Artificial Intelligence Fullstack
[Course]
Week 16 – Deep Learning –
TF - MLP Feed Forward Neural Network
[See examples / code in GitHub code repository]

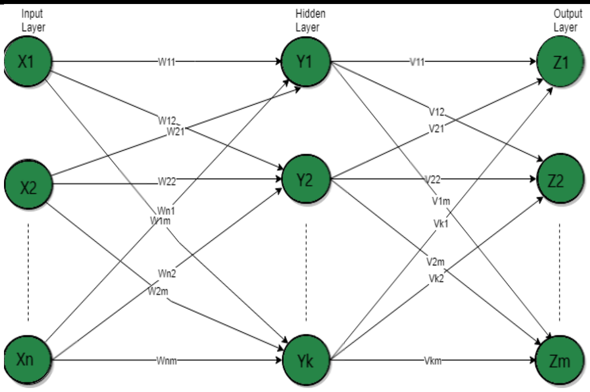
It is not about Theory, it is 20% Theory and 80% Practical – Technical/Development/Programming [Mostly Python based]

DL | What is MLP Feed Forward Neural Network ?

Multilayer Feed-Forward Neural Network(MFFNN) is an interconnected Artificial Neural Network with multiple layers that has neurons with weights associated with them and they compute the result using activation functions. It is one of the types of Neural Networks in which the flow of the network is from input to output units and it does not have any loops, no feedback, and no signal moves in backward directions that is from output to hidden and input layer

In this network there are the following layers:

- **Input Layer:** It is starting layer of the network that has a weight associated with the signals.
- **Hidden Layer:** This layer lies after the input layer and contains multiple neurons that perform all computations and pass the result to the output unit.
- **Output Layer:** It is a layer that contains output units or neurons and receives processed data from the hidden layer, if there are further hidden layers connected to it then it passes the weighted unit to the connected hidden layer for further processing to get the desired result.



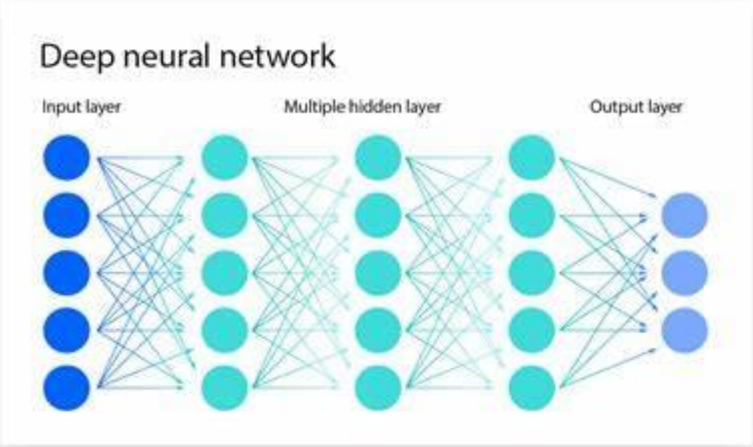
Reference:

https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning
https://aiml.com/what-is-a-multilayer-perceptron-mlp/
https://builtin.com/data-science/feedforward-neural-network-intro
www.analyticsvidhya.com/blog/2020/07/neural-networks-from-scratch-in-python-

DL | Forward Propagation - Backward Propagation

Neural networks work in a very similar manner. It takes several inputs, processes it through multiple neurons from multiple hidden layers, and returns the result using an output layer. This result estimation process is technically known as **“Forward Propagation“.**

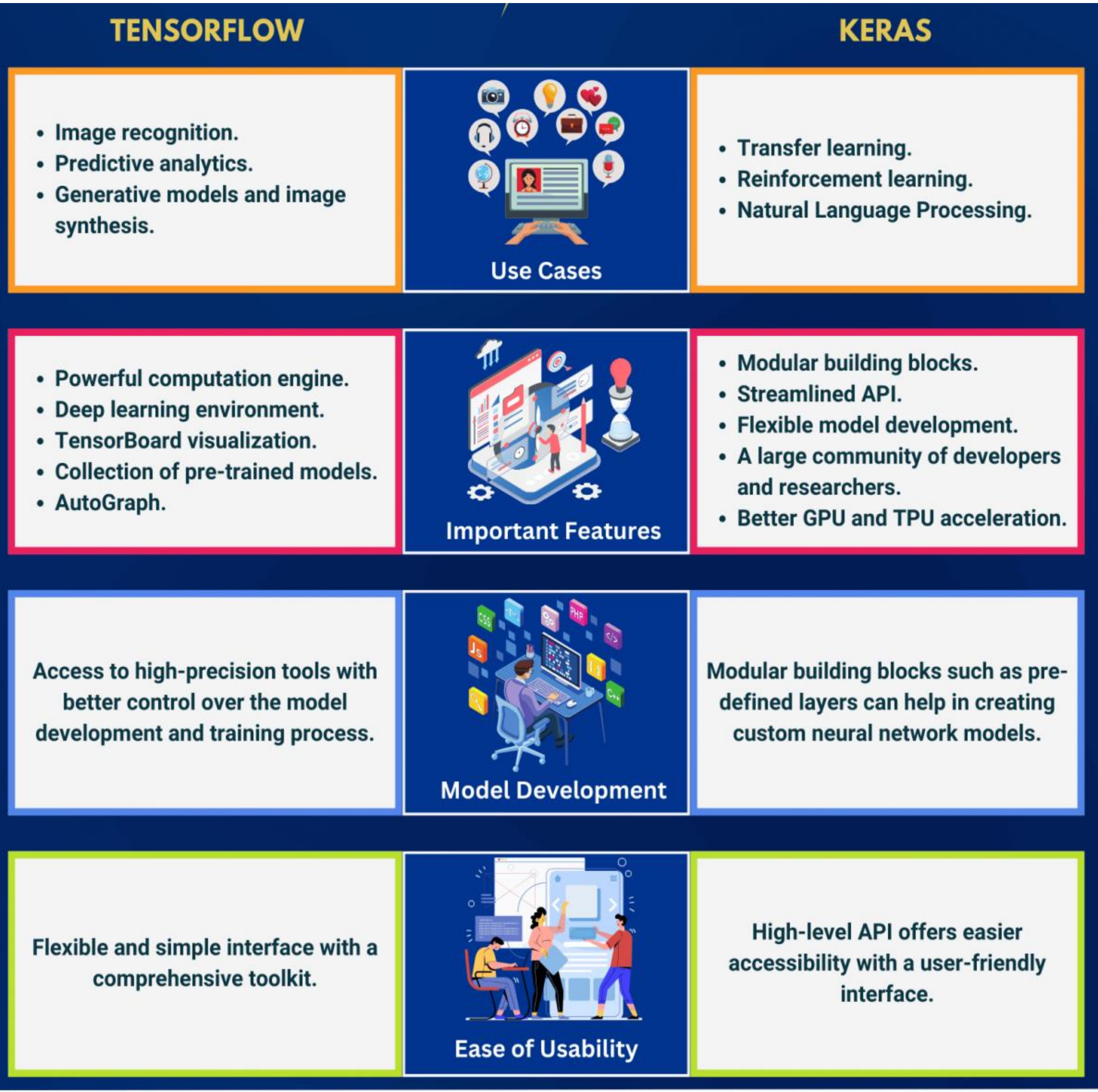
We try to minimize the value/ weight of neurons that are contributing more to the error and this happens while traveling back to the neurons of the neural network and finding where the error lies. This process is known as “**Backward Propagation**“.



Reference:

https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning
https://aiml.com/what-is-a-multilayer-perceptron-mlp/
https://builtin.com/data-science/feedforward-neural-network-intro
www.analyticsvidhya.com/blog/2020/07/neural-networks-from-scratch-in-python-and-r

DL | Difference between TensorFlow and Keras



TENSORFLOW		KERAS
• Image recognition. • Predictive analytics. • Generative models and image synthesis.	Use Cases	• Transfer learning. • Reinforcement learning. • Natural Language Processing.
• Powerful computation engine. • Deep learning environment. • TensorBoard visualization. • Collection of pre-trained models. • AutoGraph.	Important Features	• Modular building blocks. • Streamlined API. • Flexible model development. • A large community of developers and researchers. • Better GPU and TPU acceleration.
Access to high-precision tools with better control over the model development and training process.	Model Development	Modular building blocks such as pre-defined layers can help in creating custom neural network models.
Flexible and simple interface with a comprehensive toolkit.	Ease of Usability	High-level API offers easier accessibility with a user-friendly interface.

Reference:

https://www.geeksforgeeks.org/difference-between-tensorflow-and-keras/

https://www.guru99.com/tensorflow-vs-keras.html

https://mljourney.com/keras-vs-tensorflow-key-differences-use-cases-and-performance-comparison/

DL | Keras cheat sheet

Reference:

https://www.geeksforgeeks.org/keras-cheatsheet/
https://medium.com/data-science/tensorflow-keras-cheat-sheet-5ec99d9a1ccf
https://www.datacamp.com/cheat-sheet/keras-cheat-sheet-neural-networks-in-python
https://elitedatascience.com/wp-content/uploads/2018/05/Keras-Tutorial-Cheatsheet.pdf

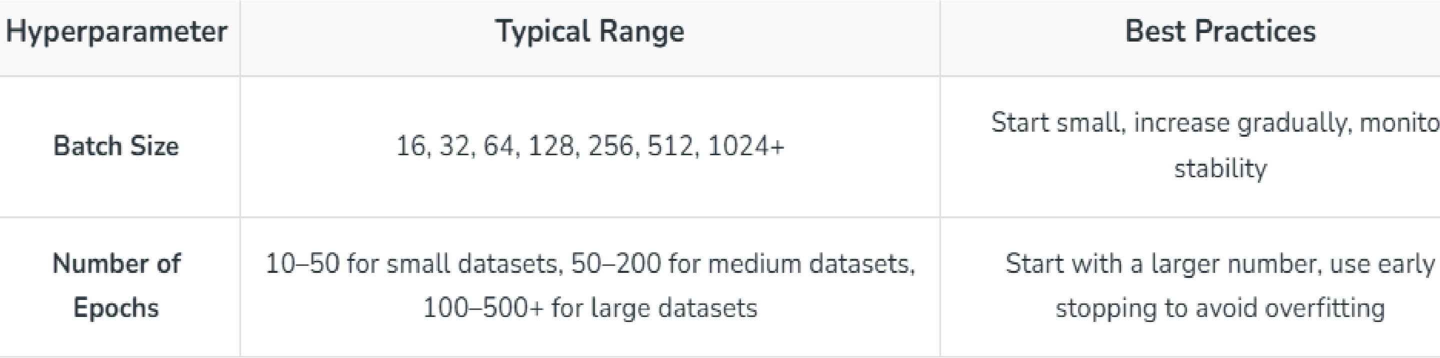
DL | What is Difference Between a Batch and an Epoch

What is Batch Size and Number of Epochs?

Before diving into the specifics, let’s clarify what these terms mean:

- **Batch Size**: The number of training samples processed before the model's internal parameters are updated. A batch size of 32 means that 32 samples are used to compute the gradient and update the model weights before the next batch of 32 samples is processed.

- **Number of Epochs**: The number of times the entire training dataset is passed through the model. If you have 1000 training samples and set the number of epochs to 10, the model will see the entire dataset 10 times.



Hyperparameter	Typical Range	Best Practices
Batch Size	16, 32, 64, 128, 256, 512, 1024+	Start small, increase gradually, monito stability
Number of Epochs	10–50 for small datasets, 50–200 for medium datasets, 100–500+ for large datasets	Start with a larger number, use early stopping to avoid overfitting

Reference:

https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/
https://www.geeksforgeeks.org/batch-size-in-neural-network/
https://www.geeksforgeeks.org/how-to-choose-batch-size-and-number-of-epochs-when-fitting-a-model/
https://www.sabrepc.com/blog/Deep-Learning-and-AI/Epochs-Batch-Size-Iterations?srsltid=AfmBOorhuyU5IIxloZTvseaoS5fprTgxdbdVwgcIEMM7o05ij7FIBtP3

DL | Difference between parameter and hyperparameter

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

Model parameters in different models:

- □ m(slope) and c(intercept) in Linear Regression
- □ weights and biases in Neural Networks

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

Model hyperparameters in different models:

- □ Learning rate in gradient descent
- □ Number of iterations in gradient descent
- □ Number of layers in a Neural Network
- □ Number of neurons per layer in a Neural Network
- □ Number of clusters(k) in k means clustering

Reference:

https://www.geeksforgeeks.org/difference-between-model-parameters-vs-hyperparameters/
https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/
https://www.geeksforgeeks.org/hyperparameter-tuning/

DL | What is an Activation Function?

An activation function is a mathematical function applied to the output of a neuron. It introduces non-linearity into the model, allowing the network to learn and represent complex patterns in the data. Without this non-linearity feature, a neural network would behave like a linear regression model, no matter how many layers it has.

Activation function decides whether a neuron should be activated by calculating the weighted sum of inputs and adding a bias term. This helps the model make complex decisions and predictions by introducing non-linearities to the output of each neuron.

Introducing Non-Linearity in Neural Network

Non-linearity means that the relationship between input and output **is not a straight line**. In simple terms, the output **does not change proportionally** with the input. A common choice is the ReLU function, defined as σ(x)=max(0,x) $\sigma(x)=\max(0,x)$.

Imagine you want to classify **apples** and **bananas** based on their **shape and color**.

- If we use a **linear function**, it can only separate them using a **straight line**.
- But real-world data is often more complex (e.g., overlapping colors, different lighting).
- By adding a **non-linear activation function** (like **ReLU, Sigmoid, or Tanh**), the network can create **curved decision boundaries** to separate them correctly.

Reference:

https://www.geeksforgeeks.org/activation-functions-neural-networks/
https://medium.com/data-science/activation-functions-non-linearity-neural-networks-101-ab0036a2e701
https://www.v7labs.com/blog/neural-networks-activation-functions

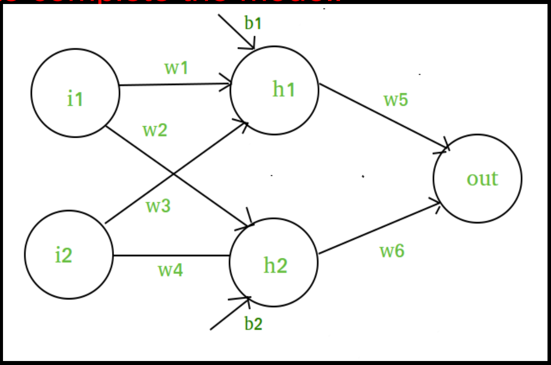
DL | Example - Activation Function?

Mathematical Proof of Need of Non-Linearity in Neural Networks

To illustrate the need for non-linearity in neural networks with a specific example, let's consider a network with two input nodes (i1and i2)($i_1$and i_2), a single hidden layer containing neurons h1 and h2h_1 and h_2, and an output neuron (out).

We will use w1,w2w_1,w_2 as weights connecting the inputs to the hidden neuron, and w5w_5 as the weight connecting the hidden neuron to the output. We'll also include biases (b1b_1 for the hidden neuron and b2b_2 for the output neuron) to complete the model.

1. **Input Layer**: Two inputs i1i_1 and i2i_2.
2. **Hidden Layer**: Two neuron h1h_1 and h2h_2
3. **Output Layer**: One output neuron.



The input to the hidden neuron h1h_1 is calculated as a weighted sum of the inputs plus a bias:

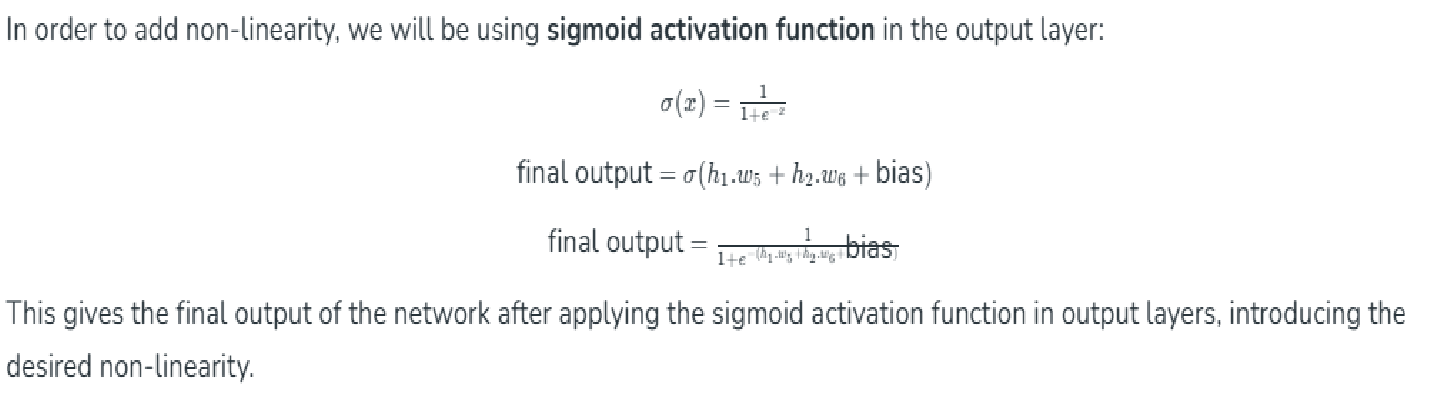
h1=i1.w1+i2.w3+b1

h2=i1.w2+i2.w4+b2

The output neuron is then a weighted sum of the hidden neuron's output plus a bias:

output=h1.w5+h2.w6+bias

Here, h_1 , h_2 \text{ and output} are linear expressions.



In order to add non-linearity, we will be using **sigmoid activation function** in the output layer:

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

$$\text{final output} = \sigma(h_1.w_5 + h_2.w_6 + \text{bias})$$

$$\text{final output} = \frac{1}{1+e^{-(h_1.w_5+h_2.w_6+\text{bias})}}$$

This gives the final output of the network after applying the sigmoid activation function in output layers, introducing the desired non-linearity.

DL | Types of Activation Functions in Deep Learning

Linear Activation Function

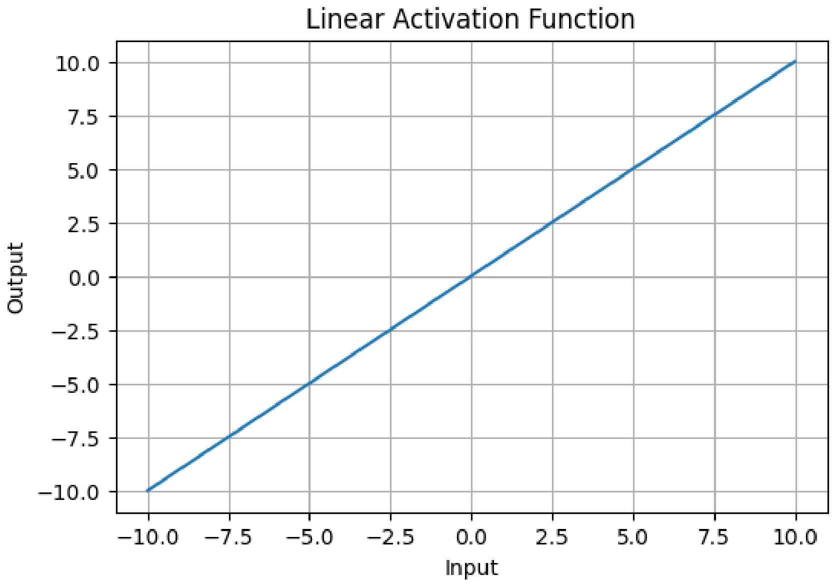
Linear Activation Function resembles straight line define by y=x. No matter how many layers the neural network contains, if they all use linear activation functions, the output is a linear combination of the input.

The range of the output spans from (−∞ to +∞).

Linear activation function is used at just one place i.e. output layer.

Using linear activation across all layers makes the network's ability to learn complex patterns limited.

Linear activation functions are useful for specific tasks but must be combined with non-linear functions to enhance the neural network's learning and predictive capabilities.

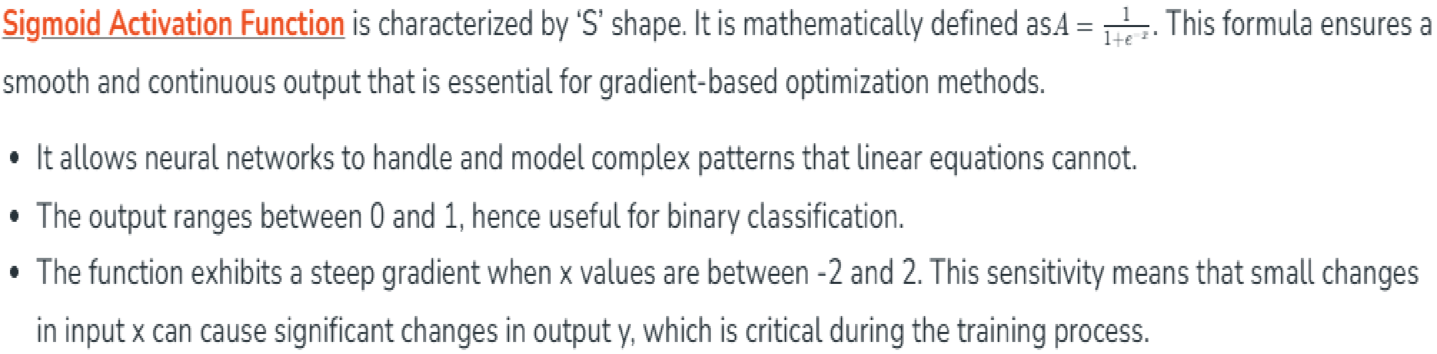


Reference:

https://www.geeksforgeeks.org/activation-functions-neural-networks/

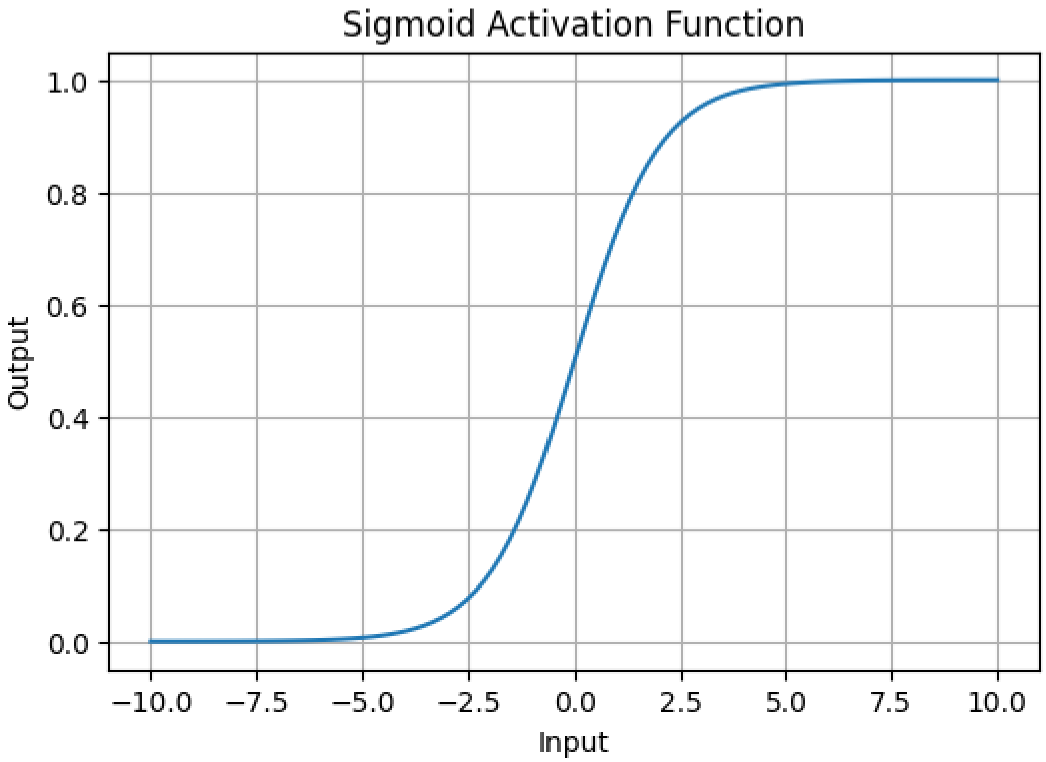
DL | Non-Linear Activation Functions

1. Sigmoid Function



Sigmoid Activation Function is characterized by 'S' shape. It is mathematically defined as$A = \frac{1}{1+e^{-x}}$. This formula ensures a smooth and continuous output that is essential for gradient-based optimization methods.

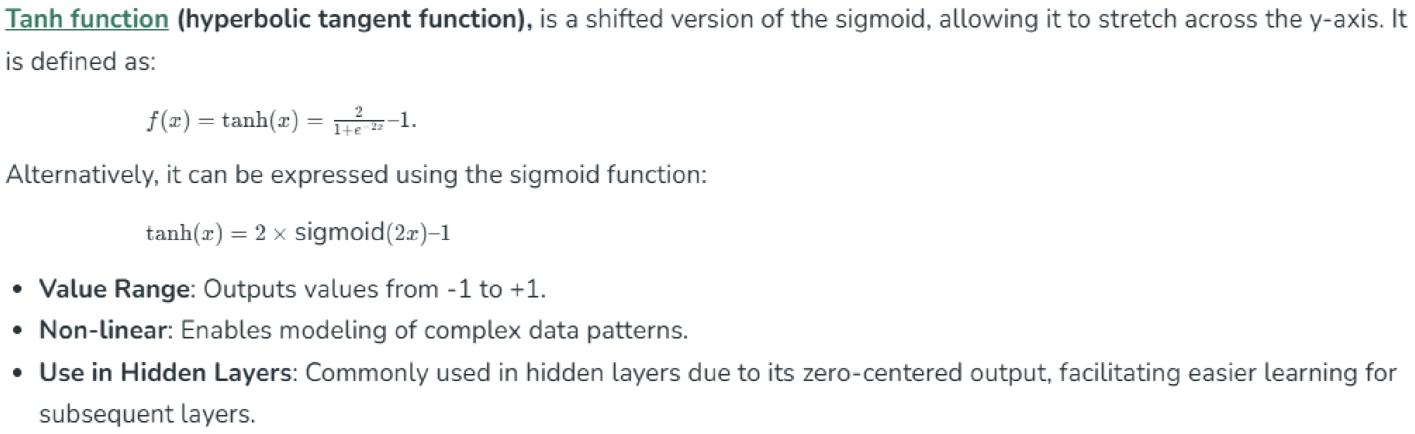
- It allows neural networks to handle and model complex patterns that linear equations cannot.
- The output ranges between 0 and 1, hence useful for binary classification.
- The function exhibits a steep gradient when x values are between -2 and 2. This sensitivity means that small changes in input x can cause significant changes in output y, which is critical during the training process.



Reference:
https://www.geeksforgeeks.org/derivative-of-the-sigmoid-function/

DL | Non-Linear Activation Functions

2. Tanh Activation Function



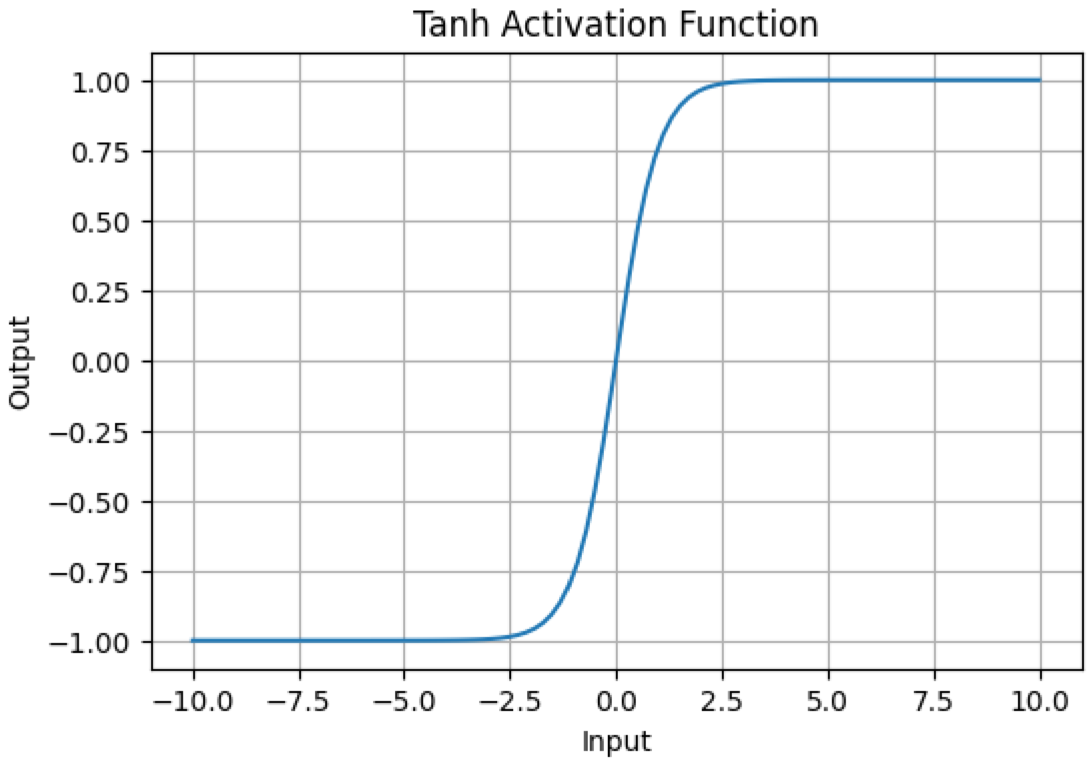
Tanh function **(hyperbolic tangent function),** is a shifted version of the sigmoid, allowing it to stretch across the y-axis. It is defined as:

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1.$$

Alternatively, it can be expressed using the sigmoid function:

$$\tanh(x) = 2 \times \text{sigmoid}(2x) - 1$$

- **Value Range**: Outputs values from -1 to +1.
- **Non-linear**: Enables modeling of complex data patterns.
- **Use in Hidden Layers**: Commonly used in hidden layers due to its zero-centered output, facilitating easier learning for subsequent layers.

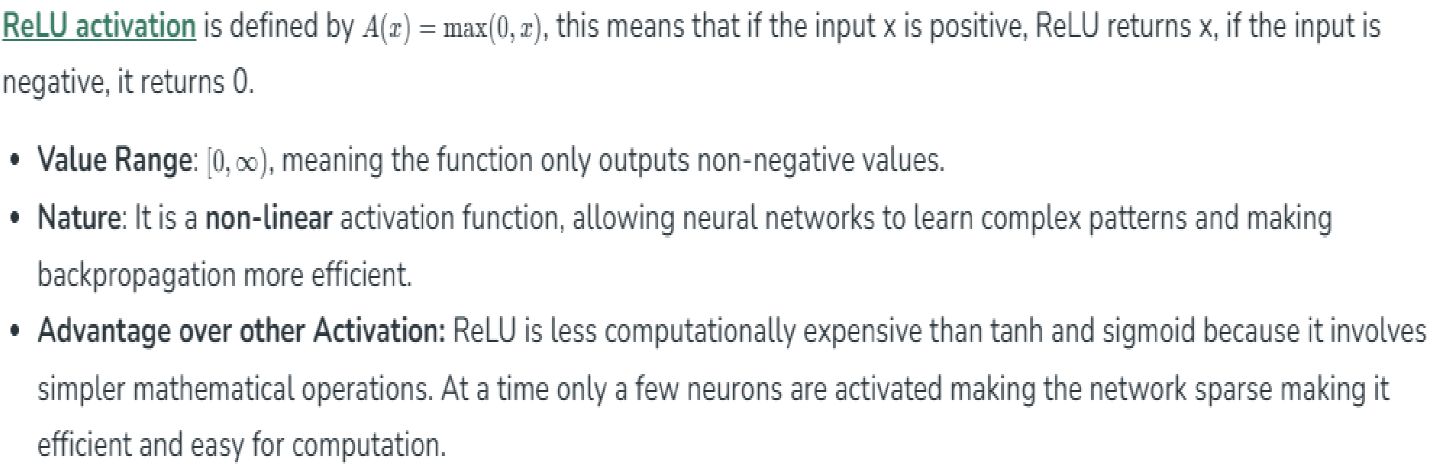


Reference:

https://www.geeksforgeeks.org/tanh-activation-in-neural-network/

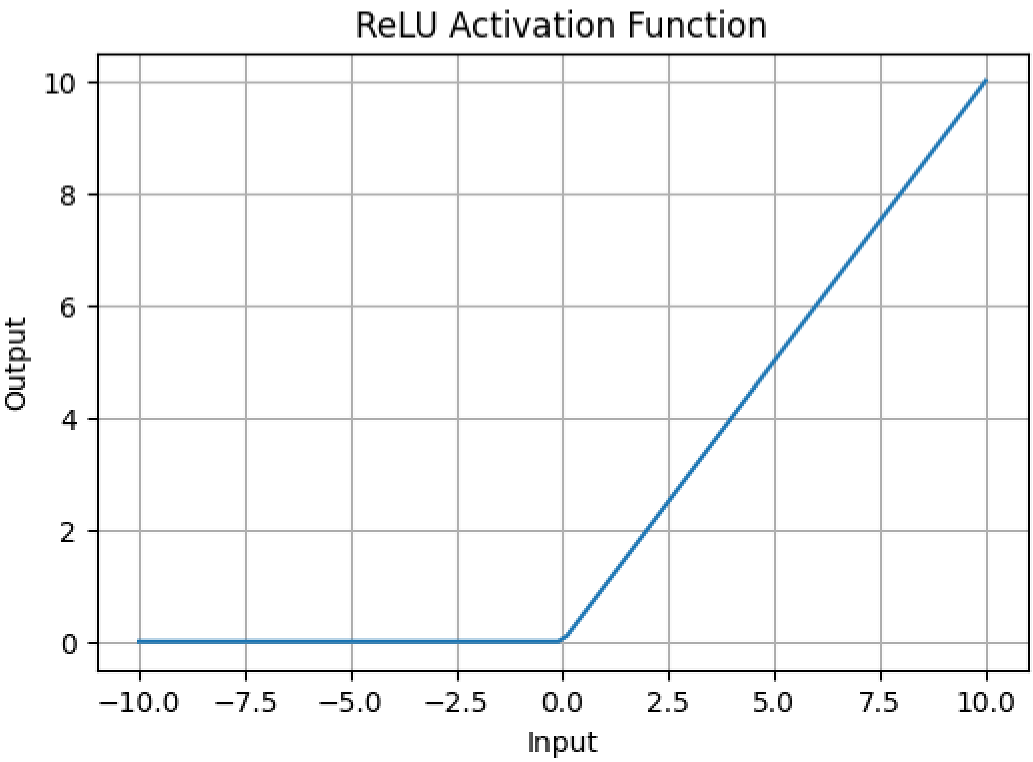
DL | Non-Linear Activation Functions

3. ReLU (Rectified Linear Unit) Function



ReLU activation is defined by $A(x) = \max(0, x)$, this means that if the input x is positive, ReLU returns x, if the input is negative, it returns 0.

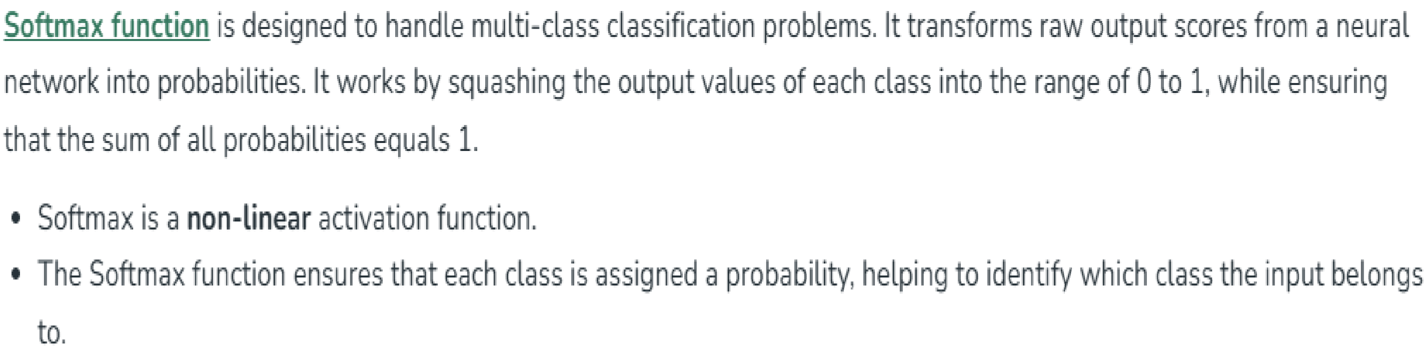
- **Value Range**: $[0, \infty)$, meaning the function only outputs non-negative values.
- **Nature**: It is a **non-linear** activation function, allowing neural networks to learn complex patterns and making backpropagation more efficient.
- **Advantage over other Activation:** ReLU is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.



Reference:
https://www.geeksforgeeks.org/relu-activation-function-in-deep-learning/

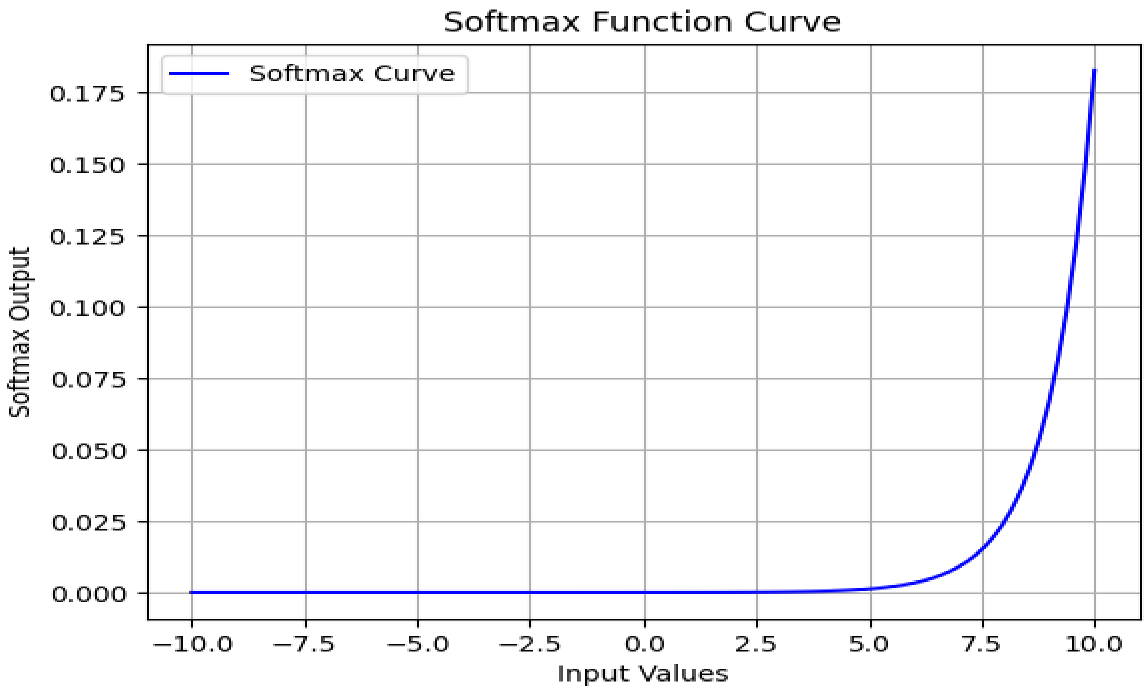
DL | Exponential Linear Units

1. Softmax Function



Softmax function is designed to handle multi-class classification problems. It transforms raw output scores from a neural network into probabilities. It works by squashing the output values of each class into the range of 0 to 1, while ensuring that the sum of all probabilities equals 1.

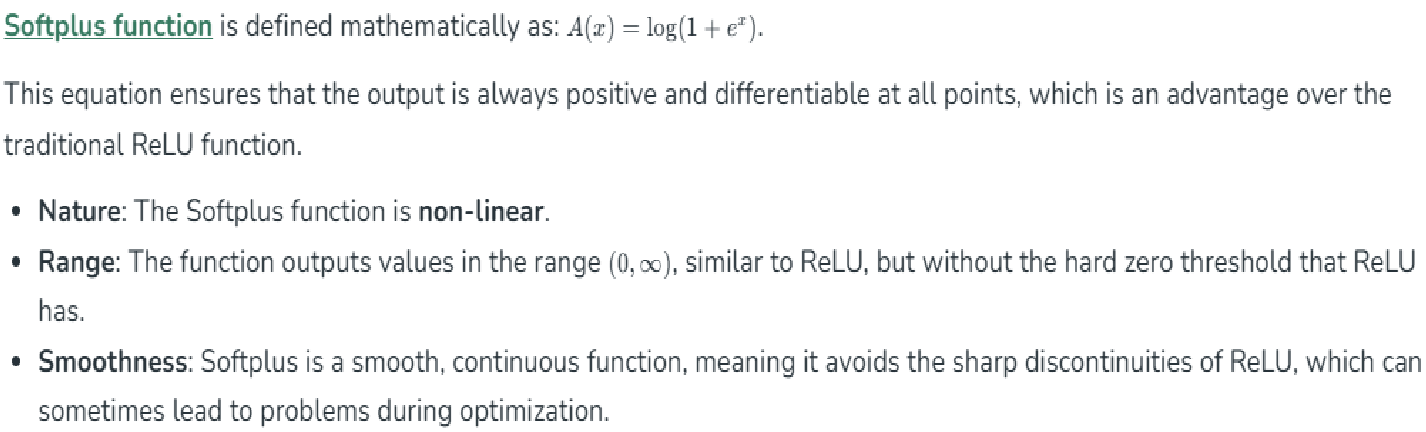
- Softmax is a **non-linear** activation function.
- The Softmax function ensures that each class is assigned a probability, helping to identify which class the input belongs to.



Reference:
https://www.geeksforgeeks.org/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applications/

DL | Exponential Linear Units

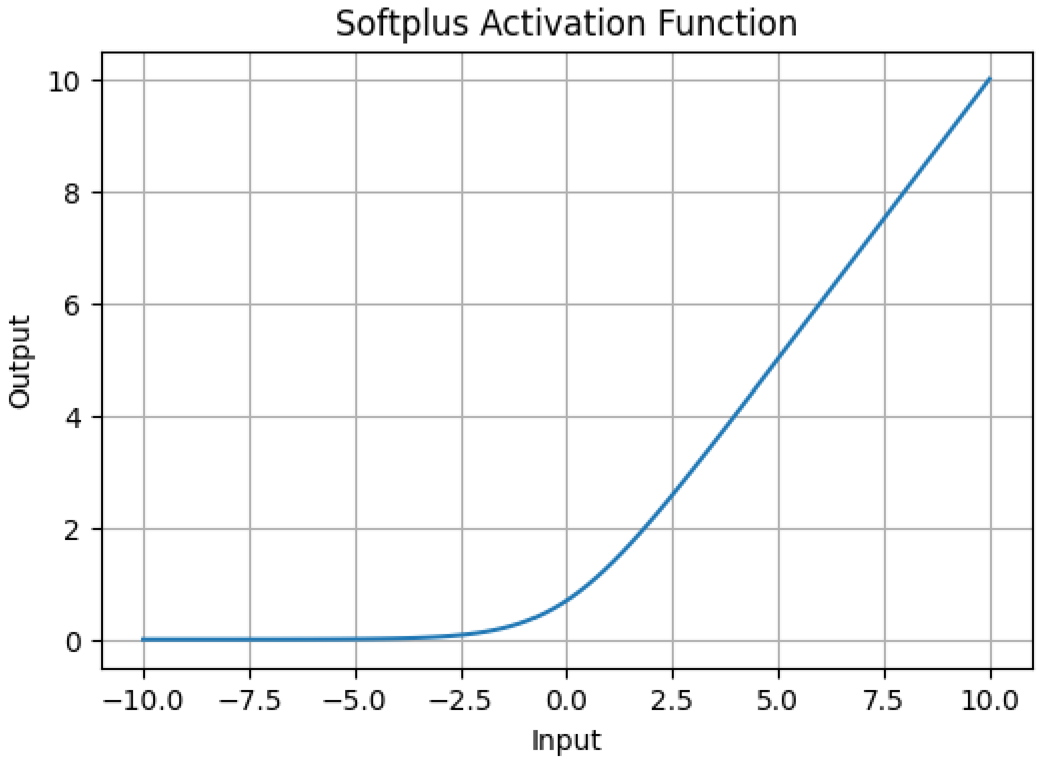
2. SoftPlus Function



Softplus function is defined mathematically as: $A(x) = \log(1 + e^x)$.

This equation ensures that the output is always positive and differentiable at all points, which is an advantage over the traditional ReLU function.

- **Nature**: The Softplus function is **non-linear**.
- **Range**: The function outputs values in the range $(0, \infty)$, similar to ReLU, but without the hard zero threshold that ReLU has.
- **Smoothness**: Softplus is a smooth, continuous function, meaning it avoids the sharp discontinuities of ReLU, which can sometimes lead to problems during optimization.



Reference:
https://www.geeksforgeeks.org/softplus-function-in-neural-network//

DL | Impact of Activation Functions on Model Performance

The choice of activation function has a direct impact on the performance of a neural network in several ways:

1. **Convergence Speed:** Functions like **ReLU** allow faster training by avoiding the vanishing gradient problem,
while **Sigmoid** and **Tanh** can slow down convergence in deep networks.

2. **Gradient Flow:** Activation functions like **ReLU** ensure better gradient flow, helping deeper layers learn effectively. In contrast, **Sigmoid** can lead to small gradients, hindering learning in deep layers.

3. **Model Complexity:** Activation functions like **Softmax** allow the model to handle complex multi-class problems, whereas simpler functions like **ReLU** or **Leaky ReLU** are used for basic layers.

Activation functions are the backbone of neural networks, enabling them to capture non-linear relationships in data. From classic functions like Sigmoid and Tanh to modern variants like ReLU and Swish, each has its place in different types of neural networks. The key is to understand their behavior and choose the right one based on your model's needs.

DL | Loss Function / Gradient descent - in Deep Learning

Loss functions in neural networks are mathematical tools that quantify the discrepancy between the predicted output of the network and the actual target values (ground truth). They serve as a measure of the model's performance during training, providing a numerical value that the optimization algorithm, typically gradient descent or its variants, aims to minimize.

Gradient descent is an optimization algorithm. It is used to find the minimum value of a function more quickly. The definition of gradient descent is rather simple. It is an algorithm to find the minimum of a convex function. To do this, it iteratively changes the parameters of the function in question. It is an algorithm that is used, for example, in linear regression.

Reference:
https://www.datacamp.com/tutorial/loss-function-in-machine-learning
https://www.geeksforgeeks.org/deep-learning/loss-functions-in-deep-learning/
https://builtin.com/machine-learning/loss-functions

Reference:
https://www.datacamp.com/tutorial/tutorial-gradient-descent

DL| Simple Test Case - Core - Exercise

See code here: https://github.com/ShahzadSarwar10/FullStackAI-B-4-SAT-SUN-10AM-TO-12PM/blob/main/Week16/Case16-3-TF-SimpleFeedforwardNeuralNetworksExampleCode.py

You should be able to analyze – each code statement, you should be able to see trace information – at each step of debugging. "DEBUGGING IS BEST STRATEGY TO LEARN A LANAGUAGE." So debug code files, line by line, analyze the values of variable – changing at each code statement. BEST STRATEGY TO LEARN DEEP.

Let's put best efforts.
Thanks.
Shahzad – Your AI – ML Instructor

Thank you - for listening and participating

- □**Questions / Queries**
- □**Suggestions/Recommendation**
- □**Ideas…..?**

Shahzad Sarwar
Cognitive Convergence
https://cognitiveconvergence.com
shahzad@cognitiveconvergence.com
voice: +1 4242530744 (USA) +92-3004762901 (Pak)