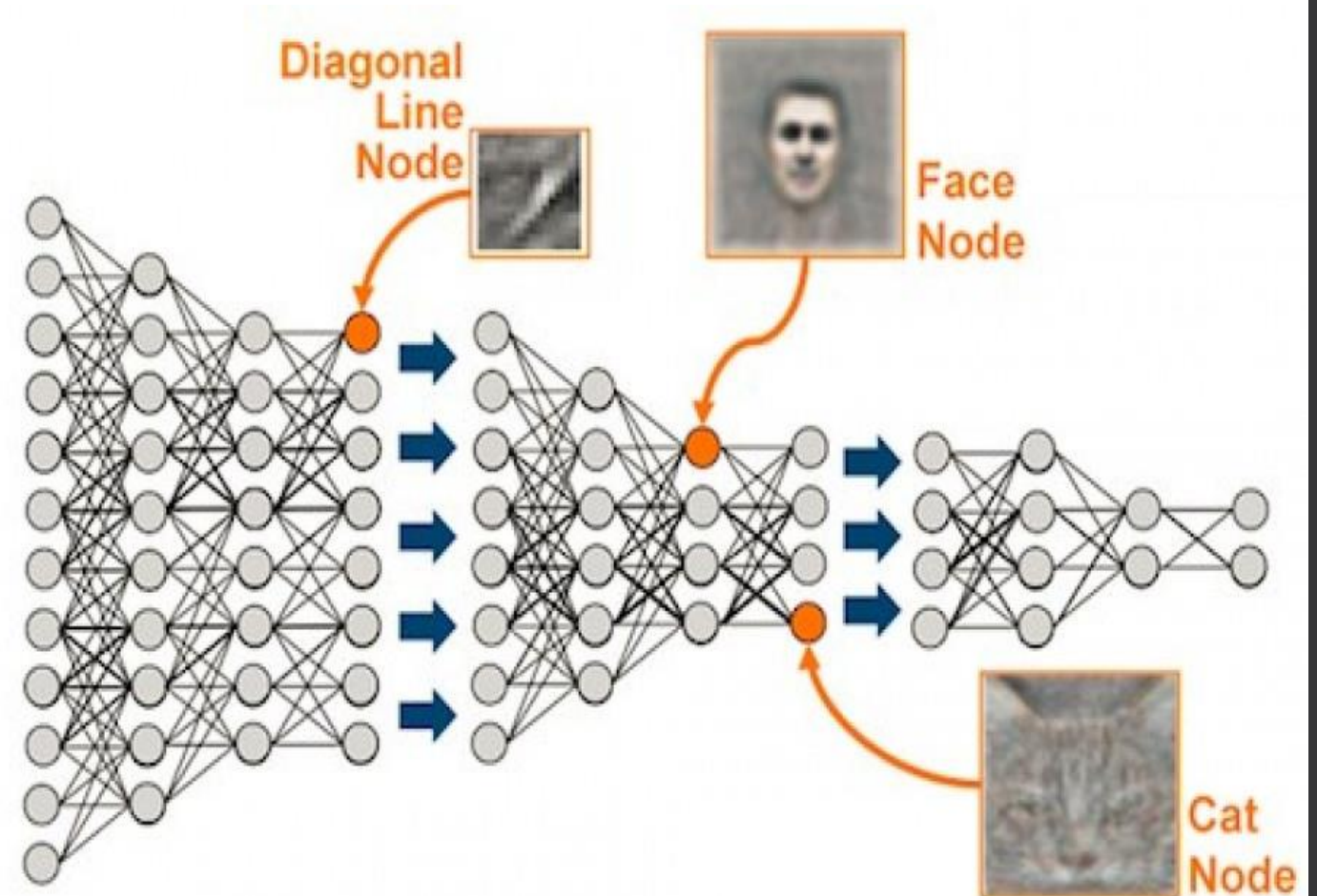
Example: Image Recognition

Imagine teaching a computer to recognize cats.

Input: A pixelated image of a cat.

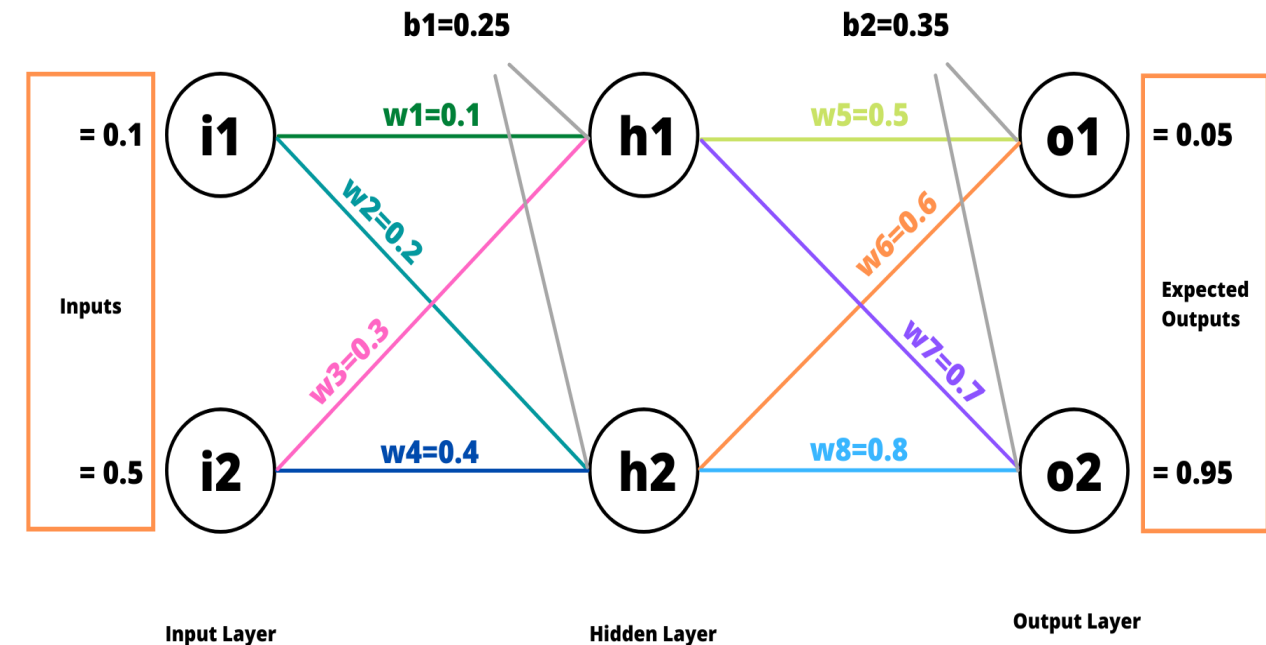
Hidden Layers: The first layer might identify basic shapes like lines and curves. The second layer could recognize features like ears, whiskers, and paws.

Output: The final layer would classify the image as a "cat."



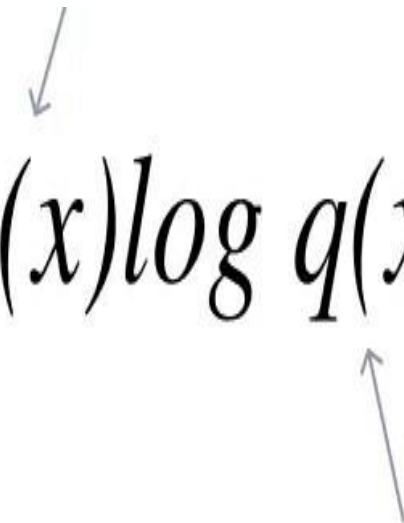
Forward Pass

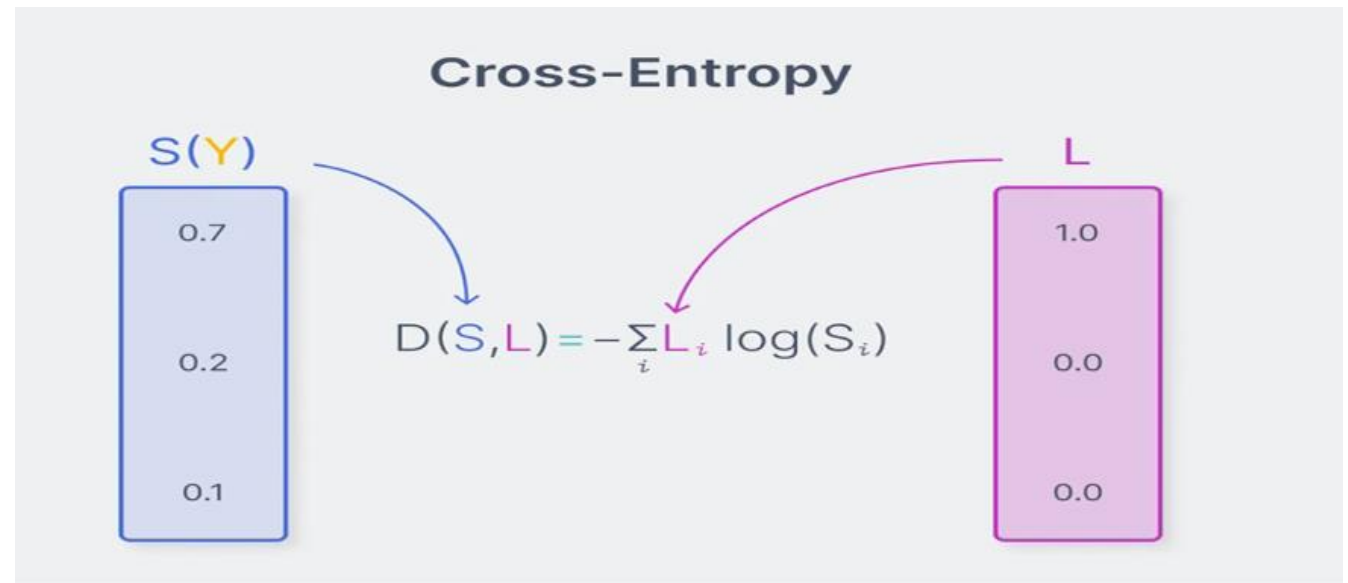
- The **forward pass** is the process of sending input data through the network to generate a prediction.
- **Input Layer**
- Suppose we have two input
 - $X = [x1, x2] = [0.5 \ 0.1]$
- **Weighted Sum at the Hidden Layer**
- The input is passed to the **first hidden layer**. Each hidden neuron computes a weighted sum of the inputs plus a bias.
 - $z1 = (w1.x1 + w2x2+ b1)$
- **w1 to w4** are the weight matrix between the input layer and the hidden layer.
- **b1** and **b2** are the biases terms for the hidden layer and output layer respectively, in the hidden layer we will **ReLU** activation function.
- Similarly **w5 to w8** are the weight matrix of output layer.
- In the output layer we will use **Sigmoid** activation function.



Cross Entropy

- **Cross-entropy** is a loss function that is used to evaluate the **performance** of a machine learning model. It is a measure of the difference between the **predicted** values and the **actual** values.
- **Cross-entropy** = $-\sum y * \log(\hat{y})$
- Practical application are Image recognition, natural language processing.
- $P(x)$ is the true probability distribution of the events, $q(x)$ is the predicted probability distribution (usually outputted by a model).
- Binary cross entropy and Categorical cross entropy are the two types for classification.
- Similarly Pleatue is

$$H(p, q) = - \sum_{x \in \text{classes}} p(x) \log q(x)$$




Binary/ Categorical Cross Entropy

- **Binary cross entropy** is a specific form of cross entropy used when there are only two classes (binary classification).
- **Categorical Cross Entropy** is a loss function commonly used in **multi-class classification** tasks, where a model needs to predict which class an input belongs to, out of multiple possible classes.
- The goal of using this loss function is to compare the **true label** of the data with the **predicted probability distribution** outputted by the model and penalize incorrect predictions. This helps the model learn by minimizing the error.

$$L_{BCE} = -\frac{1}{n} \sum_{i=1}^n (Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log (1 - \hat{Y}_i))$$

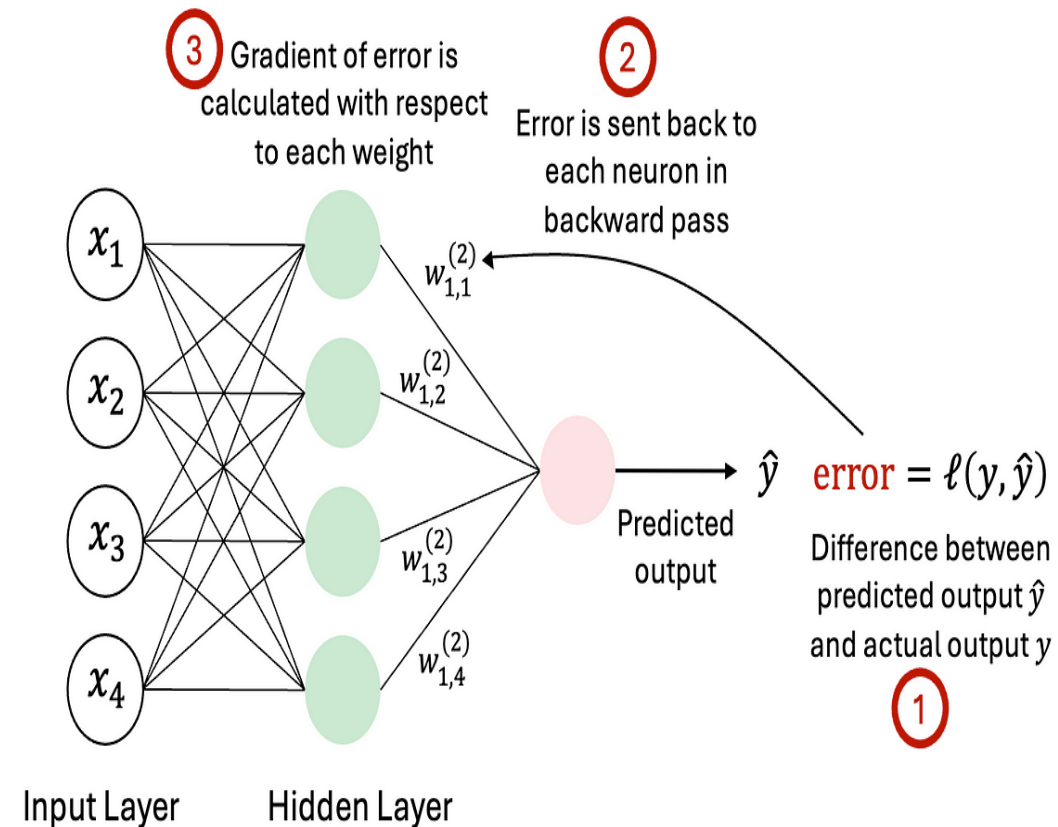
$$CE = - \sum_{i=1}^{i=N} y_{true_i} \cdot \log(y_{pred_i})$$

$$CE = - \sum_{i=1}^{i=N} y_i \cdot \log(\hat{y}_i)$$

$$\implies CE = -[y_1 \cdot \log(\hat{y}_1) + y_2 \cdot \log(\hat{y}_2) + y_3 \cdot \log(\hat{y}_3)]$$

Backward Pass

- The **backward pass** in a neural network is a critical phase of training, where we calculate the **gradients of the loss** with respect to the model's parameters (**weights and biases**) using backpropagation.
- This process allows the network to learn by **updating its parameters** to *minimize* the loss function.
- The backward pass heavily relies on the **chain rule** from **calculus**. The chain rule allows us to compute the **derivative** of composite functions.
- The backward pass calculates how the loss changes with respect to each parameter (weights and biases) in the network.
- The backward pass is thus a foundational process in training neural networks, enabling them to learn from data effectively.



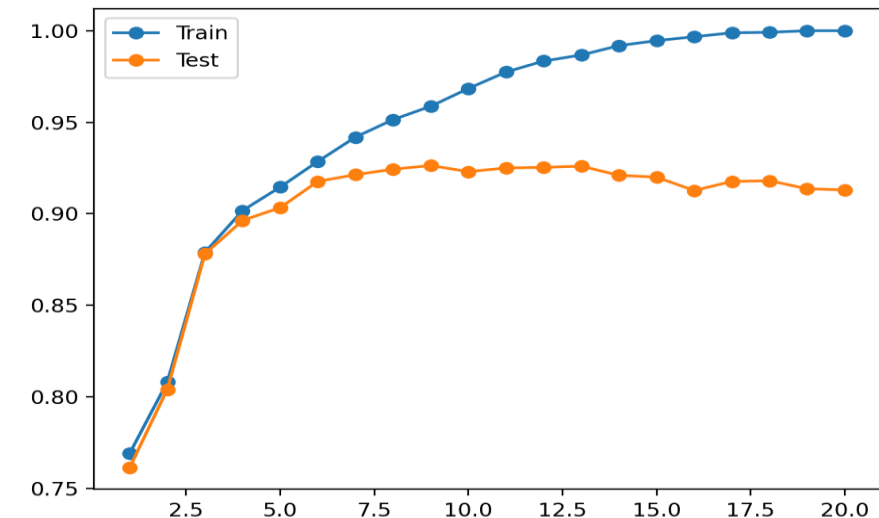
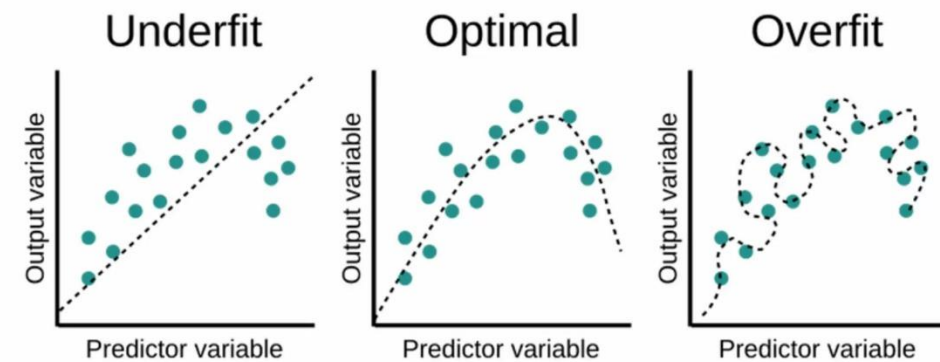
Overfit and Underfit

- **Overfitting** occurs when a model **learns** the training data **too well**, including its noise and outliers. As a result, the model performs exceptionally on the training data but **poorly** on unseen (validation or test) data. It fails to generalize well to new data.

1. **High accuracy** on training data.
2. **Poor accuracy** on validation/test data.
3. The model captures noise in the training data rather than the underlying pattern.

- **Underfitting** occurs when a model is too simple to capture the underlying pattern in the data. This results in **high errors** on both the training and validation data. The model **fails to learn the relationships within the data**.

1. **High error** on both training and validation/test data.
2. The model does not capture the underlying **trend** of the data.



Deep Learning:-

Forward Pass:-

$$[Node\ h_1] \quad z_{h1} = [w_1x_1 + w_2x_2 + w_3x_3 + \dots + b_1]$$

$$= i_1 \cdot w_1 + i_2 \cdot w_2 + b_1 \quad \begin{matrix} i \rightarrow \text{input} \\ w \rightarrow \text{weight} \end{matrix}$$

$$= (0.1 \times 0.1) + (0.5 \times 0.3) + 0.25$$

$$[z_{h1} = 0.41] \rightarrow \text{First node weighted sum.}$$

Apply activation-

$$a_{h1} = \max(0, z_{h1}) \rightarrow \text{ReLU activation}$$

$$= \max(0, 0.41)$$

$$[a_{h1} = 0.41] \text{ Activation of first node.}$$

[Node h_2]:

$$z_{h2} = (i_1 \cdot w_2) + (i_2 \cdot w_4) + b_{h2}$$

$$z_{h2} = (0.1 \times 0.2) + (0.5 \times 0.4) + 0.25$$

$$[z_{h2} = 0.47] \rightarrow \text{will be same after applying ReLU.}$$

\Rightarrow ("Output layer Nodes

Node 01

$$z_{o1} = (a_{h1} \cdot w_5) + (a_{h2} \cdot w_7) + b_2$$

$$= (0.41 \times 0.5) + (0.47 \times 0.7) + 0.35$$

$$[z_{o1} = 0.884]$$

Applying Sigmoid activation:-

$$a_{o1} = \frac{1}{1 + e^{-z_{o1}}}$$

$$= \frac{1}{1 + e^{-0.884}}$$

$$[a_{o1} = 0.707]$$

Node O_2 :-

$$z_{o2} = (a_{h1} \cdot w_6) + (a_{h2} \cdot w_8) + b_2$$

$$z_{o2} = (0.41 \times 0.6) + (0.47 \times 0.8) + 0.35$$

$$[z_{o2} = 0.972]$$

$$a_{o2} = \frac{1}{1 + e^{-z_{o2}}} = \frac{1}{1 + e^{-0.972}} = 0.725$$

After Forward Pass we will

Calculate [Loss function]

\rightarrow Difference between Predicted and True values.

Backward Pass:-

\rightarrow propagate error backward

\rightarrow Calculate gradient of loss

\rightarrow Update weights to minimize the loss.

$$\left[\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a_{out}} \cdot \frac{\partial a_{out}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial w} \right]$$

\rightarrow Chain-Rule

