# Deep Learning

# BCSE-332L

## Module 1:

## Introduction to Neural and Deep Neural Networks

**Dr . Saurabh Agrawal**

**Faculty Id: 20165**

**School of Computer Science and Engineering**

**VIT, Vellore-632014**

**Tamil Nadu, India**

# Outline

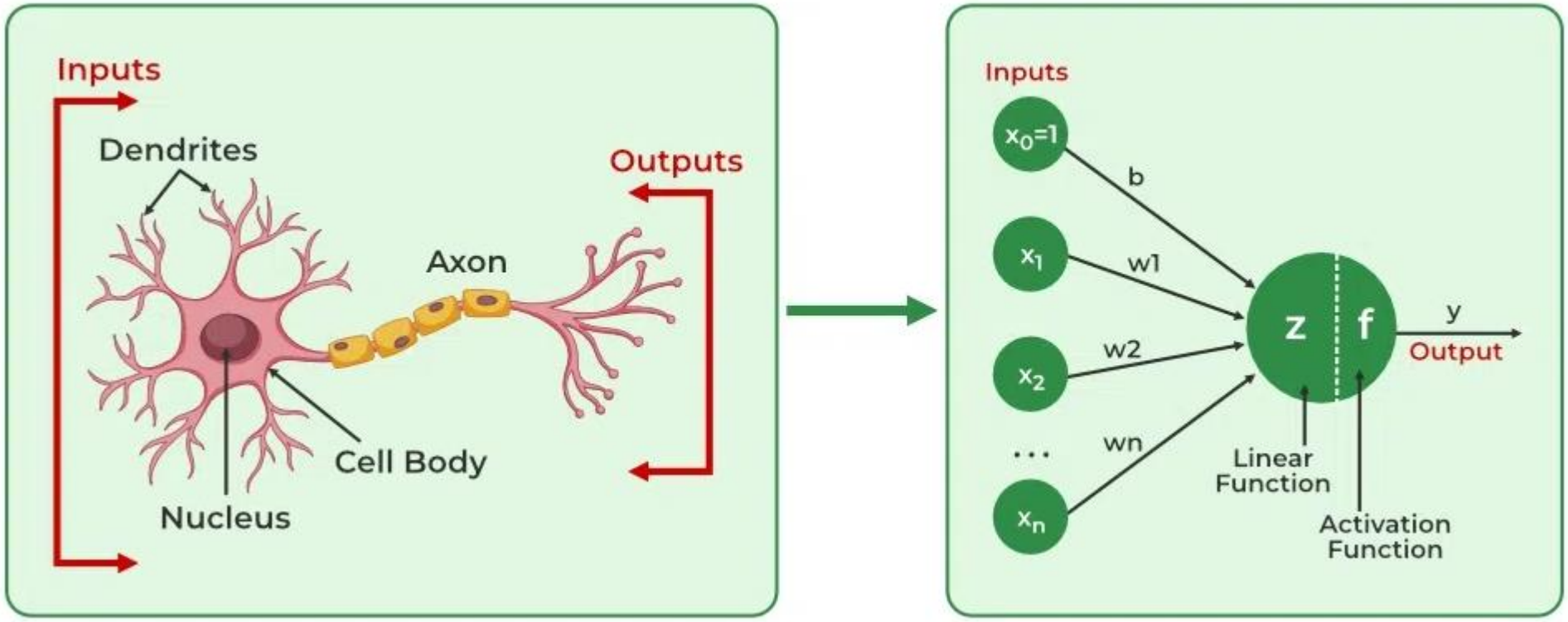
- ❑Neural Networks Basics
- ❑Functions in Neural Networks:
- ❑Activation Functions
- ❑Loss Functions
- ❑Function Approximation
- ❑Classification and Clustering Problems
- ❑Deep Networks Basics
- ❑Shallow Neural Networks
- ❑Gradient Descent
- ❑Back Propagation
- ❑Deep Neural Networks
- ❑Forward and Back Propagation
- ❑Parameters
- ❑Hyperparameters

# Neural Networks Basics

- Neural Networks are computational models that mimic the complex functions of the human brain.
- The neural networks consist of interconnected nodes or neurons that process and learn from data, enabling tasks such as pattern recognition and decision making in machine learning.
- Neural networks extract identifying features from data, lacking pre-programmed understanding.
- Network components include neurons, connections, weights, biases, propagation functions, and a learning rule.
- Neurons receive inputs, governed by thresholds and activation functions.
- Connections involve weights and biases regulating information transfer.
- Learning, adjusting weights and biases, occurs in three stages: **input computation**, **output generation**, and **iterative refinement** enhancing the network's proficiency in diverse tasks.

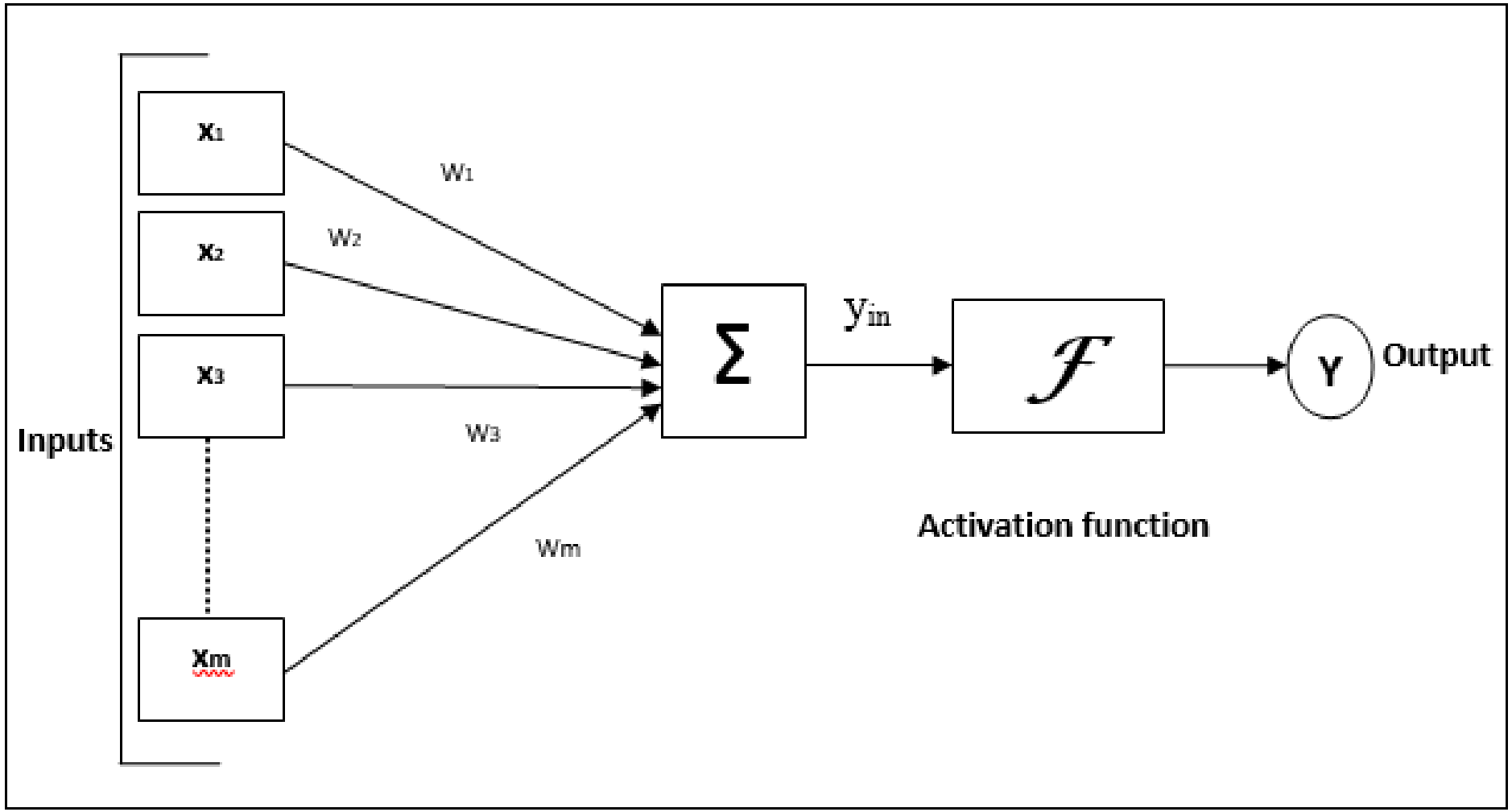
# Neural Networks Basics

1. The neural network is simulated by a new environment.
2. Then the free parameters of the neural network are changed as a result of this simulation.
3. The neural network then responds in a new way to the environment because of the changes in its free parameters.



# Neural Networks Basics

- The following diagram represents the general model of ANN followed by its processing.



# Neural Networks Basics

- For the above general model of artificial neural network, the net input can be calculated as follows:

$$y_{in} = x_1.w_1 + x_2.w_2 + x_3.w_3 \ldots x_m.w_m$$

$$\text{i.e., Net input } y_{in} = \sum_i^m x_i.w_i$$

The output can be calculated by applying the activation function over the net input.

$$Y = F(y_{in})$$

Output = function $netinputcalculated$

## ❑Importance of Neural Networks

❑The ability of neural networks to identify patterns, solve intricate puzzles, and adjust to changing surroundings is essential.

❑Their capacity to learn from data has far-reaching effects, ranging from revolutionizing technology like natural language processing and self-driving automobiles to automating decision-making processes and increasing efficiency in numerous industries.

❑The development of artificial intelligence is largely dependent on neural networks, which also drive innovation and influence the direction of technology.

# Neural Networks Basics

## ❑How does Neural Networks work?

- ❑Consider a neural network for email classification.
- ❑The input layer takes features like email content, sender information, and subject.
- ❑These inputs, multiplied by adjusted weights, pass through hidden layers.
- ❑The network, through training, learns to recognize patterns indicating whether an email is spam or not.
- ❑The output layer, with a binary activation function, predicts whether the email is spam (1) or not (0).
- ❑As the network iteratively refines its weights through backpropagation, it becomes adept at distinguishing between spam and legitimate emails, showcasing the practicality of neural networks in real-world applications like email filtering.

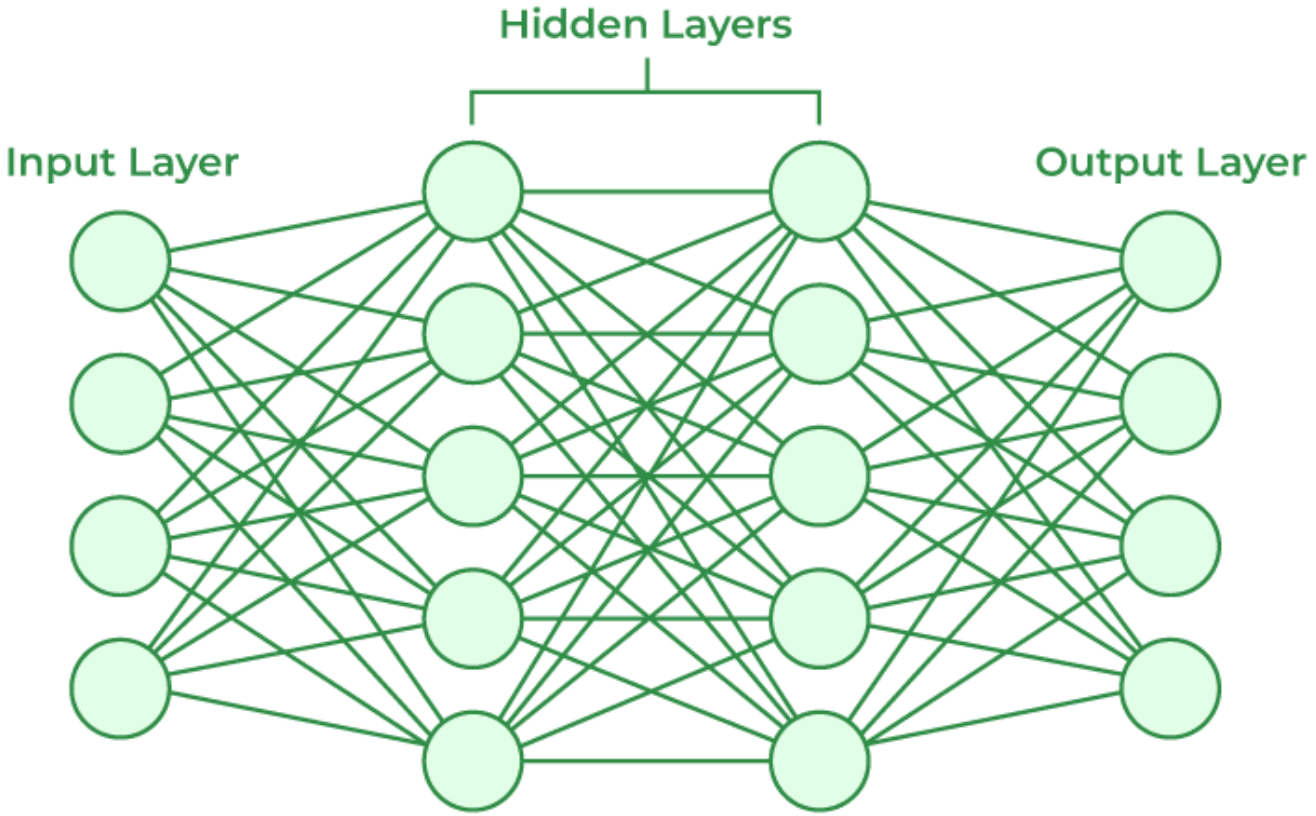
# Neural Networks Basics

## ❑How does Neural Networks work?

❑Neural networks are complex systems that mimic some features of the functioning of the human brain.

❑It is composed of an input layer, one or more hidden layers, and an output layer made up of layers of artificial neurons that are coupled.

❑The two stages of the basic process are called backpropagation and forward propagation.



# Neural Networks Basics

## ❑How does Neural Networks work?: Forward Propagation

1. **Input Layer:** Each feature in the input layer is represented by a node on the network, which receives input data.
2. **Weights and Connections:** The weight of each neuronal connection indicates how strong the connection is. Throughout training, these weights are changed.
3. **Hidden Layers:** Each hidden layer neuron processes inputs by multiplying them by weights, adding them up, and then passing them through an activation function. By doing this, non-linearity is introduced, enabling the network to recognize intricate patterns.
4. **Output:** The final result is produced by repeating the process until the output layer is reached.

# Neural Networks Basics

## ❑How does Neural Networks work?: Backpropagation

1. **Loss Calculation:** The network's output is evaluated against the real goal values, and a loss function is used to compute the difference. For a regression problem, the Mean Squared Error (MSE) is commonly used as the cost function.

   **Loss Function:** $MSE = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2$

2. **Gradient Descent:** Gradient descent is then used by the network to reduce the loss. To lower the inaccuracy, weights are changed based on the derivative of the loss with respect to each weight.
3. **Adjusting weights:** The weights are adjusted at each connection by applying this iterative process, or backpropagation, backward across the network.
4. **Training:** During training with different data samples, the entire process of forward propagation, loss calculation, and backpropagation is done iteratively, enabling the network to adapt and learn patterns from the data.
5. **Activation Functions:** Model non-linearity is introduced by activation functions like the rectified linear unit (ReLU) or sigmoid. Their decision on whether to "fire" a neuron is based on the whole weighted input.

# Neural Networks Basics

## ❑Types of Neural Networks:

1. **Feedforward Networks:** is a simple artificial neural network architecture in which data moves from input to output in a single direction. It has input, hidden, and output layers; feedback loops are absent. Its straightforward architecture makes it appropriate for a number of applications, such as regression and pattern recognition.
2. **Multilayer Perceptron (MLP):** is a type of feedforward neural network with three or more layers, including an input layer, one or more hidden layers, and an output layer. It uses nonlinear activation functions.
3. **Convolutional Neural Network (CNN):** is a specialized artificial neural network designed for image processing. It employs convolutional layers to automatically learn hierarchical features from input images, enabling effective image recognition and classification. CNNs have revolutionized computer vision and are pivotal in tasks like object detection and image analysis.

# Neural Networks Basics

## ❑Types of Neural Networks:

4. **Recurrent Neural Network (RNN):** An artificial neural network type intended for sequential data processing is called a Recurrent Neural Network (RNN). It is appropriate for applications where contextual dependencies are critical, such as time series prediction and natural language processing, since it makes use of feedback loops, which enable information to survive within the network.
5. **Long Short-Term Memory (LSTM):** LSTM is a type of RNN that is designed to overcome the vanishing gradient problem in training RNNs. It uses memory cells and gates to selectively read, write, and erase information.

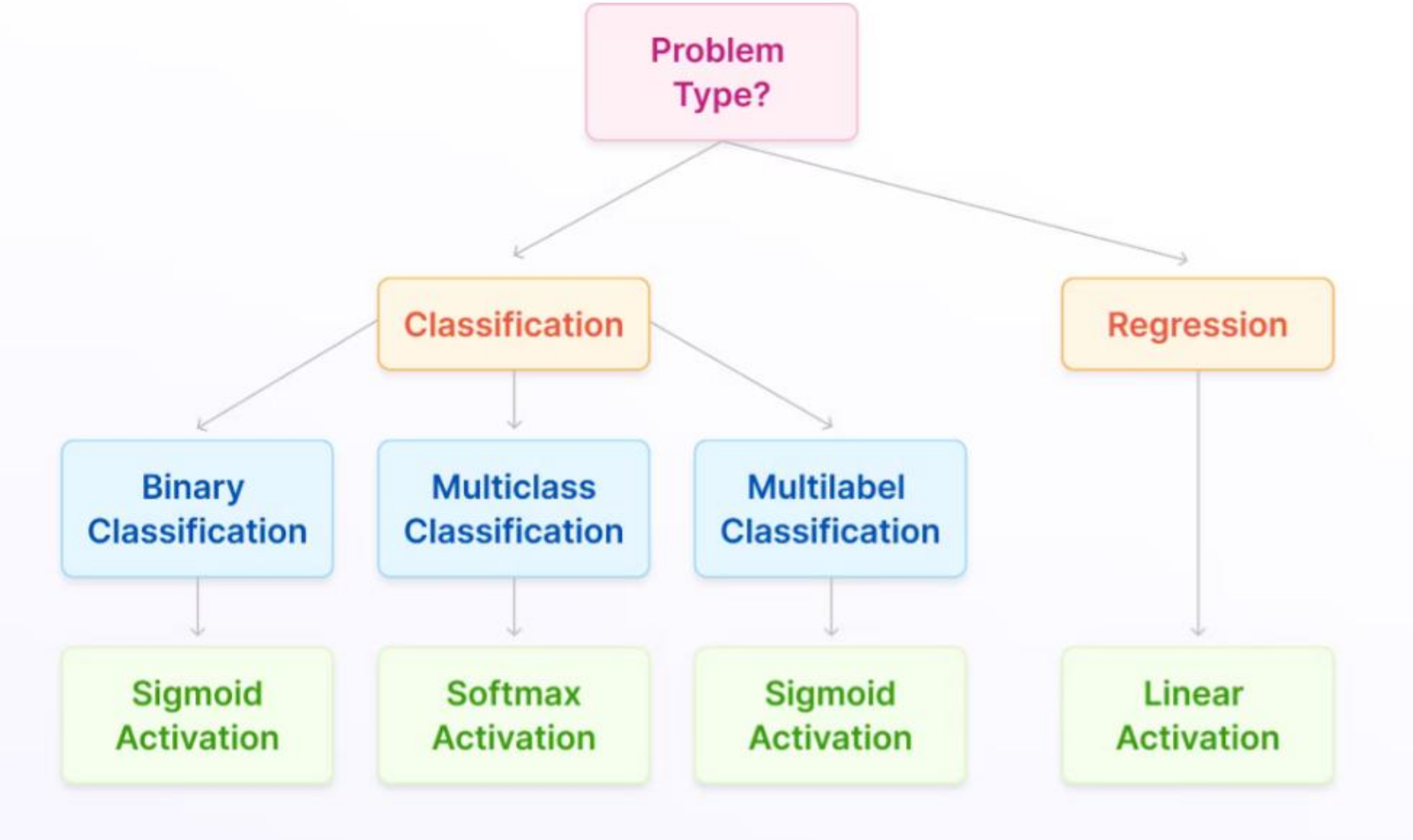
# Functions in Neural Networks: Activation Function

- An activation function in the context of neural networks is a mathematical function applied to the output of a neuron.
- The purpose of an activation function is to introduce non-linearity into the model, allowing the network to learn and represent complex patterns in the data.
- Without non-linearity, a neural network would essentially behave like a linear regression model, regardless of the number of layers it has.
- The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it.
- The purpose of the activation function is to introduce non-linearity into the output of a neuron.

# Functions in Neural Networks: Activation Function

- Activation functions can generally be classified into three main categories: binary step, linear, and non-linear, with numerous subcategories, derivatives, variations, and other calculations now being used in neural networks.
- **Binary step** is the simplest type of activation function, where the output is binary based on whether the input is above or below a certain threshold.
- **Linear** functions are also relatively simple, where the output is proportional to the input.
- **Non-linear** functions are more complex and introduce non-linearity into the model, such as Sigmoid and Tanh.
- In every case, the activation function is picked based on the specific problem and challenge that needs solving.
- It isn't always obvious which one data scientists and machine learning engineers need to use, so sometimes it's a case of trial and error.
- But that's always the starting point for choosing the right activation function for a neural network or any other kind of complicated algorithmic-based model that requires activation functions.

# Functions in Neural Networks: Activation Function



# Functions in Neural Networks: Activation Function

❑**Linear Activation Function (Identity):** In deep learning, data scientists use linear activation functions, also known as identity functions, when they want the output to be the same as the input signal.

❑Identity is differentiable, and like a train passing through a station without stopping, this activation function doesn't change the signal in any way, so it's not used within internal layers of a DL network.

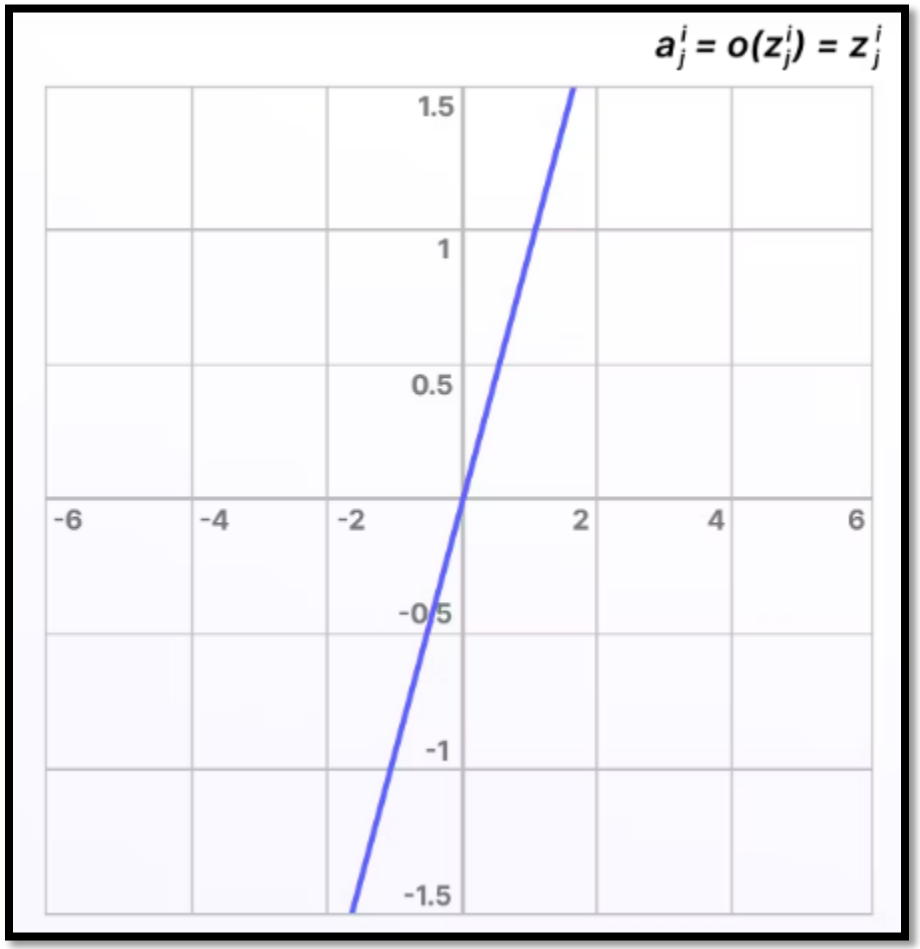
❑Although, in most cases, this might not sound very useful, it is when you want the outputs of your neural network to be continuous rather than modified or discrete. There is no convergence of data, and nothing decreases either.

❑If you use this activation function for every layer, then it would collapse the layers in a neural network into one.

❑So, not very useful unless that's exactly what you need or there are different activation functions in the subsequent hidden layers.

# Functions in Neural Networks: Activation Function

## ❑Linear Activation Function (Identity):



# Functions in Neural Networks: Activation Function

## ❑Non-Linear Activation Functions: Sigmoid, Logistic Activation Functions

❑The Sigmoid activation function, also known as the logistic activation function, takes inputs and turns them into outputs ranging between 0 and 1.

❑For this reason, sigmoid is referred to as the “squashing function” and is differentiable.

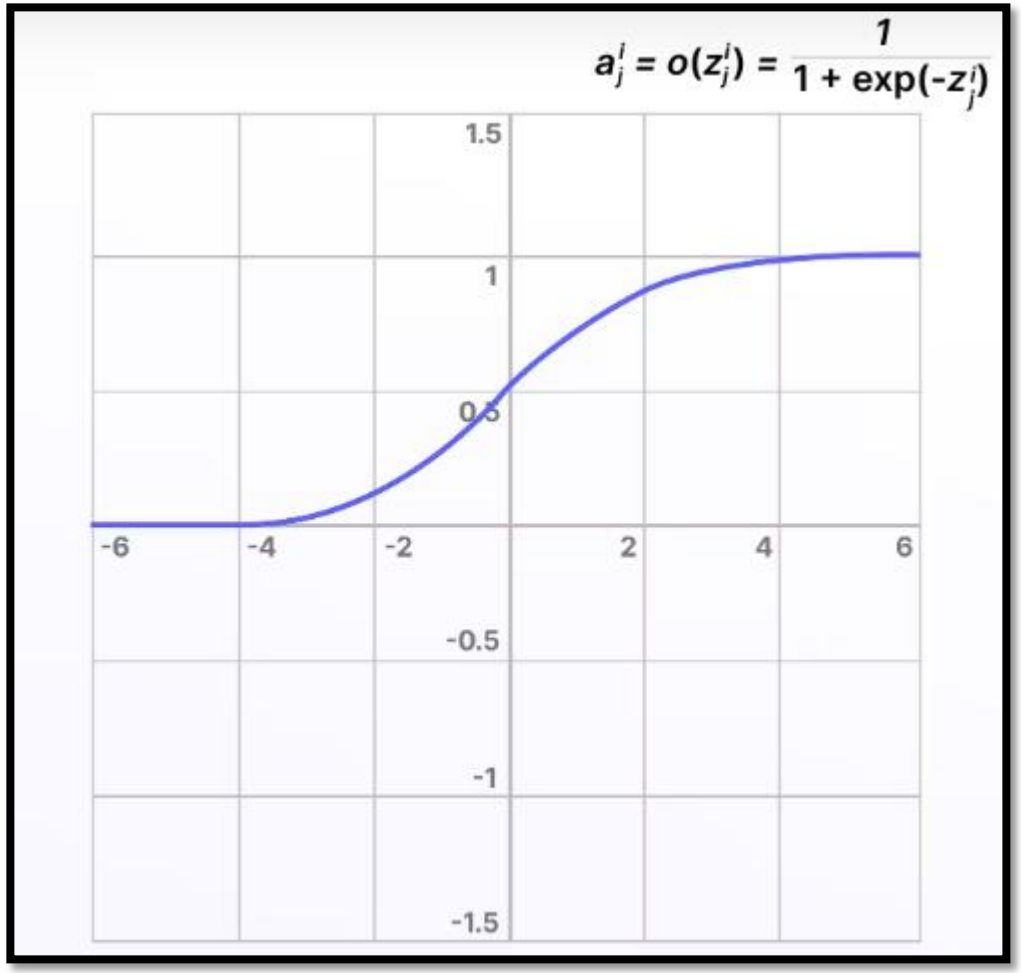
❑Larger, more positive inputs should produce output values close to 1.0, with smaller, more negative inputs producing outputs closer to 0.0.

❑It’s especially useful for classification or probability prediction tasks so that it can be implemented into the training of computer vision and deep learning networks.

❑However, vanishing gradients can make these problematic when used in hidden layers, and this can cause issues when training a model.

# Functions in Neural Networks: Activation Function

## ❑Non-Linear Activation Functions: Sigmoid, Logistic Activation Functions



# Functions in Neural Networks: Activation Function

## ❑Non-Linear Activation Functions: Tanh Function (Hyperbolic Tangent)

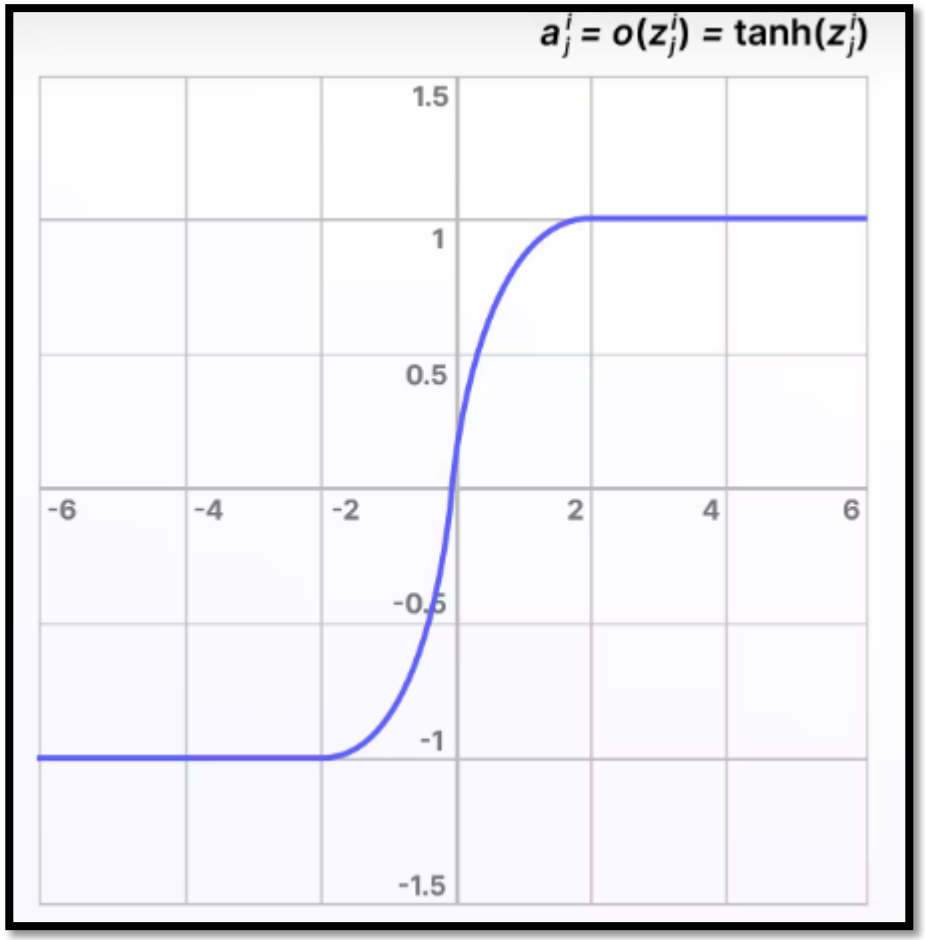
❑Tanh (or TanH), also known as the hyperbolic tangent activation function, is similar to sigmoid/logistic, even down to the S shape curve, and it is differentiable.

❑Except, in this case, the output range is -1 to 1 (instead of 0 to 1). It is a steeper gradient and also encounters the same vanishing gradient challenge as sigmoid/logistic.

❑Because the outputs of tanh are zero-centric, the values can be more easily mapped on a scale between strongly negative, neutral, or positive.

# Functions in Neural Networks: Activation Function

## ❑Non-Linear Activation Functions: Tanh Function (Hyperbolic Tangent)



# Functions in Neural Networks: Activation Function

## ❑Non-Linear Activation Functions: Rectified Linear Unit (ReLU)

❑Compared to linear functions, the rectified linear unit (ReLU) is more computationally efficient For many years, researchers and data scientists mainly used Sigmoid or Tanh, and then when ReLU came along, training performance increased significantly.

❑ReLU isn't differentiable, but this isn't a problem because derivatives can be generated for ReLU.

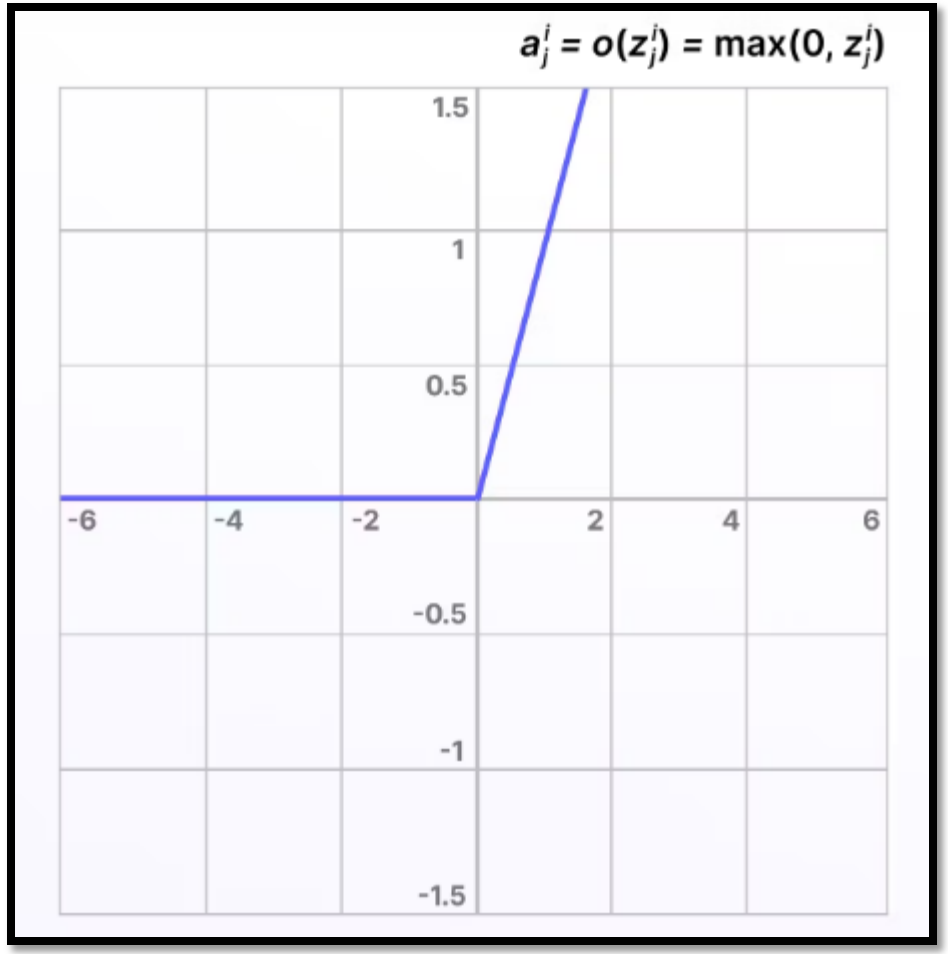
❑ReLU doesn't activate every neuron in sequence at the same time, making it more efficient than the tanh or sigmoid/logistic activation functions.

❑Unfortunately, the downside of this is that some weights and biases for neurons in the network might not get updated or activated.

❑This is known as the "dying ReLU" problem, and it can be solved in a number of ways, such as using variations on this formula, including the exponential ReLU or parametric ReLU function.

# Functions in Neural Networks: Activation Function

## ❑Non-Linear Activation Functions: Rectified Linear Unit (ReLU)



# Functions in Neural Networks: Activation Function

## ❑Non-Linear Activation Functions: Leaky ReLU Function

❑One solution to the “dying ReLU” problem is a variation on this known as the Leaky ReLU activation function.

❑With the Leaky ReLU, instead of being 0 when $z$<0, a leaky ReLU allows a small, non-zero, constant gradient $\alpha$ (Normally, $\alpha$=0.01).

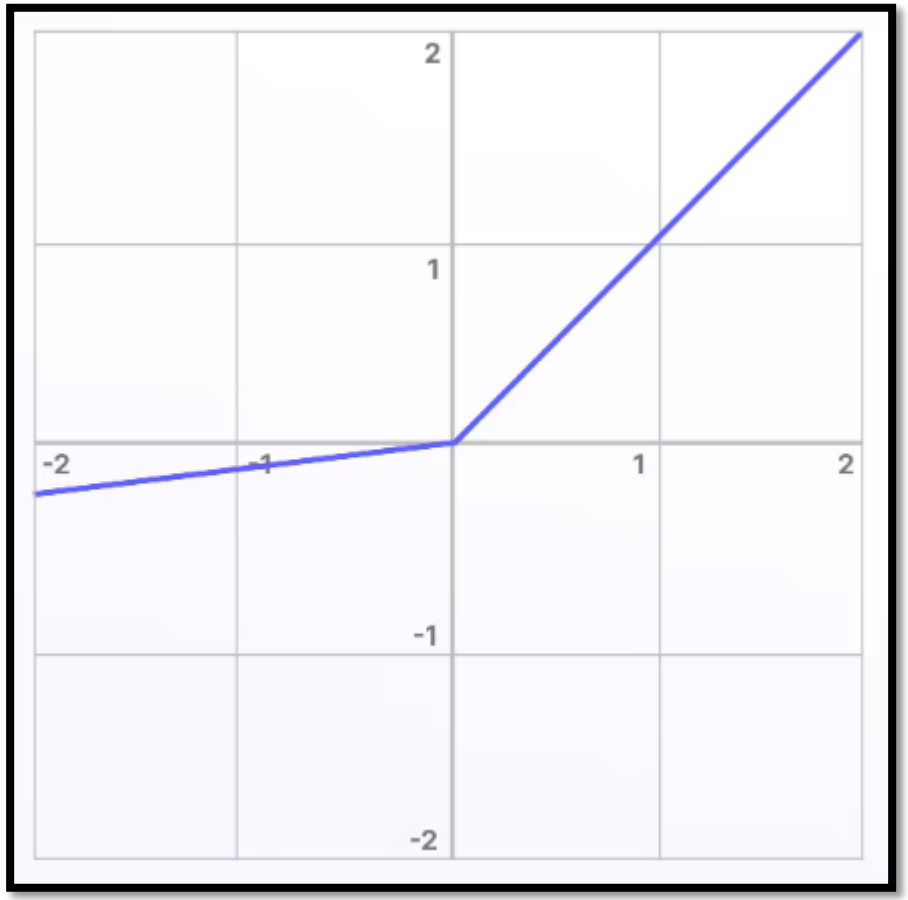
❑Leaky ReLU has been shown to perform better than the traditional ReLU activation function.

❑However, because it possesses linearity it can’t be used for more complex classification tasks and lags behind more advanced activation functions such as Sigmoid and Tanh.

$$R(z) = \begin{cases} z & z > 0 \\ \alpha z & z <= 0 \end{cases}$$

# Functions in Neural Networks: Activation Function

## ❑Non-Linear Activation Functions: Leaky ReLU Function



# Functions in Neural Networks: Activation Function

## ❑Non-Linear Activation Functions: Exponential Linear Units (ELUs) Function

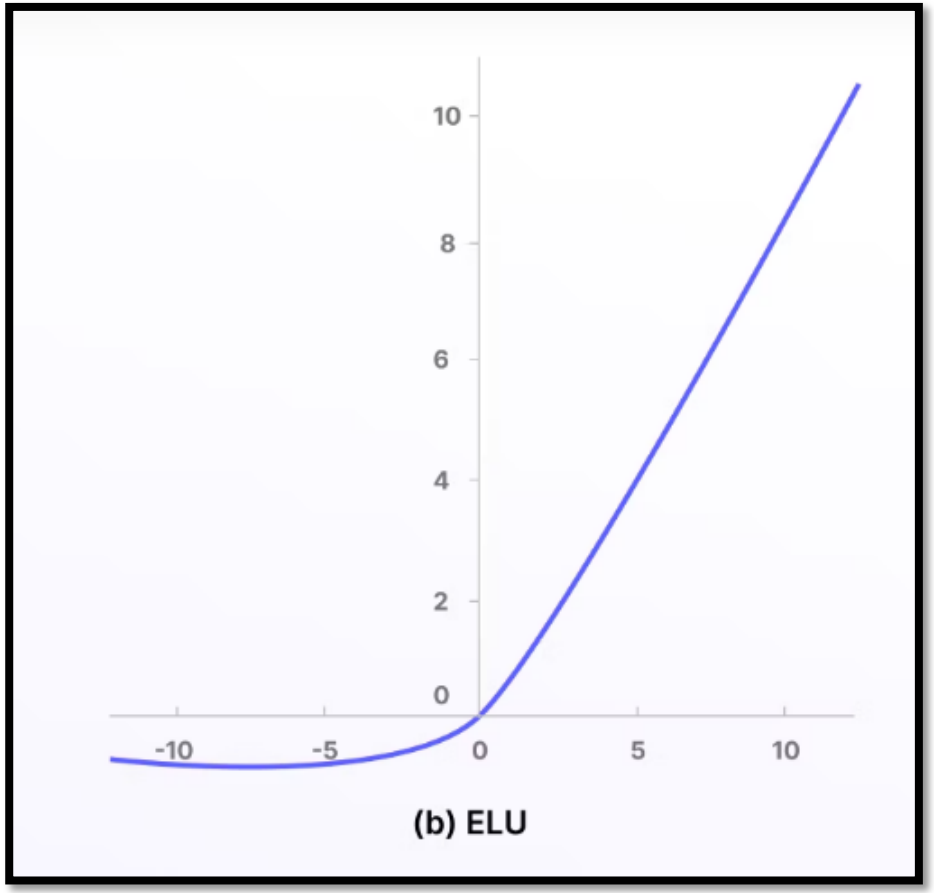
❑The exponential linear units (ELUs) function is another iteration on the original ReLU, another way to overcome the “dying ReLU” problem, and it’s also not differentiable.

❑ELUs use a log curve for negative values instead of a straight line, with it becoming smooth slowly until it reaches -α.

The exponential linear unit (ELU) with $0 < \alpha$ is:

$$f(x) = x \text{ if } x > 0$$

$$\alpha(\exp(x) - 1) \text{ if } x \leq 0$$



# Functions in Neural Networks: Activation Function

## ❑Non-Linear Activation Functions: Scaled Exponential Linear Units (SELUs)

❑Scaled exponential linear units (SELUs)is similar to ELUs, the scaled version of this is also attempting to overcome the same challenges of ReLUs.

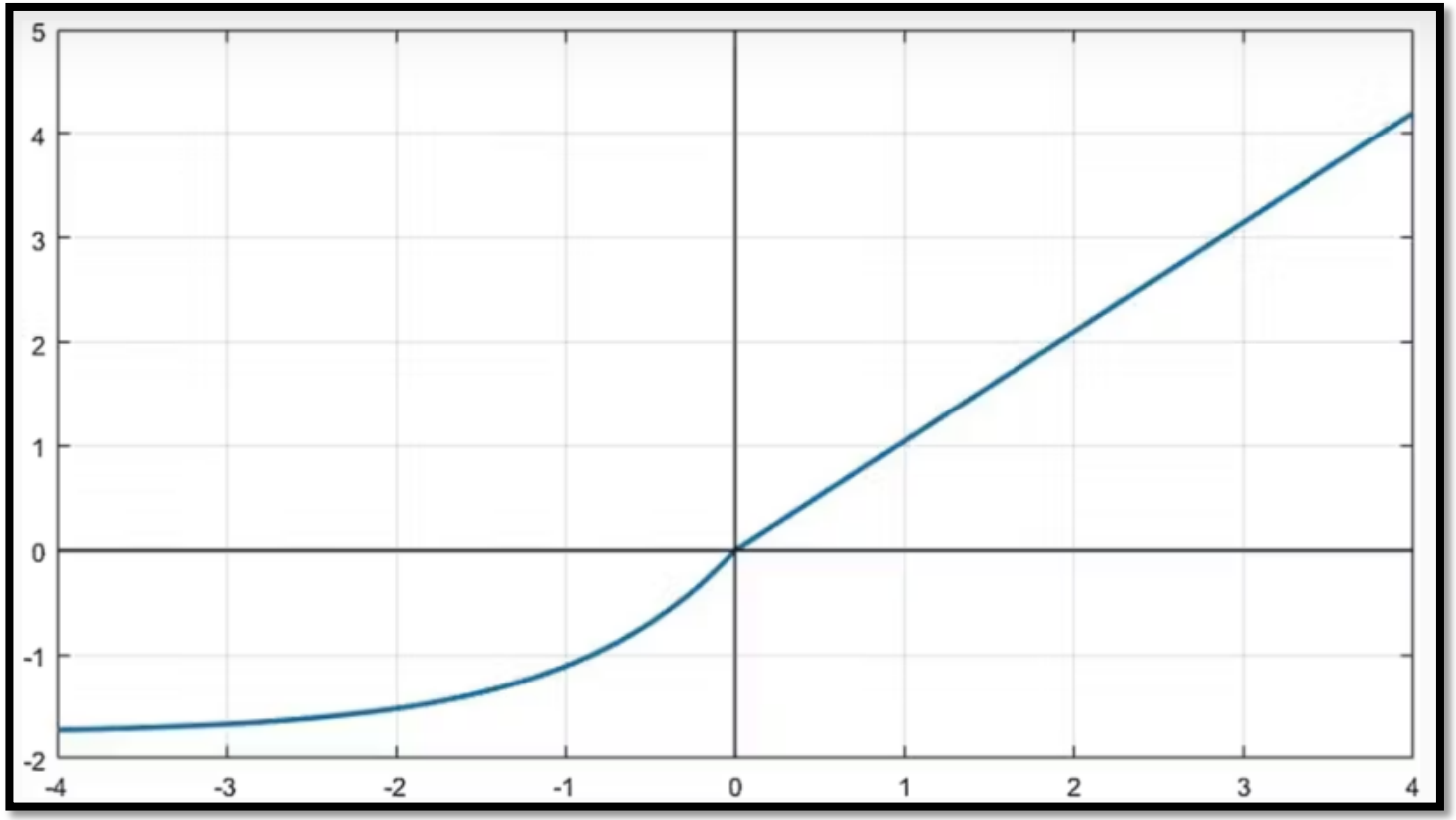
❑**SELUs** control the gradient more effectively and scale the normalization concept, and that is scales with a lambda parameter.

❑SELUs remove the problem of vanishing gradients, can't die (unlike ReLUs), and learn faster and better than other more limited activation functions.

$$\text{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leqslant 0 \end{cases}$$

# Functions in Neural Networks: Activation Function

## ❑Non-Linear Activation Functions: Scaled Exponential Linear Units (SELUs)



# Functions in Neural Networks: Activation Function

## ❑Softmax

❑The softmax function, also known as the softargmax function and the multi-class logistic regression, is one of the most popular and well-used differentiable layer activation functions.

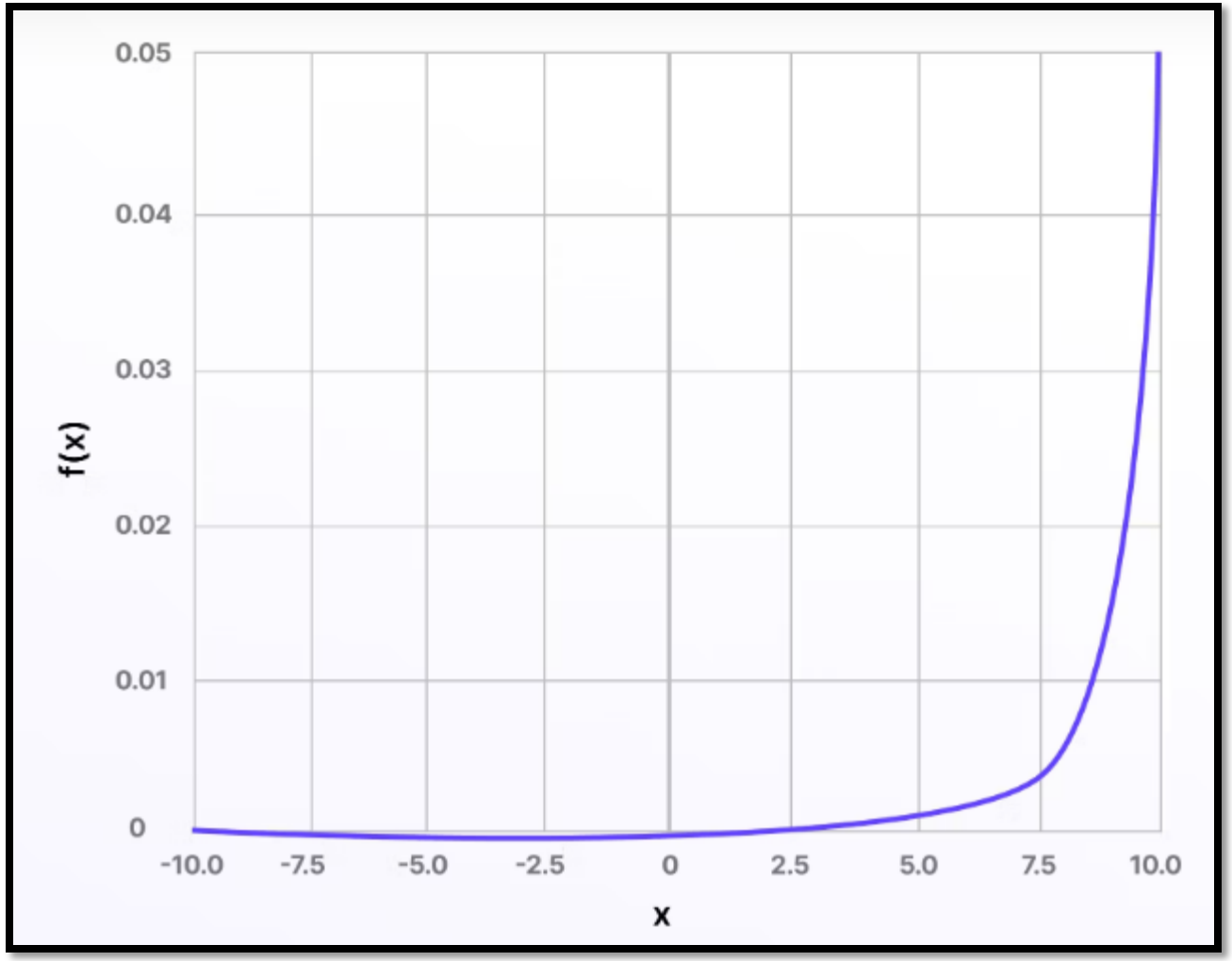
❑Softmax turns input values that are positive, negative, zero, or greater than one into values between 0 and 1.

❑By doing this, it turns input scores into a normalized probability distribution, making softmax a useful activation function in the final layer of deep learning and artificial neural networks.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

# Functions in Neural Networks: Activation Function

## ❑Softmax



# Functions in Neural Networks: Loss Functions

## ❑What is Loss Function in Deep Learning?

❑In deep learning, a loss function, also known as a cost or objective function, is a crucial component that quantifies the dissimilarity between the predicted outputs generated by a neural network and the actual target values in a given dataset.

❑The primary purpose of a loss function is to serve as a measure of how well or poorly the model is performing on a specific task.

❑It provides a numerical value that represents the error or deviation between predictions and the ground truth.

❑The ultimate goal during the training process is to minimize this loss function by iteratively adjusting the model's parameters, ensuring that the neural network becomes increasingly accurate in making predictions.

❑Different types of loss functions are employed depending on the nature of the problem, such as mean squared error for regression or cross-entropy loss for classification, and the choice of the appropriate loss function is pivotal in achieving successful model training and accurate predictions.

# Functions in Neural Networks: Loss Functions

❑**Importance of Loss Function in Deep Learning:** The importance of loss functions in deep learning cannot be overstated. They serve as the backbone of the training process and play a central role in the success of neural network models for several reasons:

1. **Quantifying Model Performance:** Loss functions provide a concrete measure of how well a deep learning model is performing on a given task. They take the difference between predicted and actual values and convert it into a single numerical value, making it easy to assess the model's accuracy and progress during training.
2. **Guiding Model Optimization:** The primary goal during training is to minimize the loss function. By continually adjusting the model's parameters in the direction that reduces the loss, the neural network learns to make more accurate predictions. This optimization process, often done through gradient descent, ensures that the model converges to a state where it performs well on the task.
3. **Customization for Task Complexity:** Deep learning is applied to a wide range of tasks, from image recognition to natural language processing. Different tasks have different characteristics, and loss functions can be tailored to suit these specifics. This adaptability allows deep learning models to excel in diverse domains.
4. **Handling Various Data Types:** Loss functions are designed to accommodate various data types, including continuous values for regression problems and discrete categories for classification tasks. The appropriate choice of loss function ensures that the model's predictions align with the nature of the target variable.

# Functions in Neural Networks: Loss Functions

## ❑Types of Loss Functions

❑Here are some common types of loss functions, categorized into three main groups:

1. Regression loss functions
2. Binary classification loss functions
3. Multi-class classification loss functions

# Functions in Neural Networks: Loss Functions

## ❑Regression Loss Function:

❑A regression loss function is a mathematical function used to quantify the error or discrepancy between the predicted values generated by a regression model and the actual observed values (or target values) in a dataset.

❑The primary purpose of a regression loss function is to measure how well the model's predictions align with the true data points.

# Functions in Neural Networks: Loss Functions

## ❑Regression Loss Function:

❑**Mean Squared Error (MSE)** is one of the most commonly used loss functions in regression analysis and machine learning.

❑It is used to measure the average squared difference between the estimated values generated by a regression model and the actual observed values (target values) in a dataset.

❑The formula for Mean Squared Error (MSE) is as follows:

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

❑Where:

n is the total number of data points in the dataset.

y_i represents the actual target value for the ith data point.

ŷ_i represents the predicted value for the ith data point generated by the regression model.

# Functions in Neural Networks: Loss Functions

## ❑Regression Loss Function:

❑**Mean Absolute Error (MAE)** is a frequently used loss function in regression analysis and machine learning.

❑It is used to measure the average absolute difference between the predicted values generated by a regression model and the actual observed values (target values) in a dataset.

❑The formula for Mean Absolute Error (MAE) is as follows:

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

❑**Where:**

❑n is the total number of data points in the dataset.

❑y_i represents the actual target value for the ith data point.

❑ŷ_i represents the predicted value for the ith data point generated by the regression model.

# Functions in Neural Networks: Loss Functions

## ❑Regression Loss Function:

❑**Huber loss** combines MSE for small errors and MAE for large errors.

❑It introduces a hyperparameter δ that defines the point at which the loss function transitions between being quadratic and linear, making it more robust to outliers.

❑The formula for Huber Loss is as follows:

$$Huber\ Loss = \begin{cases} \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 & |y^{(i)} - \hat{y}^{(i)}| \leq \delta \\ \frac{1}{n}\sum_{i=1}^{n}\delta(|y^{(i)} - \hat{y}^{(i)}| - \frac{1}{2}\delta) & |y^{(i)} - \hat{y}^{(i)}| > \delta \end{cases}$$

❑**Where:**

❑n represents the quantity of data points within the dataset.

❑y signifies the factual or true value associated with each data point.

❑ŷ denotes the anticipated or model-generated value for each data point.

❑δ determines the threshold at which the Huber loss function shifts from being quadratic to linear.

# Functions in Neural Networks: Loss Functions

## ❑Regression Loss Function:

❑**Quantile loss** finds its application in quantile regression, a technique employed when the goal is to estimate a particular quantile, such as the median, from the distribution of the target variable.

❑This method enables the modeling of diverse quantiles within the dataset.

❑The formula for Quantile Loss is as follows:

$$\text{Quantile Loss} = \sum[\alpha * |y\_i - \hat{y}\_i| * (1 - I(y\_i - \hat{y}\_i < 0)) + (1 - \alpha) * |y\_i - \hat{y}\_i| * I(y\_i - \hat{y}\_i < 0)]$$

❑**Where:**

❑α is the quantile level of interest. It's a value between 0 and 1, where α = 0.5 corresponds to estimating the median, α < 0.5 corresponds to estimating lower quantiles, and α > 0.5 corresponds to estimating upper quantiles.

❑y_i represents the actual target value (observed value) for the ith data point.

❑ŷ_i represents the predicted value generated by the quantile regression model for the ith data point.

❑|y_i – ŷ_i| represents the absolute difference between the actual and predicted values.

❑I(y_i – ŷ_i < 0) is an indicator function that equals 1 if y_i – ŷ_i is less than 0 (indicating an underestimation) and 0 otherwise.

# Functions in Neural Networks: Loss Functions

## ❑Regression Loss Function:

❑**Log-Cosh** loss is less sensitive to outliers than MSE.

❑It is a smooth approximation of the Huber loss and can be useful when you want a balance between the robustness of Huber and the differentiability of MSE.

❑The formula for Log-Cosh Loss is as follows:

$$\text{Log-Cosh Loss} = (1/n) \sum \log(\cosh(y\_i - \hat{y}\_i))$$

❑**Where:**

❑n is the total number of data points in the dataset.

❑y_i represents the actual target value for the ith data point.

❑ŷ_i represents the predicted value for the ith data point generated by the regression model.

❑cosh(x) is the hyperbolic cosine function, defined as (e^x + e^(-x))/2.

❑log(x) is the natural logarithm function.

# Functions in Neural Networks: Loss Functions

## ❑Binary Classification Loss Functions

❑Binary classification is a type of supervised learning problem where the goal is to categorize data into one of two classes or categories, typically denoted as 0 (negative or “no”) and 1 (positive or “yes”).

❑In binary classification, various loss functions can be used to measure the difference between the predicted class probabilities and the actual class labels.

# Functions in Neural Networks: Loss Functions

## ❑Binary Classification Loss Functions

❑The **Binary Cross-Entropy** Loss, also known as the Log Loss, is a common loss function used in binary classification tasks.

❑It measures the dissimilarity between predicted probabilities and actual binary labels.

❑The formula for Binary Cross-Entropy Loss is as follows:

$$BCE(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

❑**Where:**

❑y is the actual binary label, which can be either 0 or 1.

❑ŷ is the predicted probability that the given data point belongs to the positive class (class 1).

# Functions in Neural Networks: Loss Functions

## ❑Binary Classification Loss Functions

❑**Hinge Loss**, also known as SVM (Support Vector Machine) Loss, is a loss function commonly used in support vector machines and related classification algorithms.

❑It is particularly suitable for linear classifiers and aims to maximize the margin of separation between classes.

❑The formula for Hinge Loss (SVM Loss) is as follows:

$$Hinge(y, \hat{y}) = \max(0, 1 - y \cdot \hat{y})$$

❑**Where:**

❑Hinge(y, ŷ ) represents the Hinge Loss.

❑y is the actual binary label, which can be either -1 (negative class) or 1 (positive class).

❑ŷ is the raw decision value or score assigned by the classifier for a data point. It is not a probability.

❑The formula calculates the loss for each data point and returns the maximum of 0 and 1 – y . ŷ

# Functions in Neural Networks: Loss Functions

## ❑Multi-class Classification Loss Functions

❑Multi-class classification involves classifying data into one of several distinct classes or categories.

❑Unlike binary classification, where there are only two classes, multi-class classification has more than two possible outcomes.

❑Various loss functions are used in multi-class classification to measure the difference between predicted class probabilities and the actual class labels.

# Functions in Neural Networks: Loss Functions

## ❑Multi-class Classification Loss Functions

❑**Categorical Cross-Entropy**, often simply referred to as Cross-entropy, is a widely used loss function in multi-class classification tasks.

❑It measures the dissimilarity between estimated class probabilities and the true class labels in a categorical setting, where each data point belongs to one of multiple classes.

❑The formula for Categorical Cross-Entropy Loss is as follows:

$$CCE(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

❑**Where:**

❑CCE(y, ŷ ) represents the Categorical Cross-Entropy Loss.

❑y is a one-hot encoded vector representing the true class labels. Each element of y is 0 except for the element corresponding to the true class, which is 1.

❑ŷ is a vector representing the predicted class probabilities for a data point.

# Functions in Neural Networks: Loss Functions

## ❑Multi-class Classification Loss Functions

❑Kullback-Leibler Divergence (KL Divergence) Loss, also known as KL Loss, is a mathematical measure used in machine learning and statistics to quantify the difference between two probability distributions.

❑In the context of loss functions, KL divergence loss is often utilized in tasks where you want to compare or match two probability distributions, such as generative modeling or variational autoencoders.

❑The formula for Kullback-Leibler Divergence (KL Divergence) Loss is as follows:

$$KL(P \,||\, Q) = \Sigma(P\_i * \log(P\_i / Q\_i))$$

❑**Where:**

❑KL(P || Q): This represents the KL divergence from distribution P to distribution Q. It measures how distribution Q differs from the reference distribution P.

❑Σ: This symbol denotes a summation, typically taken over all possible events or outcomes.

❑P_i and Q_i: These are the probabilities of event i occurring in the distributions P and Q, respectively.

❑log(P_i / Q_i): This term computes the logarithm of the ratio of the probabilities of event i occurring in the two distributions. It quantifies how much more or less likely event i is in distribution P compared to distribution Q.

# Function Approximation

❑Function approximation using neural networks involves using a neural network model to learn and approximate a target function from input-output pairs of data.

❑Here's a structured approach to how this is typically done:

❑Steps for Function Approximation with Neural Networks:

1. **Data Collection and Preparation:** Gather a dataset that represents input-output pairs of the function you want to approximate. Ensure the dataset covers a representative range of inputs and includes corresponding outputs.
2. **Model Selection:** Choose a suitable neural network architecture. For basic function approximation tasks, a feedforward neural network (Multi-layer Perceptron, MLP) is commonly used. The number of input nodes should match the dimensionality of your input data, and the output layer should have one node for scalar output or multiple nodes for multi-dimensional output.

# Function Approximation

3. **Training Process:**

   - Split your dataset into training and validation sets. The training set is used to train the neural network, and the validation set is used to monitor the network's performance and prevent overfitting.
   - Define a loss function that measures the difference between the network's predictions and the actual outputs.
   - Choose an optimizer (e.g., SGD, Adam) to minimize the loss function during training.
   - Train the neural network by feeding the training data through the network, computing the loss, and updating the network parameters (weights and biases) using backpropagation.

4. **Hyperparameter Tuning:** Experiment with different hyperparameters such as learning rate, number of hidden layers, number of neurons per layer, activation functions, and regularization techniques (like dropout) to improve the model's performance.

# Function Approximation

5. **Validation and Testing:** Evaluate the trained model using the validation set to ensure it generalizes well to unseen data. Use the test set (if available) to further validate the model's performance.
6. **Prediction and Deployment:** Once satisfied with the model's performance, deploy it for making predictions on new data. Ensure that the input data is preprocessed in the same way as the training data.

# Function Approximation

❑**Example Workflow:** Let's say you want to approximate a function **f(x)=sin(x)** using a neural network:

1. **Data Collection:** Generate input-output pairs for a range of **x** values.
2. **Model Selection:** Choose a simple MLP with one input layer (1 node), one or more hidden layers with suitable activation functions (like ReLU), and an output layer (1 node for the scalar output).
3. **Training:** Split data into training and validation sets. Train the network using SGD optimizer with Mean Squared Error (MSE) loss function.
4. **Hyperparameter Tuning:** Adjust learning rate, number of hidden layers, neurons per layer, etc., to improve validation performance.
5. **Validation:** Validate the model's performance using the validation set. Adjust as needed.
6. **Testing:** Finally, test the model on a separate test set (if available) to ensure it performs well on unseen data.
7. **Deployment:** Deploy the trained model to predict **sin(x)** values for new input x values.

# Classification and Clustering Problems

- In deep learning, classification and clustering are fundamental problems that deal with categorizing and organizing data.
- Let’s break down each problem and discuss their approaches and challenges.

# Classification and Clustering Problems

❑**Classification Problems:** Classification involves predicting the categorical label of input data based on its features. The goal is to assign each data point to one of a predefined set of classes.

❑**Key Concepts**:

- **Supervised Learning**: Classification is a type of supervised learning where the model is trained on labeled data.
- **Loss Functions**: Common loss functions for classification include Cross-Entropy Loss for multi-class problems and Binary Cross-Entropy for binary classification.
- **Evaluation Metrics**: Metrics like Accuracy, Precision, Recall, F1 Score, and ROC-AUC are used to evaluate classification performance.

❑**Common Techniques**:

- **Neural Networks**: Multi-layer perceptrons (MLPs), Convolutional Neural Networks (CNNs) for image data, and Recurrent Neural Networks (RNNs) or Transformers for sequential data.
- **Transfer Learning**: Leveraging pre-trained models (e.g., ResNet, BERT) and fine-tuning them on specific tasks.

# Classification and Clustering Problems

❑**Clustering Problems**: Clustering involves grouping data points into clusters such that data points in the same cluster are more similar to each other than to those in other clusters. Unlike classification, clustering is an unsupervised learning problem.

❑**Key Concepts**:

▪**Unsupervised Learning**: Clustering does not require labeled data. The algorithm identifies patterns or structures in the data.

▪**Distance Metrics**: Metrics like Euclidean distance, Manhattan distance, or cosine similarity are used to measure the similarity between data points.

❑**Common Techniques**:

▪**K-Means Clustering**: Partitional clustering that minimizes within-cluster variance. The number of clusters kkk must be specified.

▪**Hierarchical Clustering**: Builds a hierarchy of clusters either by agglomeration (bottom-up) or division (top-down).

▪**DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**: Identifies clusters based on the density of points and can find arbitrarily shaped clusters and outliers.

# Classification and Clustering Problems

## ❑Classification Problems

### 1. Image Classification:

**Problem:** Assigning a label to an image from a fixed set of categories.

**Deep Learning Solution:** Convolutional Neural Networks (CNNs) are commonly used. They are adept at capturing spatial hierarchies in images through their convolutional layers.

### 2. Text Classification:

**Problem:** Categorizing text into predefined categories (e.g., spam detection, sentiment analysis).

**Deep Learning Solution:** Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformers are used. Transformers, in particular, have revolutionized text classification tasks due to their ability to handle long-range dependencies and contextual information.

### 3. Speech Classification:

**Problem:** Classifying spoken language into categories, such as identifying the speaker or transcribing speech.

**Deep Learning Solution:** Models like RNNs, LSTMs, and more recently, attention-based mechanisms and transformers, are employed to handle sequential and temporal aspects of speech data.

# Classification and Clustering Problems

## ❑Clustering Problems

### 1. Customer Segmentation:

**Problem:** Grouping customers into clusters based on their behavior or characteristics for targeted marketing.

**Deep Learning Solution:** Autoencoders can be used to learn representations of customer data, which can then be clustered using algorithms like k-means or hierarchical clustering on these learned features.

### 2. Document Clustering:

**Problem:** Grouping similar documents together based on their content.

**Deep Learning Solution:** Embedding techniques like those provided by BERT or other transformer models can represent documents in a high-dimensional space where traditional clustering algorithms can be applied.

### 3. Anomaly Detection:

**Problem:** Identifying unusual data points that do not fit the pattern of the majority.

**Deep Learning Solution:** Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) can model normal data distributions, making it easier to detect anomalies.

# Deep Network Basics

## ❑What is Deep Learning?

❑Deep learning is a type of machine learning that teaches computers to perform tasks by learning from examples, much like humans do.

❑Imagine teaching a computer to recognize cats: instead of telling it to look for whiskers, ears, and a tail, you show it thousands of pictures of cats.

❑The computer finds the common patterns all by itself and learns how to identify a cat.

❑This is the essence of deep learning.

❑In technical terms, deep learning uses something called "neural networks," which are inspired by the human brain.

❑These networks consist of layers of interconnected nodes that process information.

❑The more layers, the "deeper" the network, allowing it to learn more complex features and perform more sophisticated tasks.

# Deep Network Basics

## ❑How deep learning differs from traditional machine learning

❑While machine learning has been a transformative technology in its own right, deep learning takes it a step further by automating many of the tasks that typically require human expertise.

❑Deep learning is essentially a specialized subset of machine learning, distinguished by its use of neural networks with three or more layers.

❑These neural networks attempt to simulate the behavior of the human brain—albeit far from matching its ability—in order to "learn" from large amounts of data.

# Deep Network Basics

❑**Why is Deep Learning Important:** The reasons why deep learning has become the industry standard:

1. **Handling unstructured data:** Models trained on structured data can easily learn from unstructured data, which reduces time and resources in standardizing data sets.
2. **Handling large data:** Due to the introduction of graphics processing units (GPUs), deep learning models can process large amounts of data with lightning speed.
3. **High Accuracy:** Deep learning models provide the most accurate results in computer visions, natural language processing (NLP), and audio processing.
4. **Pattern Recognition:** Most models require machine learning engineer intervention, but deep learning models can detect all kinds of patterns automatically.

# Deep Network Basics

## ❑What are Deep Neural Networks?

❑An artificial neural network (ANN) or a simple traditional neural network aims to solve trivial tasks with a straightforward network outline.

❑An artificial neural network is loosely inspired from biological neural networks.

❑It is a collection of layers to perform a specific task. Each layer consists of a collection of nodes to operate together.

❑These networks usually consist of an input layer, one to two hidden layers, and an output layer.

❑While it is possible to solve easy mathematical questions, and computer problems, including basic gate structures with their respective truth tables, it is tough for these networks to solve complicated image processing, computer vision, and natural language processing tasks.

# Deep Network Basics

## ❑What are Deep Neural Networks?

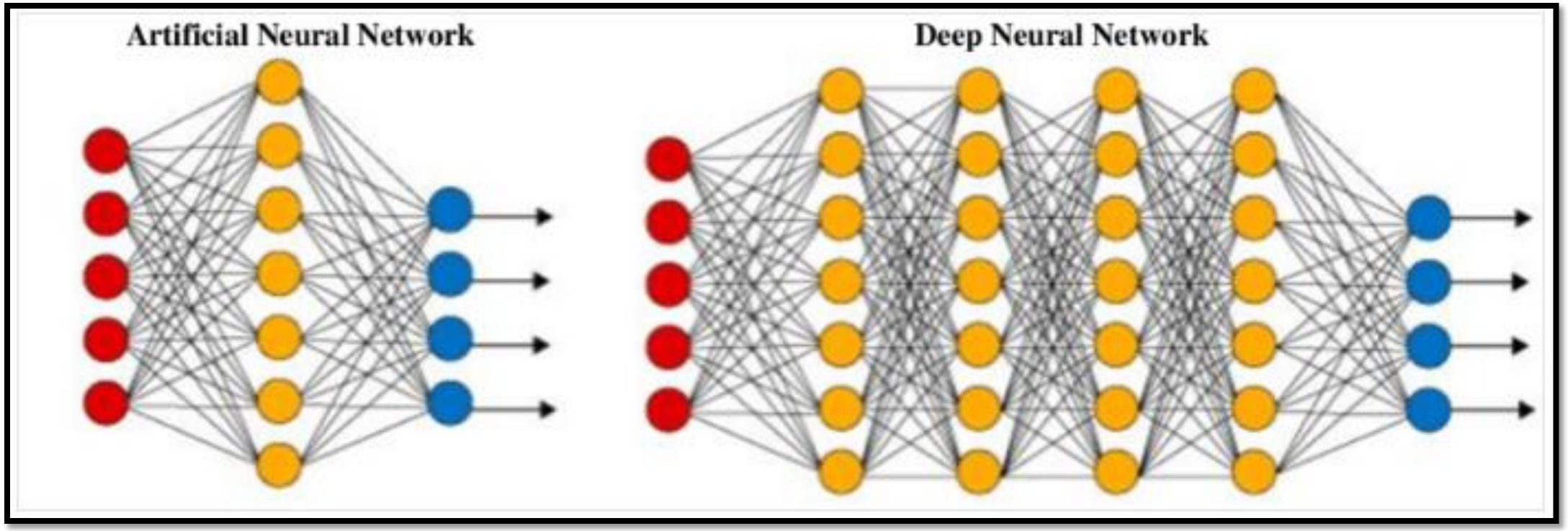
❑For these problems, we utilize deep neural networks, which often have a complex hidden layer structure with a wide variety of different layers, such as a convolutional layer, max-pooling layer, dense layer, and other unique layers.

❑These additional layers help the model to understand problems better and provide optimal solutions to complex projects.

❑A deep neural network has more layers (more depth) than ANN and each layer adds complexity to the model while enabling the model to process the inputs concisely for outputting the ideal solution.

# Deep Network Basics

## ❑What are Deep Neural Networks?



# Deep Network Basics

## ❑How Deep Learning Works

❑Deep learning uses feature extraction to recognize similar features of the same label and then uses decision boundaries to determine which features accurately represent each label.

❑In the cats and dogs classification, the deep learning models will extract information such as the eyes, face, and body shape of animals and divide them into two classes.

❑The deep learning model consists of deep neural networks. The simple neural network consists of an input layer, a hidden layer, and an output layer.

❑Deep learning models consist of multiple hidden layers, with additional layers that the model's accuracy has improved.

❑The input layers contain raw data, and they transfer the data to hidden layers' nodes.

❑The hidden layers' nodes classify the data points based on the broader target information, and with every subsequent layer, the scope of the target value narrows down to produce accurate assumptions.

❑The output layer uses hidden layer information to select the most probable label.

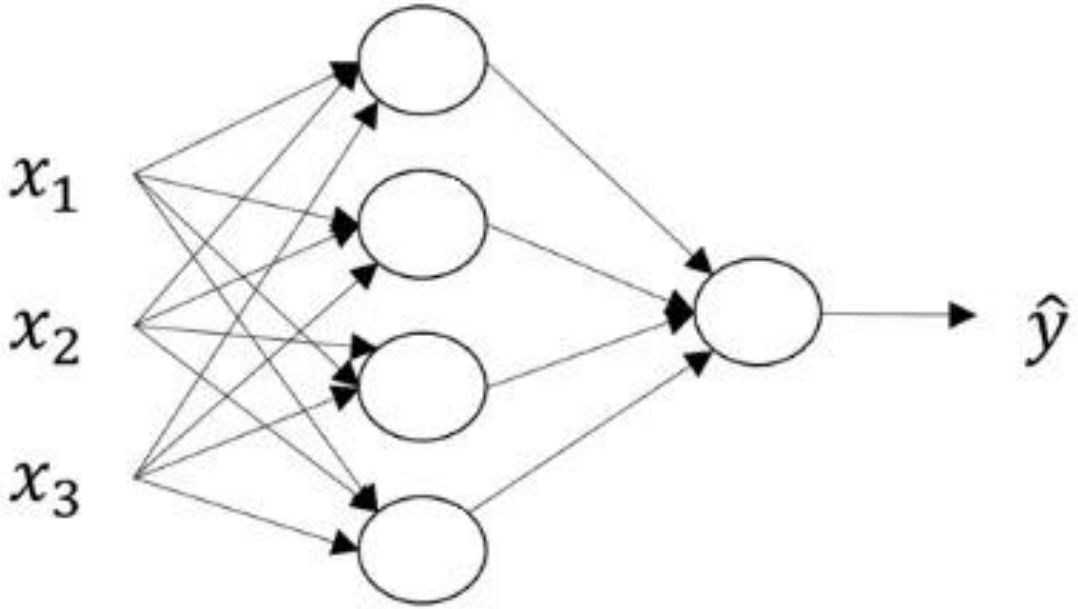
# Deep Network Basics

## ❑Deep Learning Models

1. **Convolutional Neural Networks:** (or just convolutional networks) are commonly used to analyze visual content. They are similar to regular neural networks, but they have an extra layer of processing that helps them to better identify patterns in images. This makes them particularly well suited to tasks such as image recognition and classification.
2. **Recurrent Neural Network (RNN):** is a type of artificial neural network where connections between nodes form a directed graph along a sequence. This allows it to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process sequences of inputs. This makes them valuable for tasks such as unsegmented, connected handwriting recognition or speech recognition.
3. **Long Short-Term Memory (LSTM):** networks are a type of recurrent neural network that can learn and remember long-term dependencies. They are often used in applications such as natural language processing and time series prediction. LSTM networks are well suited to these tasks because they can store information for long periods of time. They can also learn to recognize patterns in sequences of data.

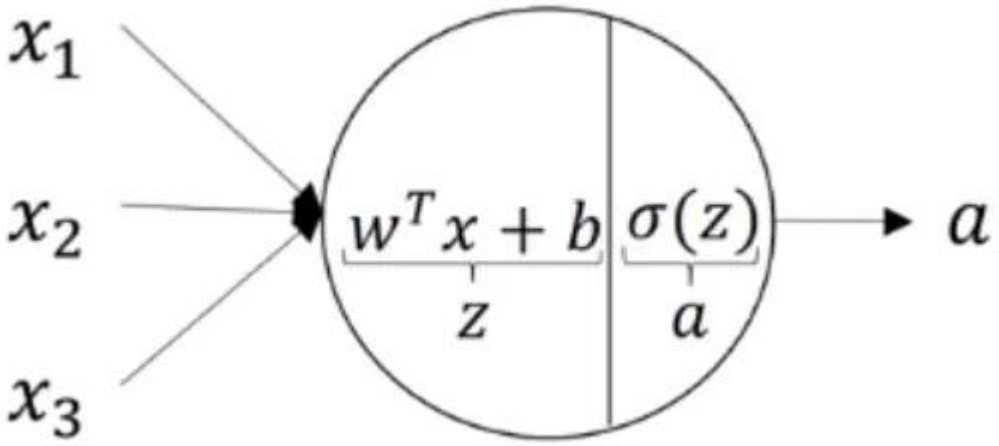
# Shallow Neural Networks

- Shallow neural networks consist of only 1 or 2 hidden layers.
- Understanding a shallow neural network gives us an insight into what exactly is going on inside a deep neural network.
- The figure below shows a shallow neural network with 1 hidden layer, 1 input layer and 1 output layer.



# Shallow Neural Networks

- **The neuron is the atomic** unit of a neural network.
- Given an input, it provides the output and passes that output as an input to the subsequent layer.
- A neuron can be thought of as a combination of 2 parts:




i. The first part computes the output ***Z***, using the inputs and the weights.

ii. The second part performs the activation on **Z** to give out the final output **A** of the neuron.

# Shallow Neural Networks

- The hidden layer comprises of various neurons, each of which performs the above 2 calculations.
- The 4 neurons present in the hidden layer of our shallow neural network compute the following:

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, a_1^{[1]} = \sigma\left(z_1^{[1]}\right)$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, a_2^{[1]} = \sigma\left(z_2^{[1]}\right)$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, a_3^{[1]} = \sigma\left(z_3^{[1]}\right)$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, a_4^{[1]} = \sigma\left(z_4^{[1]}\right)$$

# Shallow Neural Networks

❑In the above equations (Previous slide)

1. The superscript number ***[i]*** denotes the layer number and the subscript number ***j*** denotes the neuron number in a particular layer.
2. ***X*** is the input vector consisting of 3 features.
3. ***W[i]j*** is the weight associated with neuron ***j*** present in the layer ***i***.
4. ***b[i]j*** is the bias associated with neuron ***j*** present in the layer ***i***.
5. ***Z[i]j*** is the intermediate output associated with neuron ***j*** present in the layer ***i***.
6. ***A[i]j*** is the final output associated with neuron ***j*** present in the layer ***i***.
7. ***Sigma*** is the sigmoid activation function. Mathematically it is defined as:

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

# Shallow Neural Networks

- As we can see, the above 4 equations seem redundant. Therefore we will vectorize them as:

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

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

i. The first equation computes all the intermediate outputs ***Z*** in single matrix multiplication.

ii. The second equation computes all the activations **A** in single matrix multiplication.

# Shallow Neural Networks

## ❑The Shallow 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.

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

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

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

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

# Shallow Neural Networks

## ❑The Shallow Neural Network:

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.

# Gradient Descent

- Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function.
- Gradient descent in machine learning is simply used to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible.
- You start by defining the initial parameter's values and from there the gradient descent algorithm uses calculus to iteratively adjust the values so they minimize the given cost-function.
- To understand this concept fully, it's important to know about gradients.

# Gradient Descent

## ❑What Is a Gradient?

- ❑A gradient simply measures the change in all weights with regard to the change in error.
- ❑You can also think of a gradient as the slope of a function.
- ❑The higher the gradient, the steeper the slope and the faster a model can learn.
- ❑But if the slope is zero, the model stops learning. In mathematical terms, a gradient is a partial derivative with respect to its inputs.
- ❑Imagine a blindfolded man who wants to climb to the top of a hill with the fewest steps along the way as possible. He might start climbing the hill by taking really big steps in the steepest direction, which he can do as long as he is not close to the top.

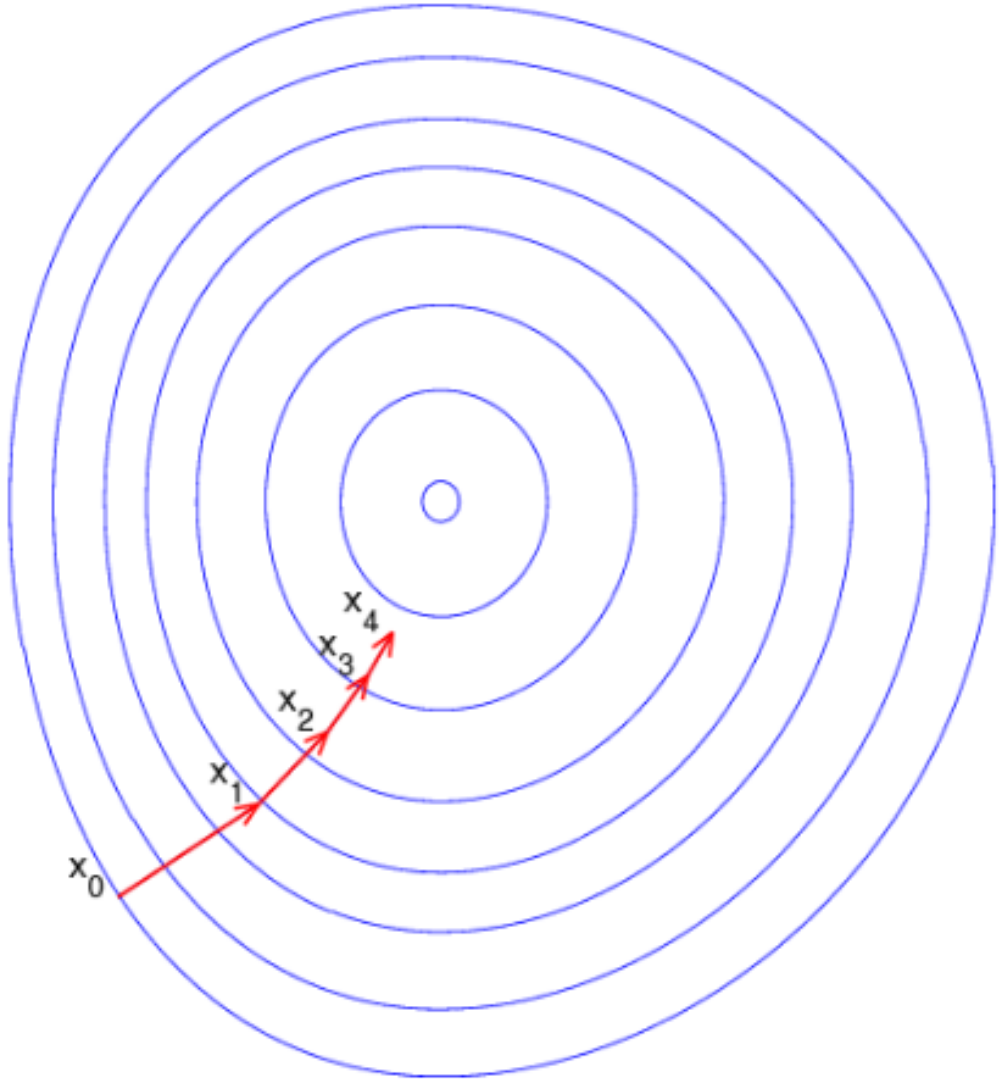
# Gradient Descent

## ❑What Is a Gradient?

❑As he comes closer to the top, however, his steps will get smaller and smaller to avoid overshooting it. This process can be described mathematically using the gradient.

❑Imagine the image below illustrates our hill from a top-down view and the red arrows are the steps of our climber.

❑Think of a gradient in this context as a vector that contains the direction of the steepest step the blindfolded man can take and also how long that step should be.



# Gradient Descent

## ❑What Is a Gradient?

- ❑Note that the gradient ranging from X0 to X1 is much longer than the one reaching from X3 to X4.
- ❑This is because the steepness/slope of the hill, which determines the length of the vector, is less.
- ❑This perfectly represents the example of the hill because the hill is getting less steep the higher it's climbed.
- ❑Therefore a reduced gradient goes along with a reduced slope and a reduced step size for the hill climber.

# Gradient Descent

## ❑How Gradient Descent Works?

❑Instead of climbing up a hill, think of gradient descent as hiking down to the bottom of a valley.

❑This is a better analogy because it is a minimization algorithm that minimizes a given function.

❑The equation below describes what the gradient descent algorithm does: *b* is the next position of our climber, while *a* represents his current position.

❑The minus sign refers to the minimization part of the gradient descent algorithm.

❑The gamma in the middle is a waiting factor and the gradient term (Δf(a)) is simply the direction of the steepest descent.

$$\mathbf{b} = \mathbf{a} - \gamma \nabla \mathbf{f}(\mathbf{a})$$

❑So this formula basically tells us the next position we need to go, which is the direction of the steepest descent. Let's look at another example to really drive the concept home.

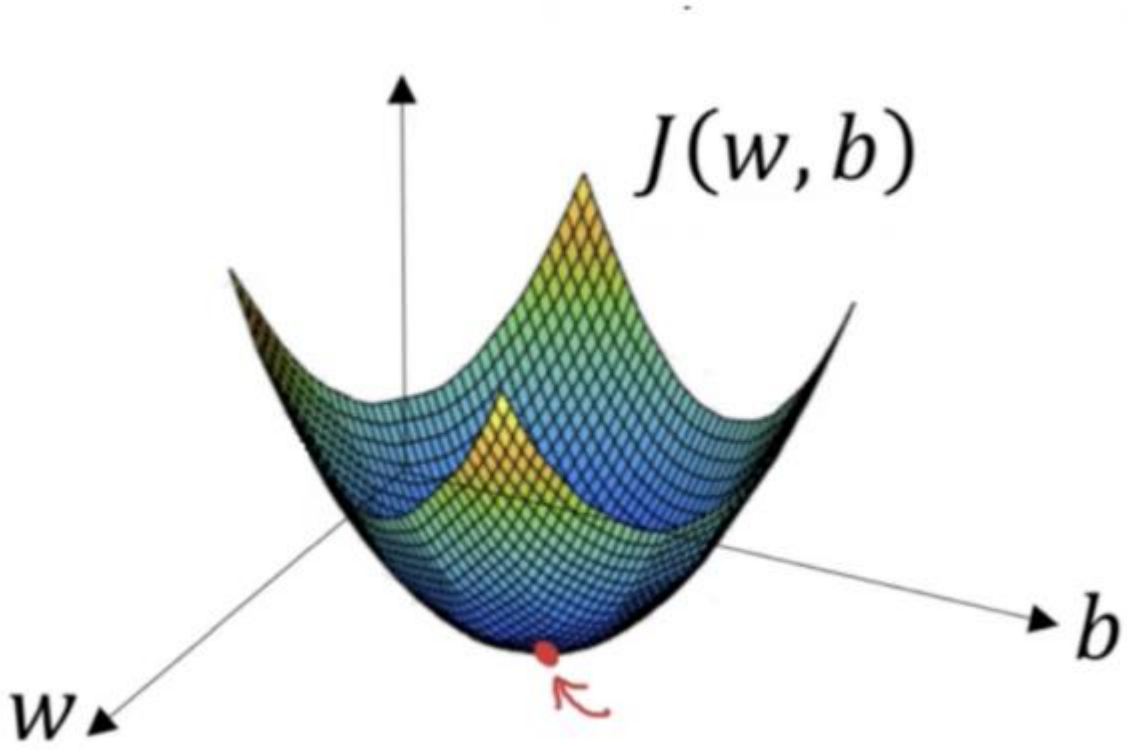
# Gradient Descent

## ❑How Gradient Descent Works?

❑Imagine you have a machine learning problem and want to train your algorithm with gradient descent to minimize your cost-function $J(w, b)$ and reach its local minimum by tweaking its parameters ($w$ and $b$).

❑The image below shows the horizontal axes representing the parameters ($w$ and $b$), while the cost function $J(w, b)$ is represented on the vertical axes.

❑Gradient descent is a convex function.



# Gradient Descent

## ❑How Gradient Descent Works?

❑We know we want to find the values of *w* and *b* that correspond to the minimum of the cost function (marked with the red arrow).

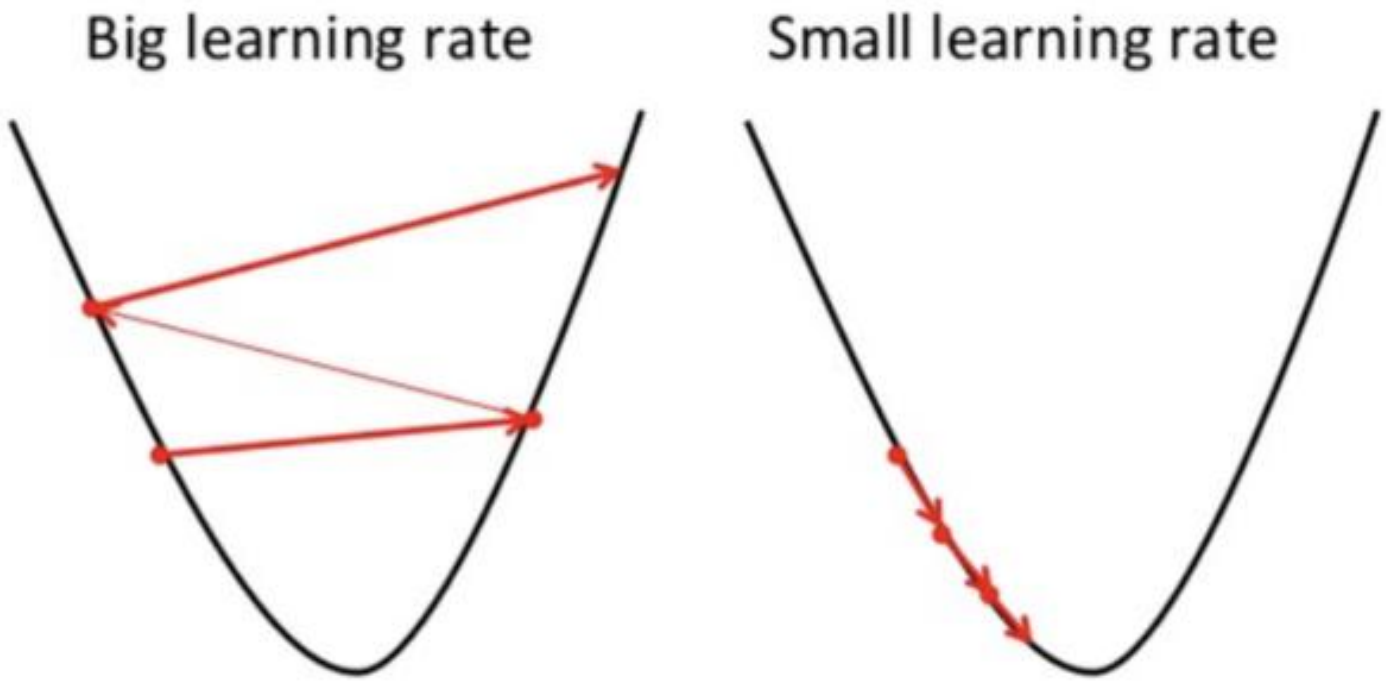
❑To start finding the right values we initialize *w* and *b* with some random numbers.

❑Gradient descent then starts at that point (somewhere around the top of our illustration), and it takes one step after another in the steepest downside direction (i.e., from the top to the bottom of the illustration) until it reaches the point where the cost function is as small as possible.

# Gradient Descent

## ❑ Gradient Descent Learning Rate

❑How big the steps gradient descent takes into the direction of the local minimum are determined by the learning rate, which figures out how fast or slow we will move towards the optimal weights.



# Gradient Descent

## ❑ Gradient Descent Learning Rate

❑For the gradient descent algorithm to reach the local minimum we must set the learning rate to an appropriate value, which is neither too low nor too high.

❑This is important because if the steps it takes are too big, it may not reach the local minimum because it bounces back and forth between the convex function of gradient descent (see left image below).

❑If we set the learning rate to a very small value, gradient descent will eventually reach the local minimum but that may take a while (see the right image).

❑So, the learning rate should never be too high or too low for this reason.

❑You can check if your learning rate is doing well by plotting it on a graph.

# Gradient Descent

## ❑ How to Solve Gradient Descent Challenges

❑A good way to make sure the gradient descent algorithm runs properly is by plotting the cost function as the optimization runs.

❑Put the number of iterations on the x-axis and the value of the cost function on the y-axis.

❑This helps you see the value of your cost function after each iteration of gradient descent, and provides a way to easily spot how appropriate your learning rate is.

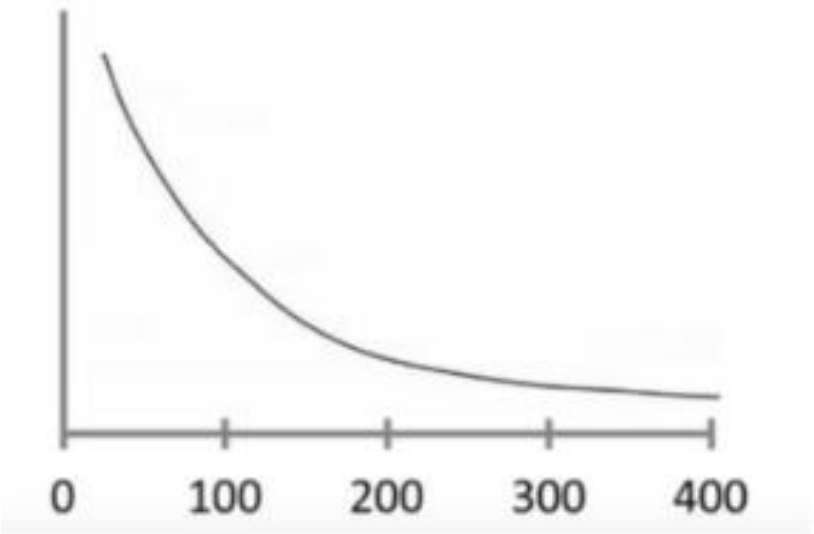
❑You can just try different values for it and plot them all together.

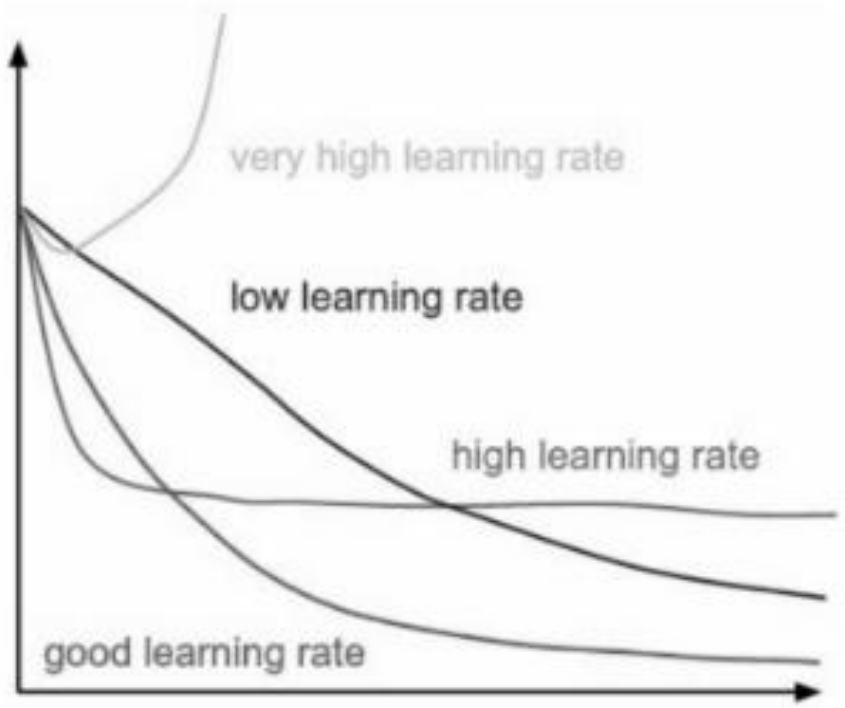
❑The left image below shows such a plot, while the image on the right illustrates the difference between good and bad learning rates.

❑If the gradient descent algorithm is working properly, the cost function should decrease after every iteration.

# Gradient Descent

## ❑ How to Solve Gradient Descent Challenges



# Gradient Descent

## ❑ Types Gradient Descent

❑**Batch gradient descent:** also called vanilla gradient descent, calculates the error for each example within the training dataset, but only after all training examples have been evaluated does the model get updated.

❑This whole process is like a cycle and it's called a training epoch.

❑Some advantages of batch gradient descent are its computational efficiency: it produces a stable error gradient and a stable convergence.

❑Some disadvantages are that the stable error gradient can sometimes result in a state of convergence that isn't the best the model can achieve.

❑It also requires the entire training dataset to be in memory and available to the algorithm.

# Gradient Descent

## ❑ Types Gradient Descent

❑**Mini-batch gradient descent** is the go-to method since it's a combination of the concepts of SGD and batch gradient descent.

❑It simply splits the training dataset into small batches and performs an update for each of those batches.

❑This creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.

❑Common mini-batch sizes range between 50 and 256, but like any other machine learning technique, there is no clear rule because it varies for different applications.

❑This is the go-to algorithm when training a neural network and it is the most common type of gradient descent within deep learning.

# Gradient Descent

## ❑ Types Gradient Descent

❑**Stochastic Gradient Descent:** By contrast, stochastic gradient descent (SGD) does this for each training example within the dataset, meaning it updates the parameters for each training example one by one.

❑Depending on the problem, this can make SGD faster than batch gradient descent.

❑One advantage is the frequent updates allow us to have a pretty detailed rate of improvement.

❑The frequent updates, however, are more computationally expensive than the batch gradient descent approach.

❑Additionally, the frequency of those updates can result in noisy gradients, which may cause the error rate to jump around instead of slowly decreasing.

# Forward Propagation

- Feedforward neural networks stand as foundational architectures in deep learning.
- Neural networks consist of an input layer, at least one hidden layer, and an output layer.
- Each node is connected to nodes in the preceding and succeeding layers with corresponding weights and thresholds.

# Forward Propagation

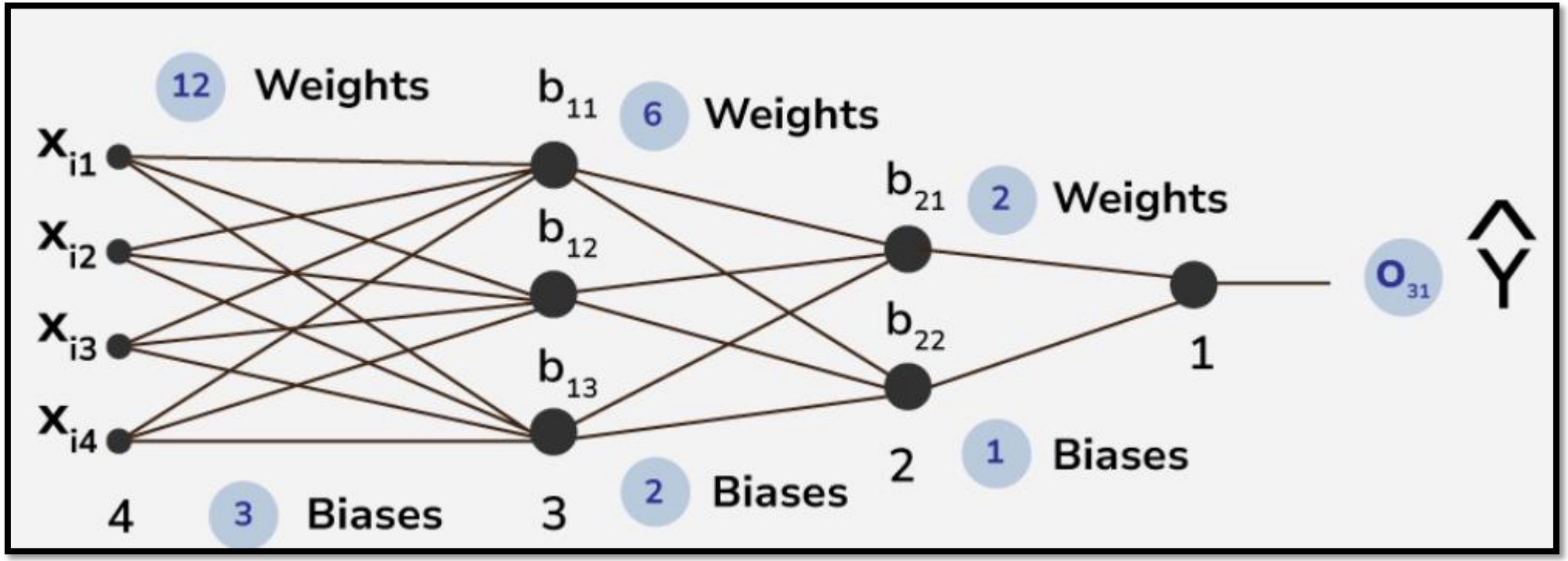
❑Forward propagation is the process in a neural network where the input data is passed through the network's layers to generate an output.

❑It involves the following steps:

1. **Input Layer:** The input data is fed into the input layer of the neural network.
2. **Hidden Layers:** The input data is processed through one or more hidden layers. Each neuron in a hidden layer receives inputs from the previous layer, applies an activation function to the weighted sum of these inputs, and passes the result to the next layer.
3. **Output Layer:** The processed data moves through the output layer, where the final output of the network is generated. The output layer typically applies an activation function suitable for the task, such as softmax for classification or linear activation for regression.
4. **Prediction:** The final output of the network is the prediction or classification result for the input data.

# Forward Propagation

- Forward propagation is essential for making predictions in neural networks.
- It calculates the output of the network for a given input based on the current values of the weights and biases.
- The output is then compared to the actual target value to calculate the loss, which is used to update the weights and biases during the training process.



# Forward Propagation

- In the above picture , the first layer of $X_{i1}, X_{i2}, X_{i3}, X_{i4}$ are the input layer and last layer of $b_{31}$ is the output layer .
- Other layers are hidden layers in this structure of ANN . this is a 4 layered deep ANN where first hidden layer consists of 3 neuron and second layer consists of 2 neuron .
- There are total 26 trainable parameters . here , in hidden layer 1 , the top to bottom biases are $b_{11}, b_{12}, b_{13}$ and in hidden layer 2 , the top to bottom biases are $b_{21}, b_{22}$ . the output layer contains the neuron having bias b31 . likewise the weights of corresponding connections are assigned like $W_{111}, W_{112}, W_{113}, W_{121}, W_{122}$etc.
- Here, considering we are using sigmoid function as the activation function

# Forward Propagation

❑**Steps for Forward Propagation in Neural Network:** The steps can be described at once as:

**Step 1: Parameters Initialization**

- We will first initialize the weight matrices and the bias vectors. It's important to note that we shouldn't initialize all the parameters to zero because doing so will lead the gradients to be equal and on each iteration the output would be the same, and the learning algorithm won't learn anything.
- Therefore, it's important to randomly initialize the parameters to a value between 0 and 1. The learning rate will be recommended as 0.01 to make the activation function active.

**Step 2: Activation function**

There is no definitive guide for which activation function works best on specific problems, based on specific requirements activation function must be choosen.

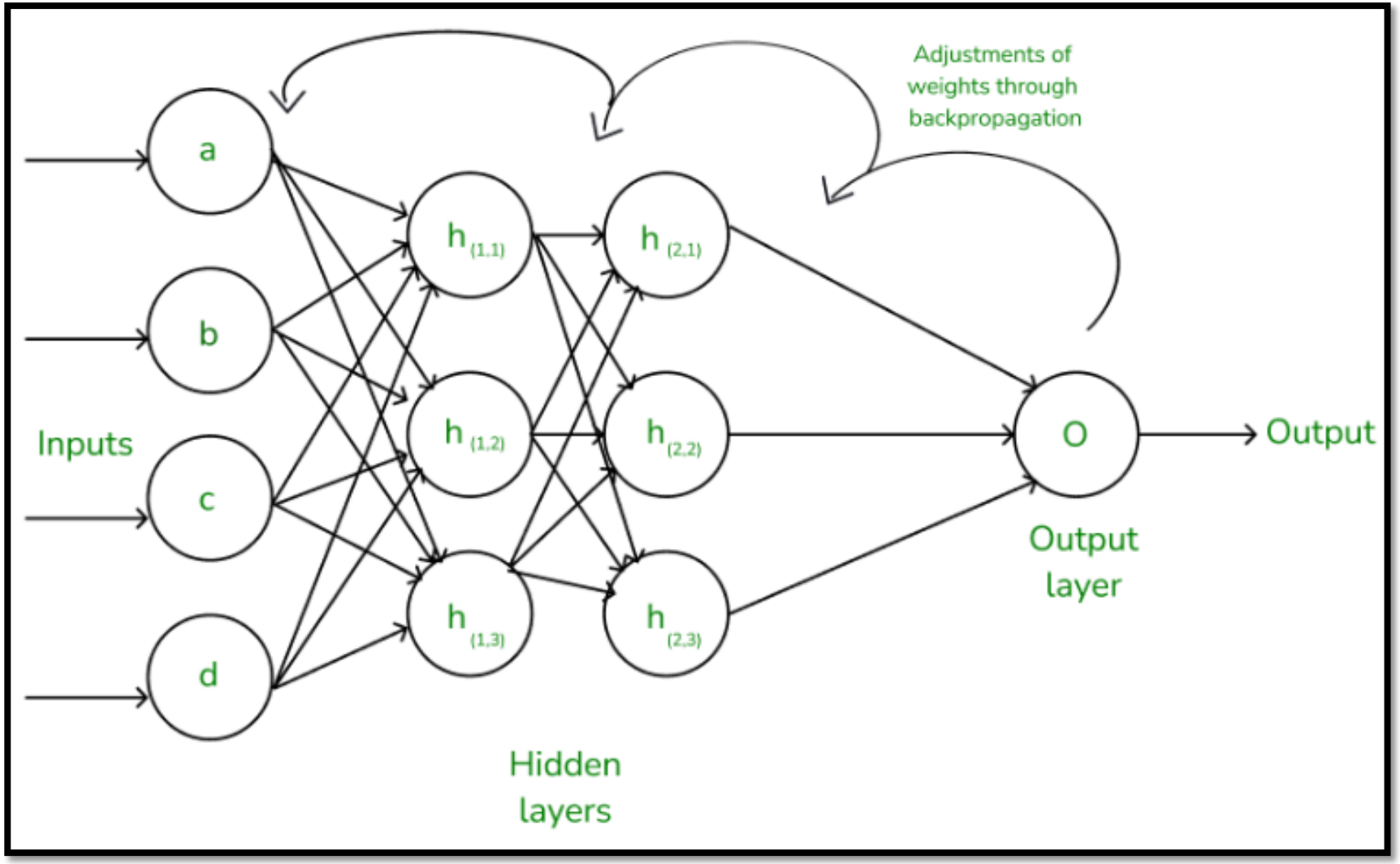
**Step 3: Feeding forward**

Given inputs from the previous layer, each unit computes an affine transformation $z = WTx + b$ and then applies an activation function $g(z)$such as ReLU element-wise. During this process, we will store all the variables computed and used on each layer to be used in backpropagation.

# Back Propagation

- In machine learning, backpropagation is an effective algorithm used to train artificial neural networks, especially in feed-forward neural networks.
- Backpropagation is an iterative algorithm, that helps to minimize the cost function by determining which weights and biases should be adjusted.
- During every epoch, the model learns by adapting the weights and biases to minimize the loss by moving down toward the gradient of the error.
- Thus, it involves the two most popular optimization algorithms, such as gradient descent or stochastic gradient descent.
- Computing the gradient in the backpropagation algorithm helps to minimize the cost function and it can be implemented by using the mathematical rule called chain rule from calculus to navigate through complex layers of the neural network.

# Back Propagation



# Back Propagation

❑**Working of Backpropagation Algorithm:** The Backpropagation algorithm works by two different passes, they are:

1. Forward pass
2. Backward pass

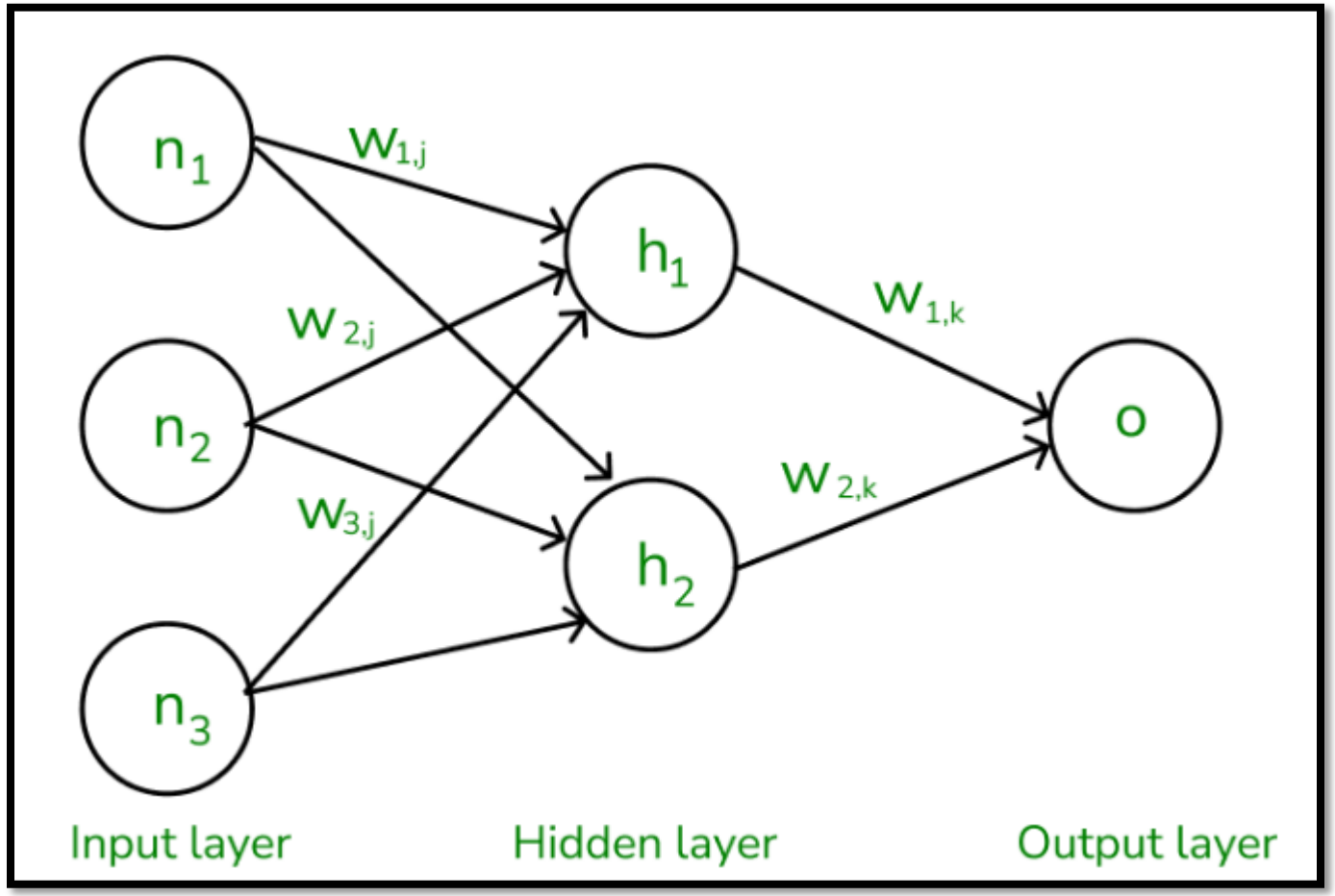
# Back Propagation

## ❑ Forward Pass

1. In forward pass, initially the input is fed into the input layer. Since the inputs are raw data, they can be used for training our neural network.
2. The inputs and their corresponding weights are passed to the hidden layer. The hidden layer performs the computation on the data it receives. If there are two hidden layers in the neural network, for instance, consider the illustration fig(a), $h_1$ and $h_2$ are the two hidden layers, and the output of $h_1$ can be used as an input of $h_2$. Before applying it to the activation function, the bias is added.
3. To the weighted sum of inputs, the activation function is applied in the hidden layer to each of its neurons. One such activation function that is commonly used is ReLU can also be used, which is responsible for returning the input if it is positive otherwise it returns zero. By doing this so, it introduces the non-linearity to our model, which enables the network to learn the complex relationships in the data. And finally, the weighted outputs from the last hidden layer are fed into the output to compute the final prediction, this layer can also use the activation function called the softmax function which is responsible for converting the weighted outputs into probabilities for each class.

# Back Propagation

## ❑ Forward Pass



# Back Propagation

## ❑ Backward Pass

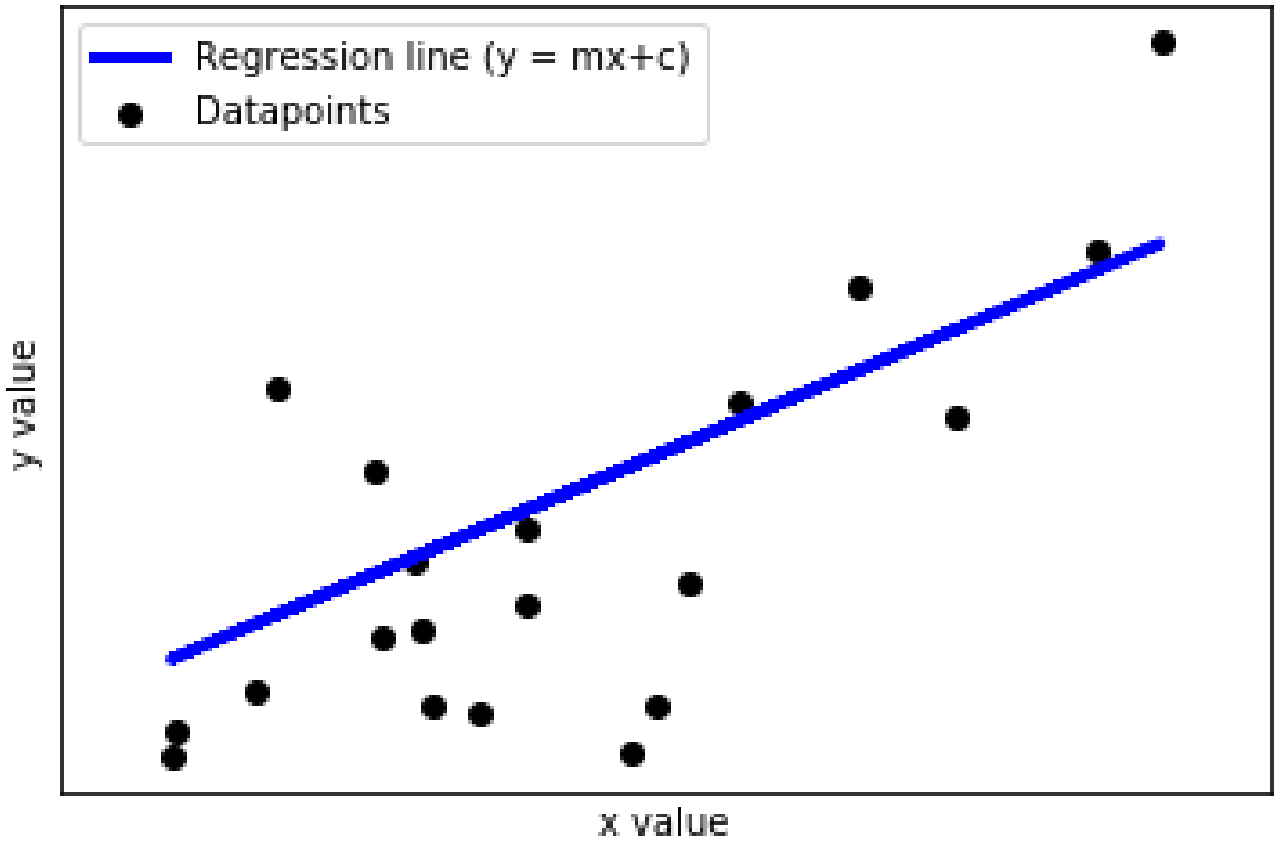
1. In the backward pass process shows, the error is transmitted back to the network which helps the network, to improve its performance by learning and adjusting the internal weights.
2. To find the error generated through the process of forward pass, we can use one of the most commonly used methods called mean squared error which calculates the difference between the predicted output and desired output. The formula for mean squared error is:

$$Meansquarederror = (predictedoutput - actualoutput)^2$$

3. Once we have done the calculation at the output layer, we then propagate the error backward through the network, layer by layer.
4. The key calculation during the backward pass is determining the gradients for each weight and bias in the network. This gradient is responsible for telling us how much each weight/bias should be adjusted to minimize the error in the next forward pass. The chain rule is used iteratively to calculate this gradient efficiently.
5. In addition to gradient calculation, the activation function also plays a crucial role in backpropagation, it works by calculating the gradients with the help of the derivative of the activation function.

# Parameters and Hyperparameters

❑**Parameters:** A model parameter is a variable of the selected model which can be estimated by fitting the given data to the model.



# Parameters and Hyperparameters

❑**Parameters:** In the above plot, x is the independent variable, and y is the dependent variable.

❑The objective is to fit a regression line to the data. This line(the model) is then used to predict the y-value for unseen values of x.

❑Here, m is the slope and c is the intercept of the line.

❑These two parameters(m and c) are estimated by fitting a straight line to the data by minimizing the RMSE(root mean squared error).

❑Hence, these parameters are called the model parameters.

❑**Model parameters in different models:**

1. m(slope) and c(intercept) in Linear Regression
2. weights and biases in Neural Networks

# Parameters and Hyperparameters

## ❑Parameters:

❑Definition: Model parameters are internal configuration variables that the model learns from historical training data. They are essential for making predictions.

❑Estimation: These values are automatically estimated from the data during training.

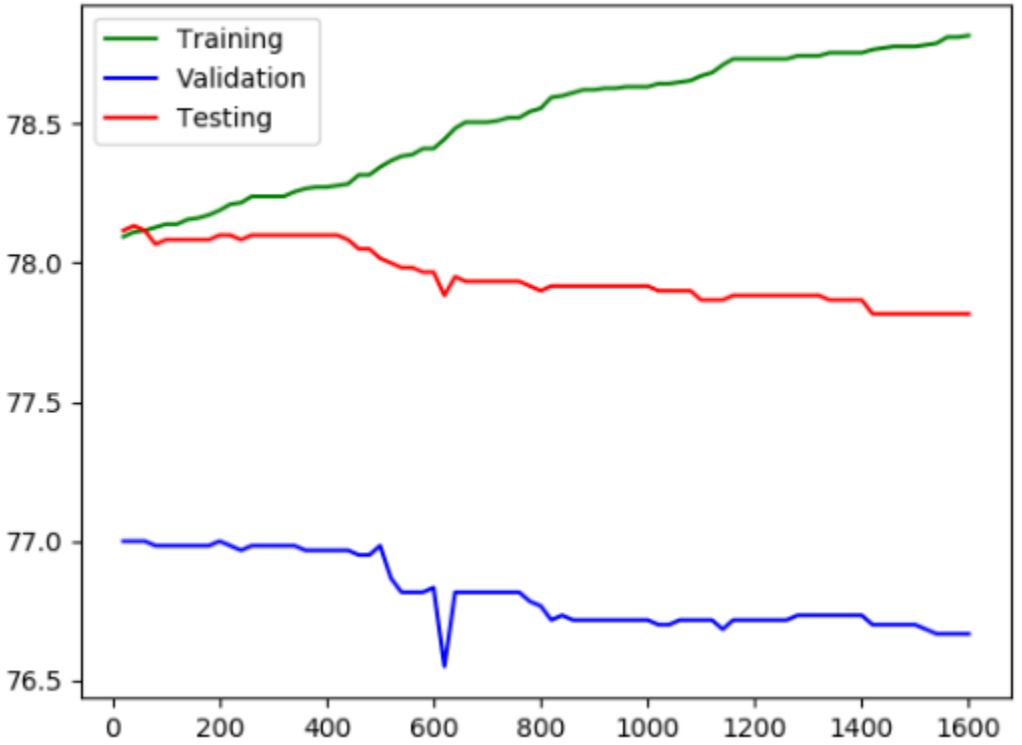
❑Examples:

- In an artificial neural network, the weights connecting neurons are model parameters.
- For linear regression, the coefficients (slope and intercept) are model parameters.

# Parameters and Hyperparameters

## ❑Hyperparameters:

❑A model hyperparameter is the parameter whose value is set before the model start training. They cannot be learned by fitting the model to the data.



# Parameters and Hyperparameters

## ❑Hyperparameters:

❑In the above plot the x-axis represents the number of epochs and the y-axis represents the training loss..

❑We can see after a certain point when epochs are more than then although the training accuracy increases but the validation and test accuracy starts decreasing.

❑This is a case of overfitting.

❑Here number of epochs is a hyperparameter and is set manually.

❑Setting this number to a small value may cause underfitting and high value may cause overfitting.

## ❑Model hyperparameters in different models:

1. Learning rate in gradient descent
2. Number of iterations in gradient descent
3. Number of layers in a Neural Network
4. Number of neurons per layer in a Neural Network
5. Number of clusters(k) in k means clustering
6. Activation functions (e.g., ReLU, sigmoid) used in the model.

# Parameters and Hyperparameters

❑**Hyperparameters:**

❑Definition: Model hyperparametersare external configuration settings that cannot be estimated from data. They are specified manually by the practitioner.

❑Role: Hyperparametersinfluence the learning process and help estimate model parameters.

❑Examples:

- Learning rate: Controls the step size during gradient descent optimization.
- Number of hidden layers and nodes in a neural network.
- Activation functions (e.g., ReLU, sigmoid) used in the model.

# Parameters and Hyperparameters

| PARAMETERS | HYPERPARAMETER |
| --- | --- |
| They are required for making predictions | They are required for estimating the model parameters |
| They are estimated by optimization algorithms(Gradient Descent, Adam, Adagrad) | They are estimated by hyperparameter tuning |
| They are not set manually | They are set manually |
| The final parameters found after training will decide how the model will perform on unseen data | The choice of hyperparameters decide how efficient the training is. In gradient descent the learning rate decide how efficient and accurate the optimization process is in estimating the parameters |

# Note for Students

- This power point presentation is for lecture, therefore it is suggested that also utilize the text books and lecture notes.