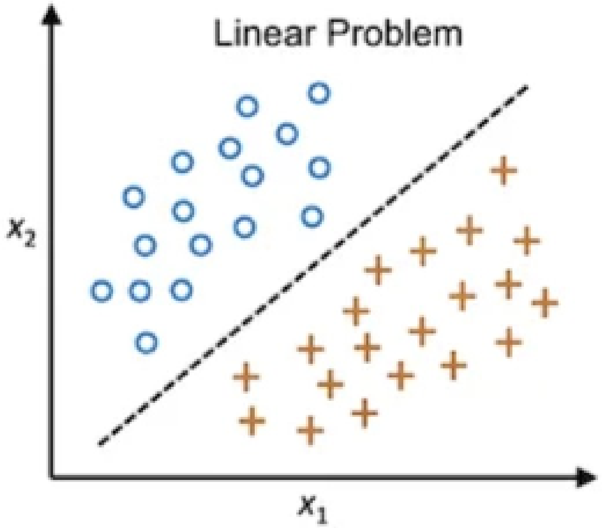
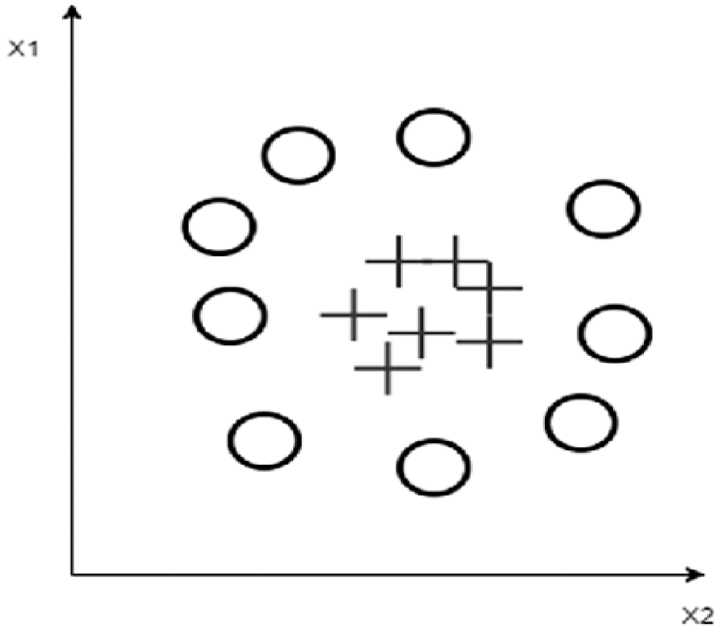
Nonlinearity

- Let the feature vectors are of 2-D and of two different classes.
- They may be linearly separable or nonlinearly separable.
- Here, it is a linear discriminator or classifier that can separate the samples of two different classes.
- For 2-D it is a straight line, for 3-D is a plane and for more dimensions it is a hyperplane.
- Here, the accuracy is 100%, i. e., without any error the



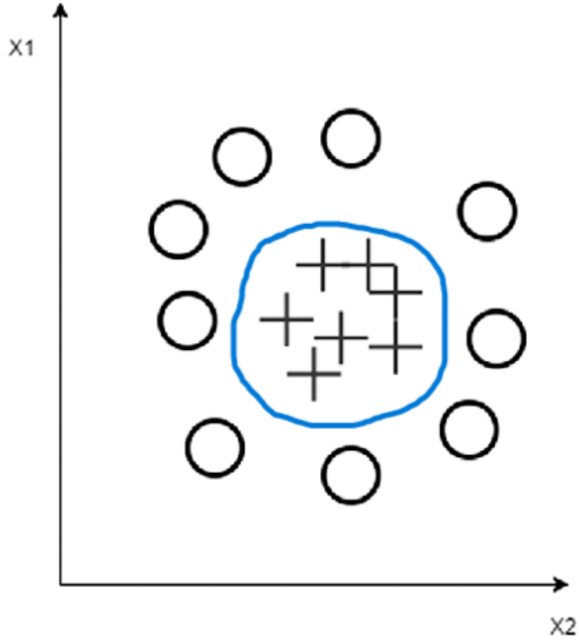
Nonlinearity



- Let the feature vectors distribution is as above.
- Let all the vectors of '+' symbol are of one class and all other of symbols '-' are of another class.
- Clearly they are not linearly separable.

Nonlinearity

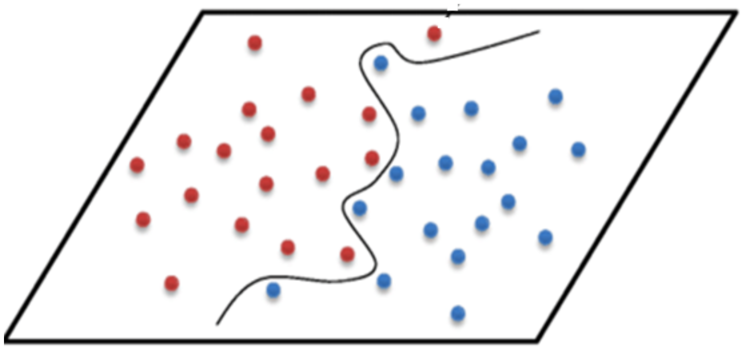
- Here, all the feature vectors of class ‘+’ are within a circle and all others are outside the circle.
- In case of 3-D, all samples of symbol ‘+’ are in a sphere and others are outside the sphere, and so on.
- So we may have many such complicated distribution of the feature vectors.
- Here, we cannot imagine any linear separator to separate the samples of



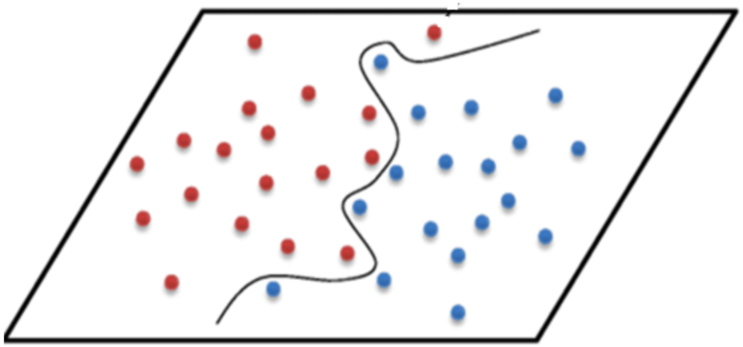
- So, if we train our linear classifier with such nonlinearly separable objects, there must be error or misclassification in the classifier.
- This is known as linearly non separable problem.

Nonlinearity

- Here, we cannot separate the classes without using nonlinear functions.
- What we can do in such cases?
- Let us consider the 2-D features for simplicity and samples of two different classes as shown below:.

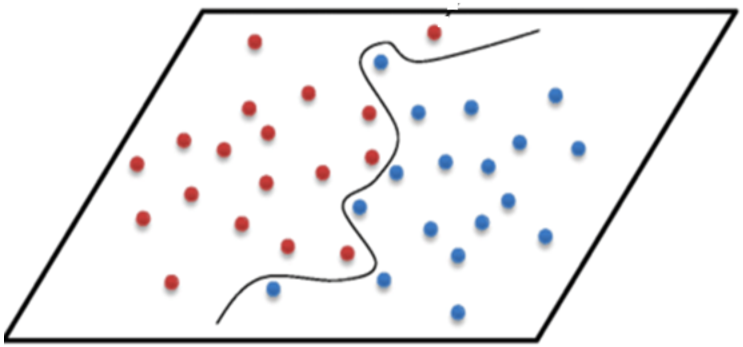


Nonlinearity



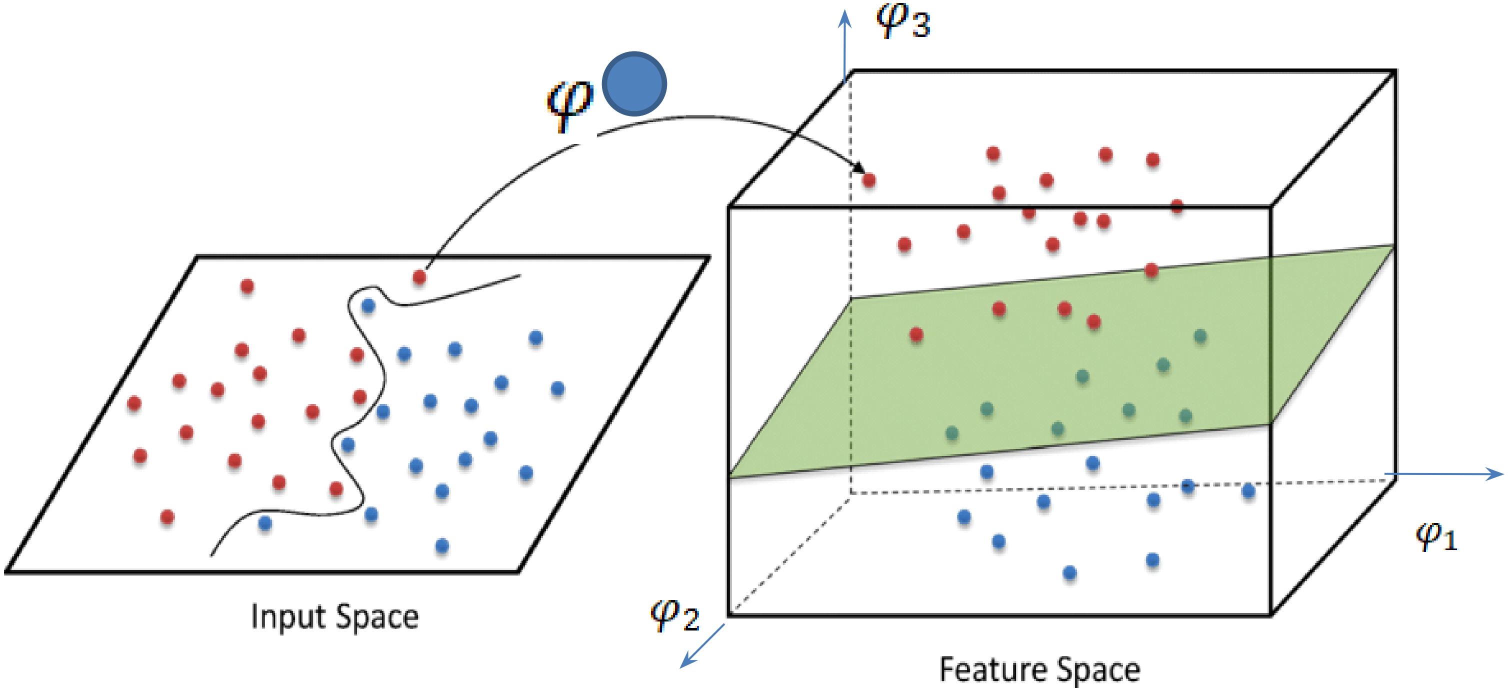
- Let, all samples right side of the classifier are of class C1, and others are of class C2.
- Here, we cannot find any linear classifier, i.e., straight line by which we can separate the samples of two classes, i.e., they are not linearly separable.
- Actually, they are nonlinearly separable.

Nonlinearity



- In fact, there is a curve (i.e., nonlinear function) by which the samples are separable.
- So, it is a nonlinear problem.
- But still possible to solve this problem using linear classifier – How?

Nonlinearity



- Use nonlinear mapping of the feature vectors, so that the feature vectors of 2-D are mapped into 3-D.
- Here, say they mapped in such a way that the they are now separable by a plane, as shown in the right side of the figure.

Nonlinearity

- So we are increasing the dimension of the feature vectors using nonlinear mapping.
- So even though the samples of different classes are mixed with each other, still without using nonlinear classifier, it is possible to separate the samples by using the nonlinear mapping.
- Nonlinear mapping maps the feature vectors or samples from lower dimensional space to higher dimensional space where they are separable by linear classifier.

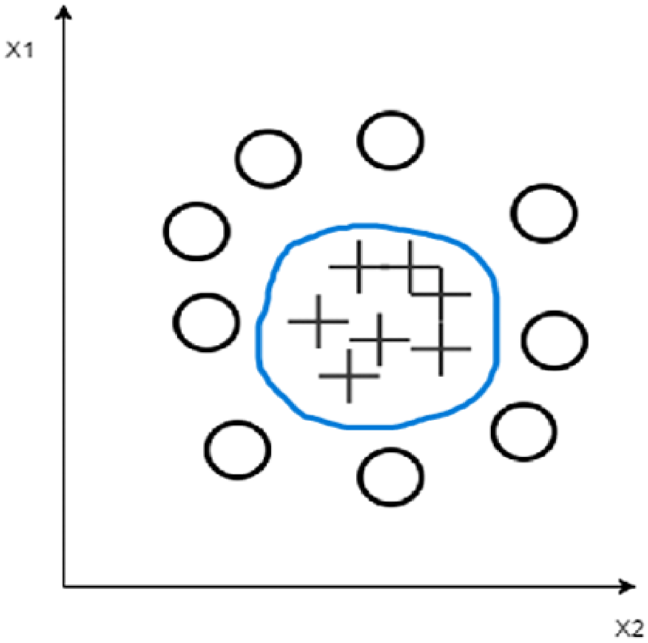
Nonlinearity

- Let us consider the previous 2-D samples.
- Here, all samples within circle are of class C1 and others are of C2.
- Nonlinear mapping is done using a nonlinear function, say

φ

.

- Here, we have the samples of two dimensions, X_1 and X_2.
- Let, cφe sample is $X= <x_1, x_2 >$, then is defined as $\varphi\ (X) = <\varphi_1(X), \varphi_2(X), \varphi_3(X)>$



- So we are considering 3-Ds with a nonlinear mapping on 2-D such th$\varphi\ (X) = <\varphi_1(X), \varphi_2(X), \varphi_3(X)>$

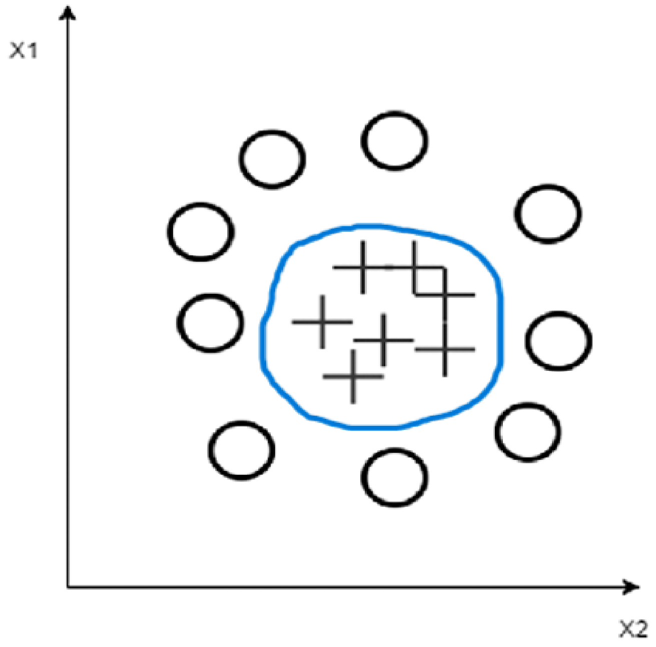
$$\varphi_1(X) = X_1$$

$$\varphi_2(X) = X_2$$

$$\varphi_3(X) = {X_1}^2 + {X_2}^2$$

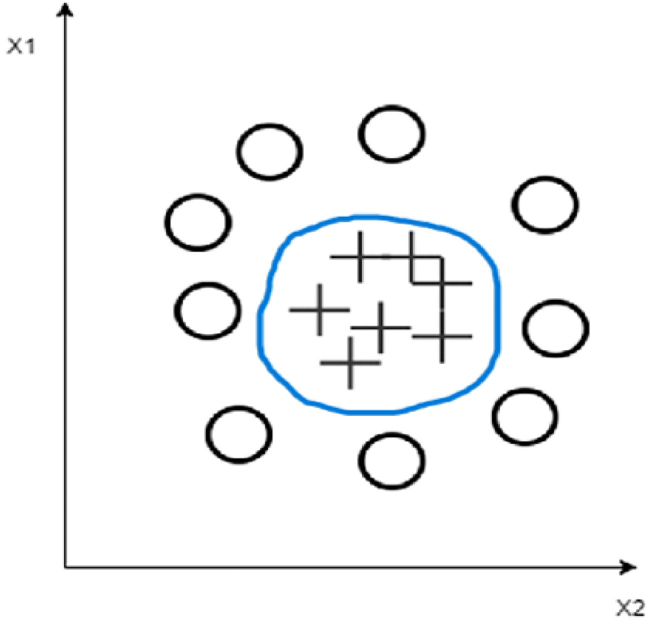
Nonlinearity

- Here, equation of the circle is $X_1^{\,2} + X_2^{\,2} = R^2$, where R is the radius of the circle.
- If a sample point (x_1, x_2) lies inside, then $X_1^{\,2} + X_2^{\,2} < R^2$
- If a sample point (x_1, x_2) lies outside, then $X_1^{\,2} + X_2^{\,2} > R^2$
- If a sample point (x_1, x_2) lies on the circle, then $X_1^{\,2} + X_2^{\,2} = R^2$



Nonlinearity

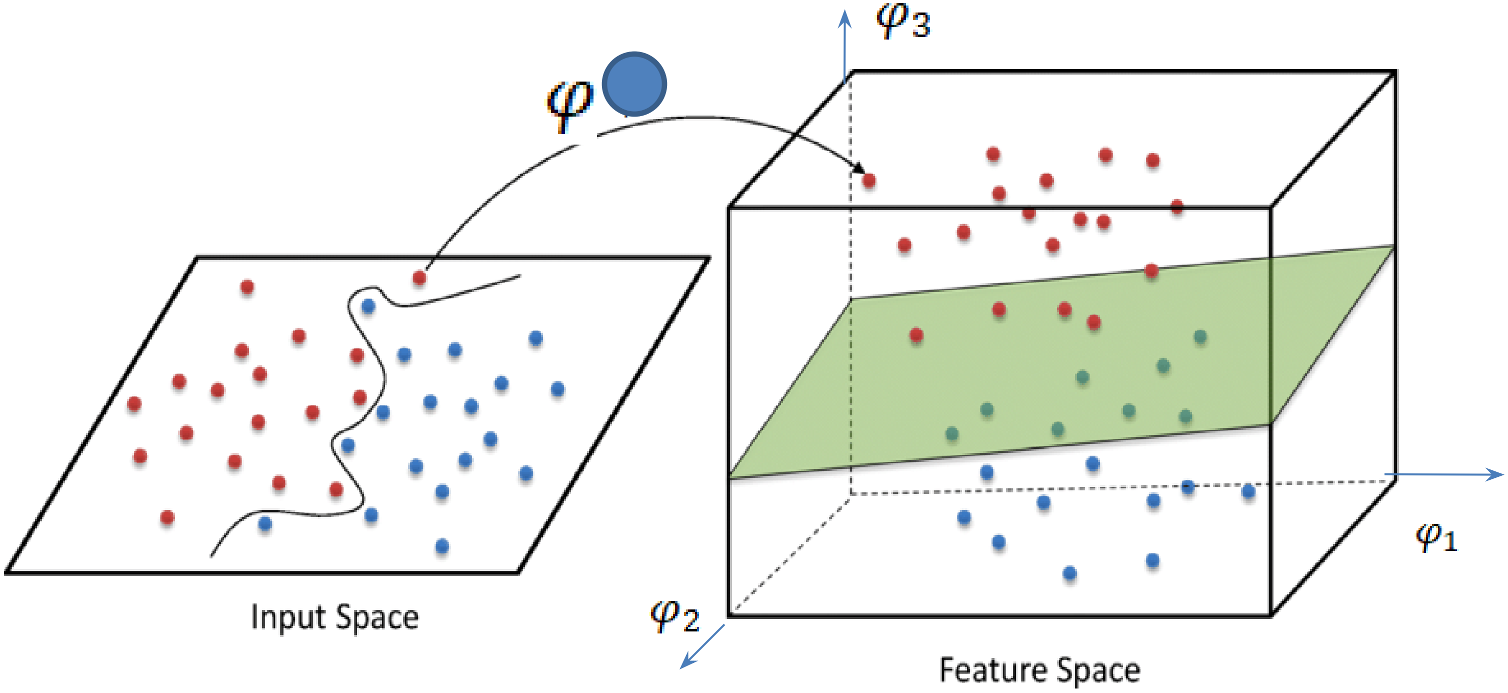
- So value of $\varphi(x_1, x_2)$ for any sample within the circle is always less than that of outside samples.
- So the third coordinate of all samples outside the circle (class C2) is greater than that of all samples inside the circle (class C1).
- The feature vectors in 3-D are of the form $< X_1, X_2, {X_1}^2 + {X_2}^2 >$
- Thus our sample points in 2-D are nonlinearly mapped to 3-D space by the nonlinear mapping: $\varphi\,(X) = < \varphi_1(X), \varphi_2(X), \varphi_3(X) >$
  $\varphi_1(X) = X_1$ $\varphi_2(X) = X_2$where $\varphi_3(X) = {X_1}^2 + {X_2}^2$ and for any feature vector X = [X_1, X_2].



- Value of the third coordinate is higher for all the samples of class C2.
- So there must be a plane by which the samples of class C1 is linearly separable from samples of class C2, such that samples lower to the plane are of class C1 and upper to the plane of class C2.

Nonlinear to linear transformation

- Thus, it is possible to transform the nonlinearly separable lower dimensional feature vectors to higher dimensional feature vectors using nonlinear mapping so that they are linearly separable by plane in 3-D, and hyperplane in higher dimensions.

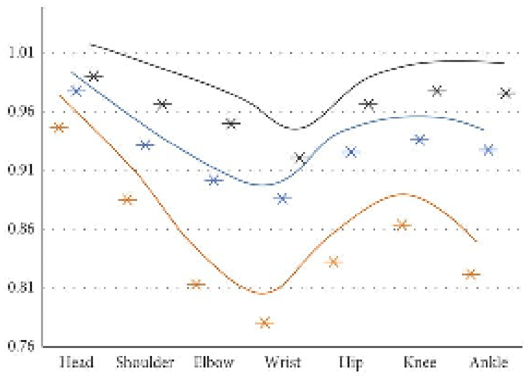


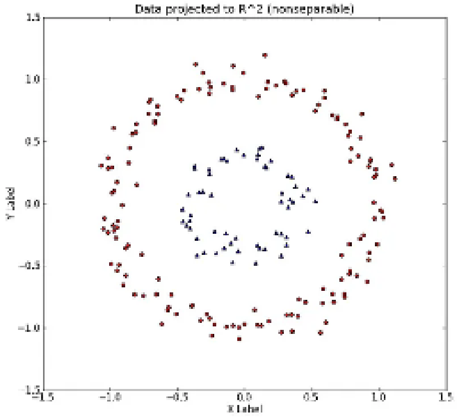
Nonlinear to linear transformation

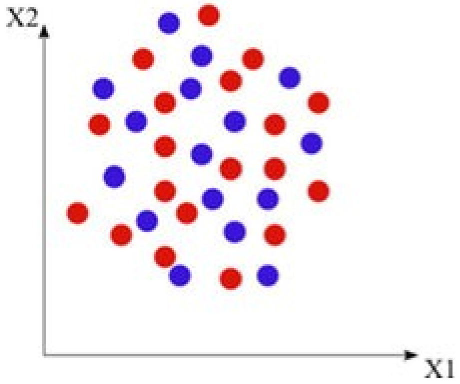
- So this transformation is done by nonlinear mapping $\varphi(X) = <\varphi_1(X), \varphi_2(X), \varphi_3(X)>$ where X=[X_1, X_2], such $\varphi_1(X) = X_1$ $\varphi_2(X) = X_2$ $\varphi_3(X) = {X_1}^2 + {X_2}^2$
- Thus we map the sample X = [X_1, X_2] in 2-D to $\varphi(X) = <\varphi_1(X), \varphi_2(X), \varphi_3(X)>$ in 3-D using the nonlinear φ mapping .
- The hyperplane in 3-D may be written as $w_1\varphi_1 + w_2\varphi_2 + w_3\varphi_3 = 0$ => $w_1X_1 + w_2X_2 + w_3({X_1}^2 + {X_2}^2) = 0$ by which the feature vectors are linearly separable.

Nonlinear to linear transformation

- This type nonlinearity mapping from lower dimension to higher dimension is not so simple.
- But theoretically, it is proven that if we map a feature vector to a infinite dimension space then whatever be the complexity of the nonlinearity, every nonlinear problem can be solved as a linear problem. So for nonlinear mapping we need some nonlinear functions.







Some nonlinear functions

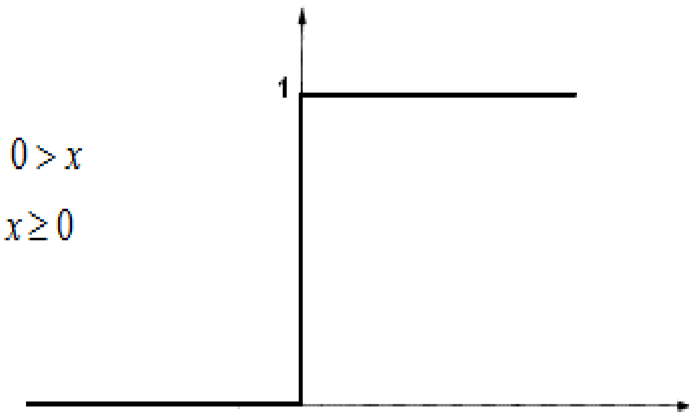
- The nonlinear functions are also used in deep learning as activation functions to handle nonlinearity. Some of these types of nonlinear functions ar

1. Threshold Function:

- This is the simplest type of nonlinearity function.

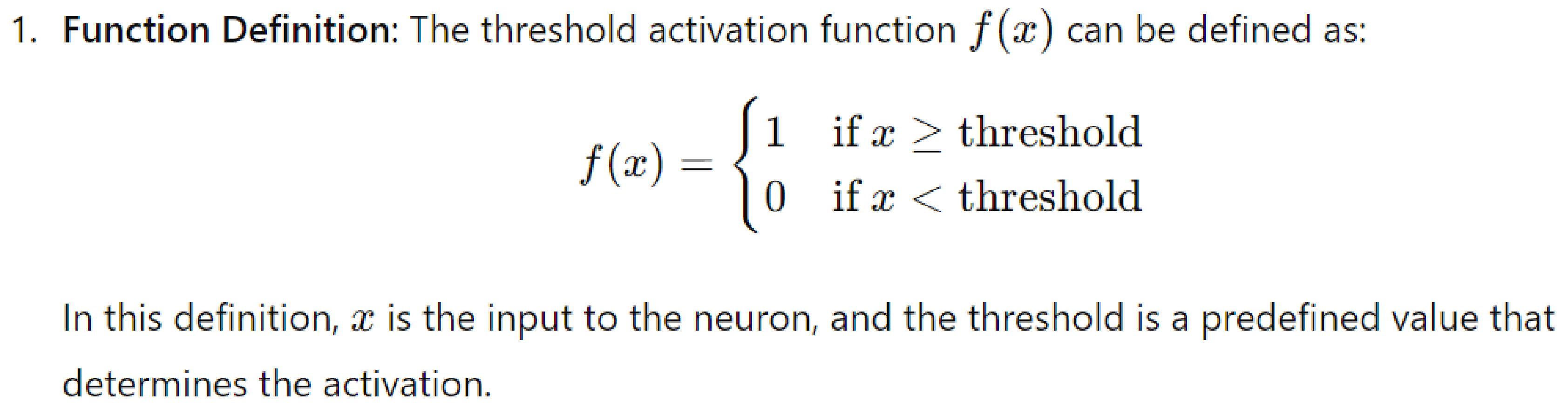
$$f(x)=\begin{cases}0 \text{ if } 0>x\\1 \text{ if } x\geq 0\end{cases}$$

Unit step (threshold)



1. Threshold Function

- The threshold activation function, also known as the step function, is a type of activation function used in artificial neural networks.
- It's a simple function that activates a neuron only when the input exceeds a certain threshold. General definition is:

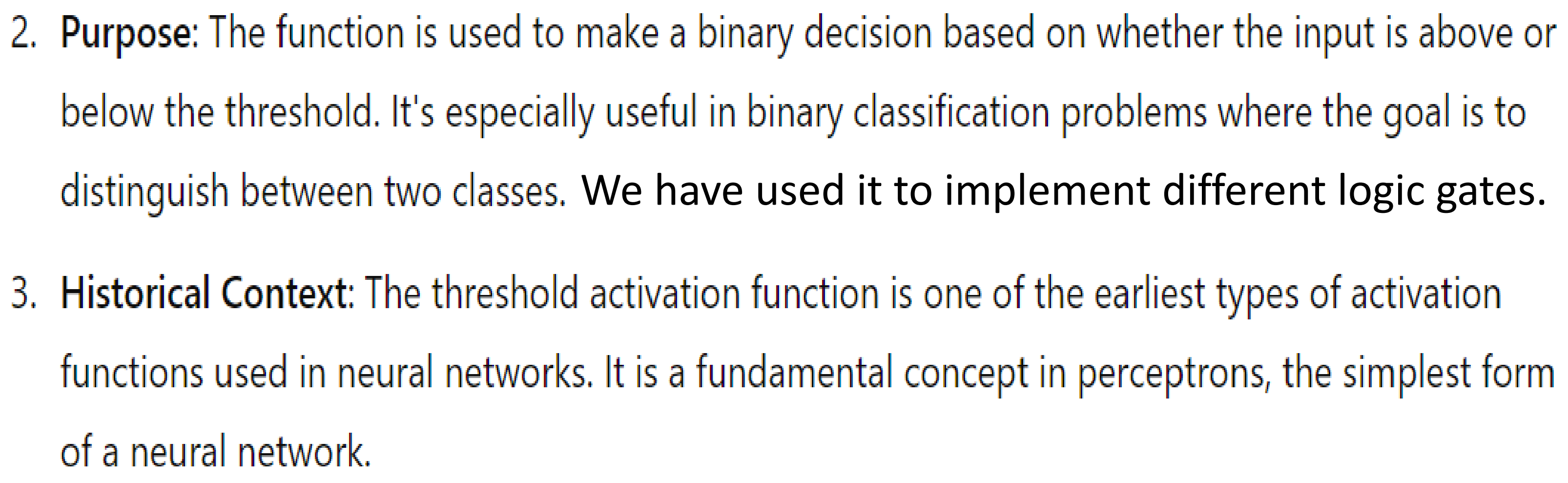


1. **Function Definition:** The threshold activation function $f(x)$ can be defined as:

$$f(x) = \begin{cases} 1 & \text{if } x \geq \text{threshold} \\ 0 & \text{if } x < \text{threshold} \end{cases}$$

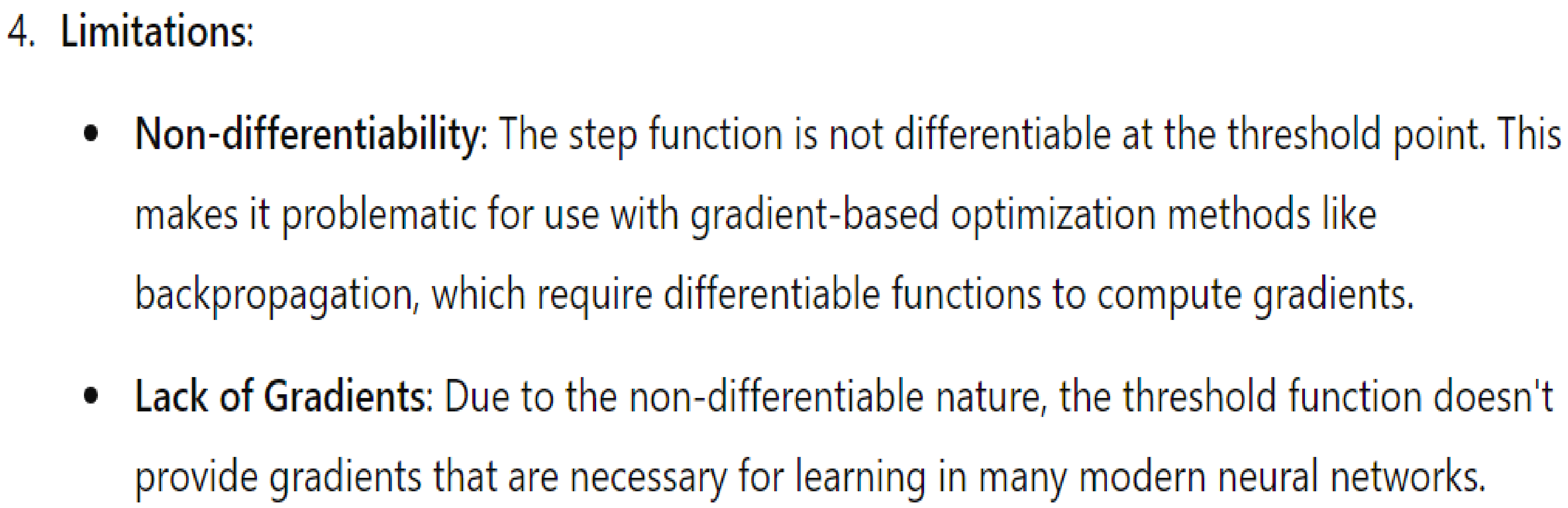
In this definition, x is the input to the neuron, and the threshold is a predefined value that determines the activation.

1. Threshold Function



2. **Purpose**: The function is used to make a binary decision based on whether the input is above or below the threshold. It's especially useful in binary classification problems where the goal is to distinguish between two classes. We have used it to implement different logic gates.

3. **Historical Context**: The threshold activation function is one of the earliest types of activation functions used in neural networks. It is a fundamental concept in perceptrons, the simplest form of a neural network.



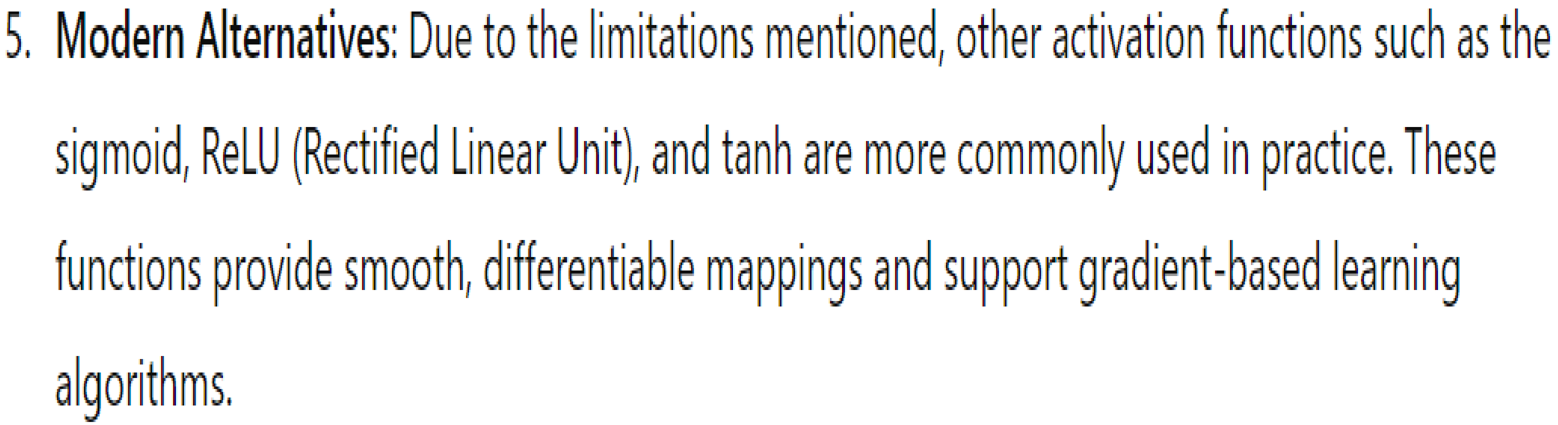
4. **Limitations**:

- **Non-differentiability**: The step function is not differentiable at the threshold point. This makes it problematic for use with gradient-based optimization methods like backpropagation, which require differentiable functions to compute gradients.

- **Lack of Gradients**: Due to the non-differentiable nature, the threshold function doesn't provide gradients that are necessary for learning in many modern neural networks.

1. Threshold Function

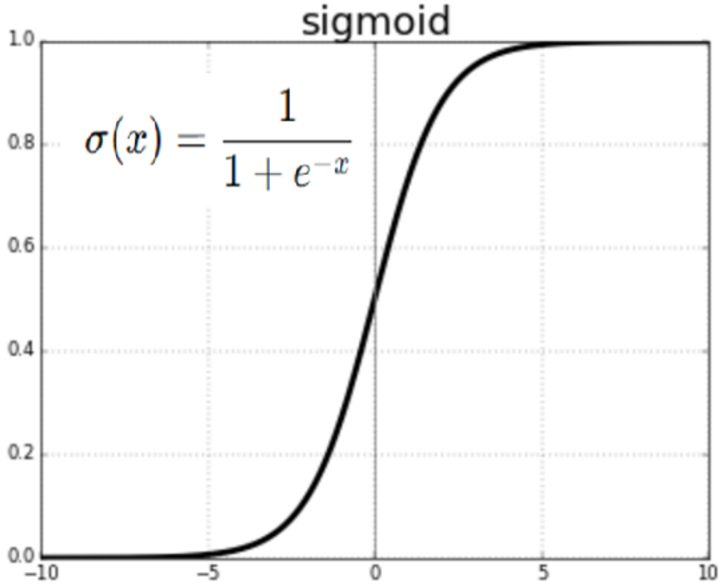
- Despite its simplicity, the threshold activation function is a foundational concept that has paved the way for more advanced and sophisticated activation functions used in contemporary neural networks.



5. **Modern Alternatives:** Due to the limitations mentioned, other activation functions such as the sigmoid, ReLU (Rectified Linear Unit), and tanh are more commonly used in practice. These functions provide smooth, differentiable mappings and support gradient-based learning algorithms.

2. Sigmoid Function

- We have discussed it during construction of logistic regression model.
- The sigmoid activation function is a widely used activation function in neural networks, particularly in the context of binary classification problems.
- It maps any real-valued number into a value between 0 and 1, which is useful for scenarios where we want to model probabilities or outcomes that need to be in this range.



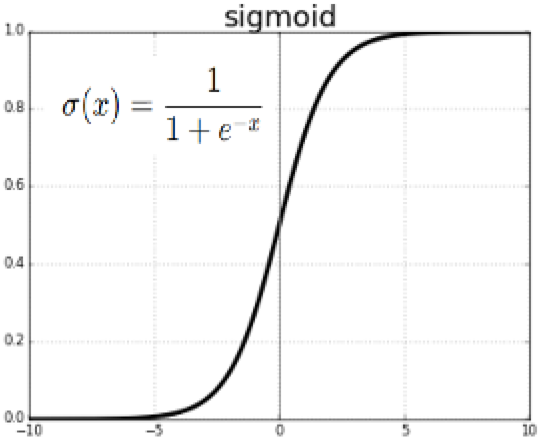
2. Sigmoid Function

Here's a detailed overview of the sigmoid activation function:

Definition

The sigmoid activation function $\sigma(x)$ is defined by the formula:

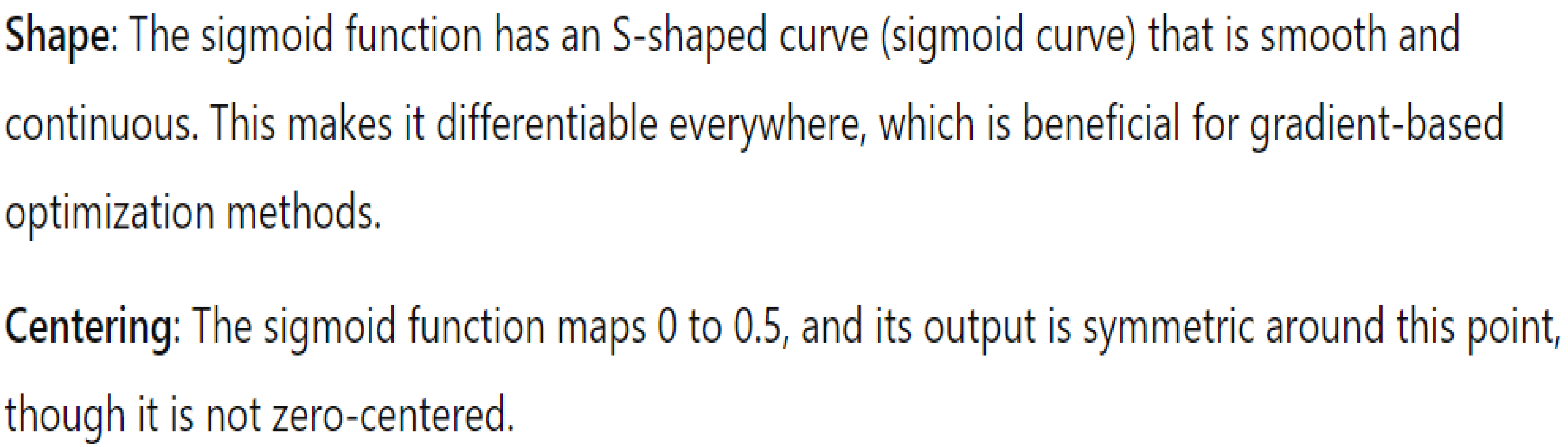
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



where e is the base of the natural logarithm (approximately 2.71828), and x is the input to the function.

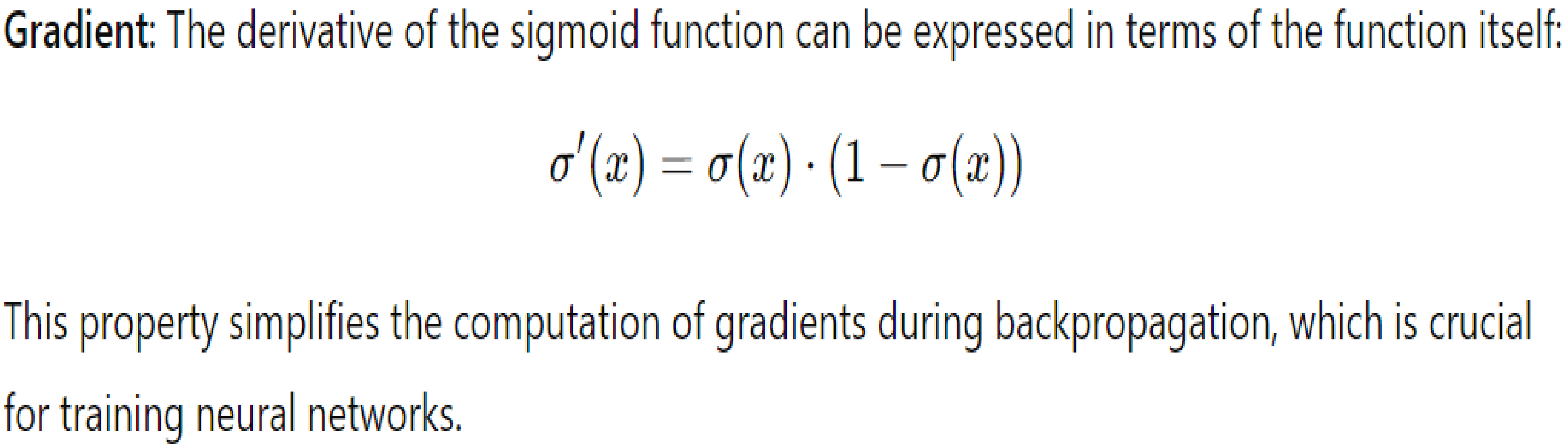
Range: The output of the sigmoid function is between 0 and 1. This is because the function asymptotically approaches 0 as x goes to negative infinity and approaches 1 as x goes to positive infinity.

2. Sigmoid Function



Shape: The sigmoid function has an S-shaped curve (sigmoid curve) that is smooth and continuous. This makes it differentiable everywhere, which is beneficial for gradient-based optimization methods.

Centering: The sigmoid function maps 0 to 0.5, and its output is symmetric around this point, though it is not zero-centered.



Gradient: The derivative of the sigmoid function can be expressed in terms of the function itself:

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

This property simplifies the computation of gradients during backpropagation, which is crucial for training neural networks.

2. Sigmoid Function

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

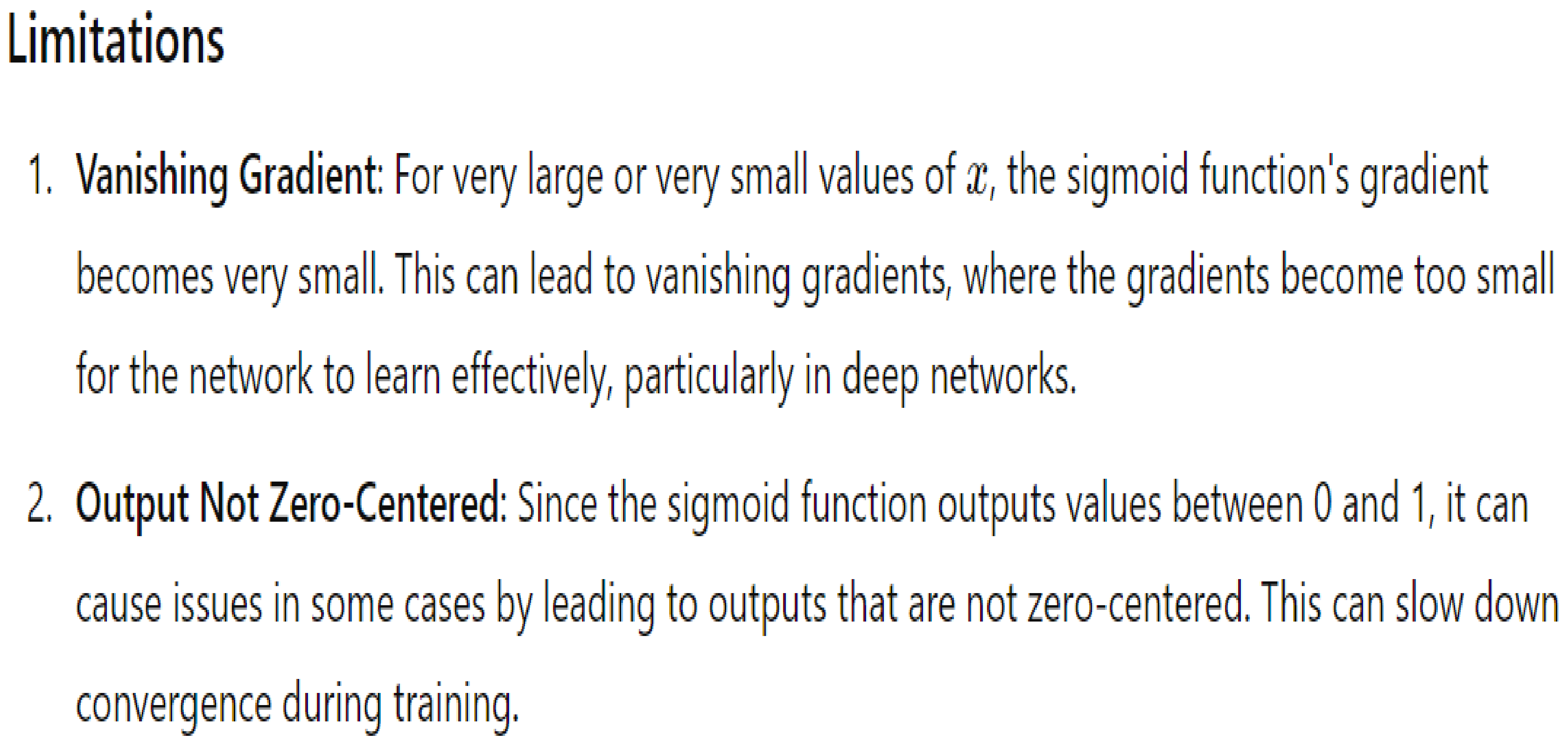
$$\sigma'(x) = +\frac{1}{(1+e^{-x})^2} e^{-x} = \frac{1}{1+e^{-x}} \frac{e^{-x}}{1+e^{-x}}$$

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

Use Cases

- **Binary Classification:** The sigmoid function is often used in the output layer of a neural network for binary classification tasks. The output can be interpreted as a probability that the input belongs to the positive class.
- **Logistic Regression:** It is also used in logistic regression models, where the output represents the probability of the positive class given the input features.

2. Sigmoid Function



Limitations

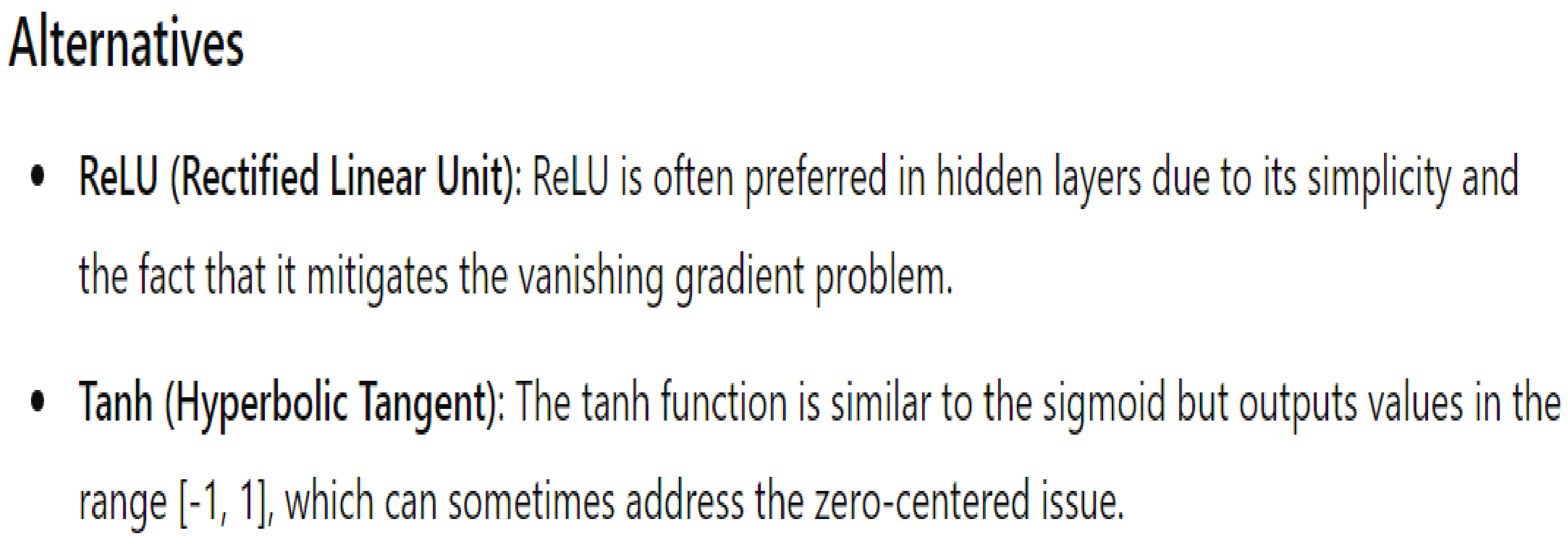
1. **Vanishing Gradient:** For very large or very small values of x, the sigmoid function's gradient becomes very small. This can lead to vanishing gradients, where the gradients become too small for the network to learn effectively, particularly in deep networks.
2. **Output Not Zero-Centered:** Since the sigmoid function outputs values between 0 and 1, it can cause issues in some cases by leading to outputs that are not zero-centered. This can slow down convergence during training.

Non-Zero-Centered Output

- Non-Zero-Centered Output implies the range of values the function produces. The sigmoid function's output is not distributed around zero. Here, the inputs map to values in the range [0,1]. This means that the function only produces positive outputs, with no values below zero. This can slow down the convergence during training.

  - **Bias in Gradient Updates**: The gradients can become skewed because the outputs are always positive. This can lead to a situation where the network's updates are not symmetric, potentially causing inefficient learning and slower convergence.
  - Example: Consider a simple neural network with sigmoid activation functions in its hidden layers. The activations will be in the range of 0 to 1, meaning that any neuron output will be positive. If you have a large number of neurons in the network, the activations can become biased towards higher values, and the gradients may not be centered around zero, leading to less effective gradient updates.

2. Sigmoid Function



Alternatives

- **ReLU (Rectified Linear Unit):** ReLU is often preferred in hidden layers due to its simplicity and the fact that it mitigates the vanishing gradient problem.
- **Tanh (Hyperbolic Tangent):** The tanh function is similar to the sigmoid but outputs values in the range [-1, 1], which can sometimes address the zero-centered issue.

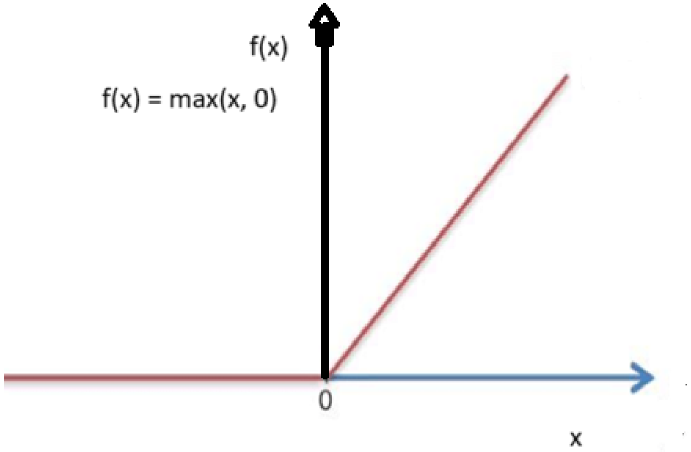
- Conclusion: The sigmoid function is a fundamental concept in neural network design and provides a good starting point for understanding activation functions and their role in network training.

3. Rectifier Function

- Rectifier functions are often called Rectified Linear Unit activation functions, or ReLUs for short.
- A Rectified Linear Unit (ReLU) is a widely used activation function in neural networks.
- It helps introduce non-linearity into the model, allowing it to learn complex patterns in data.
- The ReLU function is defined mathematically as:

  f(x) = max(0, x)

- This means that if the input x is positive, the output is x. If x is negative, the output is 0.



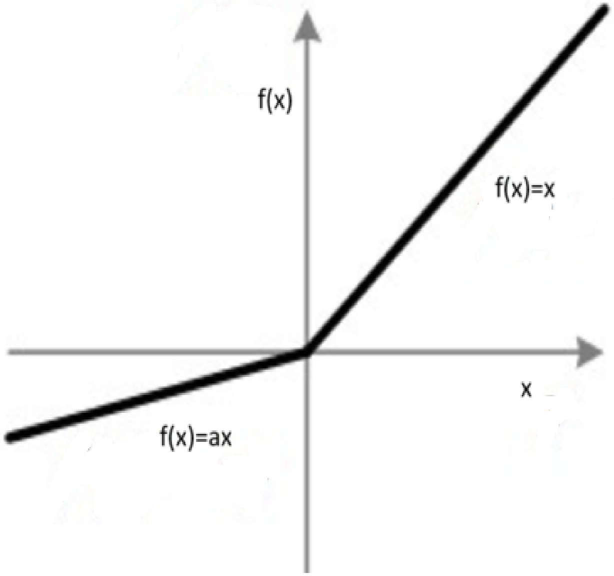
Key Features of ReLU:

- **Non-linearity**: Despite being a simple piecewise linear function, ReLU introduces non-linearity, which is crucial for learning complex patterns.
- **Sparsity**: It leads to sparse activation. For any given input, a significant portion of neurons will be inactive (outputting zero), which can help with efficient computation and reduce overfitting.
- **Computational Efficiency**: ReLU is computationally efficient because it only involves a simple comparison operation.
- **But the disadvantage of ReLU is that it can** lead to "dying ReLU" problem where neurons output zero and stop learning. Variants like Leaky ReLU or ELU can help mitigate this issue.

Leaky ReLU

- Leaky ReLU is a variant of the standard ReLU activation function designed to address the "dying ReLU" problem, where neurons can become inactive and stop learning if their outputs are zero.
- Leaky ReLU introduces a small, non-zero slope for negative inputs, which helps keep the neurons active during training.
- It is defined mathematically as:

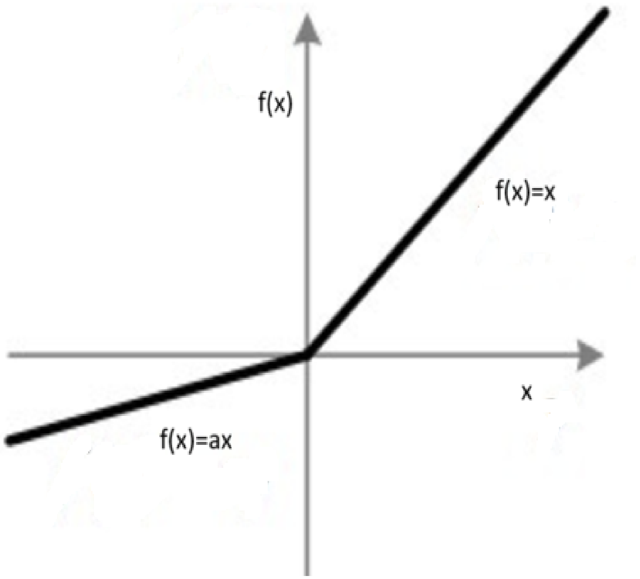
  f(x)=max(a*x , x)



- It returns x if it receives any positive input, but for any negative value of x, it returns a really small value which is ’a’ times x.
- Thus it gives an output for negative values as well.

Leaky ReLU

- Thus, Leaky ReLU has a small slope for negative values instead of a flat slope.
- The slope coefficient (a) is determined before training, i.e. it is not learnt during training.
- This type of activation function is popular in tasks where we may suffer from sparse gradients, for example training generative adversarial networks.



What is Sparse Gradient?

- **Gradient**: In the context of machine learning and neural networks, the gradient of a function (such as a loss function) with respect to a model's parameters indicates how much the function would change if the parameters were adjusted. The gradient is used to update the parameters during training.
- **Sparsity**: Sparse gradients are those where many of the gradient values are zero or very small. This often means that only a few parameters are actively contributing to the loss, and the rest do not change significantly in response to small variations in the parameters.

How Sparse Gradients Arise?

- **Activation Functions:** Activation function like ReLU can lead to sparse gradients. For example, ReLU outputs zero for all negative inputs, which can cause gradients to be zero for neurons with negative activations.
- **Feature Selection**: In models where feature selection or sparsity is encouraged (e.g., through L1 regularization), some parameters or features may end up with gradients of zero, reflecting that they do not contribute to the loss function.
- **Optimization Algorithms**: Algorithms like stochastic gradient descent (SGD) might result in sparse gradients due to the nature of the data batches used for updates.

Advantages of Sparse Gradient

- **Computational Efficiency**: Sparse gradients can lead to more efficient computations and updates since fewer parameters need to be adjusted. This can be beneficial in high-dimensional spaces where only a small subset of parameters are active.
- **Reduced Overfitting**: Sparse gradients might help with model generalization by focusing the learning process on a smaller number of important parameters, potentially reducing overfitting.

Disadvantages of Sparse Gradient

- **Learning Dynamics**: If too many parameters have sparse gradients, the model might learn slowly or inadequately if the gradients are not informative enough about the loss function.
- **Gradient Noise**: Sparse gradients can introduce noise into the training process, particularly if the gradients are zero for many parameters, making the optimization process less stable.

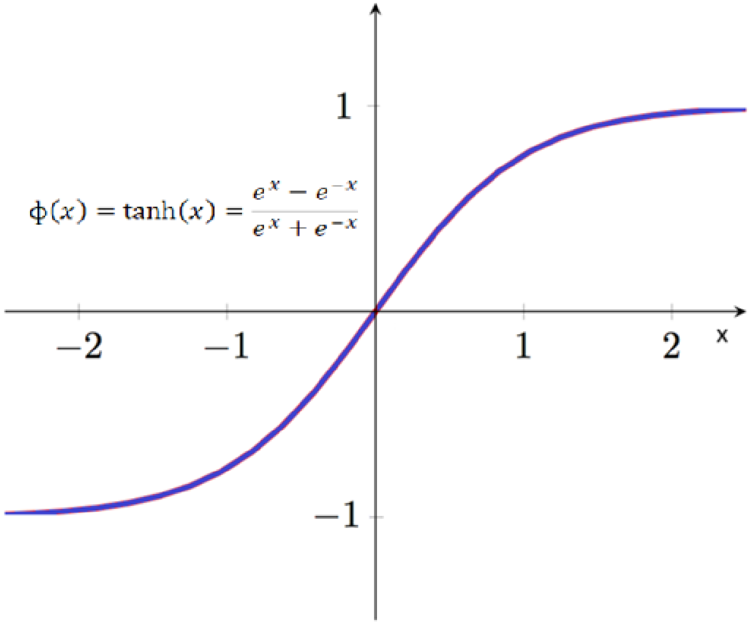
The Hyperbolic Tangent Function

- The **hyperbolic tangent function**, often abbreviated as **tanh**, is a mathematical function used as an activation function in neural networks and other applications.
- It is defined as follows:

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

- Alternatively, it can also be expressed in terms of exponential functions:

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



Key Properties of tanh

- **Range**: The output of the tanh function ranges between -1 and 1. This is in contrast to the sigmoid function, which ranges between 0 and 1.
- **Non-linearity**: The tanh function introduces non-linearity into the model, allowing it to learn complex patterns in the data.
- **Odd Function**: The tanh function is odd, meaning that $\tanh(-x) = -\tanh(x)$. This symmetry around the origin can help in learning representations that are centered around zero.
- **Smooth Gradient**: The gradient of the tanh function is smooth and continuous, which can help with gradient-based optimization methods.

Key Properties of tanh

- **Gradient Descent and Symmetry**: In tanh, outputs range from -1 to 1, which helps in keeping gradients symmetric around zero. This symmetry can lead to more efficient learning as the gradient steps are balanced in all directions.
- tanh functions produce outputs in the range [−1,1]. This zero-centered nature helps maintain a balanced gradient flow, making the learning process more stable and often faster.

Mathematical Behavior of tanh

- **For large positive inputs**: tanh(x) approaches 1.
- **For large negative inputs**: tanh(x) approaches -1.
- **For inputs around zero**: tanh(x) behaves approximately like the input x, which makes it approximately linear in the small input region.

Advantages

- **Zero-Centered Output**: Unlike the sigmoid function, which outputs values between 0 and 1, tanh outputs values between -1 and 1. This can help in centering the data and speeding up learning because the activations are zero-centered, reducing bias in gradients.
- **Non-saturation in the Middle Region**: The tanh function is less likely to saturate around zero compared to the sigmoid function, making it less prone to the vanishing gradient problem in the central part of its domain.

Disadvantages

- **Vanishing Gradient Problem**: For very large or very small inputs, the gradient of the tanh function can become very small, leading to the vanishing gradient problem where the model learns very slowly. This can hinder training, particularly in deep networks.
- **Computational Complexity**: While tanh is more complex to compute than simpler functions like ReLU, modern computational libraries and hardware optimizations typically make this a minor issue.

Use in Neural Networks

- **Activation Function**: Tanh is often used as an activation function in the hidden layers of neural networks. It can help in ensuring that activations have zero mean and maintain some level of gradient flow.
- **Recurrent Neural Networks (RNNs)**: Tanh is particularly popular in RNNs, including Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks, where its ability to manage gradients and zero-centered output is beneficial for learning temporal sequences.