DEEP LEARNING

PART- A

**1.**

**A) What is the difference between Graph execution and Eager execution?**

| **Aspect** | **Graph Execution** | **Eager Execution** |
| --- | --- | --- |
| Computation model | Computation is defined as a static graph. | Computation is defined and executed dynamically. |
| Definition of operations | Define and build the computation graph first. | Define and execute operations immediately. |
| Debugging | Debugging can be more challenging. | Easier to debug, similar to traditional Python. |
| Flexibility | Less flexible, as the graph is static. | More flexible, as operations can change on-the-fly. |
| Python interaction | Requires a separate session to run the graph. | No need for a separate session; uses Python naturally. |

**B) What is Size, Shape and axis of Tensor?**

**A.** The terms "size," "shape," and "axis" are fundamental concepts that describe various aspects of the tensor's structures:

1. Size:

* Size refers to the total number of elements in a tensor.
* It is the product of the dimensions (length, width, height, etc.) of the tensor.

2. Shape:

* The shape of a tensor defines its dimensions along each axis.
* It is represented as a tuple (or list) of integers, where each integer corresponds to the length of a specific axis.

3. Axis:

* An axis is a specific dimension along which data is stored in a tensor.
* Each dimension in the shape of a tensor is associated with a unique axis.
* In a 2D tensor (matrix), the first axis typically represents rows, and the second axis represents columns.

.

**C) What are the limitations of Tensor Flow?**

**A. Performance:** While TensorFlow offers strong performance for many deep learning tasks, the optimal performance of models often depends on careful optimization and tuning, which can be time-consuming.

**Compatibility:** As TensorFlow evolves, models and code written for older versions may require updates or modifications to run on newer versions.

**Dependency:** Even though TensorFlow reduces the size of the program and makes it user-friendly, it adds a layer of complexity to it. Every code needs some platform for its execution which increases dependency.

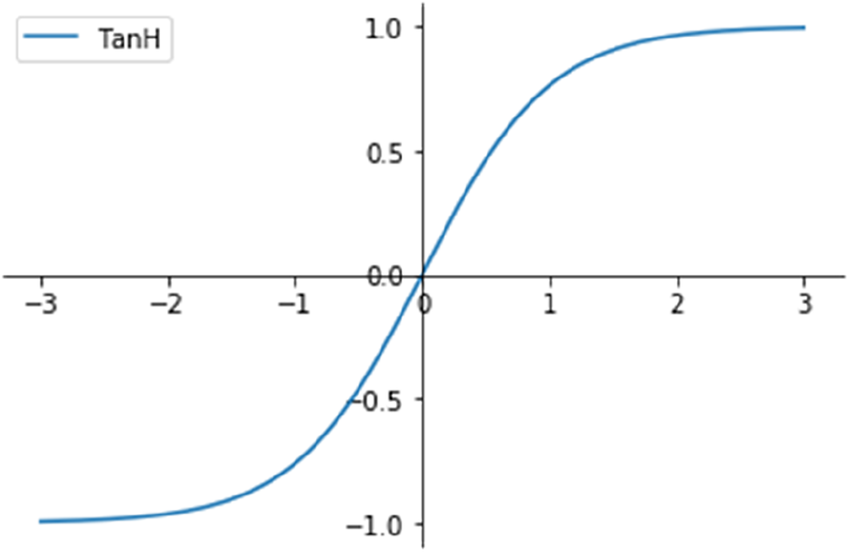
**Slow:** It is comparatively slower and less usable compared to its competing frameworks.

**Frequent Updates:** Tensor flow undergoes frequent updates making it overhead for a user to time to time uninstall and reinstall it so that it can bind and be blended with its latest updates.

**D) Plot the graph of tanH activation function?**

**A.** **Tanh:**

The tanh function is very similar to the sigmoid function. The only difference is that it is symmetric around the origin. The range of values in this case is from -1 to 1. Thus the inputs to the next layers will not always be of the same sign. The tanh function is defined as



tanh(x)=2sigmoid(2x)-1

**E) Given an Image size n\*n and filter m\*m, assuming stride =1, what is the size of feature map**

**A.** When you convolve an image of size n x n with a filter (also called a kernel) of size m x m and use a stride of 1, the size of the resulting feature map can be calculated as follows:

* If you don't use any padding, the feature map size will be (n - m + 1) x (n - m + 1).
* If you apply padding to the input image tIn this case, the feature map size will be n x n. So in this case the size of the padding is typically (m - 1) / 2 pixels on each side.

**F) Why do we need Flattening?**

**A.** Flattening refers to the process of converting a multi-dimensional tensor or array into a one-dimensional vector. This operation is commonly used for specific layers in a neural network architecture for several important reasons:

1. Compatability
2. Parameter Reduction
3. Reducing the Dimensionality
4. Model Simplicity

**G) How are multiple filters applied to CNN?**

**A.**

* In a Convolutional Neural Network (CNN), multiple filters are applied to an input data by performing a series of convolution operations. Each filter is a small, learnable matrix that slides or convolves across the input data to produce a feature map. This process is repeated for each filter to extract different features from the input.
* CNNs apply multiple filters to input data to extract various features, and these features are then used for tasks like image classification, object detection, and segmentation. The combination of multiple filters allows CNNs to learn hierarchical representations of features in the data.

**H) Why do we use RNN in NLP?**

**A.** Recurrent Neural Networks (RNNs) are a class of neural networks commonly used in Natural Language Processing (NLP) tasks for several reasons:

1. Speech Recognition

2. Machine Translation

3. Sentiment Analysis

4. Name Entity Recognition

5. Sequential Data Handling etc…

**PART – B**

**Section 1**

**2A. What are the different components of Tensor flow. How do we run a graph in a session**

**Ans:** The word TensorFlow is made by two words, i.e., Tensor and Flow

1. **Tensor** is a multidimensional array
2. **Flow** is used to define the flow of data in operation.

TensorFlow is used to define the flow of data in operation on a multidimensional array or Tensor.

1. Tensor : Tensors are multi-dimensional arrays with a uniform type (called a dtype).Tensors are (kind of) like nparrays.

All tensors are immutable like Python numbers and strings: you can never update the contents of a tensor, only create a new one.

COMPUTATIONAL GRAPHS :

* TensorFlow operations are executed by Python, operation by operation, and returning results back to Python.While eager execution has several unique advantages, graph execution enables portability outside Python and tends to offer better performance.
* Graph execution means that tensor computations are executed as a TensorFlow graph, sometimes referred to as a tf.Graph or simply a "graph." Graphs are data structures that contain a set of tf.Operation objects, which represent units of computation; and tf.Tensor objects, which represent the units of data that flow between operations.

**2B) Explain Linear Regression with Gradient descent**

**Ans:** Linear regression with gradient descent in the context of deep learning is similar to traditional linear regression, but it's applied within a neural network framework. Deep learning typically involves more complex models with multiple layers, but linear regression can serve as a basic building block or as the output layer of a neural network.

1. Linear Regression Model:

In deep learning, linear regression is often used as a simplified case, especially in the output layer of a neural network for regression tasks. The model's prediction is calculated as a linear combination of the input features. Mathematically, it can be represented as:

y = Wx + b

Here,

* y is the predicted output (a scalar value)
* x is the input feature vector.
* W represents the weight matrix, where each element **W\_ij** corresponds to the weight or coefficient for the connection between the i-th input feature and the j-th output.
* b is the y-intercept (bias)

2. Cost Function:

The cost function (also known as the loss function) measures how well the model's predictions match the actual target values. For linear regression, a commonly used cost function is the Mean Squared Error (MSE), defined as:

MSE = (1/N) \* Σ(yi - (mxi + b))^2

Here,

* N is the number of data points
* yi is the actual target value for the i-th data point.
* xi is the corresponding input feature.
* m and **b** are the model parameters.

3. Gradient Descent:

Gradient descent is an iterative optimization algorithm used to minimize the cost function. It works by updating the model parameters in the opposite direction of the gradient of the cost function with respect to those parameters. The update rule for gradient descent in linear regression is as follows:

m = m - learning\_rate \* (∂MSE/∂m)

b = b - learning\_rate \* (∂MSE/∂b)

Here,

* learning\_rate is a hyperparameter that controls the step size in each iteration.
* (∂MSE/∂m) is the partial derivative of the MSE with respect to m.
* (∂MSE/∂b) is the partial derivative of the MSE with respect to b.

4. Training:

The training process involves repeatedly applying the gradient descent update rule for a specified number of iterations or until the cost converges to a minimum. During training, the algorithm adjusts the values of **m** and **b** to minimize the MSE.

5. Prediction:

Once the training is complete, you can use the learned values of m and b to make predictions for new input data points using the linear equation y = mx + b.

Gradient descent continues to update the model parameters until convergence, ensuring that the linear regression model fits the data as closely as possible. The choice of learning rate and the number of iterations are hyperparameters that need to be carefully tuned to achieve good model performance.

**3A. Explain the lifecycle of variable ?**

**Ans:** In deep learning, the lifecycle of a variable refers to the stages that a variable goes through during the training and inference process. Variables are an essential part of deep learning models as they store and represent the model's parameters (weights and biases) and intermediate results. Here's an overview of the lifecycle of a variable in deep learning:

1. Intialization:

The first stage in the lifecycle of a variable is initialization. When we create a deep learning model, we define various layers and their associated variables (weights and biases). These variables are typically initialized with random values or predefined initializations before training begins. Proper initialization is crucial because it can affect the convergence and performance of the model.

2. Training:

During the training phase, the variables are updated iteratively using an optimization algorithm (e.g., gradient descent) to minimize a chosen loss function. The process involves the following steps:

* Forward Pass: Input data is passed through the model, and predictions are made.
* Loss Calculation: The difference between the predictions and the actual target values is computed using the loss function.
* Backward Pass (Backpropagation): Gradients of the loss with respect to the model's variables are computed. These gradients guide the updates to the variables in the next step.
* Update Variables: The values of the variables (weights and biases) are adjusted in the direction that minimizes the loss, typically using gradient descent or its variants

3. Validation and Hyperparameter Tuning:

After training, it's common to evaluate the model's performance on a separate validation dataset. You may adjust hyperparameters such as the learning rate, batch size, and network architecture based on the validation results. Variables remain unchanged during this phase unless you decide to retrain the model with different hyperparameters.

4. Deployment:

Once the model is trained and validated, it can be deployed for inference in real-world applications. During deployment, the trained variable values are used to make predictions on new, unseen data. The model's architecture and variable values are typically saved to a file

5. Fine tuning and transfer Learning:

In some cases, you may want to fine-tune a pre-trained model on a specific task or adapt it to new data. In transfer learning, the pre-trained model's variables are partially or fully retained, and only certain layers or variables are updated during further training.

6. Model Maintainance and Updates:

In practice, deep learning models may require periodic maintenance and updates. This could involve retraining the model on new data, fine-tuning it to adapt to changing requirements, or updating the model architecture. During these processes, the variables may undergo changes.

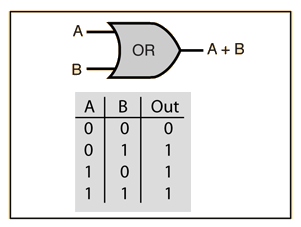
7. End of Lifecycle:

The lifecycle of a variable in deep learning ends when a model is no longer in use or has been replaced by a more advanced model. At this point, the model and its variables may be archived or discarded.

**3B) Implement ANN for OR Gate?**

**Ans:** OR Neural Network:

A single-layer perceptron can solve any linearly separable function such as the OR function since it is possible to draw a single straight line (hyperplane) to separate and group the output patterns. The data is linearly separable using a 1-dimensional hyperplane.



From the diagram, the OR gate is 0 only if both inputs are 0.

**Row 1**

* From w1x1+w2x2+b, initializing w1, w2, as 1 and b as –1, we get;

*x1(1)+x2(1)–1*

* Passing the first row of the OR logic table (x1=0, x2=0), we get;

*0+0–1 = –1*

* From the Perceptron rule, if Wx+b≤0, then y`=0. Therefore, this row is correct.

**Row 2**

* Passing (x1=0 and x2=1), we get;

*0+1–1 = 0*

* From the Perceptron rule, if Wx+b <**=** 0, then y`=0. Therefore, this row is incorrect.
* So we want values that will make inputs x1=0 and x2=1 give y` a value of 1. If we change w2 to 2, we have;

*0+2–1 = 1*

* From the Perceptron rule, this is correct for both the row 1 and 2.

**Row 3**

* Passing (x1=1 and x2=0), we get;

*1+0–1 = 0*

* From the Perceptron rule, if Wx+b <**=** 0, then y`=0. Therefore, this row is incorrect.
* Since it is similar to that of row 2, we can just change w1 to 2, we have;

*2+0–1 = 1*

* From the Perceptron rule, this is correct for both the row 1, 2 and 3.

**Row 4**

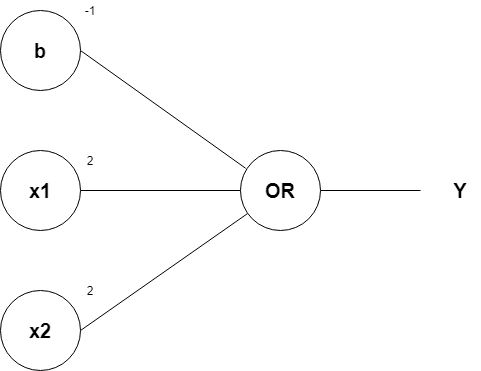
* Passing (x1=1 and x2=1), we get;

*2+2–1 = 3*

* Again, from the perceptron rule, this is still valid. Quite Easy!

Therefore, we can conclude that the model to achieve an OR gate, using the Perceptron algorithm is;

*2x1+2x2–1*



SECTION -2

4A. What is the objective of Gradient Decent ?

Ans. The objective of Gradient Descent is to minimize a given cost or loss function in the context of machine learning and optimization. In particular, Gradient Descent is used to find the set of model parameters that result in the lowest possible value of the cost or loss function. This process of finding the optimal parameters is crucial in training machine learning models, including linear regression, neural networks, and many other algorithms.

1. Minimize the Cost Function: The primary goal of Gradient Descent is to minimize a cost or loss function, which measures the error or discrepancy between the model's predictions and the actual target values. Minimizing this function means making the model's predictions as close as possible to the true target values.

2. To Find Optimal Model Parameters: Gradient Descent iteratively updates the model's parameters (e.g., weights and biases) to reach a configuration where the cost function is minimized. These optimized parameters represent the best possible model for the given dataset and problem.

3. Convergence: Gradient Descent aims to reach a point of convergence where the algorithm finds the minimum value (or a very low value) of the cost function. At this point, further updates to the parameters do not significantly reduce the cost. Convergence indicates that the algorithm has found a suitable solution.

4. Efficiency: Gradient Descent seeks to minimize the cost function efficiently by iteratively adjusting the parameters based on the gradient (derivative) of the cost function with respect to each parameter. It takes steps in the direction that leads to a decrease in the cost, and the size of each step is controlled by a hyperparameter known as the learning rate

4B. Elaborate the structure of an Artificial Neural Network

Ans: Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of [machine learning](https://www.ibm.com/topics/machine-learning) and are at the heart of [deep learning](https://www.ibm.com/topics/deep-learning) algorithms.

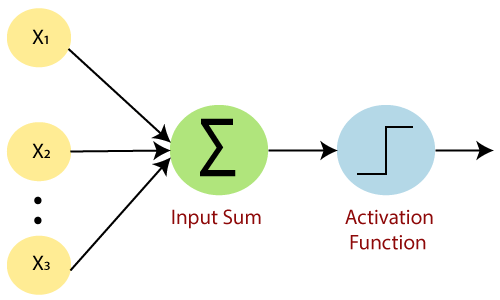
ANNs are used for regression or classification problems and they consists of two basic architecture:

1. Single-Layer Artificial Neural Networks

2. Multi-Layer Artifical Neural Networks

1. Single-Layer Artificial Neural Networks:

A single-layer neural network, often referred to as a single-layer perceptron, is the simplest form of an artificial neural network. It consists of only one layer of neurons, which serves as both the input and output layer. Here's an overview of a single-layer neural network:



Structure:

1.Input Layer/Neurons: The input layer consists of input neurons, each representing a feature or input variable. These neurons pass their inputs directly to the output layer without any intermediate hidden layers.

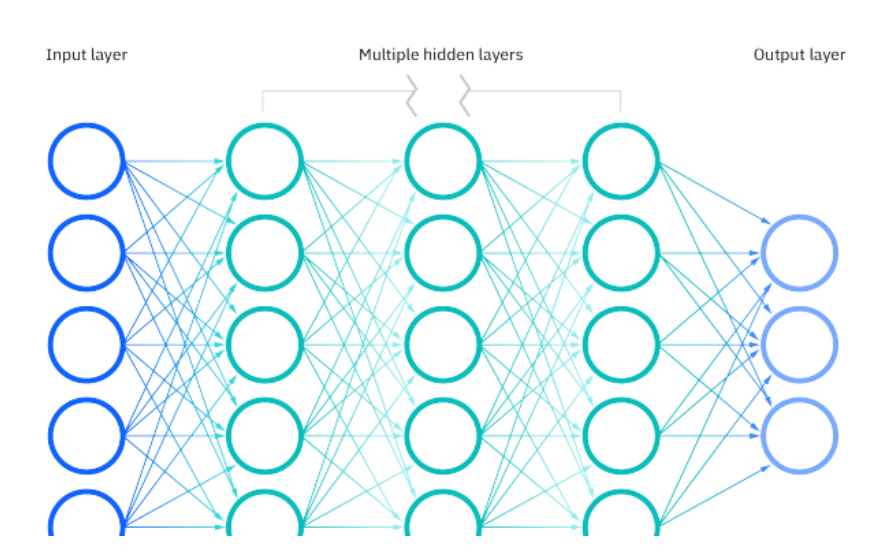
2.Weights: Each input neuron is associated with a weight. These weights represent the strength or importance of each input feature in determining the network's output.

3.Bias: In addition to the input neurons and weights, a single-layer neural network includes a bias neuron. The bias is similar to a weight but is not associated with any specific input feature. It allows the network to shift its decision boundary or decision surface.

4.Activation Function: Typically, a single-layer neural network uses a step function (also known as a threshold function) as the activation function. The step function outputs a binary value (e.g., 0 or 1) based on a threshold.

2. Multi Layer Neural Network:

A multi-layer neural network in deep learning, also known as a multi-layer perceptron (MLP) or feedforward neural network, consists of multiple layers of artificial neurons, including an input layer, one or more hidden layers, and an output layer. These networks are designed to capture complex patterns and representations in data. Here's an overview of a multi-layer neural network:



Structure:

1.Input Layer:

The input layer consists of neurons (also known as nodes) representing input features.Each neuron in the input layer corresponds to a feature in the input data.The number of neurons in the input layer is determined by the dimensionality of the input data.

2. Hidden Layers:

Hidden layers are intermediate layers between the input and output layers.Each hidden layer consists of multiple neurons or units.The number of hidden layers and the number of neurons in each layer are design choices and can vary depending on the problem.Hidden layers introduce nonlinearity into the model, enabling it to capture complex patterns and representations.

3.Output Layer:

The output layer is the final layer of the neural network.Neurons in the output layer produce the network's predictions or classifications.The number of neurons in the output layer depends on the problem type:

* For binary classification, there is typically one neuron with a sigmoid activation function.
* For multi-class classification, the number of neurons equals the number of classes, often with softmax activation.
* For regression, there is typically one neuron with a linear activation function.

4. Connections (Weights and Biases):

Each neuron in one layer is connected to every neuron in the previous and subsequent layers (if present).These connections are associated with weights, which represent the strengths of the connections.Each neuron also has an associated bias, which allows the network to shift its decision boundary.

5. Activation Functions:

Neurons in hidden and output layers typically apply activation functions to their weighted inputs.Common activation functions include ReLU (Rectified Linear Unit), sigmoid, tanh, and softmax. Activation functions introduce nonlinearity into the model, enabling it to learn complex relationships.

5A. Explain the Sigmoid activation function

## Ans: Sigmoid Function:

The sigmoid function is a special form of the logistic function and is usually denoted by σ(x) or sig(x). It is given by:

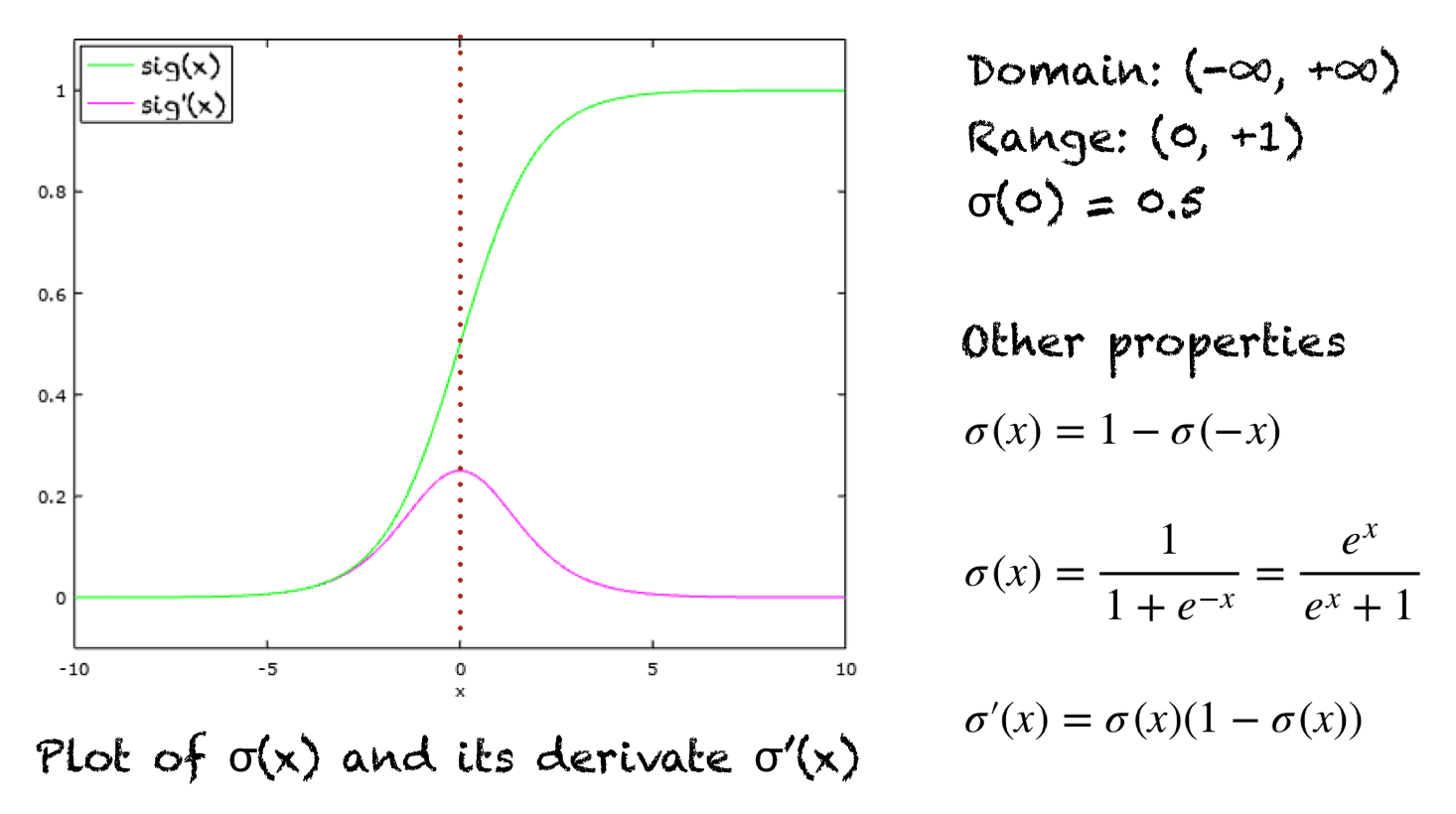
σ(x) = 1/(1+exp(-x))

Properties of Sigmoid Functions:

1. A sigmoid function can take any real number as an input. So, the **domain** of a sigmoid function is **(-∞, ∞)**.

1. The output of a sigmoid function is a probability value between 0 and 1. So, the **range** of a sigmoid function is **(0, +1)**.
2. The value of a sigmoid function at x=0 is 0.5. So, we can say that σ(0) = 0.5.
3. The sigmoid function is **monotonically increasing**. This means that as the input value increases, the output value increases or remains constant, but it never decreases.
4. The function is **continuous** in the domain. This means that we can find the sigmoid value for any point on the curve.
5. The function is **differentiable** everywhere in its domain.
6. Sigmoid function has a **non-negative derivative** at each point. It shows that the graph of a sigmoid function never decreases.
7. The sigmoid function has only **one inflection point** at x=0. (An inflection point is a point where the curve changes the sign.)
8. The derivative of a sigmoid function is bell-shaped

* The graph of sigmoid function is an S-shaped curve as shown by the green line in the graph below. The figure also shows the graph of the derivative in pink color. The expression for the derivative, along with some important properties are shown on the right.



* When the activation function for a neuron is a sigmoid function it is a guarantee that the output of this unit will always be between 0 and 1. Also, as the sigmoid is a non-linear function, the output of this unit would be a non-linear function of the weighted sum of inputs. Such a neuron that employs a sigmoid function as an activation function is termed as a sigmoid unit.
* A sigmoid function placed as the last layer of a machine learning model can serve to convert the model's output into a probability score, which can be easier to work with and interpret.

Problems with Sigmoid activation function:

 The main problems with the Sigmoid function are:

1. Vanishing gradient: looking at the function plot, you can see that when inputs become small or large, the function saturates at 0 or 1, with a derivative extremely close to 0. Thus it has almost no gradient to propagate back through the network, so there is almost nothing left for lower layers.

 2. Computationally expensive: the function has an exponential operation

3. The output is not zero centered

5B. Compare and Contrast between Relu and Softmax activation function

Ans: ReLU (Rectified Linear Unit) and Softmax are two different activation functions used in neural networks, and they serve distinct purposes. Let's compare and contrast these two activation functions:

1. ReLU (Rectified Linear Unit):

A. Purpose:

* ReLU is primarily used in hidden layers of neural networks.
* It introduces non-linearity into the model, allowing it to learn complex patterns and representations in the data.
* ReLU is especially effective at mitigating the vanishing gradient problem, which can occur with other activation functions like sigmoid and tanh.

B. Activation Formula:

* The ReLU activation function is defined as follows:

f(x) = max(0, x)

* It outputs 0 for all negative input values and returns the input itself for positive input values.

C. Output Range:

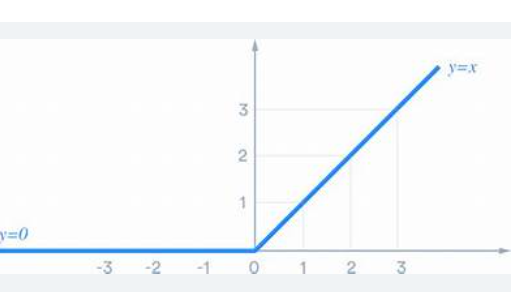
* The output of ReLU is in the range [0, ∞). Negative values are "clipped" to 0.

D. Advantages:

* Computationally efficient compared to some other activation functions.
* Helps in overcoming the vanishing gradient problem.
* Empirically, ReLU has been successful in training deep neural networks.

E. Disadvantages:

* ReLU can suffer from the "dying ReLU" problem, where neurons can become inactive (output 0) for all inputs, effectively "dying" and not contributing to learning.
* Not suitable for all types of problems, especially those that require outputs in a specific range.



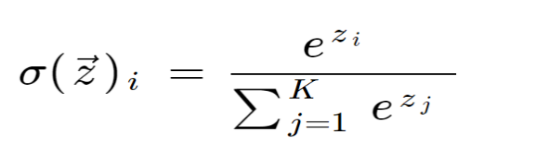
Soft Max Activaton Function :

A. Purpose:

* Softmax is typically used in the output layer of neural networks for multi-class classification problems.
* It converts raw model outputs (logits) into probability distributions over multiple classes.

B. Activation Formula:

* The Softmax activation function is defined as follows for each output neuron (j):



* It exponentiates the logits and normalizes them such that the sum of all class probabilities equals 1.

C. Output Range:

* The output of Softmax is a probability distribution over all classes, with values in the range (0, 1).

D. Advantages:

* Converts raw model outputs into interpretable class probabilities.
* Useful for multi-class classification problems where each instance belongs to one of several exclusive classes.
* The gradients are well-defined, making it suitable for training.

E. Disadvantages:

* Softmax is not suitable for regression or binary classification problems.
* It can lead to numerical instability when dealing with very large or very small logits.

|--- e^(z\_1) ---|

|--- e^(z\_2) ---|

| ... | | e^(z\_i) |

Input -> |--- e^(z\_k) -- -| -- > Softmax -- > |---------| --> Output

| ... | | Σ(e^(z\_j))|

|--- e^(z\_n) ---|