Tekhnê
January - June 2022, Vol. 19, No. 1, pp. 23 – 32

ISSN 1692-8407

How to choose an activation function for deep learning

Cómo elegir una función de activación para el aprendizaje profundo

Albert I. Rodríguez P.[1] and Xiomara D. Buitrago R.[2]
[1]Facultad Tecnológica, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia
airodriguezp@correo.udistrital.edu.co
[2]Facultad Tecnológica, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia
xdbuitragor@correo.udistrital.edu.co

Activation functions are important in each layer of the neural network because they allow the network to learn complex relationships between the input data and the output data. They also introduce nonlinearity into the network, which is essential for learning patterns in data. Activation functions play a critical role in the training and optimization of deep learning models, and choosing the right activation function can significantly impact the model's performance. This article presents a summary of the features of these functions.

Keywords: Activation function, deep learning, neural network, nonlinearity

Las funciones de activación son importantes en cada capa de la red neuronal porque permiten a la red aprender relaciones complejas entre los datos de entrada y los de salida. También introducen la no linealidad en la red, que es esencial para aprender patrones en los datos. Las funciones de activación desempeñan un papel fundamental en el entrenamiento y la optimización de los modelos de aprendizaje profundo, y la elección de la función de activación adecuada puede influir significativamente en el rendimiento del modelo. Este artículo presenta un resumen de las características de estas funciones.

Palabras clave: Aprendizaje profundo, función de activación, no linealidad, red neuronal

Article typology: Research

Received: March 16, 2022
Accepted: May 27, 2022

Research funded by: Universidad Distrital Francisco José de Caldas (Colombia).

How to cite: Rodríguez, A., and Buitrago, X. (2022). *How to choose an activation function for deep learning*. Tekhnê, 19(1), 23 -32.

Introduction

Activation functions play a crucial role in deep learning and can greatly impact the performance of a neural network (Kim & Cho, 2019). Choosing the right activation function can be challenging, as there are a wide variety of options to choose from and the optimal choice may depend on the specific characteristics of the data and the problem being solved (Maguolo et al., 2019). In this paper, we will explore the most important considerations when selecting an activation function for deep learning.

One of the primary considerations when selecting an activation function is the range of the output (Martínez, Martínez, & Montiel, 2020). Activation functions that produce output in a limited range, such as between 0 and 1 or between -1 and 1, can be more stable and easier to train than those with a wider range (Montiel et al., 2021). Sigmoid and Tanh are examples of activation functions that produce output in a limited range, while ReLU (Rectified Linear Unit) has a range of 0 to infinity.

Another important consideration is the shape of the activation function. Activation functions with a smooth, continuous curve can help the neural network to converge faster and perform better, as the gradient is well-behaved and easy to compute (Rendón & Martínez, 2021). On the other hand, activation functions with a more complex or discontinuous curve can be more difficult to optimize and may lead to slower convergence (Alonso et al., 2021). For example, the Sigmoid function has a smooth curve, while the ReLU function has a piecewise linear shape.

In addition to the range and shape of the activation function, it is also important to consider the computational efficiency of the function (Martínez et al., 2022). Activation functions that require more computation, such as Sigmoid and Tanh, can be more expensive to compute and may slow down training (Jacinto et al., 2022). On the other hand, activation functions that are more computationally efficient, such as ReLU and its variants, can speed up training and make it more efficient.

Another important factor to consider when selecting an activation function is the ability to handle non-linearity (Martínez, Martínez, & Jacinto, 2020). Neural networks are able to model complex relationships and patterns in data due to their ability to learn non-linear functions (Martínez, Penagos, et al., 2020). Activation functions that are able to introduce non-linearity into the network can help the network to better capture these patterns and improve its performance (Montiel et al., 2017). However, it is important to find the right balance between introducing non-linearity and maintaining stability, as too much non-linearity can lead to overfitting and poor generalization.

It is also worth considering the vanishing gradient problem when selecting an activation function (Rendón et al., 2017). During training, the gradients of the weights in the neural network are updated based on the error between the predicted output and the true output. If the gradients are too small, the network may have difficulty learning and convergence may be slow (Martínez et al., 2017). Activation functions that have a slope close to 0 for large input values can lead to the vanishing gradient problem, as the gradients will be small and the network will have difficulty learning. On the other hand, activation functions that have a slope that remains relatively large for a wide range of input values can help to mitigate the vanishing gradient problem and improve the network's ability to learn.

Finally, it is also worth considering the specific characteristics of the data and the problem being solved when selecting an activation function. Some activation functions may be better suited for certain types of data or problems, while others may be less effective. For example, Sigmoid and Tanh may be more effective for binary classification problems, while ReLU may be better suited for regression tasks.

In summary, there are a number of important considerations when selecting an activation function for deep learning. These include the range and shape of the output, computational efficiency, ability to handle non-linearity, and the vanishing gradient problem. It is also important to consider the specific characteristics of the data and the problem being solved.

Literature review

(Deng et al., 2013) provide an overview of the invited and contributed papers presented at the special session at ICASSP-2013, entitled "New Types of Deep Neural Network Learning for Speech Recognition and Related Applications," as organized by the authors. (Chang & Tsai, 2017) choose to use exponential linear units (ELUs) as the activation function and stochastic gradient descent (SGD) as the optimizer. (Ramachandran et al., 2017) propose to leverage automatic search techniques to discover new activation functions. (X. Zhu et al., 2018) propose a new method based on a multilayer perceptron (MLP), a classic network in deep learning, to predict the EDH. Auto-Encoder will be utilised as a feature learning algorithm to practice the recommended model to excerpt the useful features from the surface electromyography signal (Ibrahim & Al-Jumaily, 2018). Based on the big data of rail transit IC card (Public Transportation Card) (H. Zhu et al., 2018) analyze the data of major dynamic factors having effect on entrance passenger flow and exit passenger flow of rail transit stations: weather data, atmospheric temperature data, holiday and festival data, ground index data, and elevated road data and calculates the daily entrance passenger flow and daily exit passenger flow of individual rail transit stations with data reduction. Krizhevsky outlines the construction of an image recognition model using the Rectified Linear Unit activation function

(ReLU) for the ImageNET LSVRC-2010 competition which outperformed the state-of-the-art image recognition systems at the time (Pomerat et al., 2019). (Szandala, 2021) evaluate the commonly used additive functions, such as swish, ReLU, Sigmoid, and so forth. There is no existing work that provides sound information for peers to choose appropriate deep AUROC maximization techniques. (D. Zhu et al., 2022) fill this gap, they compared the existing top ten loss functions (pairwise or composite) for deep maximization of AUROC. They performed experiments on six image datasets and six molecule graph datasets. The large-scale CheXpert dataset contains 191,027 samples and five recognition targets.

Problem statement

The activation functions are fundamental for the design of a neural network, this activation function allows for example in the hidden layer to verify how well the network model learns with the training data provided, so it is important a good choice of the function because in the outputs we will see the predictions that the model presents us.

Something that we will learn with this article and that is very important is:

- Activation functions are a key part of neural network design.
- The modern default activation function for hidden layers is the ReLU function.
- The activation function of the output layers depends on the type of prediction problem.

This paper is divided into three parts:

- Activation functions
- Activation of Hidden Layers
- Activation for output layers

Activation functions

An input weighted sum that is converted into an output of one or more nodes in a layer of a neural network is known as an activation function in a neural network. Since many activation functions are not linear, the design of the network or layer is non-linear, as is well known (Non-linearity).

On the other hand, the activation function is often referred to as a transfer function, but if the output range of the activation function is limited, then it can be called a squashing function.

A good choice or choices of the activation function will allow the neural network to have good capacity and performance, basically, the activation function can be used within or after the processing of each node in the network, although we know that networks are designed to use the same activation function for all nodes in a layer. On the other hand, it is necessary to know that activation functions are also usually differentiable, which means that the first order derivative can be calculated for a given input value. This is because neural networks are usually trained using the error back propagation algorithm that requires the derivative of the prediction error to update the model weights.

Therefore, the hidden layers usually use the same activation function, but the output layer will use a different activation function from the hidden layers, this may be because of the type of prediction required by the model.

The network is composed of three different types of layers: input layers, which take the domain's raw input, hidden layers, which take the input from another layer and transmit the acquired output to another layer, and output layers, which perform the prediction.

Activation of hidden layers

A hidden layer in a neural network is one that, in general, neither directly interacts with input data nor produces outputs for models; rather, it accepts input from other layers as another input layer and permits output to other layers as an output layer.

A neural network may contain zero hidden layers or more. In the hidden layers of a neural network, a nonlinear differentiable function is frequently utilized for the activation function. In comparison to a network trained using a linear activation function, the model can learn more complicated functions as a result.

There are three activation functions that can be considered for use in hidden layers:

- Rectified Linear Activation (ReLU).
- Logistic (Sigmoid)
- Hyperbolic Tangent (Tanh)

ReLU hidden layer activation function

The most typical function employed in hidden layers is the rectified linear activation function, or ReLU activation function.

It is popular because it is easy to use and gets around the drawbacks of other activation algorithms like Sigmoid and Tanh. It can experience other issues like saturated units, but it is less prone to vanishing gradients that prohibit deep models from being trained.

This is how the ReLU function is determined:

$$max\,(0.0,\ x) \tag{1}$$

Accordingly, a value of 0.0 is returned if the input value x is negative; otherwise, the value is returned. The solved

Figure 1

Example of the ReLU activation function.

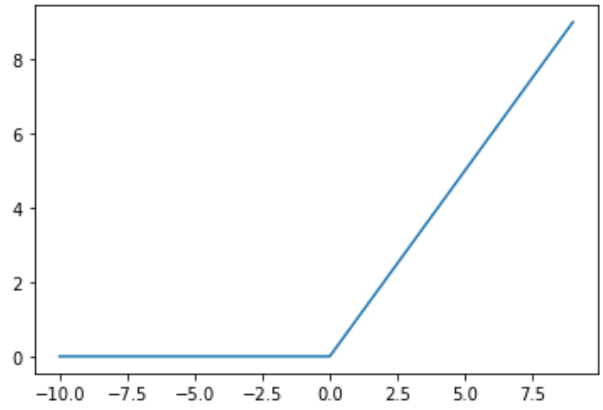
```
# example plot for the relu activation function
from matplotlib import pyplot

# rectified linear function
def rectified(x):
  return max(0.0, x)

# define input data
inputs = [x for x in range(-10, 10)]
# calculate outputs
outputs = [rectified(x) for x in inputs]
# plot inputs vs outputs
pyplot.plot(inputs, outputs)
pyplot.show()
```

Figure 2

Input vs. output graph for ReLU activation function.



example below can be used to get a general idea of the shape of this function (Fig. 1).

When the example is run, the outputs are calculated for a variety of values, and an input-to-output graph is produced (Fig. 2). We can see the ReLU activation function's well-known kink shape.

Sigmoid hidden layer activation function

The logistic function is another name for the sigmoid activation function. The logistic regression classification approach uses the same function.

The function produces numbers between 0 and 1 and accepts any real value as an argument. The output value will be nearer to 1.0 the greater the input, or positive, whereas the output will be nearer to 1.0 the smaller the input, or negative. 0.0 will be the result.

Figure 3

Example of the Sigmoid activation function.

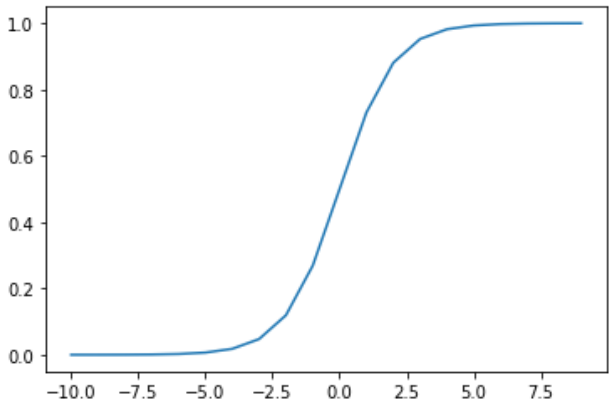
```
# example plot for the sigmoid activation function
from math import exp
from matplotlib import pyplot

# sigmoid activation function
def sigmoid(x):
  return 1.0 / (1.0 + exp(-x))

# define input data
inputs = [x for x in range(-10, 10)]
# calculate outputs
outputs = [sigmoid(x) for x in inputs]
# plot inputs vs outputs
pyplot.plot(inputs, outputs)
pyplot.show()
```

Figure 4

Input vs. output plot for Sigmoid activation function.



Following are the calculations for the sigmoid activation function (Eq. 2).

$$\frac{1.0}{(1.0 + e^{-x})} \tag{2}$$

Where the natural logarithm's base, e, is a mathematical constant. With the help of the below-mentioned example's solution, we can acquire a general idea of this function's shape (Fig. 3).

When the example is run, the outputs are calculated for a range of values, and an input-to-output plot is produced. It is possible to view the sigmoid activation function's well-known S-shape (Fig. 4).

Figure 5

Example of the Tanh activation function.

```
# example plot for the tanh activation function
from math import exp
from matplotlib import pyplot

# tanh activation function
def tanh(x):
  return (exp(x) - exp(-x)) / (exp(x) + exp(-x))

# define input data
inputs = [x for x in range(-10, 10)]
# calculate outputs
outputs = [tanh(x) for x in inputs]
# plot inputs vs outputs
pyplot.plot(inputs, outputs)
pyplot.show()
```

Tanh hidden layer activation function

The Tanh function, also referred to as the hyperbolic tangent activation function, is remarkably close to the sigmoid activation function and even has the same S-shape.

The function accepts any real number as an input and outputs values between -1 and 1. The output value will be nearer to 1.0 the greater the input, or positive, and nearer to -1.0 the smaller the input, or negative.

This is how the Tanh activation function is calculated (Eq. 3).

$$\frac{e^x - e^{-x}}{(e^x + e^{-x})} \tag{3}$$

Where the natural logarithm's base, e, is a mathematical constant. With the help of the below-posted solved example, we may gain a general understanding of this function's form (Fig. 5).

When the example is run, the outputs are calculated for a variety of values, and an input-to-output graph is produced. It is possible to see Tanh's activation function's well-known S-shape (Fig. 6).

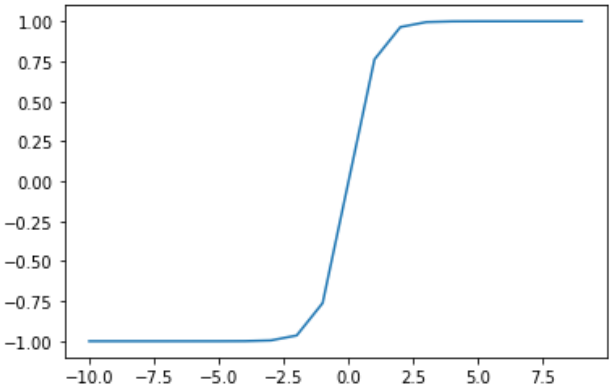
How to choose a hidden layer activation function

The most unusual thing is to change the activation function across a network model; a neural network will usually always have the same activation function in all hidden layers.

In the 1990s, the sigmoid activation function was frequently used as the activation function. Then, between the mid-to-late 1990s and 2010, the Tanh function served as the standard activation function for hidden layers since it frequently outperforms the logistic Sigmoid.

Figure 6

Inputs vs. outputs plot for Tanh activation function.



On the other hand, the so-called leakage gradient problem caused by sigmoid functions like Tanh's can make the model more prone to issues during training.

In modern neural network models, the most prevalent architectures are MLP and CNN, which use the activation function or extensions of ReLU. In these neural networks, the default recommendation is to use the rectified linear unit, or ReLU. The activation function used in the hidden layers is chosen depending on the type of neural network architecture.

The Tanh or sigmoid activation functions, or possibly both, are still used in recurrent networks. For instance, the LSTM frequently employs Tanh activation for output and Sigmoid activation for recurrent connections.

Some examples of neural networks using the ReLU activation function.

- Multilayer perceptron (MLP): ReLU activation function.
- Convolutional neural network (CNN): ReLU activation function.

When you are not sure which activation function to use for your network, try a few and compare the results (Fig. 7).

Activation for output layers

We know that all feed-forward neural network models have an output layer, which is the layer in a neural network model that directly generates a prediction.

There are other activation mechanisms; however, the three that are most frequently employed can be utilized in the output layer.

- Linear
- Logistic (Sigmoid)
- Softmax

Figure 7

How to choose an activation function for hidden layers (Brownlee, 2021).

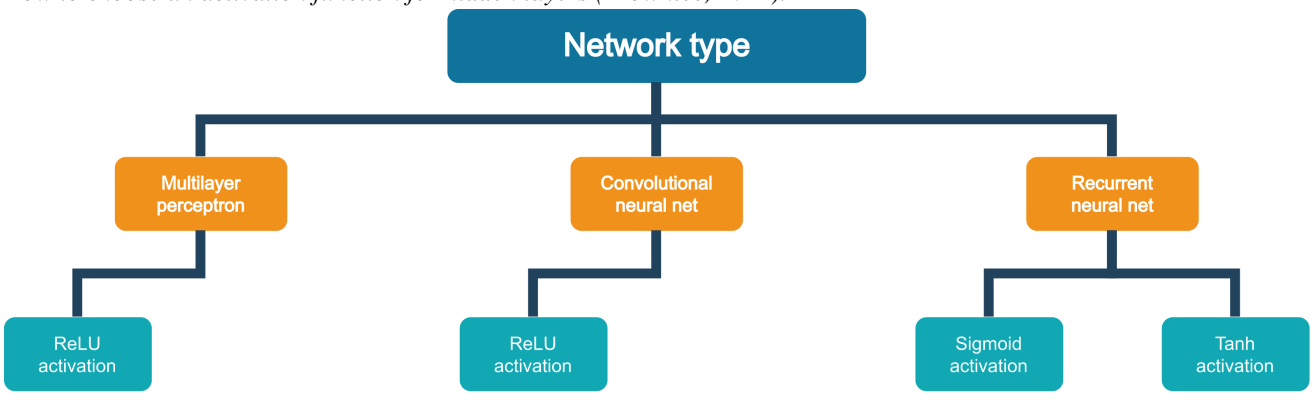


Figure 8

Example of the Linear activation function.

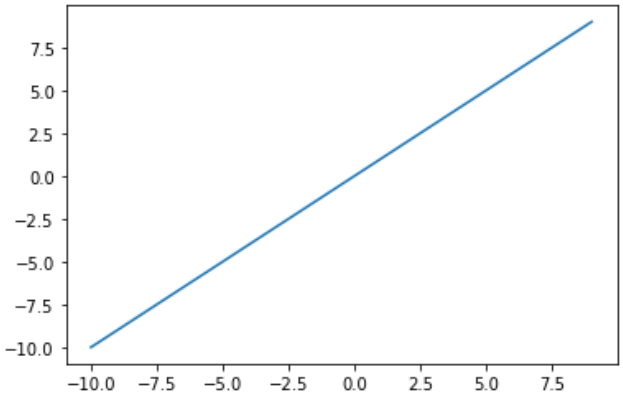
```
# example plot for the linear activation function
from matplotlib import pyplot

# linear activation function
def linear(x):
  return x

# define input data
inputs = [x for x in range(-10, 10)]
# calculate outputs
outputs = [linear(x) for x in inputs]
# plot inputs vs outputs
pyplot.plot(inputs, outputs)
pyplot.show()
```

Figure 9

Inputs vs. outputs plot for Linear activation function.



Linear output activation function

Because the activation function does not alter the weighted sum of the input in any way and instead returns the value straight, the linear activation function is also known as identity or no activation because it is like multiplying by 1.0.

This function, also referred to as identity, enables the input and output to be equal, so if I apply a linear function to a multilayer neural network, it is considered to be a linear regression. Because the neural network to which the function is applied will only produce one value, this linear activation function is utilized when the output requires a linear regression.

With the help of the below-posted solved example, we may gain a general understanding of this function's form (Fig. 8).

When the example is run, the outputs are calculated for a range of values, and an input-to-output plot is produced. If the inputs and outputs are plotted against each other, a diagonal line will appear (Fig. 9).

Sigmoid output activation function

As was already discussed, the so-called leakage gradient problem might render the model more vulnerable to issues during training when using sigmoid functions like Tanh's.

The output of this function, also known as the logistic function, is understood as a probability because the range of possible output values is between zero and one. When a function is evaluated with extremely negative input values, such as x0, it will result in a value of zero, 0.5 when evaluated at zero, and nearly 1 when evaluated at high values. As a result, the last layer uses this function to divide data into two categories.

The sigmoid function is currently underutilized because it is not centered, which has an impact on the neuron's capacity

Figure 10

Example of the Sigmoid activation function.

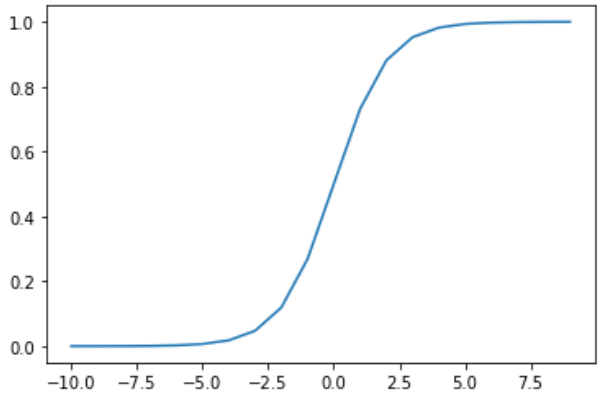
```
# example plot for the sigmoid activation function
from math import exp
from matplotlib import pyplot

# sigmoid activation function
def sigmoid(x):
  return 1.0 / (1.0 + exp(-x))

# define input data
inputs = [x for x in range(-10, 10)]
# calculate outputs
outputs = [sigmoid(x) for x in inputs]
# plot inputs vs outputs
pyplot.plot(inputs, outputs)
pyplot.show()
```

Figure 11

Input vs. output plot for Sigmoid activation function.



for learning and training and, consequently, on the gradient disappearance issue.

To add some symmetry, we can study this function's form using the example that has been solved below (Fig. 10).

When the example is run, the outputs are calculated for a range of values, and an input-to-output plot is produced. It is possible to view the sigmoid activation function's well-known S-shape (Fig. 11).

A model with a sigmoid activation function in the output layer will be trained using target labels that are either 0 or 1.

Softmax output activation function

The relationship to the argmax function is that it generates a 0 for all alternatives and a 1 for the selected option. The softmax function creates a vector of values that total to 1,0 that can be read as probability of class membership. A "softer" variation of argmax called softmax enables an output that resembles the probability of a winner-take-all function.

As a result, the function accepts a vector of real numbers as input, and produces a vector of the same length that has probabilities represented by values that add to 1.0.

As seen below, the softmax function is computed (Eq. 4).

$$\frac{e^x}{sum\,(e^x)} \tag{4}$$

Where e is a mathematical constant that serves as the natural logarithm's basis and x is a vector of outputs.

This function is used to categorize data. For instance, if we are asked to identify the type of fruit from an image, applying softmax to the network will reveal the likelihood that the fruit is either 0.3 or 30% melon, 0.2 or 20% watermelon, or 0.5 or 50% papaya. The outcome will be the fruit with the highest probability, and it should be noted that the sum of these probabilities will be equal to 1. In other words, Softmax is utilized for determining probability for each class that is a part of many classes.

How to choose an output activation function

The type of prediction problem being handled, more especially the kind of variable being predicted, should be considered when selecting the activation function for the output layer.

For instance, you can categorize prediction issues into two main groups: classification and regression. If the problem is a regression problem, you should use a linear activation function; however, there are three types of classification difficulties, so you can have a different function for each (Eq. 5).

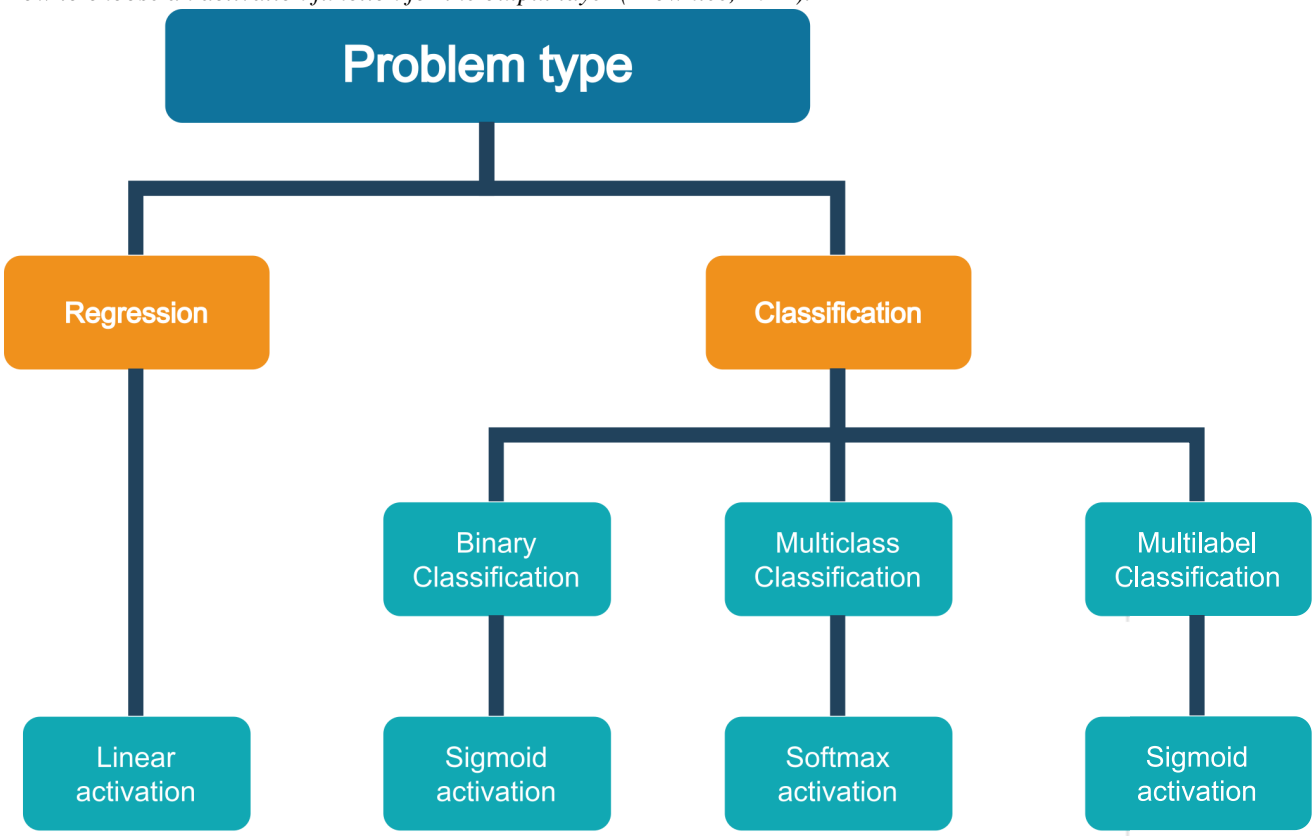
Regression = A node, i.e. a linear activation function (5)

Probability prediction is a classification problem, not a regression one. In each of these scenarios, your model will forecast the likelihood that an example belongs to each class, which you can then turn into a precise class label by rounding (for sigmoid) or argmax (for softmax).

- Binary classification: one node, sigmoid activation.
- Multi-class classification: one node per class, softmax activation.
- Multi-label classification: one node per class, sigmoid activation (Fig. 12).

Figure 12

How to choose an activation function for the output layer (Brownlee, 2021).



Conclusion

In conclusion, choosing the right activation function is an important decision in the design of a neural network. The activation function plays a crucial role in the performance of the network and can greatly impact its ability to learn and generalize to new data. In this paper, we have explored the various activation functions that are commonly used in deep learning, including Sigmoid, Tanh, ReLU, and their variants.

Based on our analysis, it is clear that there is no one-size-fits-all activation function that is universally best for all tasks. Each activation function has its own strengths and weaknesses, and the optimal choice will depend on the specific characteristics of the data and the problem being solved.

In general, activation functions with a smooth, continuous curve, such as Sigmoid and Tanh, can help the network to converge faster and perform better. These functions are also well-suited for binary classification tasks. On the other hand, activation functions with a more complex or discontinuous curve, such as ReLU and its variants, can be more difficult to optimize but may be better suited for certain types of data or tasks. For example, ReLU and its variants are often used in convolutional neural networks for image classification tasks, as they are able to introduce non-linearity and improve the network's ability to capture complex patterns in the data.

In terms of computational efficiency, activation functions that are more computationally efficient, such as ReLU and its variants, can speed up training and make it more efficient. These functions may be particularly useful for large-scale tasks where computational resources are limited.

It is also important to consider the vanishing gradient problem when selecting an activation function. Activation functions that have a slope that remains relatively large for a wide range of input values can help to mitigate the vanishing gradient problem and improve the network's ability to learn. On the other hand, activation functions that have a slope close to 0 for large input values can lead to the vanishing gradient problem and slow down convergence.

In summary, the choice of activation function is an important decision in the design of a neural network. By carefully evaluating the range and shape of the output, computational efficiency, ability to handle non-linearity, and the vanishing gradient problem, it is possible to select the best activation function for a given task. By considering the specific characteristics of the data and the problem being

solved, it is possible to choose an activation function that is well-suited to the task at hand and can help to improve the performance of the neural network.

References

Alonso, A., Peña, A., & Martínez, F. (2021). Autonomous identification of high-contact surfaces from convolutional neural networks. *Journal of Physics: Conference Series*, *2135*(1), 012001.

Brownlee, J. (2021). Machine learning Mastery. https : / / machinelearningmastery . com / choose - an - activation-function-for-deep-learning/

Chang, Y.-W., & Tsai, C.-Y. (2017). Apply deep learning neural network to forecast number of tourists. *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. https : / / doi . org / 10 . 1109 / waina.2017.125

Deng, L., Hinton, G., & Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. https://doi.org/10.1109/icassp.2013.6639344

Ibrahim, M. F. I., & Al-Jumaily, A. A. (2018). Auto-encoder based deep learning for surface electromyography signal processing. *Advances in Science, Technology and Engineering Systems Journal*, *3*(1), 94–102. https://doi.org/10.25046/aj030111

Jacinto, E., Martínez, F., & Martínez, F. (2022). Performance evaluation of temporal and frequential analysis approaches of electromyographic signals for gestures recognition using neural networks. *International Journal of Advanced Computer Science and Applications*, *13*(3), 1–8.

Kim, J.-Y., & Cho, S.-B. (2019). Evolutionary optimization of hyperparameters in deep learning models. *2019 IEEE Congress on Evolutionary Computation (CEC)*. https://doi.org/10.1109/cec.2019.8790354

Maguolo, G., Nanni, L., & Ghidoni, S. (2019). Ensemble of convolutional neural networks trained with different activation functions. *arXiv*, 1–13.

Martínez, F., Hernández, C., & Rendón, A. (2017). A study on machine learning models for convergence time predictions in reactive navigation strategies. *Contemporary Engineering Sciences*, *10*(25), 1223–1232.

Martínez, F., Martínez, F., & Jacinto, E. (2020). Performance evaluation of the nasnet convolutional network in the automatic identification of covid-19. *International Journal on Advanced Science, Engineering and Information Technology*, *10*(2), 662.

Martínez, F., Martínez, F., & Montiel, H. (2020). Low cost, high performance fuel cell energy conditioning system controlled by neural network. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, *18*(6), 3116–3122.

Martínez, F., Montiel, H., & Martínez, F. (2022). A machine learning model for the diagnosis of coffee diseases. *International Journal of Advanced Computer Science and Applications*, *13*(4), 1–8.

Martínez, F., Penagos, C., & Pacheco, L. (2020). Scheme for motion estimation based on adaptive fuzzy neural network. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, *18*(2), 1030–1037.

Montiel, H., Jacinto, E., & Martínez, F. (2021). A double-loop hybrid approach for the recognition of fissures in bone structures. *ARPN Journal of Engineering and Applied Sciences*, *16*(11), 1151–1156.

Montiel, H., Martínez, F., & Jacinto, E. (2017). Visual patterns recognition in robotic platforms through the use of neural networks and image processing. *International Journal of Applied Engineering Research*, *12*(18), 7770–7774.

Pomerat, J., Segev, A., & Datta, R. (2019). On neural network activation functions and optimizers in relation to polynomial regression. *2019 IEEE International Conference on Big Data (Big Data)*. https : / / doi . org / 10 . 1109 / bigdata47090 . 2019 . 9005674

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. *arXiv*, 1–13.

Rendón, A., & Martínez, F. (2021). Intelligent sensor for thermal process control using convolutional neural network. *Journal of Physics: Conference Series*, *1993*(1), 012027.

Rendón, A., Martínez, F., & Hernández, C. (2017). Deep regression model for predictive control in a vegetable waste carbonization plant. *Contemporary Engineering Sciences*, *10*(21), 1047–1055.

Szandala, T. (2021). *Bio-inspired neurocomputing* (A. K. Bhoi, P. K. Mallick, C.-M. Liu, & V. E. Balas, Eds.; Vol. 903). Springer Singapore. https://doi.org/10.1007/978-981-15-5495-7

Zhu, D., Wu, X., & Yang, T. (2022). Benchmarking deep auroc optimization: Loss functions and algorithmic choices. *arXiv*, 1–32.

Zhu, H., Yang, X., & Wang, Y. (2018). Prediction of daily entrance and exit passenger flow of rail transit stations by deep learning method. *Journal of Advanced Transportation*, *2018*(1), 1–11. https://doi.org/10.1155/2018/6142724

Zhu, X., Li, J., Zhu, M., Jiang, Z., & Li, Y. (2018). An evaporation duct height prediction method based on deep learning. *IEEE Geoscience and Remote Sensing Letters*, *15*(9), 1307–1311. https://doi.org/10.1109/lgrs.2018.2842235