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

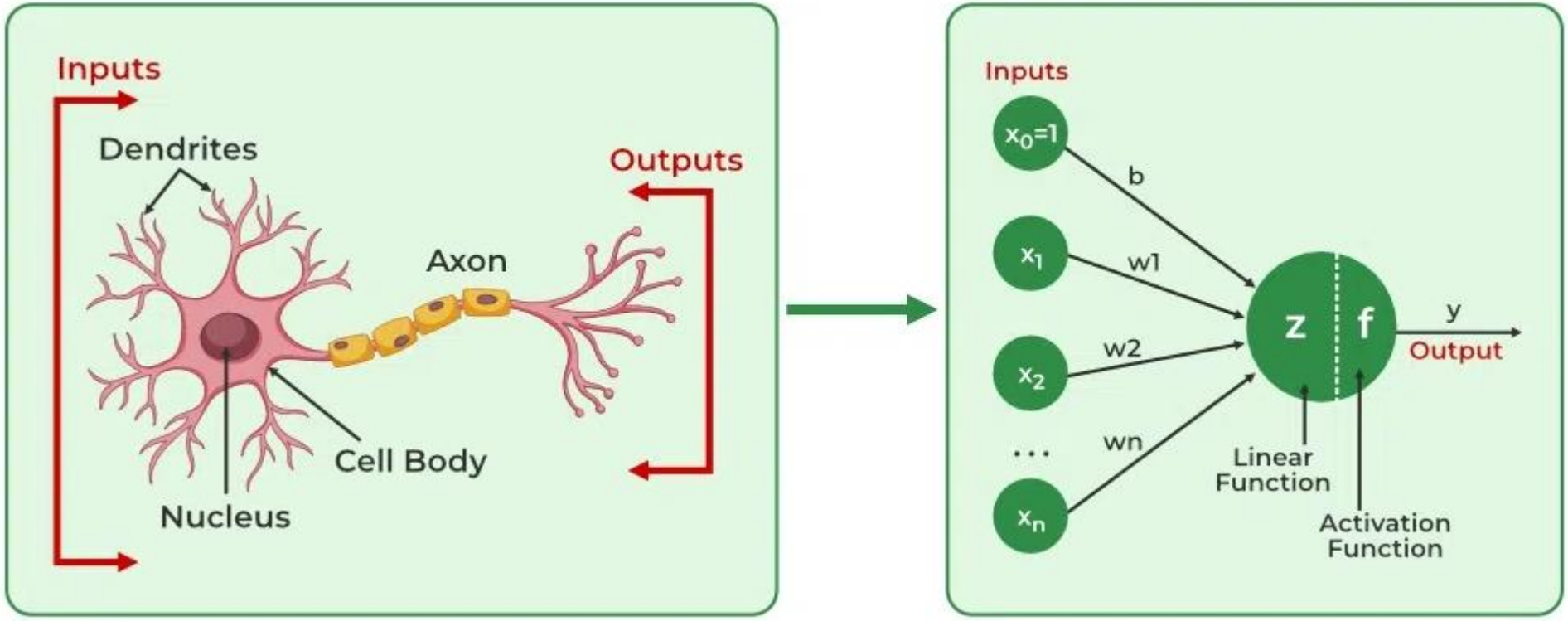
- Neural Networks Basics
- Functions in Neural Networks:
- Activation Functions
- Loss Functions
- Function Approximation
- Classification and Clustering problems
- Deep Networks basics
- Shallow Neural Networks
- Gradient Descent
- Back Propagation
- Deep Neural Networks
- Forward and Back Propagation
- 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.

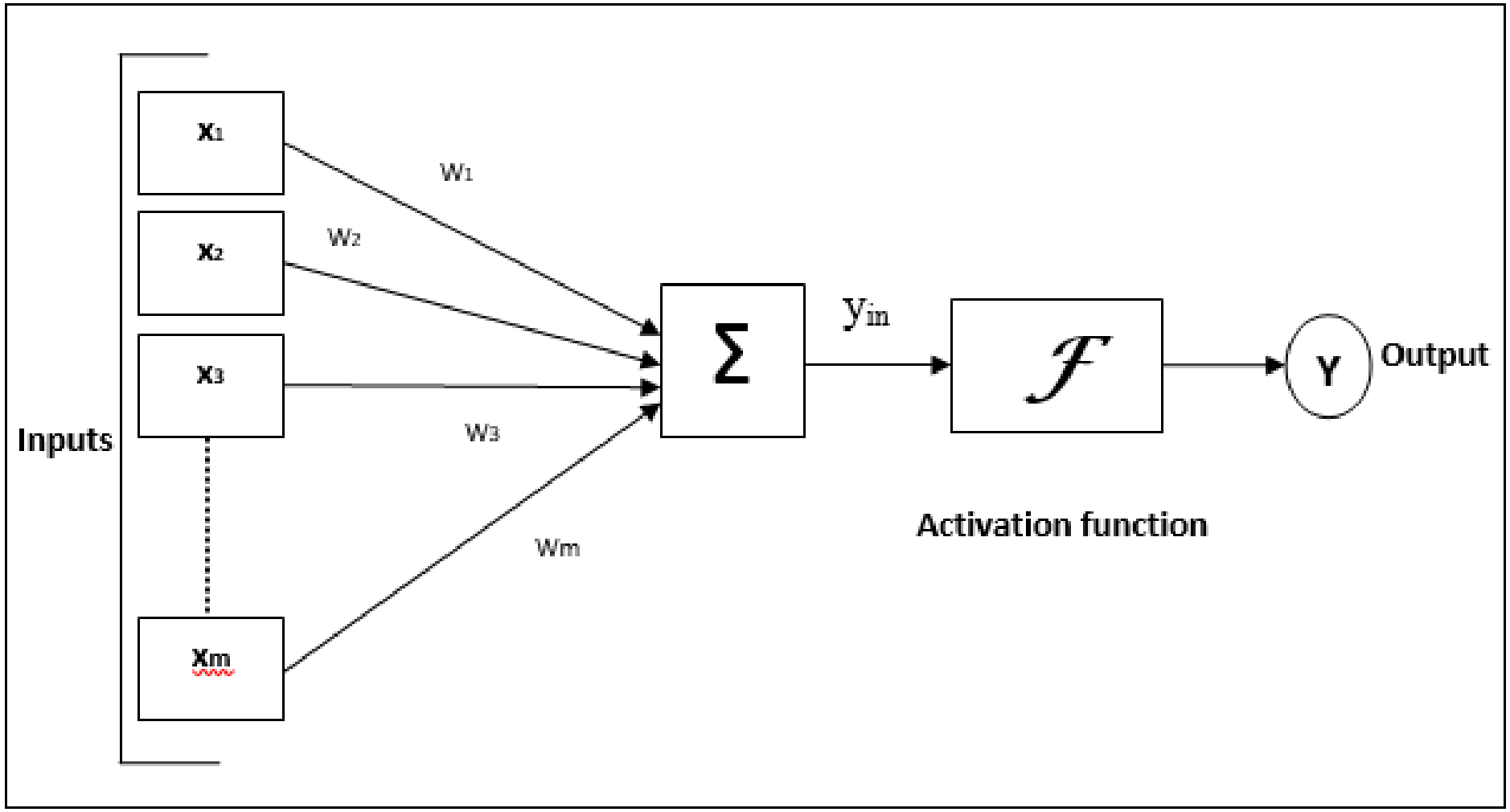
Neural Networks Basics

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 $netinputcalculated$

Neural Networks Basics

❑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.

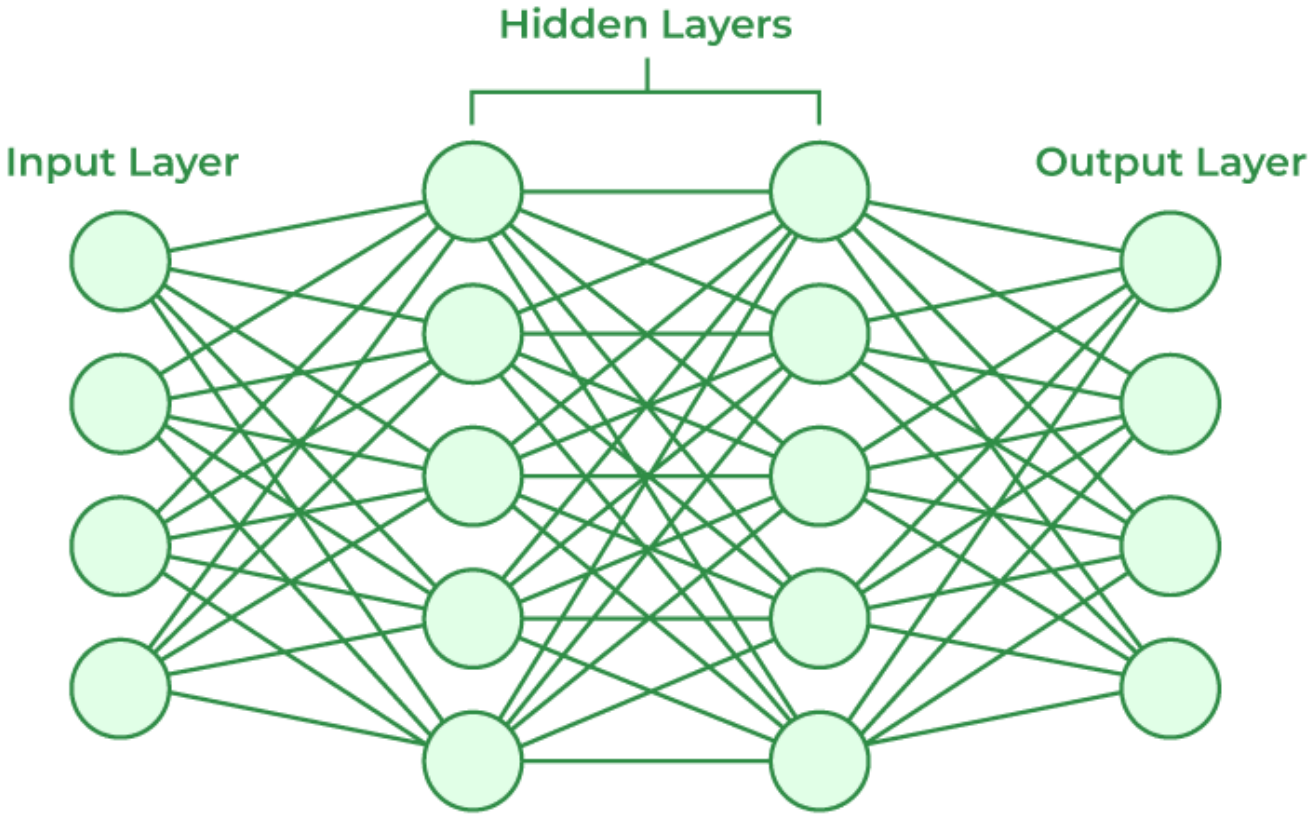
Neural Networks Basics

❑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.

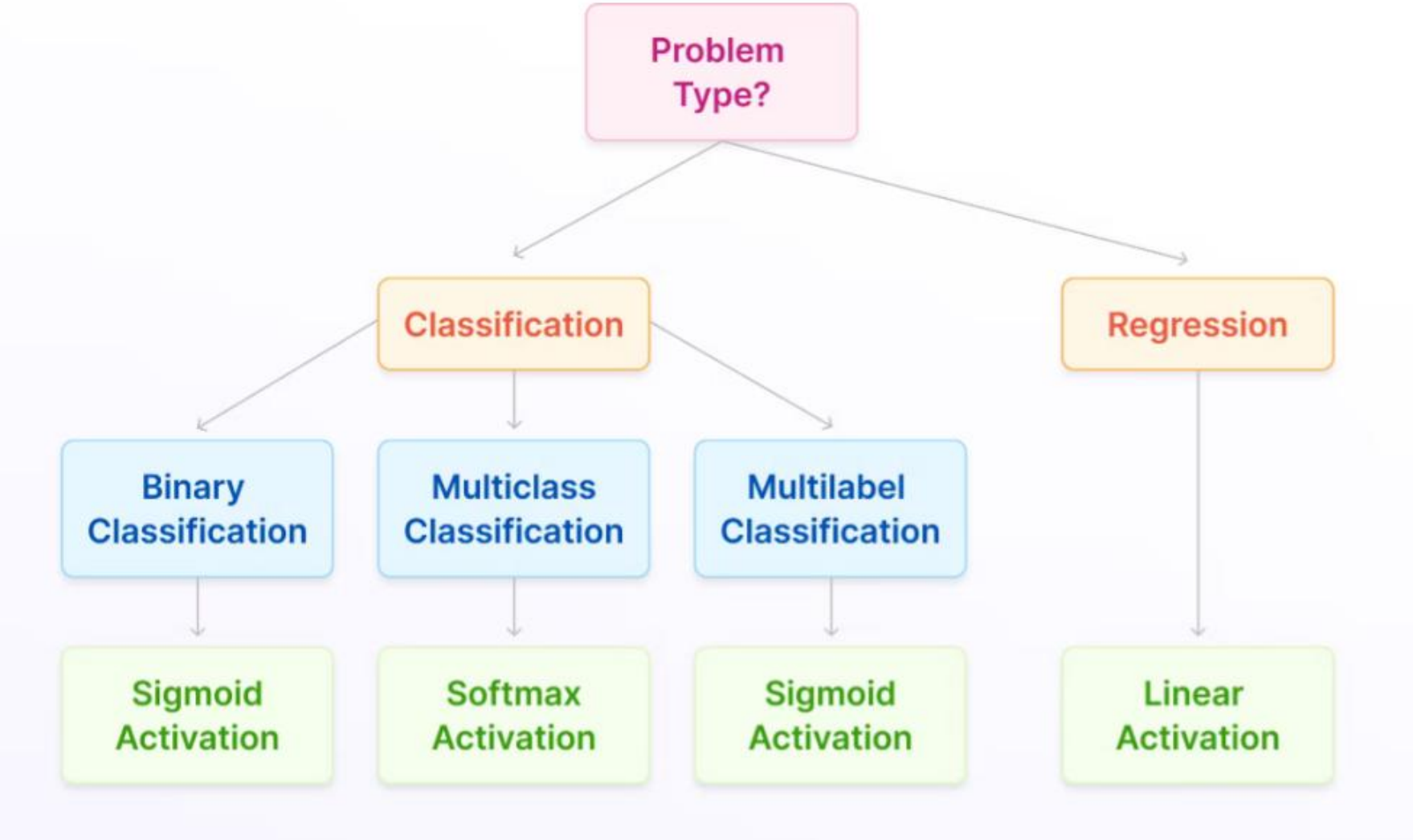
Functions in Neural Networks: Activation Function

- 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.

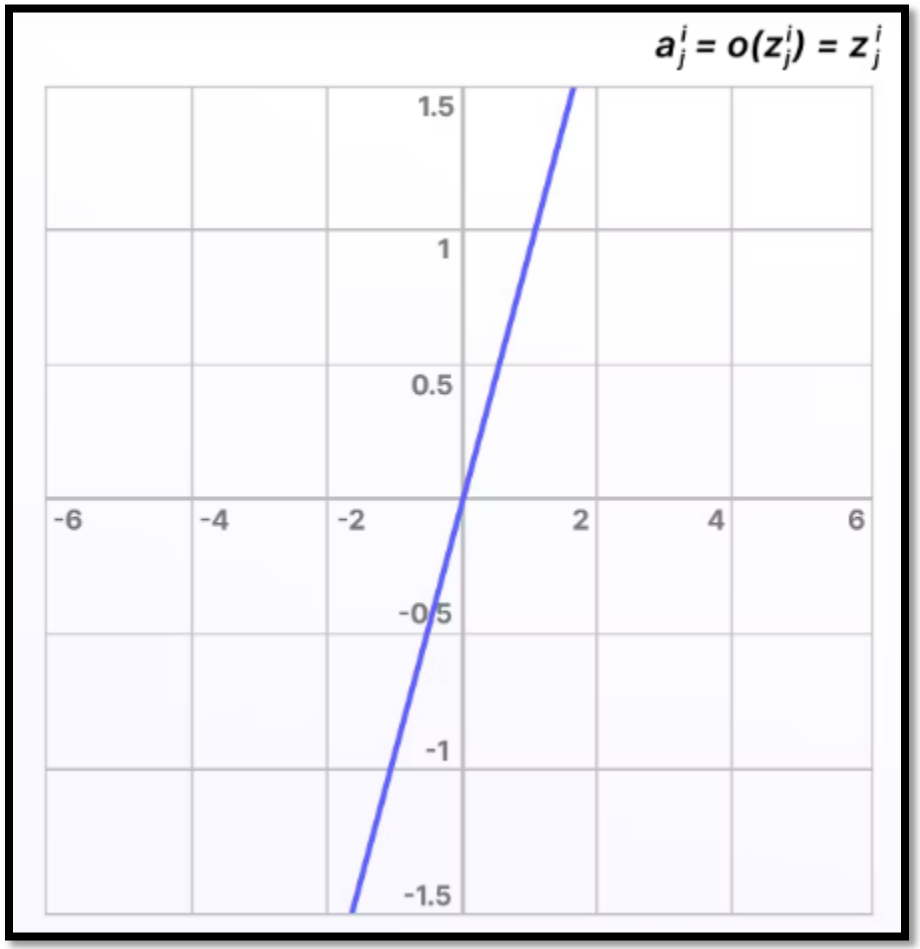
❑Although, in most cases, this might not sound very useful, it is when you want the outputs of your neural network to be continuous rather than modified or discrete. There is no convergence of data, and nothing decreases either.

❑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.

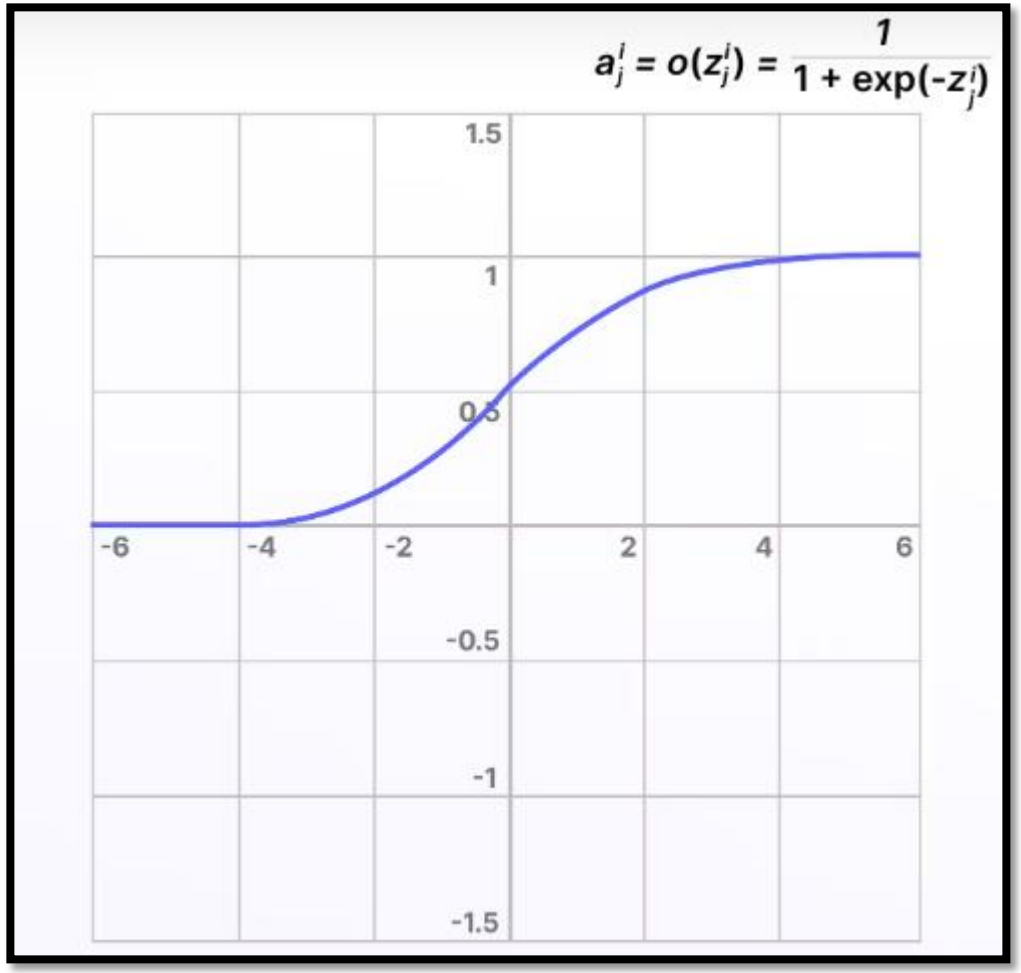
❑Larger, more positive inputs should produce output values close to 1.0, with smaller, more negative inputs producing outputs closer to 0.0.

❑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)

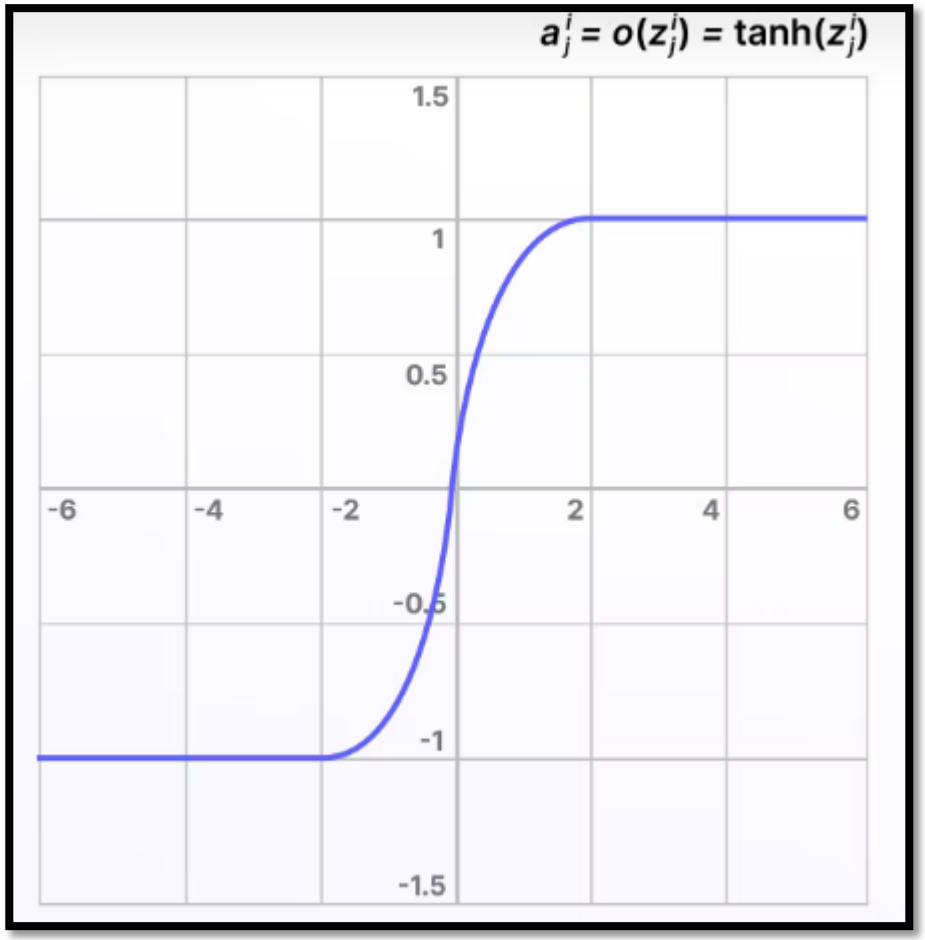
❑Tanh (or TanH), also known as the hyperbolic tangent activation function, is similar to sigmoid/logistic, even down to the S shape curve, and it is differentiable.

❑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.

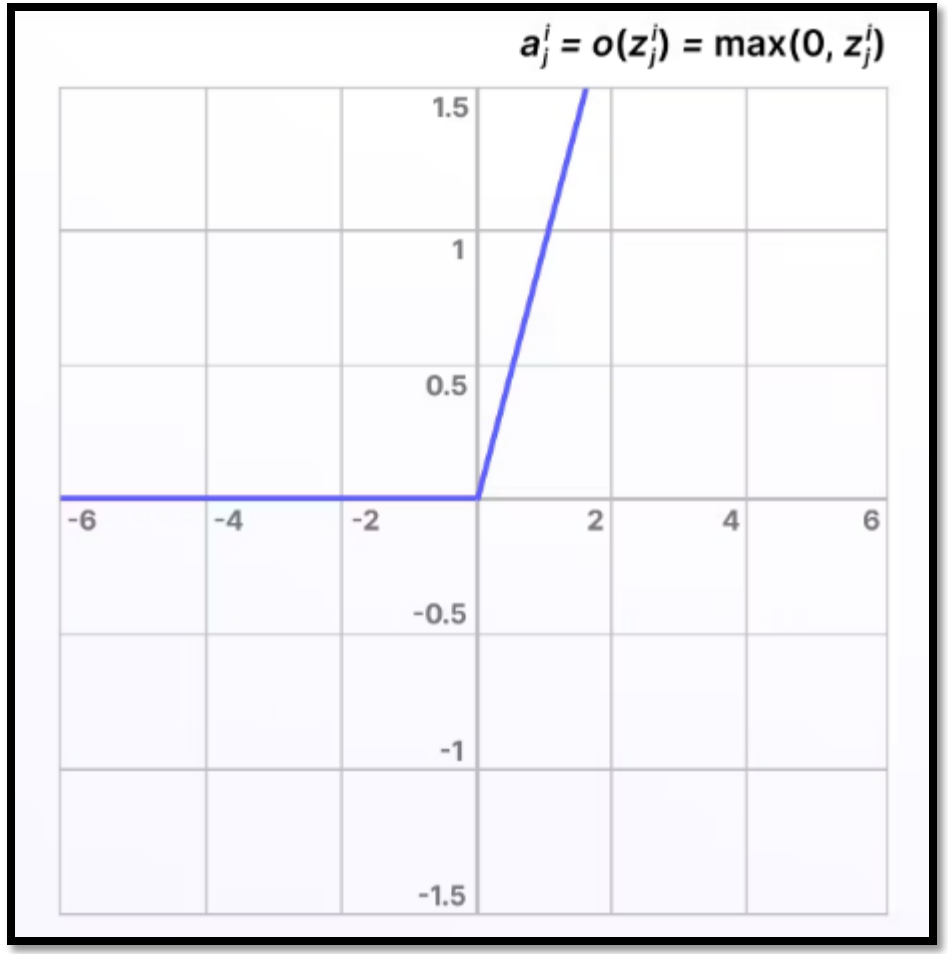
❑ReLU doesn't activate every neuron in sequence at the same time, making it more efficient than the tanh or sigmoid/logistic activation functions.

❑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 α (Normally, α=0.01).

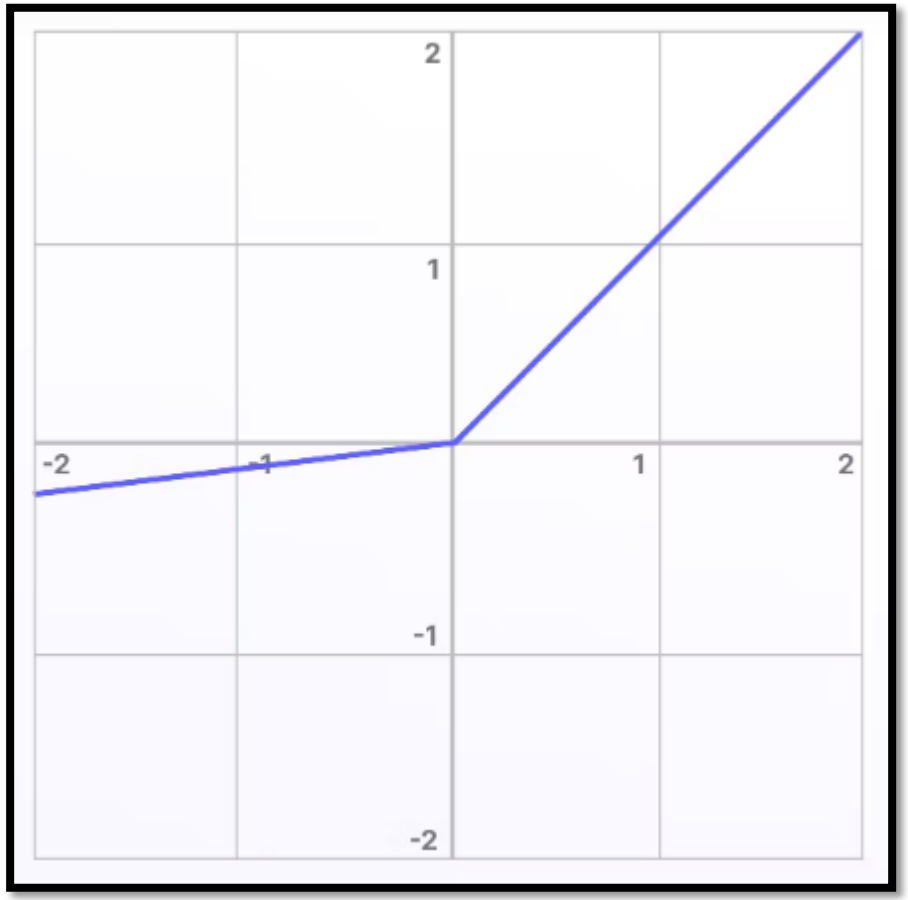
❑Leaky ReLU has been shown to perform better than the traditional ReLU activation function.

❑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

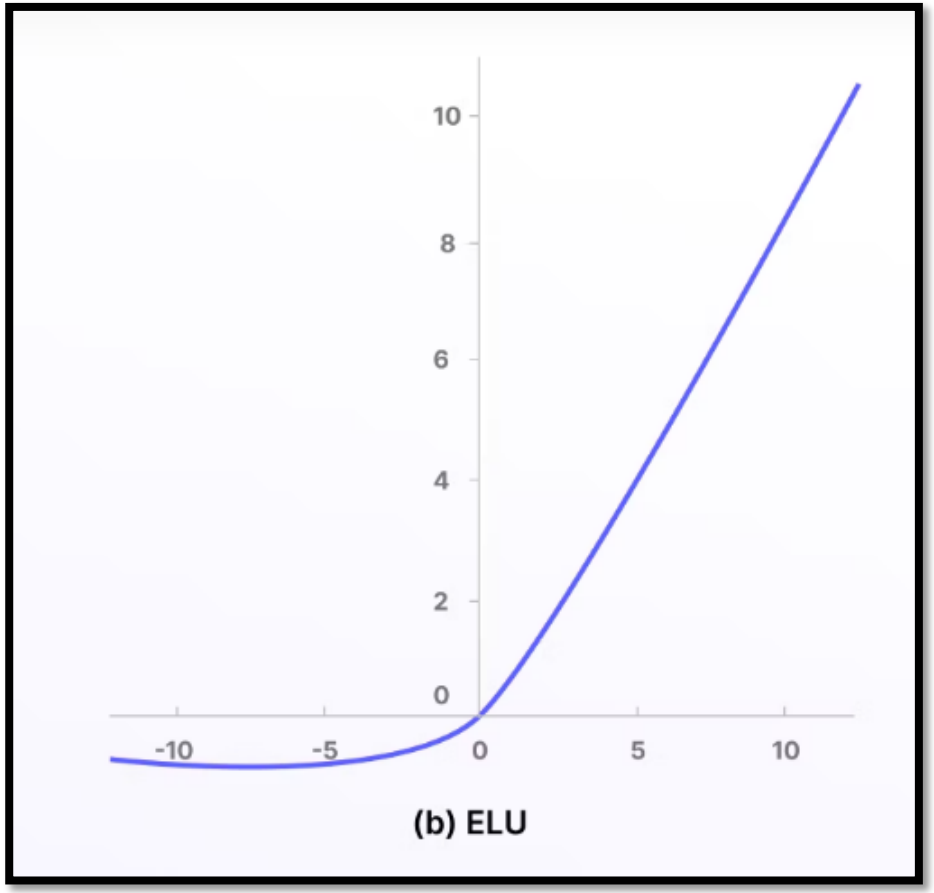
❑The exponential linear units (ELUs) function is another iteration on the original ReLU, another way to overcome the “dying ReLU” problem, and it’s also not differentiable.

❑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.

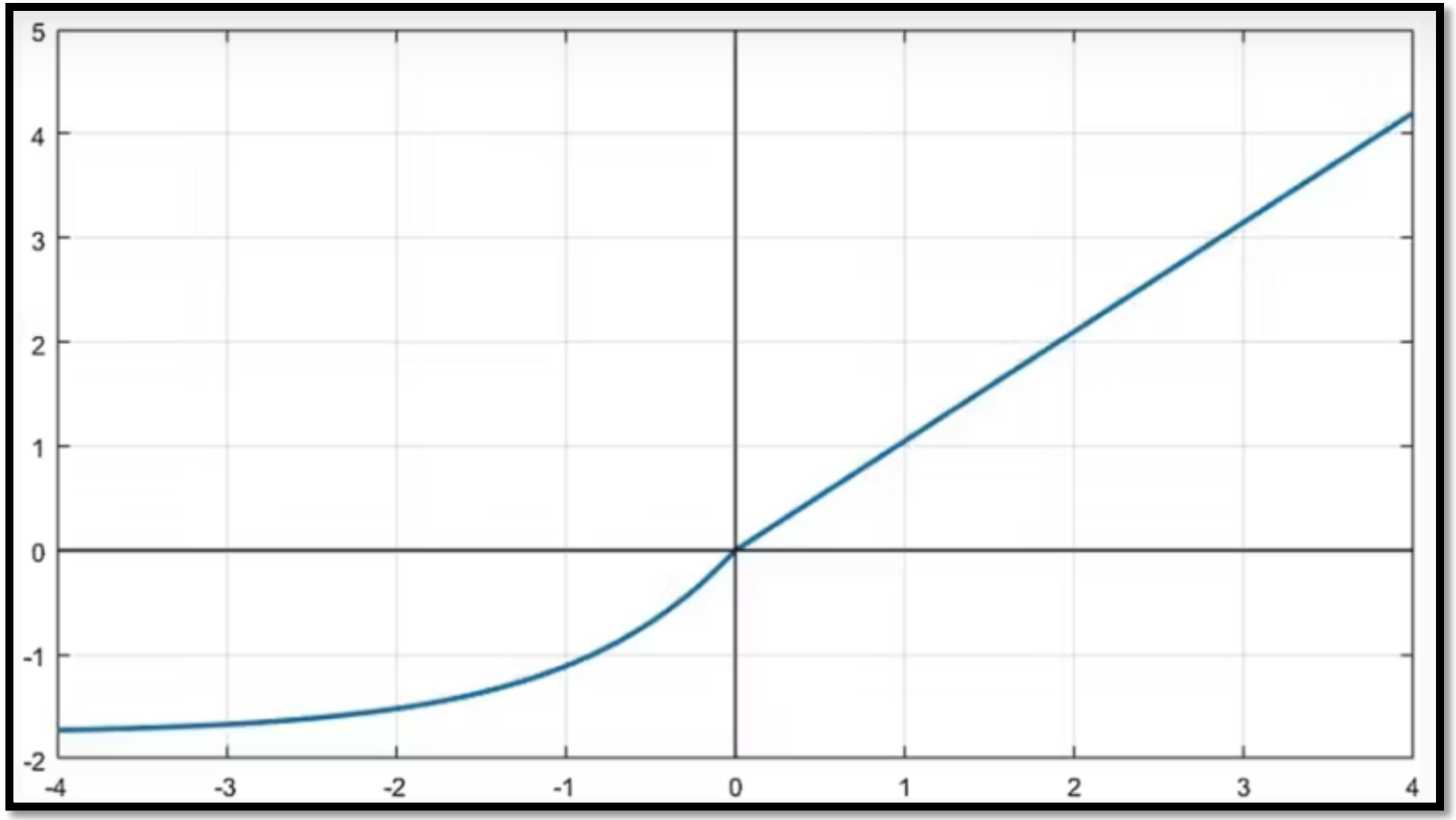
❑**SELUs** control the gradient more effectively and scale the normalization concept, and that is scales with a lambda parameter.

❑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.

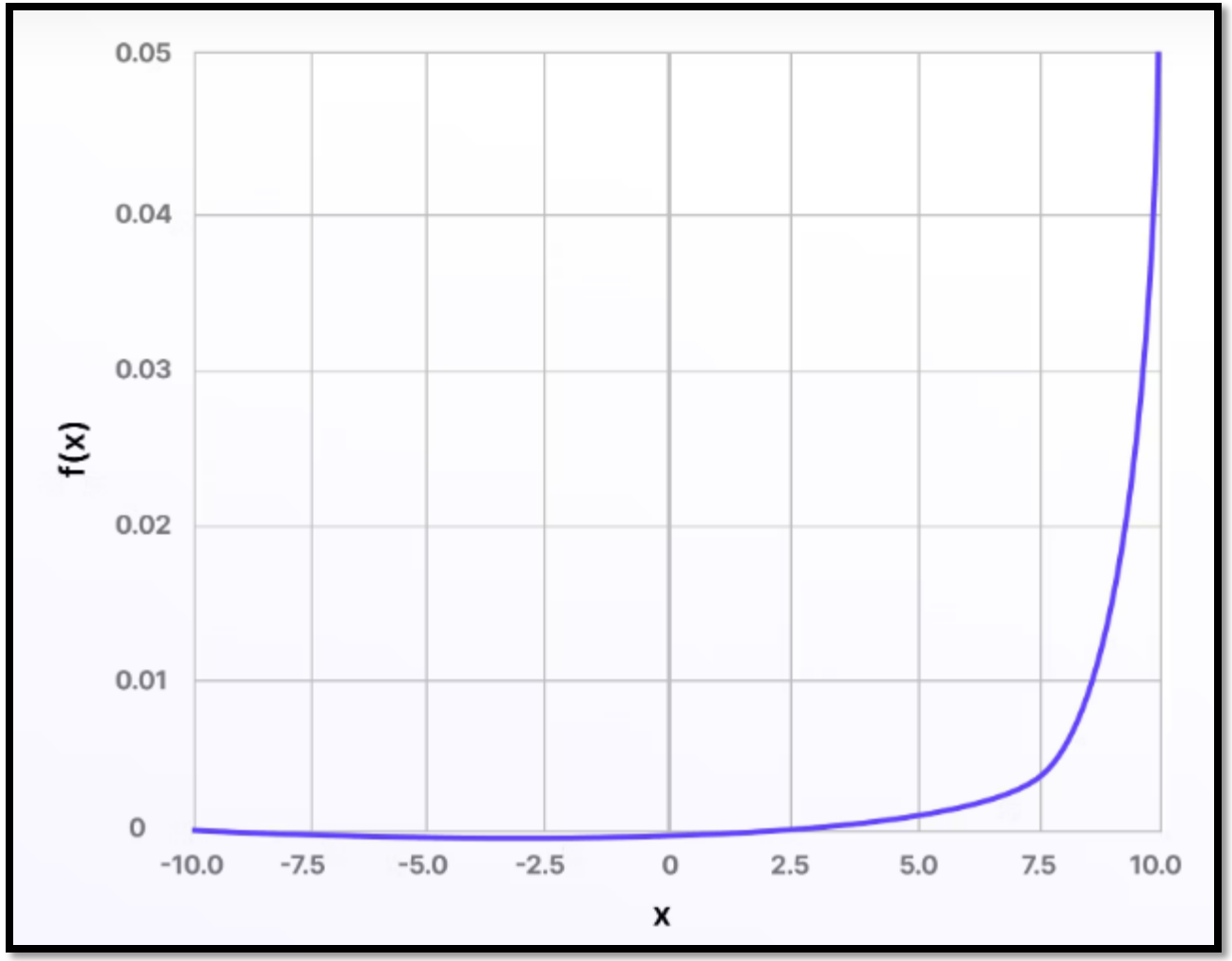
❑Softmax turns input values that are positive, negative, zero, or greater than one into values between 0 and 1.

❑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.

Note for Students

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