How to Choose the Right Activation Function for Neural Networks

Analyzing different types of activation functions with visual representations — Neural Networks and Deep Learning Course: Part 5

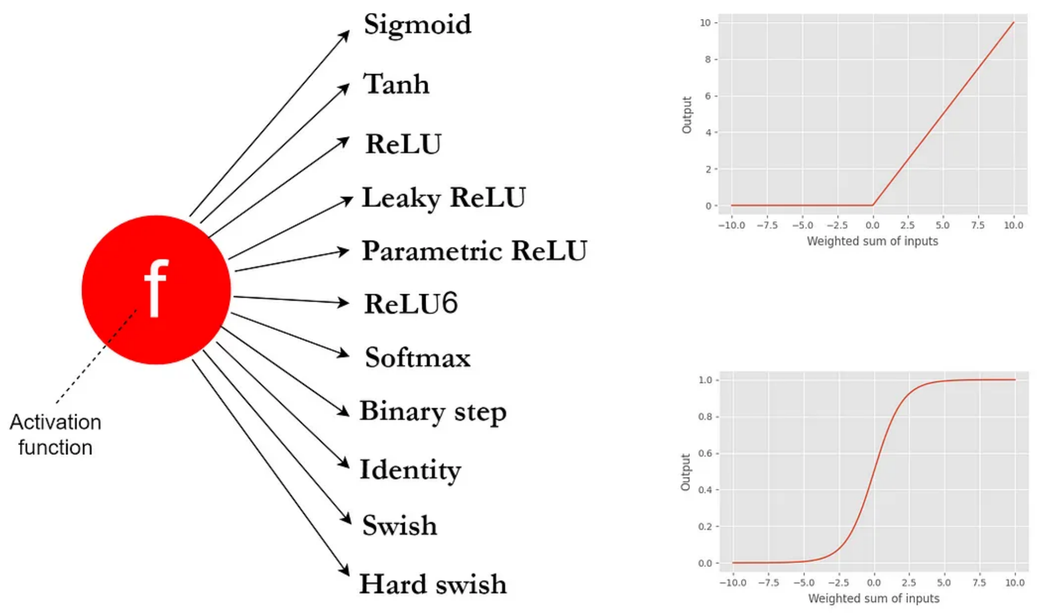


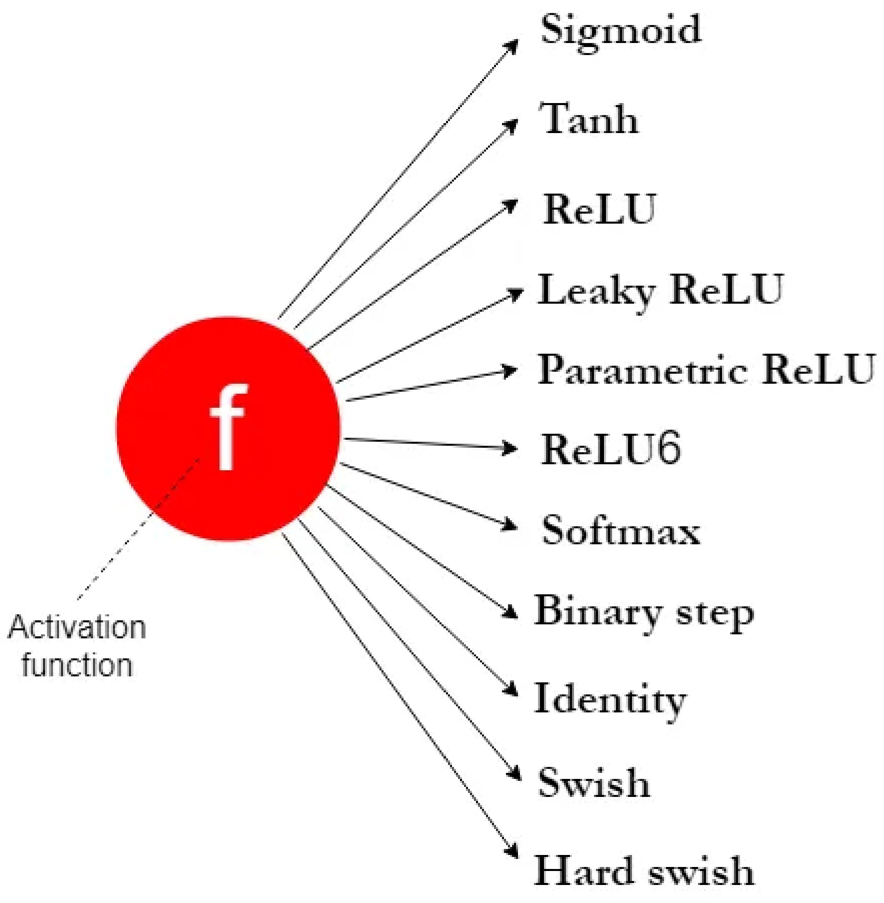
Image by author, made with draw.io and matplotlib

Introduction

In Part 1 of our **Neural Networks and Deep Learning Course** as introduced here, we've discussed the main purpose of using activation functions in neural network models.

Activation functions are applied to the weighted sum of inputs called **z** (here the input can be raw data or the output of a previous layer) at every node in the hidden layer(s) and the output layer.

Today, we're going to discuss the following different types of activation functions used in neural networks.



Different types of activation functions (Image by author, made with draw.io)

The visual representations will help you to understand the function definitions and different usage scenarios of activation functions.

At the end of the article:

- You will have a clear understanding of when to use which activation function.
- You will understand the definitions of different activation functions.

Activations functions in different layers in a neural network

A neural network typically consists of three types of layers: Input Layer, Hidden Layer(s) and Output Layer.

The input layer just holds the input data and no calculation is performed. Therefore, no activation function is used there.

We must use a non-linear activation function inside hidden layers in a neural network. This is because we need to introduce non-linearity to the network to learn complex patterns. Without non-linear activation functions, a neural network with many hidden layers would become a giant linear regression model that is useless for learning complex patterns from real-world data. The performance of a neural network model will vary significantly depending on the type of activation function we use inside the hidden layers.

We must also use an activation function inside the output layer in a neural network. The choice of the activation function depends on the type of problem that we want to solve.

Linear vs non-linear functions

Most of the activation functions are non-linear. However, we also use linear activation functions in neural networks. For example, we use a linear activation function in the output layer of a neural network model that solves a regression problem. Some activation functions are made up of two or three linear components. Those functions are also classified as non-linear functions.

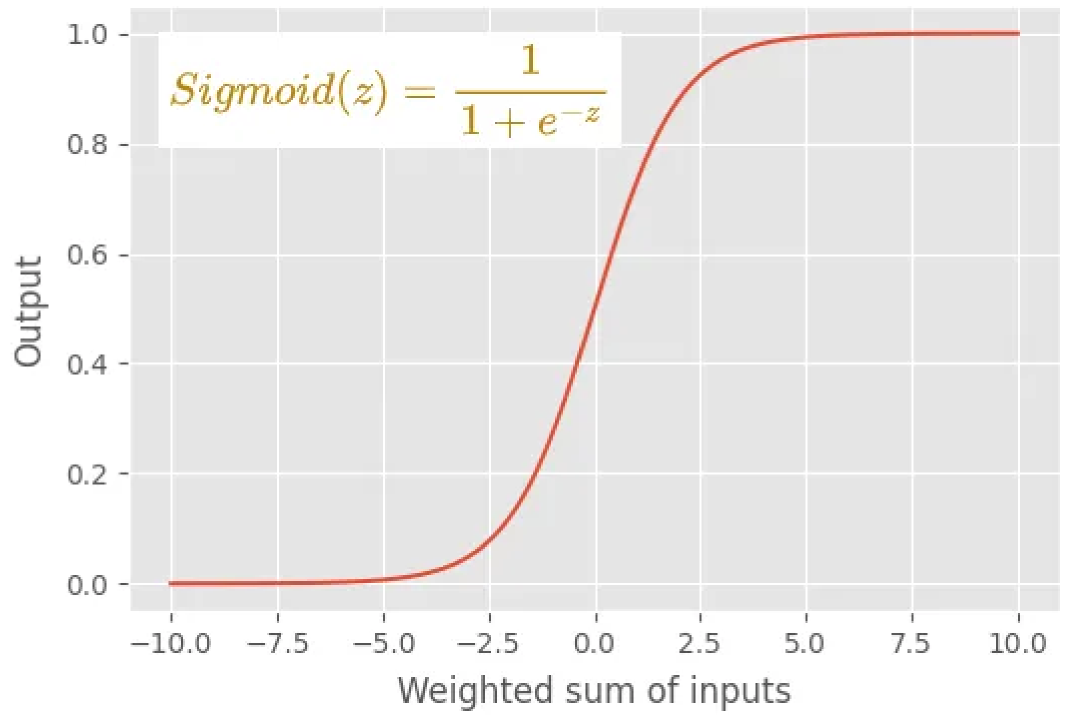
It will be useful to distinguish between linear and non-linear functions. A linear function (called **f**) takes the input, **z** and returns the output, **cz** which is the multiplication of the input by the constant, **c**. Mathematically, this can be expressed as **f(z) = cz**. When c=1, the function returns the input as it is and no change is made to the input. The graph of a linear function is a *single straight line*.

Any function that is not linear can be classified as a non-linear function. The graph of a non-linear function is not a single straight line. It can be a complex pattern or a combination of two or more linear components.

Different types of activation functions

We'll discuss commonly-used activation functions in neural networks.

1. Sigmoid activation function



Sigmoid activation function (Image by author, made with latex editor and matplotlib)

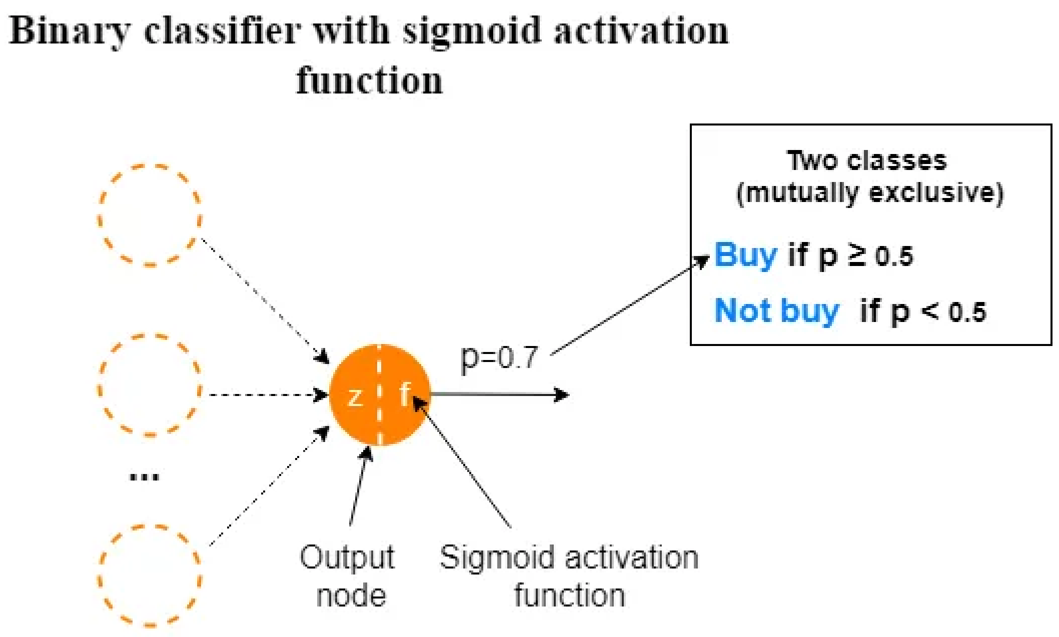
Key features:

- This is also called the *logistic function* used in logistic regression models.
- The sigmoid function has an **s-shaped** graph.
- Clearly, this is a non-linear function.
- The sigmoid function converts its input into a probability value between 0 and 1.
- It converts large negative values towards 0 and large positive values towards 1.
- It returns 0.5 for the input 0. The value 0.5 is known as the threshold value which can decide that a given input belongs to what type of two classes.

Usage:

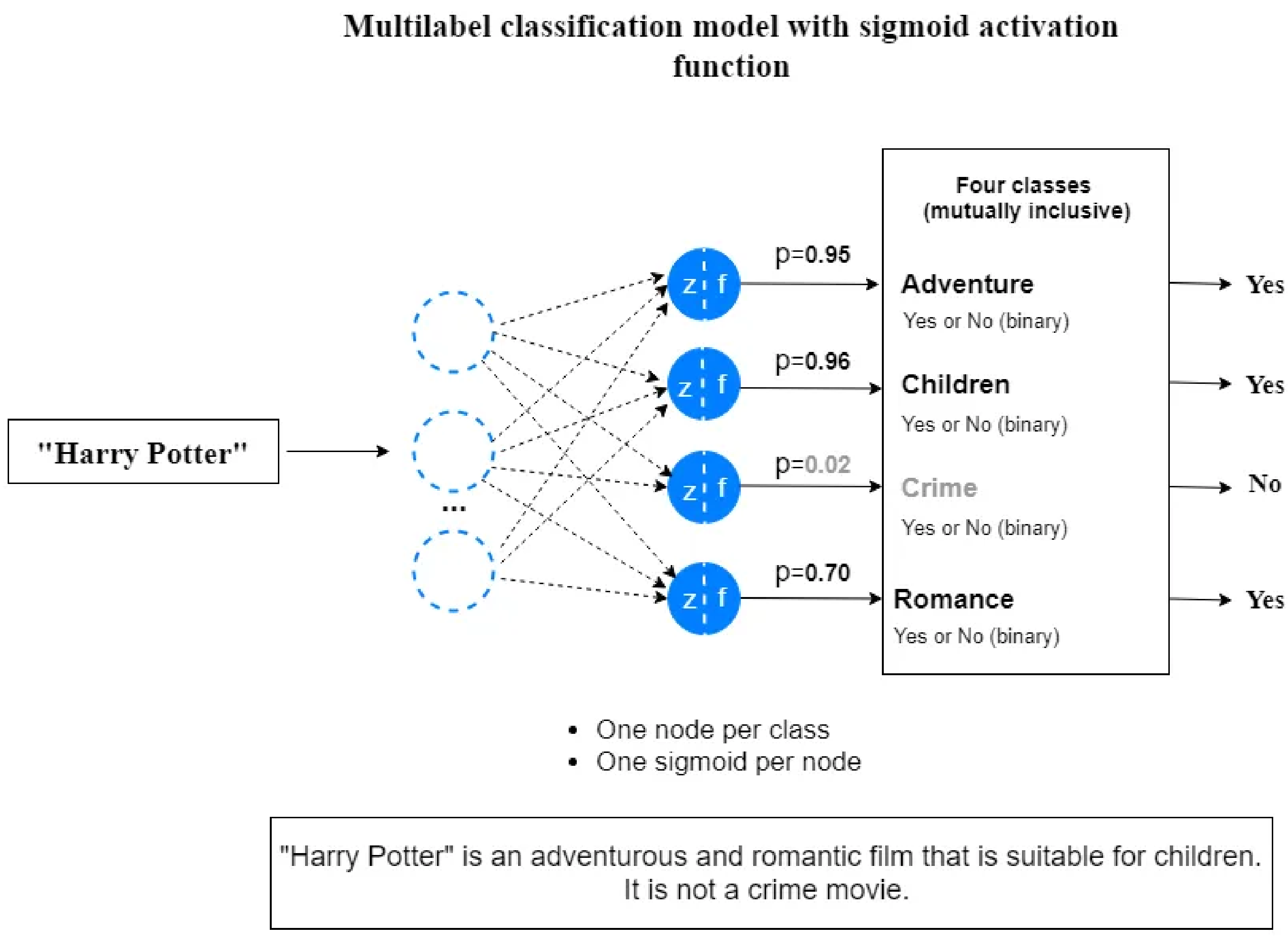
- In the early days, the sigmoid function was used as an activation function for the hidden layers in MLPs, CNNs and RNNs.
- However, the sigmoid function is still used in RNNs.

- Currently, we do not usually use the sigmoid function for the hidden layers in MLPs and CNNs. Instead, we use ReLU or Leaky ReLU there.
- The sigmoid function must be used in the output layer when we build a binary classifier in which the output is interpreted as a class label depending on the probability value of input returned by the function.


Binary classifier with sigmoid activation function
Two classes (mutually exclusive)
Buy if p ≥ 0.5
Not buy if p < 0.5
p=0.7
z
f
...
Output node
Sigmoid activation function

Binary classification with sigmoid function (Image by author, made with draw.io)

- The sigmoid function is used when we build a multilabel classification model in which each mutually inclusive class has two outcomes. Do not confuse this with a multiclass classification model.



Multilabel classification model with sigmoid activation function
"Harry Potter"
...
z f
p=0.95
p=0.96
p=0.02
p=0.70
Four classes (mutually inclusive)
Adventure
Yes or No (binary)
Children
Yes or No (binary)
Crime
Yes or No (binary)
Romance
Yes or No (binary)
Yes
Yes
No
Yes
One node per class
One sigmoid per node
"Harry Potter" is an adventurous and romantic film that is suitable for children. It is not a crime movie.

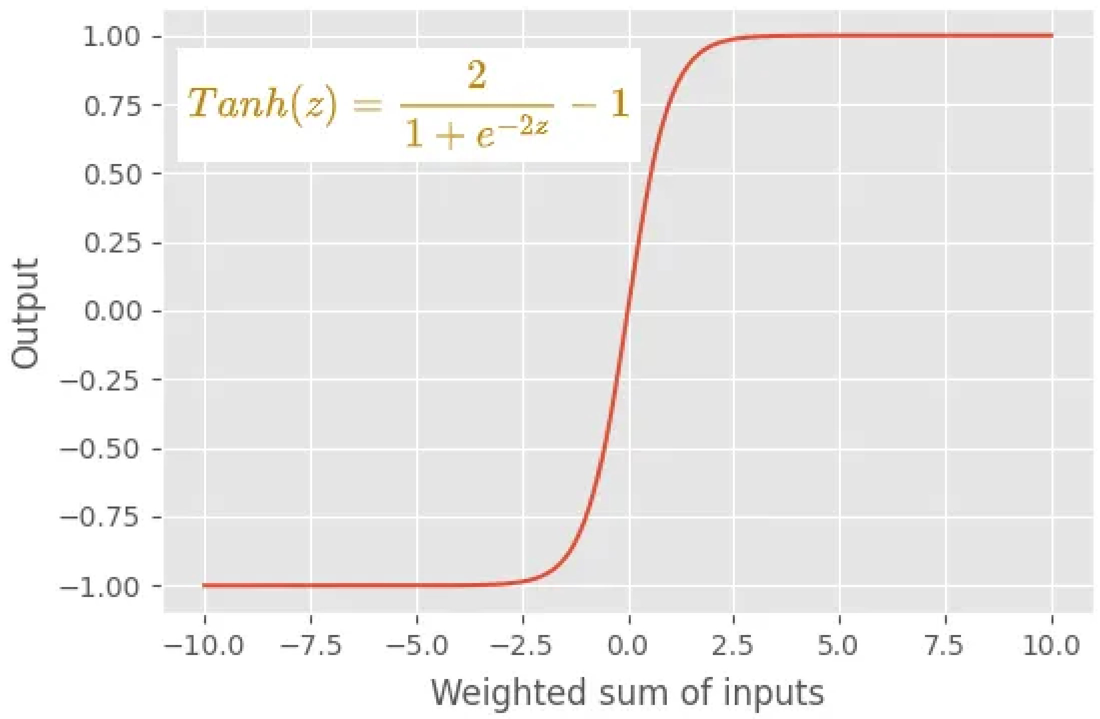
Multilabel classification with sigmoid function (Image by author, made with draw.io)

Drawbacks:

We do not usually use the sigmoid function in the hidden layers because of the following drawbacks.

- The sigmoid function has the vanishing gradient problem. This is also known as saturation of the gradients.
- The sigmoid function has slow convergence.
- Its outputs are not zero-centered. Therefore, it makes the optimization process harder.
- This function is computationally expensive as an **e^z** term is included.

2. Tanh activation function



Tanh activation function (Image by author, made with latex editor and matplotlib)

Key features:

- The output of the tanh (tangent hyperbolic) function always ranges between -1 and +1.
- Like the sigmoid function, it has an s-shaped graph. This is also a non-linear function.
- One advantage of using the tanh function over the sigmoid function is that the tanh function is zero centered. This makes the optimization process much easier.
- The tanh function has a steeper gradient than the sigmoid function has.

Usage:

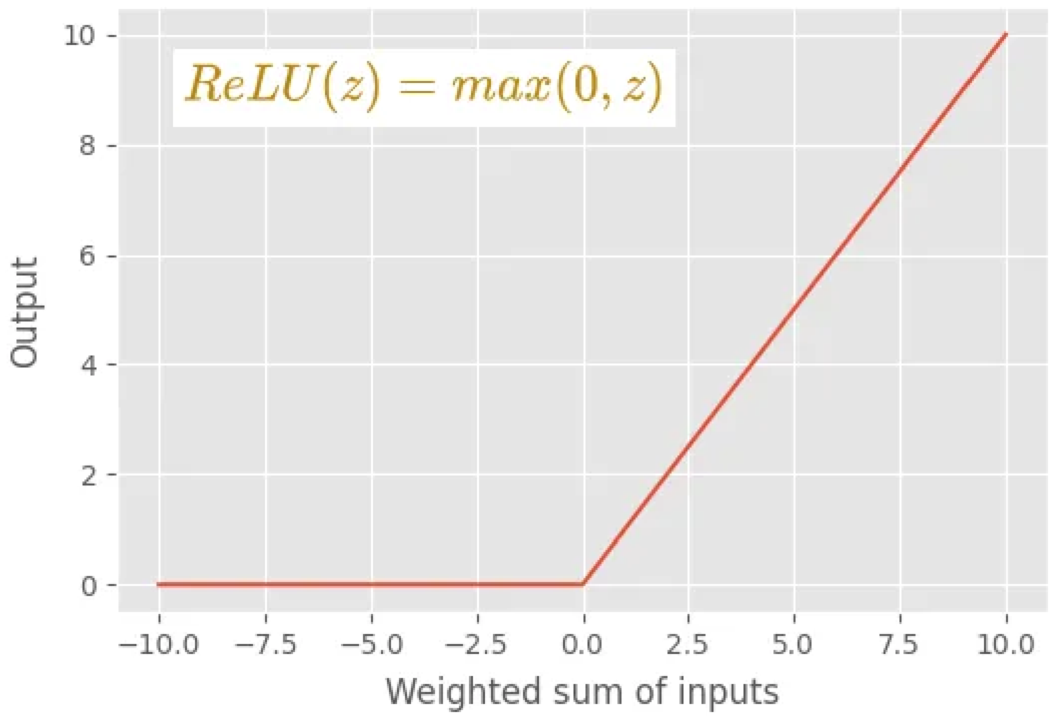
- Until recently, the tanh function was used as an activation function for the hidden layers in MLPs, CNNs and RNNs.
- However, the tanh function is still used in RNNs.
- Currently, we do not usually use the tanh function for the hidden layers in MLPs and CNNs. Instead, we use ReLU or Leaky ReLU there.
- We never use the tanh function in the output layer.

Drawbacks:

We do not usually use the tanh function in the hidden layers because of the following drawback.

- The tanh function has the vanishing gradient problem.
- This function is computationally expensive as an **e^z** term is included.

3. ReLU activation function



ReLU activation function (Image by author, made with latex editor and matplotlib)

Key features:

- The ReLU (**Re**ctified **L**inear **U**nit) activation function is a great alternative to both sigmoid and tanh activation functions.
- Inventing ReLU is one of the most important breakthroughs made in deep learning.
- This function does not have the vanishing gradient problem.
- This function is computationally inexpensive. It is considered that the convergence of ReLU is 6 times faster than sigmoid and tanh functions.

- If the input value is 0 or greater than 0, the ReLU function outputs the input as it is. If the input is less than 0, the ReLU function outputs the value 0.
- The ReLU function is made up of two linear components. Because of that, the ReLU function is a piecewise linear function. In fact, the ReLU function is a non-linear function.
- The output of the ReLU function can range from 0 to positive infinity.
- The convergence is faster than sigmoid and tanh functions. This is because the ReLU function has a fixed derivate (slope) for one linear component and a zero derivative for the other linear component. Therefore, the learning process is much faster with the ReLU function.
- Calculations can be performed much faster with ReLU because no exponential terms are included in the function.

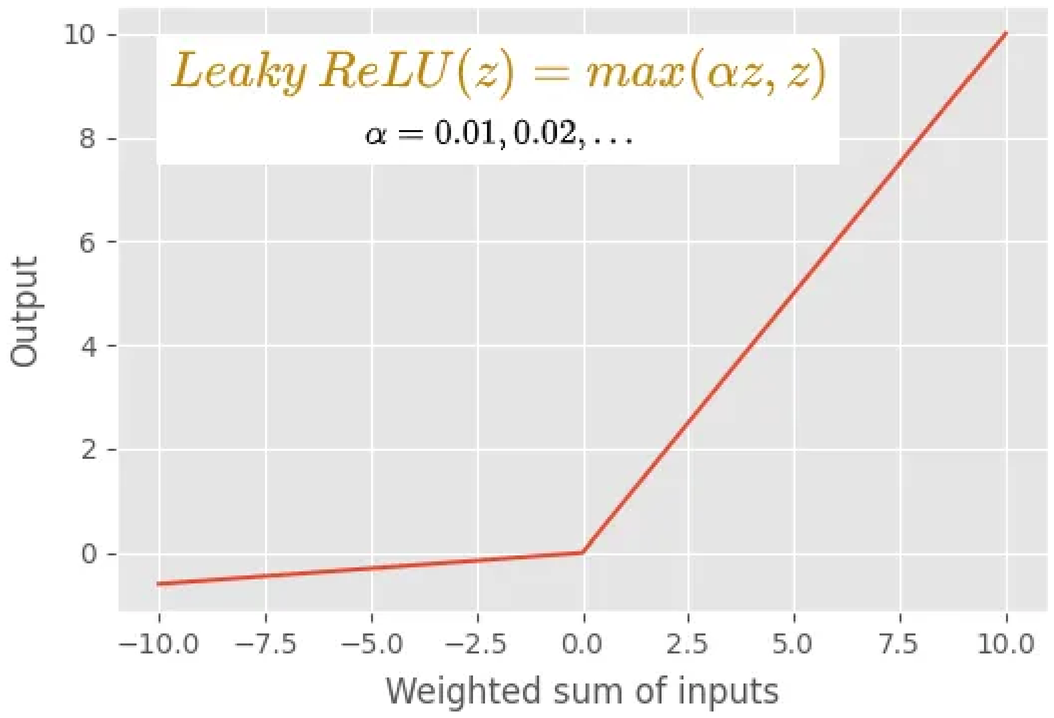
Usage:

- The ReLU function is the default activation function for hidden layers in modern MLP and CNN neural network models.
- We do not usually use the ReLU function in the hidden layers of RNN models. Instead, we use the sigmoid or tanh function there.
- We never use the ReLU function in the output layer.

Drawbacks:

- The main drawback of using the ReLU function is that it has a dying ReLU problem.
- The value of the positive side can go very high. That may lead to a computational issue during the training.

4. Leaky ReLU activation function



Leaky ReLU activation function (Image by author, made with latex editor and matplotlib)

Key features:

- The leaky ReLU activation function is a modified version of the default ReLU function.
- Like the ReLU activation function, this function does not have the vanishing gradient problem.
- If the input value is 0 greater than 0, the leaky ReLU function outputs the input as it is like the default ReLU function does. However, if the input is less than 0, the leaky ReLU function outputs a small negative value defined by **αz** (where **α** is a small constant value, usually 0.01 and **z** is the input value).
- It does not have any linear component with zero derivatives (slopes). Therefore, it can avoid the dying ReLU problem.
- The learning process with leaky ReLU is faster than the default ReLU.

Usage:

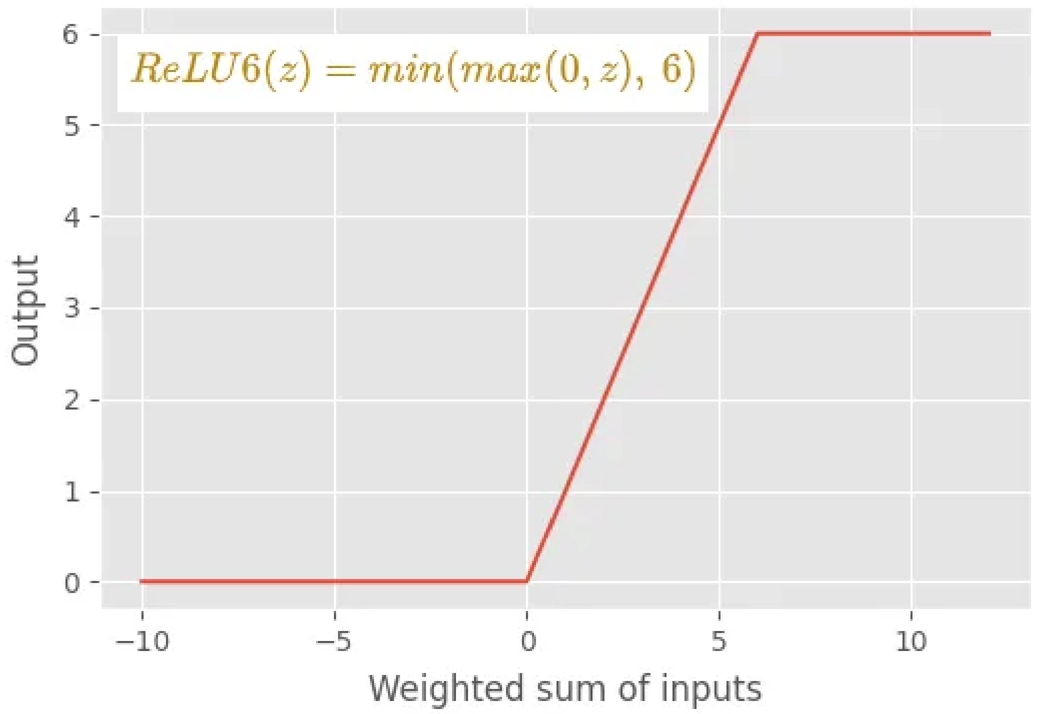
- The same usage of the ReLU function is also valid for the leaky ReLU function.

5. Parametric ReLU (PReLU) activation function

Key features:

- This is another variant of the ReLU function.
- This is almost similar to the leaky ReLU function. The only difference is that the value **α** becomes a learnable parameter (hence the name). We set **α** as a parameter for each neuron in the network. Therefore, the optimal value of **α** learns from the network.

6. ReLU6 activation function



ReLU6 activation function (Image by author, made with latex editor and matplotlib)

Key features:

- The main difference between ReLU and ReLU6 is that ReLU allows very high values on the positive side while ReLU6 restricts to the value 6 on the positive side. Any input value which is 6 or greater than 6 will be restricted to the value 6 (hence the name).
- The ReLU6 function is made up of three linear components. It is a non-linear function.

7. Softmax activation function

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

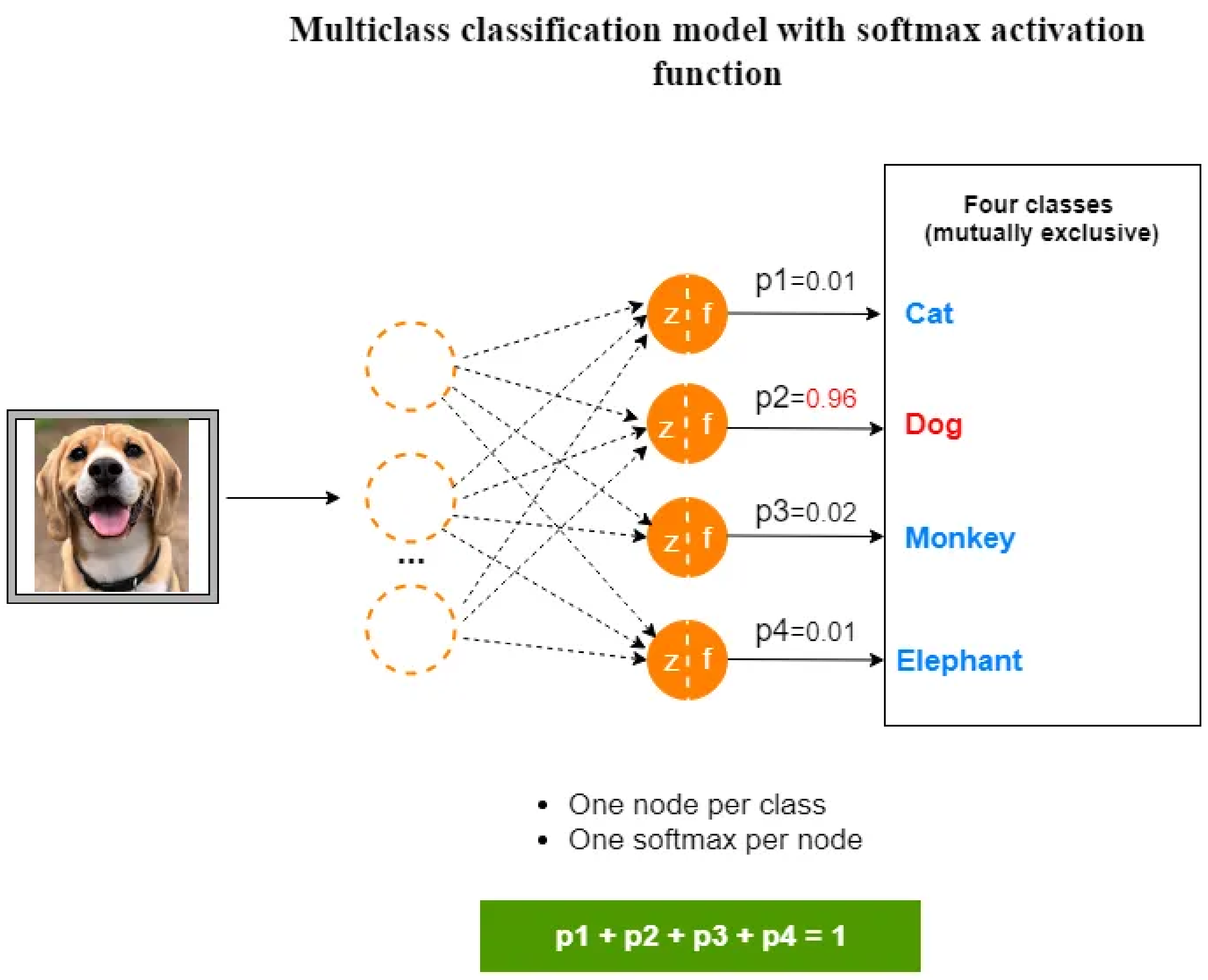
Softmax activation function (Image by author, made with latex editor)

Key features:

- This is also a non-linear activation function.
- The softmax function calculates the probability value of an event (class) over **K** different events (classes). It calculates the probability values for each class. The sum of all probabilities is 1 meaning that all events (classes) are mutually exclusive.

Usage:

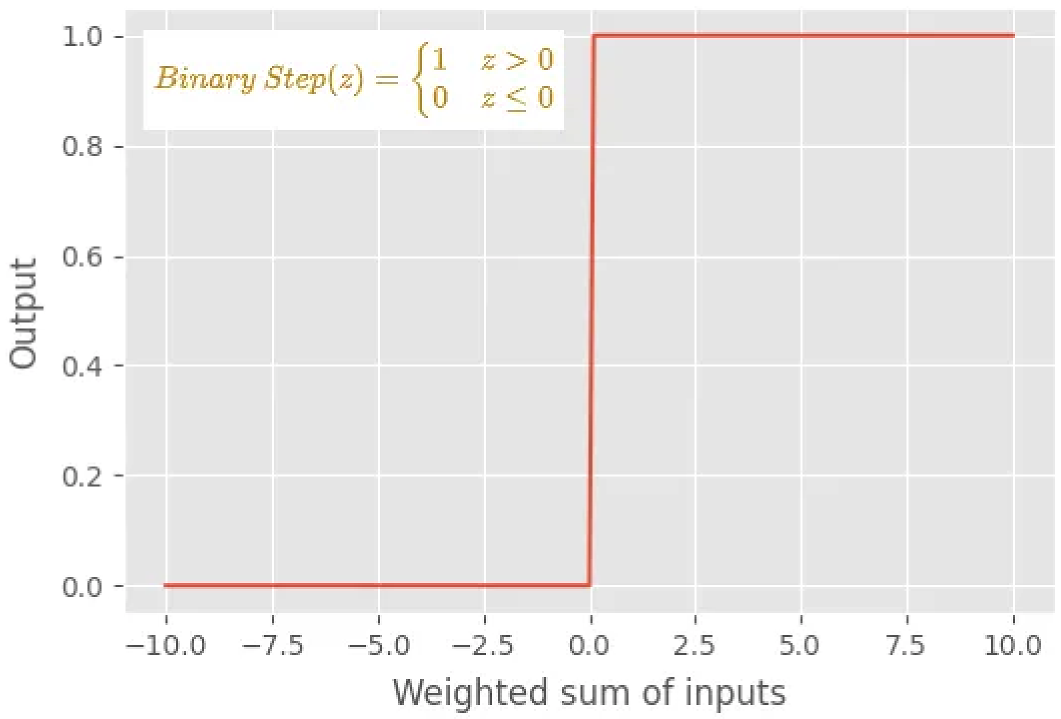
- We must use the softmax function in the output layer of a multiclass classification problem.



Multiclass classification with softmax function (Image by author, made with draw.io)

- We never use the softmax function in the hidden layers.

8. Binary step activation function



Binary step activation function (Image by author, made with latex editor and matplotlib)

Key features:

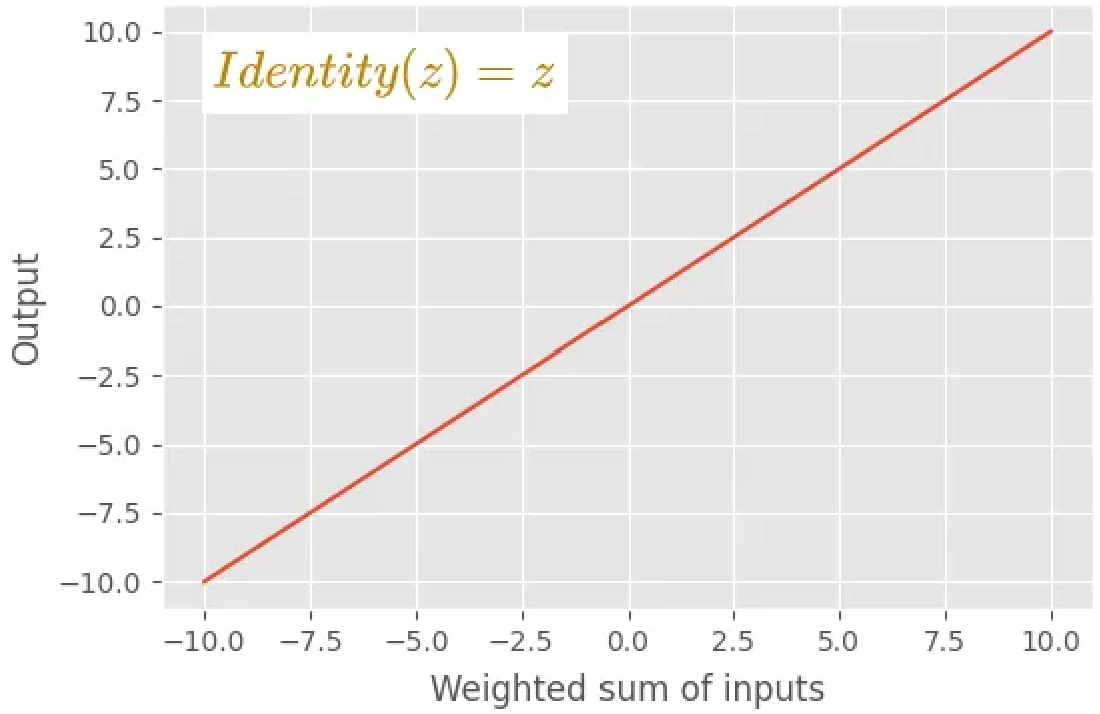
- This function is also known as the *threshold activation function*. We can set any value to the threshold and here we specify the value 0.
- If the input is greater than the threshold value, this function outputs the value 1. If the input is equal to the threshold value or less than it, this function outputs the value 0.
- This outputs a binary value, either 0 or 1.
- The binary step function is made up of two linear components. Because of that, this function is a piecewise linear function. In fact, the binary step function is a non-linear function.
- This function is not a smooth function.

Usage:

- In practice, we do not usually use this function in modern neural network models.
- However, we can use this function to explain theoretical concepts such as “firing a neuron”, “inner workings of a perceptron”. Therefore, the step function is theoretically

important.

9. Identity activation function



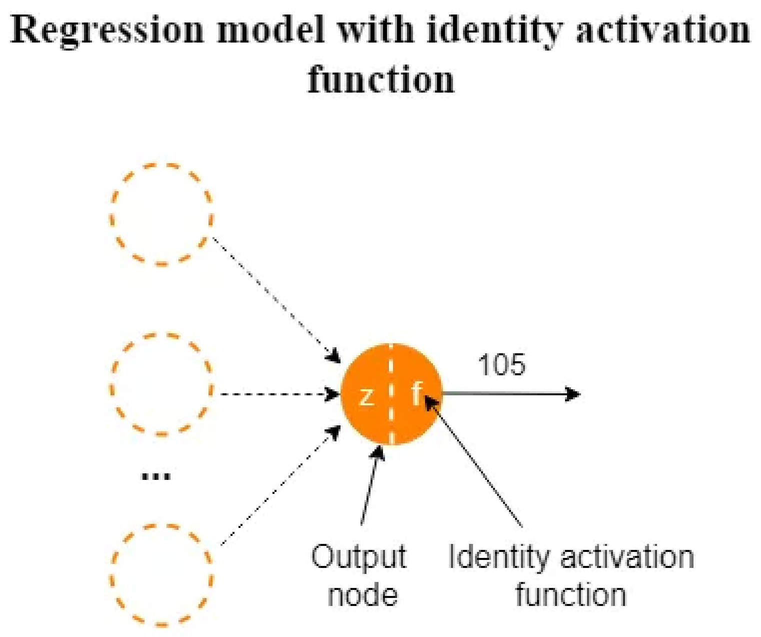
Identity activation function (Image by author, made with latex editor and matplotlib)

Key features:

- This is also known as the *linear activation function*.
- This is the only function that is considered as a linear function when we talk about activation functions.
- This function outputs the input value as it is. No changes are made to the input.

Usage:

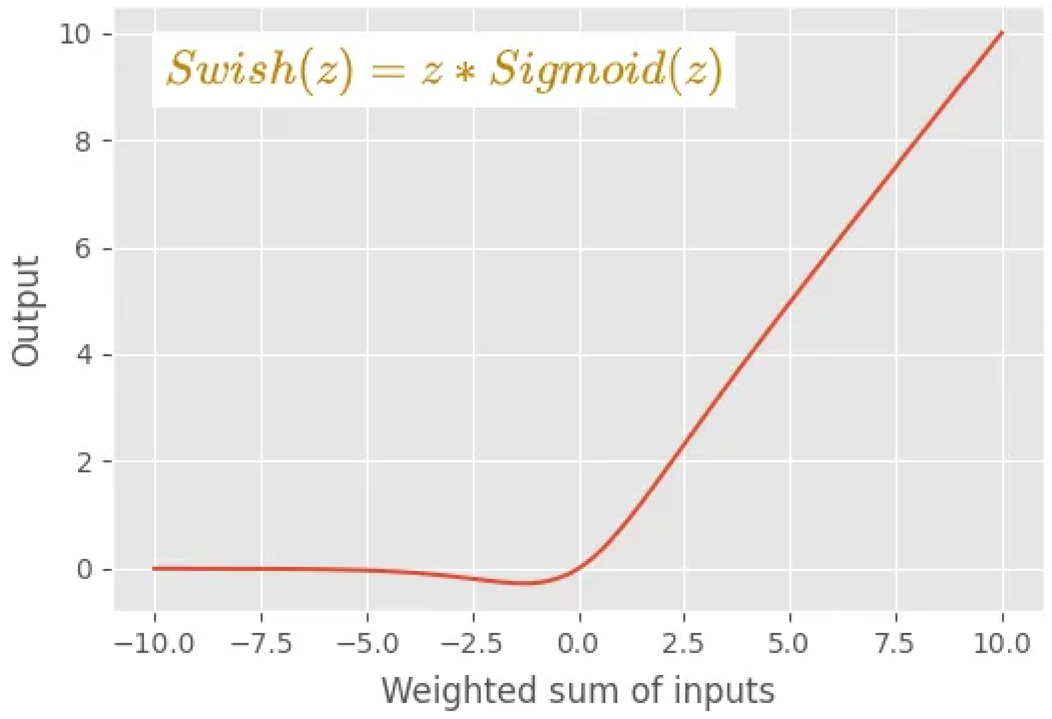
- This function is only used in the output layer of a neural network model that solves a regression problem.


Regression model with identity activation function
z
f
105
...
Output node
Identity activation function

Regression with identity function (Image by author, made with draw.io)

- We never use the identity function in the hidden layers.

10. Swish activation function



Swish activation function (Image by author, made with latex editor and matplotlib)

Key features:

- This function is made of by multiplying the sigmoid function by the input **z**.
- This is a non-linear function.
- The graph is much similar to the graph of the ReLU activation function.
- The curve is more smooth than the ReLU activation function. This smoothness is important when training the model. The function converges easily while training.

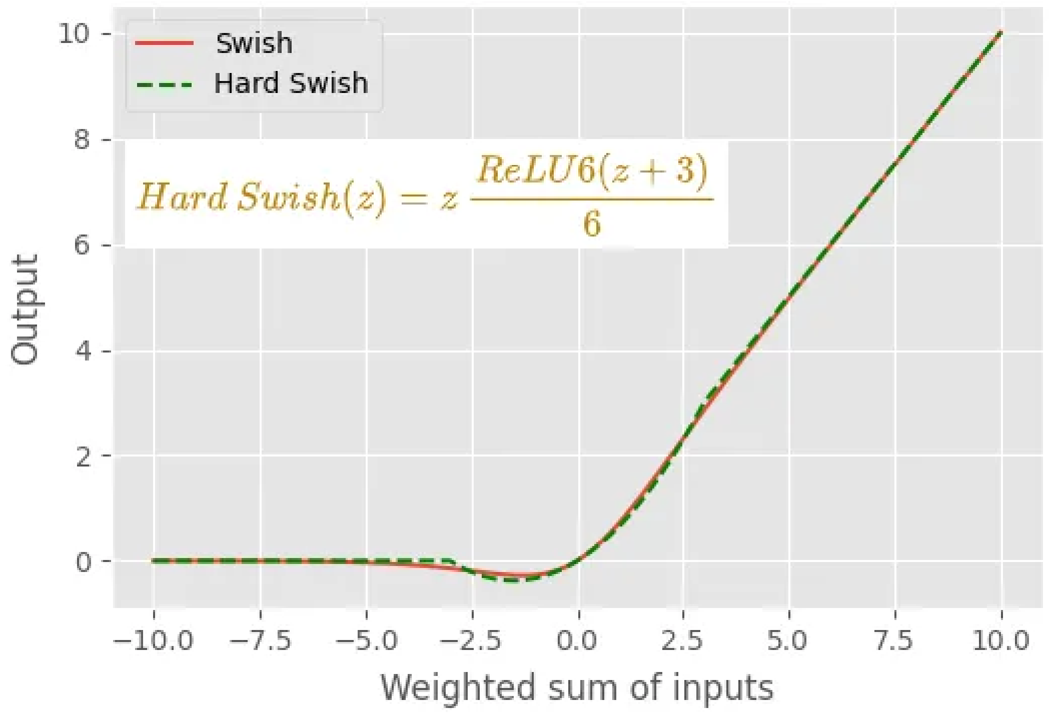
Usage:

- This function is only used in the hidden layers.
- We never use this function in the output layer of a neural network model.

Drawbacks:

- The main drawback of the Swish function is that it is computationally expensive as an **e^z** term is included in the function. This can be avoided by using a special function called "Hard Swish" defined below.

11. Hard Swish (H-Swish) activation function



Hard Swish (H-Swish) activation function (Image by author, made with latex editor and matplotlib)

Key features:

- The graph is identical to the graph of the Swish function.
- This is computationally inexpensive as the sigmoid function was replaced with a linear analogue.

Usage:

- The usage of hard Swish is similar to the usage of the Swish activation function.

Summary

Activation functions are just mathematical functions. The main feature that an activation function should have is that the function should be differentiable as this is a requirement for backpropagation in model training.

Choosing the right activation function is the main challenge and it can be considered as a type of hyperparameter tuning in which the programmer manually chooses the activation function by understanding the problem definition and considering the performance of the model and the convergence of the loss function.

General guidelines to choose the right activation function

Here is the summary of usage scenarios of different activation functions discussed above. You may find this useful when you train your own neural network models.

- No activation function is required in the input layer nodes of a neural network. So, you don't need to worry about activation functions when you define the input layer.
- The output layer activation function depends on the type of problem that we want to solve. In a regression problem, we use the linear (identity) activation function with one node. In a binary classifier, we use the sigmoid activation function with one node. In a multiclass classification problem, we use the softmax activation function with one node per class. In a multilabel classification problem, we use the sigmoid activation function with one node per class.
- We should use a non-linear activation function in hidden layers. The choice is made by considering the performance of the model or convergence of the loss function. Start with

the ReLU activation function and if you have a dying ReLU problem, try leaky ReLU.

- In MLP and CNN neural network models, ReLU is the default activation function for hidden layers.
- In RNN neural network models, we use the sigmoid or tanh function for hidden layers. The tanh function has better performance.
- Only the identity activation function is considered linear. All other activation functions are non-linear.
- We never use softmax and identity functions in the hidden layers.
- We use tanh, ReLU, variants of ReLU, swish and hard swish functions only in the hidden layers.
- The swish and hard swish functions have been found recently from the latest researches.