A MORE DETAILED RESPONSE ON EACH

1. WHAT ARE NEURAL NETWORKS

Neural networks are a class of machine learning algorithms inspired by the structure and functioning of the human brain. They are used to model complex relationships and patterns in data. Neural networks consist of interconnected units, called neurons, which are organized in layers. These networks are capable of learning and making predictions from data through a process of training.

The basic building block of a neural network is a neuron, which takes inputs, processes them using weights, and produces an output. Neurons are organized in layers: an input layer, one or more hidden layers, and an output layer. The connections between neurons have associated weights that are adjusted during the training process to optimize the network's performance.

Neural networks are used for various tasks such as image and speech recognition, natural language processing, recommendation systems, and more. They excel at capturing complex and nonlinear relationships in data that may be difficult for traditional algorithms to model.

The process of training a neural network involves iteratively presenting the network with input data and adjusting its weights based on the differences between predicted outputs and actual targets. This is achieved using optimization techniques like gradient descent and backpropagation, where the network learns to minimize a cost function that quantifies the difference between predictions and targets.

In recent years, deep neural networks, also known as deep learning, have gained significant attention. These networks have multiple hidden layers and are capable of learning intricate patterns in large and complex datasets. The success of deep learning has revolutionized fields like computer vision, natural language processing, and artificial intelligence.

1. WHAT ARE ANN?

Artificial Neural Networks (ANNs) are a specific type of neural network model designed to mimic the behavior of the human brain and its interconnected neurons. ANNs are used in machine learning and deep learning to solve complex problems by learning patterns and relationships from data. They consist of layers of interconnected artificial neurons that process input data and produce output predictions.

Key characteristics of artificial neural networks:

1. Layers: ANNs consist of multiple layers, including an input layer, one or more hidden layers, and an output layer. Each layer contains a set of artificial neurons.

2. Neurons: Neurons in ANNs are analogous to neurons in the human brain. They receive input from the previous layer, calculate a weighted sum of inputs, and apply an activation function to produce an output.

3. Weights and Biases: The connections between neurons are represented by weights. Each connection has an associated weight that determines the strength of the connection. Biases are additional values added to the weighted sum before applying the activation function.

4. Activation Functions: Activation functions introduce nonlinearity to the network, allowing it to learn complex relationships. Common activation functions include sigmoid, tanh, and Rectified Linear Unit (ReLU).

5. Training: ANNs learn from data through a training process. During training, the network adjusts its weights and biases to minimize a cost function that measures the difference between predicted outputs and actual targets.

6. Backpropagation: Backpropagation is a fundamental algorithm used to train ANNs. It involves calculating the gradients of the cost function with respect to the network's weights and biases and then updating these parameters to minimize the cost.

7. Deep Learning: Deep learning refers to the use of deep artificial neural networks, which have multiple hidden layers. These deep networks are capable of learning complex and hierarchical patterns in data.

8. Applications: ANNs are used for various tasks such as image recognition, speech recognition, natural language processing, autonomous vehicles, recommendation systems, and more.

9. Architecture Variations: There are different types of ANN architectures, including feedforward neural networks, recurrent neural networks (RNNs), convolutional neural networks (CNNs), and more. Each architecture is designed to handle specific types of data and tasks.

1. WHAT ARE THE TYPES OF ARTIFICIAL NEURAL NETWORKS WE HAVE?

There are several types of artificial neural network (ANN) architectures, each designed to handle specific types of data and tasks. Here are some of the most common types:

1. Feedforward Neural Networks (FNNs) : These are the simplest type of neural networks, where data flows in one direction, from the input layer through hidden layers to the output layer. They are used for tasks like regression and classification.

2. Multilayer Perceptrons (MLPs) : MLPs are a type of feedforward neural network with one or more hidden layers. They can learn complex patterns and are widely used for various tasks, including image and text classification.

3. Convolutional Neural Networks (CNNs) : CNNs are designed for processing grid-like data, such as images. They use convolutional layers to automatically detect features in images and are particularly effective in computer vision tasks like image recognition and object detection.

4. Recurrent Neural Networks (RNNs) : RNNs are designed for sequential data, where the order of input matters. They have loops that allow information to be passed from one step to the next, making them suitable for tasks like natural language processing, speech recognition, and time series analysis.

5. Long Short-Term Memory Networks (LSTMs) : LSTMs are a type of RNN that addresses the vanishing gradient problem and is capable of learning longer-term dependencies in sequential data. They are commonly used in tasks requiring memory of past inputs, like language modeling and machine translation.

6. Gated Recurrent Units (GRUs) : GRUs are another variant of RNNs that address the vanishing gradient problem. They have a simpler architecture than LSTMs and are suitable for tasks similar to those of LSTMs.

7. Autoencoders : Autoencoders are used for unsupervised learning and dimensionality reduction. They consist of an encoder that compresses input data into a lower-dimensional representation and a decoder that reconstructs the original data from the compressed representation.

8. Generative Adversarial Networks (GANs) : GANs consist of two neural networks, a generator and a discriminator, that work together in a game-theoretic framework to generate realistic data. GANs are commonly used for generating images, videos, and other types of creative content.

9. Self-Organizing Maps (SOMs) : SOMs are used for clustering and visualization of high-dimensional data. They map input data onto a lower-dimensional grid while preserving certain topological properties of the input space.

10. Transformers : Transformers are a type of architecture that has gained significant attention in natural language processing tasks. They use self-attention mechanisms to process input data in parallel, making them highly efficient for capturing contextual relationships in sequential data.

11. Capsule Networks : Capsule networks are designed to capture hierarchical relationships among features in an image. They offer potential improvements in handling variations in pose and appearance compared to traditional CNNs.

1. WHAT IS FORWARD PROPAGATION

Forward propagation is the process by which input data is passed through a neural network to generate predictions or outputs. It involves calculating the outputs of each neuron in the network, layer by layer, until the final output is obtained. Forward propagation is the first step in the overall process of using a neural network to make predictions.

Here's how forward propagation works:

1. Input Layer: The input data is fed into the input layer of the neural network. Each neuron in the input layer represents a feature or input value.

2. Weighted Sum and Activation: For each neuron in the subsequent hidden layers and the output layer, a weighted sum of its inputs is calculated. Each input is multiplied by its corresponding weight, and the bias term (if present) is added. This weighted sum is then passed through an activation function to produce the neuron's output.

3. Passing Through Hidden Layers: The calculated outputs from the previous layer serve as inputs to the next layer. This process continues through the hidden layers until the output layer is reached.

4. Final Output: The final output of the neural network is obtained from the output neurons. The output can be a single value for regression tasks or a set of probabilities for classification tasks.

5. Predictions: The obtained output represents the neural network's prediction based on the input data. The network has "learned" from its training data, adjusting its weights during the training process, to make accurate predictions.

Mathematically, forward propagation can be represented as follows:

For each neuron in layer l:

- Calculate the weighted sum of inputs: `z = w \* x + b`

- Apply an activation function: `a = activation\_function(z)`

Where:

- `w` represents the weights associated with the neuron's inputs.

- `x` represents the inputs to the neuron.

- `b` is the bias term.

- `z` is the weighted sum of inputs.

- `a` is the output after applying the activation function.

Forward propagation is a crucial step in training and inference for neural networks. It enables the network to transform input data into meaningful predictions by passing it through the layers and capturing complex patterns and relationships in the data.

1. WHAT IS BACKWARD PROPAGATION?

Backward propagation, also known as backpropagation, is the process used to train neural networks by updating the network's weights and biases based on the calculated gradients of the loss function with respect to these parameters. It's the mechanism through which a neural network learns from its mistakes and adjusts its parameters to improve its predictions.

The goal of backward propagation is to minimize the difference between the network's predicted outputs and the actual target values. This process involves calculating the gradient of the loss function with respect to the network's parameters using the chain rule of calculus, and then using these gradients to update the parameters in the opposite direction of the gradient's ascent, which minimizes the loss.

Here's how backward propagation works:

1. Forward Propagation: The input data is passed through the network's layers using forward propagation to generate predictions.

2. Calculate Loss: The loss function, which quantifies the difference between predicted outputs and actual targets, is computed based on the predictions.

3. Calculate Gradients: The gradient of the loss function with respect to each parameter (weight and bias) in the network is calculated using the chain rule. This gradient represents the direction and magnitude of the change needed to minimize the loss.

4. Update Parameters: The weights and biases of the network are updated using an optimization algorithm (e.g., gradient descent) in the opposite direction of the gradients calculated in step 3. The learning rate determines the size of each update.

5. Repeat: Steps 1 to 4 are repeated for multiple iterations or epochs until the network's performance improves and the loss is minimized.

Backward propagation is critical for training deep neural networks. It allows the network to adjust its internal parameters to better capture the underlying patterns in the training data. The process of calculating gradients and updating parameters iteratively drives the network towards better predictions and improved generalization to unseen data.

It's important to note that while forward propagation computes predictions, backward propagation computes gradients and updates parameters. These two processes work together in the training phase to fine-tune the network's weights and biases for optimal performance.

1. WHAT ARE THE TYPES OF COST FUNCTIONS WE USE IN DEEP LEARNING AND UNDER WHAT SCENARIOS ARE THEY APPLIED

Cost functions, also known as loss functions or objective functions, are used to measure the difference between the predicted outputs of a neural network and the actual target values. Different types of cost functions are used based on the nature of the task at hand, whether it's a regression or classification problem, and the specific characteristics of the data. Here are some common types of cost functions and the scenarios in which they are applied:

1. Mean Squared Error (MSE):

- Formula: MSE = (1/n) \* Σ(y\_pred - y\_actual)^2

- Scenario: Used for regression tasks, where the goal is to predict continuous numerical values. It penalizes large prediction errors more heavily.

2. Mean Absolute Error (MAE):

- Formula: MAE = (1/n) \* Σ|y\_pred - y\_actual|

- Scenario: Similar to MSE, used for regression tasks, but it penalizes errors linearly instead of quadratically.

3. Binary Cross-Entropy (Log Loss):

- Formula: BCE = - (y\_actual \* log(y\_pred) + (1 - y\_actual) \* log(1 - y\_pred))

- Scenario: Used for binary classification tasks, where there are two classes (0 and 1). Suitable when dealing with probabilities.

4. Categorical Cross-Entropy:

- Formula: CCE = - Σ(y\_actual \* log(y\_pred))

- Scenario: Used for multi-class classification tasks, where there are more than two classes. It extends binary cross-entropy to multiple classes.

5. Sparse Categorical Cross-Entropy:

- Formula: Similar to categorical cross-entropy, but the targets are provided as integer indices rather than one-hot encoded vectors.

- Scenario: Used when the target labels are integers representing class indices.

6. Hinge Loss (SVM Loss):

- Formula: Hinge Loss = max(0, 1 - y\_actual \* y\_pred)

- Scenario: Used in support vector machines (SVMs) and for binary classification tasks with margin-based optimization objectives.

7. Huber Loss:

- Formula: Huber Loss = 0.5 \* (y\_pred - y\_actual)^2, if |y\_pred - y\_actual| <= δ; δ \* |y\_pred - y\_actual| - 0.5 \* δ^2, otherwise

- Scenario: A combination of MAE and MSE, used for regression tasks when you want a balance between handling outliers and having quadratic penalties.

8. Custom Loss Functions:

- Scenario: In some cases, custom loss functions are designed based on the specific requirements of the problem. For example, in tasks like object detection or sequence-to-sequence modeling, specialized loss functions might be used.

The choice of the cost function depends on the nature of the task, the type of data, and the desired characteristics of the model's predictions. The primary goal is to select a cost function that guides the training process toward minimizing the prediction error and improving the model's performance on the specific task.

1. WHAT IS A PERCEPTRON?

A perceptron is a fundamental building block of artificial neural networks, specifically in the context of single-layer neural networks. It is a mathematical model inspired by the way neurons in the human brain work, designed to make simple binary decisions. A perceptron takes input values, applies weights to them, calculates a weighted sum, adds a bias term, and then passes the result through an activation function to produce an output. Here's how a perceptron works:

1. Input Values: A perceptron takes a set of input values, each of which is associated with a weight.

2. Weighted Sum: Each input value is multiplied by its corresponding weight. The weighted inputs are then summed together.

3. Bias Term: A bias term is added to the weighted sum. The bias term allows the perceptron to shift its decision boundary.

4. Activation Function: The result of the weighted sum and bias is passed through an activation function. The activation function introduces nonlinearity into the perceptron's output.

5. Output: The output of the activation function is the perceptron's final output, which can be binary (0 or 1) or continuous, depending on the specific task.

Mathematically, the operation of a perceptron can be described as follows:

Output = Activation\_Function(Weighted\_Sum + Bias)

The activation function in the perceptron is usually a step function that maps the weighted sum to either 0 or 1. For example, the step function might produce 0 if the weighted sum is less than a threshold and 1 otherwise. This makes the perceptron function as a simple binary classifier.

It's important to note that a single perceptron can only solve linearly separable problems, meaning it can only classify data that can be separated by a straight line. However, by combining multiple perceptrons in layers (known as multi-layer perceptrons or neural networks), more complex patterns and relationships in data can be captured, allowing for the modeling of more intricate tasks.

1. HOW DOES A PERCEPTRON APPLY TO DEEP NEURAL NETWORKS?

A perceptron is the basic building block of a deep neural network, which is also referred to as a multi-layer perceptron (MLP). In deep neural networks, perceptrons are used as individual neurons in the network's layers to process input data and contribute to the overall computation of the network's output.

Here's how a perceptron applies to deep neural networks:

1. Neuron in a Layer: In a deep neural network, each neuron or node in a layer represents a perceptron. Each neuron takes input values, applies weights, computes a weighted sum, adds a bias term, and passes the result through an activation function.

2. Input Layer: The input layer of a deep neural network consists of neurons that represent the input features of the data. Each input value is associated with a weight.

3. Hidden Layers: Deep neural networks often have one or more hidden layers composed of perceptrons. These hidden layers enable the network to learn complex patterns and relationships in the data.

4. Output Layer: The output layer of the network produces the final prediction or classification based on the processed information from the hidden layers. Each neuron's output contributes to the overall prediction.

5. Activation Functions: Activation functions introduce nonlinearity to the network's computations. Each perceptron's output is determined by the activation function applied to the weighted sum of inputs.

6. Weights and Biases: The weights and biases associated with the connections between neurons in different layers of the network are learned during the training process. These parameters determine the behavior of the perceptrons and their contribution to the network's predictions.

7. Backward Propagation: During training, backpropagation is used to adjust the weights and biases of the perceptrons based on the calculated gradients of the loss function. This iterative process fine-tunes the network's parameters to improve its predictions.

8. Deep Learning Capabilities: The stacking of multiple layers of perceptrons in deep neural networks allows them to capture complex and hierarchical patterns in data. The intermediate layers progressively learn more abstract features from the input data, leading to improved representation and performance.

In summary, perceptrons are the essential computational units in deep neural networks, and they apply by serving as the basic processing elements in each layer of the network. When arranged in multiple layers, perceptrons enable deep neural networks to model intricate relationships in data, making them powerful tools for a wide range of machine learning tasks.

1. TYPES OF PERCEPTRON

When discussing the types of perceptrons, it's important to clarify that in modern deep learning, the term "perceptron" is often used interchangeably with "neuron" or "node" in neural networks. However, historically, the term "perceptron" referred specifically to a single-layer binary classifier based on a linear threshold unit.

In contemporary deep learning, different types of neural network architectures utilize perceptron-like nodes, each adapted for specific tasks and challenges. Here are some common types:

1. Single-Layer Perceptron: As mentioned, this refers to the historical model of a linear binary classifier. It takes input features, computes the weighted sum, adds a bias, and applies a step function to classify into one of two classes.

2. Multi-Layer Perceptron (MLP): An extension of the single-layer perceptron, MLPs consist of multiple layers of interconnected nodes, including an input layer, one or more hidden layers, and an output layer. Each node (perceptron) in these layers performs the weighted sum and activation function operations, enabling the network to model complex patterns.

3. Convolutional Neural Network (CNN) Neuron: In CNNs, the "neurons" perform convolutions, applying filters to local regions of the input data. These filters detect features like edges, textures, and more complex patterns, enabling CNNs to excel in image and spatial data tasks.

4. Recurrent Neural Network (RNN) Cell: In RNNs, cells (equivalent to nodes) process sequences of data and maintain memory of past inputs. Various types of RNN cells exist, including vanilla RNN, LSTM, and GRU cells, each with unique properties for handling sequential data.

5. Transformers Self-Attention Mechanism: In transformer-based architectures, self-attention mechanisms, often represented as nodes, capture relationships between different words in a sentence or elements in a sequence. These mechanisms allow transformers to excel in natural language processing tasks.

6. Capsule Networks (Capsules): Capsules are inspired by biological neurons and aim to capture richer information about features in images. Capsules represent features using vectors, enabling them to handle variations in pose and appearance more effectively than traditional convolutional layers.

Remember, while these types of nodes or neurons have their unique characteristics, they're all built on the basic principles of processing input, applying weights, and applying activation functions to produce outputs. Their adaptability and combinations in various architectures are what enable neural networks to perform a wide range of tasks.

1. HOW ARE PERCEPTRON IMPLEMENTED?

Implementing a perceptron involves setting up the basic structure of the model, defining the weighted sum and activation function, and providing the training process. Here's a simple example of how to implement a perceptron using Python:

```python

import numpy as np

class Perceptron:

def \_\_init\_\_(self, input\_size, learning\_rate=0.1):

self.weights = np.random.rand(input\_size)

self.bias = np.random.rand()

self.learning\_rate = learning\_rate

def activation\_function(self, x):

# Step function: 1 if x >= 0, 0 otherwise

return 1 if x >= 0 else 0

def predict(self, inputs):

weighted\_sum = np.dot(inputs, self.weights) + self.bias

return self.activation\_function(weighted\_sum)

def train(self, training\_data, labels, epochs):

for \_ in range(epochs):

for inputs, label in zip(training\_data, labels):

prediction = self.predict(inputs)

error = label - prediction

self.weights += self.learning\_rate \* error \* inputs

self.bias += self.learning\_rate \* error

# Example usage

training\_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

labels = np.array([0, 0, 0, 1])

perceptron = Perceptron(input\_size=2)

perceptron.train(training\_data, labels, epochs=100)

# Test predictions

print(perceptron.predict([0, 0])) # Output: 0

print(perceptron.predict([1, 1])) # Output: 1

```

In this example:

- The `Perceptron` class is defined, with methods for initializing the perceptron, predicting outputs, and training the model.

- The `activation\_ function` is a simple step function, where the output is 1 if the input is greater than or equal to 0, and 0 otherwise.

- The `predict` method calculates the weighted sum of inputs, applies the activation function, and returns the prediction.

- The `train` method adjusts the weights and bias based on the error between the prediction and the actual label. This process is repeated for a specified number of epochs.

Keep in mind that this example uses a simple perceptron for a logical OR operation. For more complex tasks, you would need to use more sophisticated architectures like multi-layer perceptrons (MLPs) and implement backpropagation for training. Libraries like TensorFlow or PyTorch provide higher-level abstractions to implement and train neural networks efficiently.

1. EXPLAINING MULTI-LAYER PERCEPTRON

A Multi-Layer Perceptron (MLP) is a type of artificial neural network architecture that consists of multiple layers of interconnected neurons (perceptrons). It's a versatile and widely used neural network model for various machine learning tasks, including classification, regression, and pattern recognition. The key feature that sets MLPs apart from single-layer perceptrons is the presence of hidden layers, which enable the network to capture complex relationships in data.

Here's how a Multi-Layer Perceptron works:

1. Input Layer: The input layer consists of neurons that represent the input features of the data. Each neuron is connected to every neuron in the next layer (hidden layer) through weighted connections.

2. Hidden Layers: MLPs have one or more hidden layers situated between the input and output layers. These hidden layers are responsible for learning and capturing complex patterns in the data. Each neuron in a hidden layer takes the outputs of neurons from the previous layer as inputs.

3. Activation Function: Each neuron in a hidden layer calculates the weighted sum of its inputs, adds a bias term, and then passes the result through an activation function. Common activation functions include sigmoid, tanh, and Rectified Linear Unit (ReLU). The activation function introduces nonlinearity, allowing the network to model nonlinear relationships.

4. Output Layer: The final layer of the MLP is the output layer, which produces the network's predictions. The number of neurons in the output layer depends on the task. For example, for binary classification, there might be one neuron with a sigmoid activation function, while for multi-class classification, there could be multiple neurons with softmax activations.

5. Weights and Biases: Each connection between neurons has an associated weight, and each neuron has a bias term. During training, these weights and biases are adjusted to minimize the difference between predicted outputs and actual targets.

6. Forward Propagation: The process of passing input data through the network to generate predictions is called forward propagation. It involves calculating the outputs of neurons layer by layer, from the input layer through the hidden layers to the output layer.

7. Backward Propagation: After forward propagation, the loss between predictions and targets is computed. Backward propagation involves calculating gradients of the loss with respect to weights and biases and using these gradients to update the parameters using optimization algorithms like gradient descent.

8. Training: The network is trained iteratively using batches of data. The goal is to minimize the loss function over multiple epochs by adjusting the weights and biases through backward propagation.

MLPs are capable of learning complex representations of data, making them suitable for a wide range of tasks such as image classification, natural language processing, and more. However, the number of hidden layers and the number of neurons in each layer must be chosen carefully, as too many can lead to overfitting and too few might result in underfitting. This process often involves experimentation and tuning to find the right architecture for a specific task.

1. WHAT ARE ACTIVATION FUNCTIONS?

Activation functions are mathematical functions applied to the output of a neuron or node in an artificial neural network. They introduce nonlinearity to the network, allowing it to learn and model complex relationships in data. Activation functions determine the firing or activation level of a neuron based on its input.

In a neural network, each neuron receives inputs, multiplies them by corresponding weights, and then computes a weighted sum. The activation function is then applied to this weighted sum to determine whether and to what extent the neuron should "fire" or be activated. The transformed output of the activation function becomes the input to the next layer of neurons.

The primary purpose of activation functions is to introduce nonlinearity, as a network consisting of only linear operations would be equivalent to a linear transformation, unable to capture complex patterns in data. Nonlinearity allows neural networks to model intricate relationships, handle variations, and perform tasks like classification, regression, and feature extraction.

Commonly used activation functions include sigmoid, tanh, rectified linear unit (ReLU), and variants like Leaky ReLU, Parametric ReLU (PReLU), exponential linear unit (ELU), and more. Each activation function has its own characteristics and strengths, making it suitable for different scenarios and tasks.

In summary, activation functions are crucial components of neural networks that introduce nonlinearity, enabling the networks to approximate complex functions and learn from data effectively.

1. TYPES OF ACTIVATION FUNCTIONS?

Activation functions introduce nonlinearity to neural networks, allowing them to model complex relationships and capture different types of patterns in data. Here are some common types of activation functions used in neural networks:

1. \*\*Sigmoid Function\*\*:

- Formula: σ(x) = 1 / (1 + e^(-x))

- Output Range: (0, 1)

- Characteristics: S-shaped curve that maps inputs to a probability-like output. Used historically, but less common due to vanishing gradient problem.

2. \*\*Hyperbolic Tangent (Tanh) Function\*\*:

- Formula: tanh(x) = (e^x - e^(-x)) / (e^x + e^(-x))

- Output Range: (-1, 1)

- Characteristics: Similar to sigmoid, but centered at 0. Outputs can be both negative and positive.

3. \*\*Rectified Linear Unit (ReLU)\*\*:

- Formula: ReLU(x) = max(0, x)

- Output Range: [0, ∞)

- Characteristics: Most widely used activation function. Efficient and avoids vanishing gradient problem. Can suffer from the "dying ReLU" problem for large negative inputs.

4. \*\*Leaky ReLU\*\*:

- Formula: Leaky ReLU(x) = x if x > 0, αx if x ≤ 0 (where α is a small positive constant)

- Output Range: (-∞, ∞)

- Characteristics: Similar to ReLU but allows small gradients for negative inputs, addressing the "dying ReLU" problem.

5. \*\*Parametric ReLU (PReLU)\*\*:

- Formula: PReLU(x) = x if x > 0, αx if x ≤ 0 (where α is a learnable parameter)

- Output Range: (-∞, ∞)

- Characteristics: Similar to Leaky ReLU, but α is learned during training.

6. \*\*Exponential Linear Unit (ELU)\*\*:

- Formula: ELU(x) = x if x > 0, α(e^x - 1) if x ≤ 0 (where α is a positive constant)

- Output Range: (-α, ∞)

- Characteristics: Similar to ReLU, but with a smoother transition for negative inputs. Addresses "dying ReLU" problem and can lead to faster convergence.

7. \*\*Scaled Exponential Linear Unit (SELU)\*\*:

- Formula: SELU(x) = λx if x > 0, λα(e^x - 1) if x ≤ 0 (where α and λ are constants)

- Output Range: Varies based on constants

- Characteristics: Introduced for self-normalizing neural networks. Maintains mean and variance of activations, promoting stable training.

8. \*\*Softmax Function\*\*:

- Formula: softmax(x\_i) = e^(x\_i) / Σ(e^(x\_j)) for all j

- Output Range: [0, 1], sums to 1 across all dimensions

- Characteristics: Converts a vector of arbitrary real numbers into a probability distribution, often used in multi-class classification.

These activation functions serve different purposes and have varying characteristics. Choosing the appropriate activation function depends on the nature of the problem, the architecture of the neural network, and considerations such as addressing vanishing gradients and enabling efficient training.

1. LEARNING WHEN TO USE THE DIFFERENT TYPES OF ACTIVATION FUNCTIONS