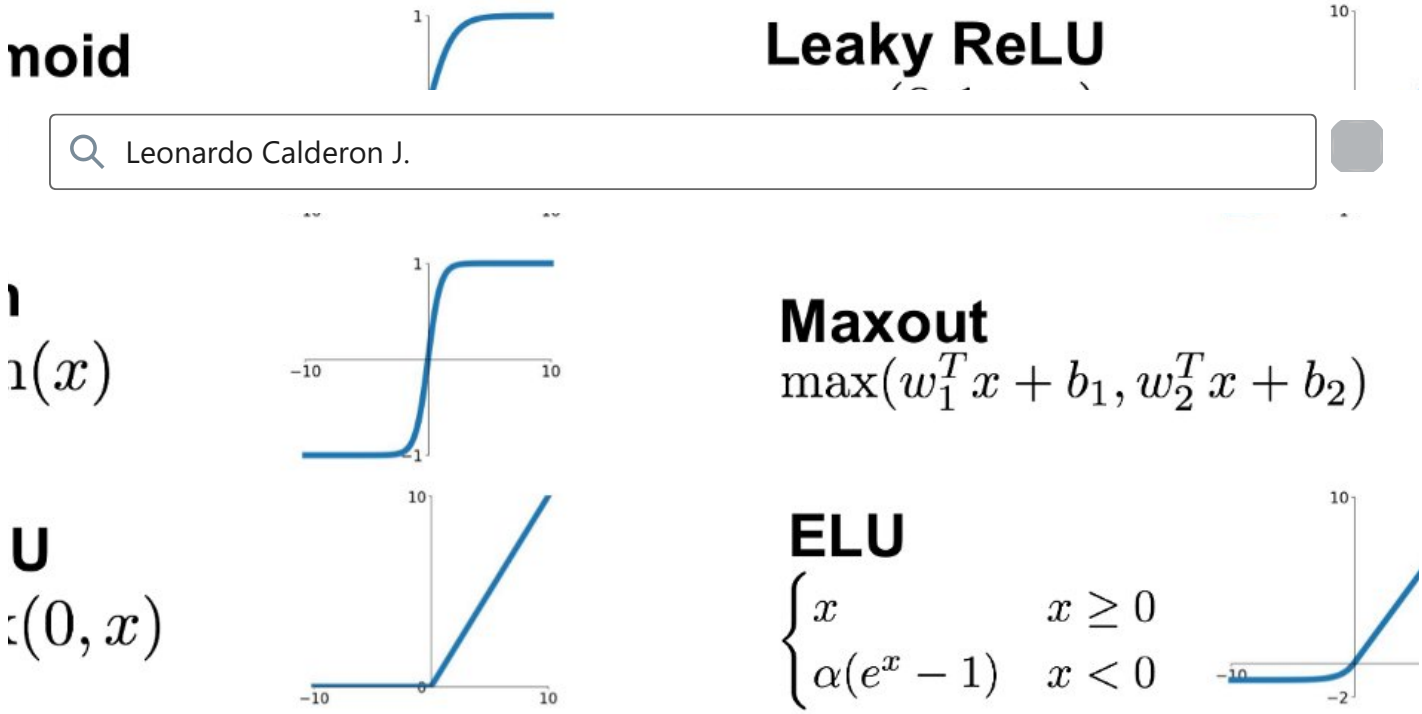

https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17

Activation functions for Neural Networks

Click here

Leonardo Calderon J.
Senior Developer, Endava
Published May 12, 2021

+ Follow

The activation function defines the output given an input or set of inputs for the neuron in a neural network. It is also known as *Transfer Function*. These functions add non-linearity into the network in order to describe patterns in data that are more complicated than a straight line. It is an unavoidable choice because *activation functions* are the foundations for a *neural network* to learn and approximate any kind of complex and continuous relationship between variables.

It is used to determine the output of a neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function). So, for each neuron, we have the following model where each variable is a vector.

```
neuron_output = activation_function(sum(weights * inputs) + bias)
```

The activation function makes it easy for the model to generalize or adapt to a variety of data and to differentiate between the output.

The activation functions can be one of the next functions explained in detail with pros and cons.

Identity Activation Function (Linear)

Is a line, therefore, the output of the functions will not be confined between any range and will be always proportional to the input.

Pros:

- The simplicity of the function and its computation.

Cons:

- It doesn’t help with the complexity of various parameters that are fed to the neural networks.
- The output of the entire network will be always linear (see Linear Transformation), so it doesn't matter the deep of the neural network, it can be always reduced to one single

layer network.

- This network is similar to a ***linear regression model*** which can only address the linear relationship between variables

Logistic Activation Function (Sigmoid)

The Sigmoid Function curve looks like an s-shape. The main reason why we use this function is that it exists between **(0 to 1).** Therefore, it is especially used for models where we have to **predict the probability** as an output.

Pros:

- Since the probability of anything exists only between the range of **0 and 1,** sigmoid is the right choice.
- The function is **differentiable**. That means, we can find the slope of the sigmoid curve at any two points. Note: the function is monotonic but its derivative is not.

Cons:

- Can cause a neural network to get stuck at the training time because it does saturate at both negative and positive regions, this problem is known as the **vanishing gradient problem.** The saturated neuron refuses to learn or keep learning at a very small rate.
- The output is always positive and the output is always accumulated only towards one

side (positive side) so it is not a zero-centered function. Hence during the update step, the weights are only allowed to move in certain directions, not in all the possible directions. It makes the optimization harder.

- e^x, it is highly ***compute-intensive*** which makes convergence slower.
- Can't be used directly for multiclass classification. The **softmax function** is a more generalized logistic activation function that is used for that purpose.

Hyperbolic tangent Activation Function (tanh)

This looks like the sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s-shaped).

Pros:

- The negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.
- The function is **differentiable**. The function is monotonic while its derivative is not monotonic.
- It is a ***zero-centered*** function. Hence optimization becomes comparatively easier than *logistic* and it is always preferred over *logistic*.

Cons:

- It does saturate at both negative and positive regions. May lead to *saturation* and

cause a *vanishing gradient* problem.

- Can't be used directly for multiclass classification. The tanh function is mainly used classification between two classes.
- e^x, it is highly ***compute-intensive.***

Rectified Linear Unit Activation Function (ReLU)

Is half rectified (from bottom), is zero when z is less than zero, and f(z) is equal to z when z is above or equal to zero.

Pros:

- It is the most widely used activation function in neural networks today. It is used in almost all convolutional neural networks or deep learning because of its simplicity.
- It does not activate all neurons at the same time. Set all negative inputs to zero and the neuron does not get activated. This is computationally efficient as few neurons are activated per time.
- It does not saturate at the positive region. In practice, ReLU converges six times faster than tanh and sigmoid activation functions.
- The function is **differentiable**. The function and its derivative **both are monotonic**.

Cons:

- All the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. Affects the resulting graph by not mapping the negative values appropriately.
- With the gradient equal to zero, during backpropagation, all the weights will not be updated.
- Is not zero-centered. This means that for it to get to its optimal point, it will have to use a zig-zag path which may be longer.

Two solutions can be proposed:

1. Initialize the *bias* of the neuron to a large positive value.
2. Use **Leaky ReLu**.

Leaky ReLU Activation Function

Attempt to solve the dying ReLU problem.

Pros are the same as ReLu plus:

- No ***saturation*** problem in the negative region. The leak helps to increase the range of the ReLU function. Usually, the value of **a** is 0.01 or so. When **a is not 0.01** then it is called **Randomized ReLU**.

Cons:

- Is not zero-centered. This means that for it to get to its optimal point, it will have to use a zig-zag path which may be longer.

- The value **a** is selected arbitrary, so it can affect the resulting graph by not mapping the negative values appropriately.

Softmax activation function

Is a type of s*igmoid* function, is a flavor of the *max* function where instead of selecting only one maximum value, it assigns the maximal element largest portion of the distribution.

For a *multiclass* problem, the output needs to be a probability distribution containing a probability value for each class. For a binary classification problem (i.e. two-class problem), the *logistic* activation function works well but not for a *multiclass* classification problem, so is used for this case.

It is generally preferred in the output layer where we are trying to get probabilities for different classes in the output.

Pros:

- Can be used for multiclass classification problem.

- Gives a probability value for each class. If the input value is negative then also *softmax* returns a positive value because e^x always gives positive values.

Cons:

- Same as the Sigmoid activation function.

Logmoid activation function

As a personal contribution, I created my own activation function using the natural logarithm function having the following features in mind:

- Does not saturate at the positive region. Can map big positive values.
- Does not saturate at the negative region. Can map bit negative values.
- The function is **differentiable**.
- Don't have arbitrary parameters to set.
- Zero-centered.
- The gradient is not too close to zero everywhere.

This logmoid function is a piecewise-defined function using ln(x+1) for x equals or greater than zero and -ln(-x+1) for x less than zero, it is continuos in x=0 and is derivable for all x.

This is the derivative of the Logmoid function (orange plus green curves) which are really simple to compute, 1/(x+1) for x equals or greater than zero and 1/(1-x) for x less than zero:

This new function will be tested to compare its behavior against sigmoid, tanh, ReLu and Leaky ReLu with a binary classification problem.

Future work will be focused on a variation of Logmoid for multiclass classification problem (Logmax) that can be of the form:

```
Logmax(Zi) = (Logmoid(Zi) + min(Logmoid(Zi)) / sum((Logmoid(Zi) + min(Logmoid(Zi)))
```


Leonardo Calderon

Holberton School Student

Machine Learning Advanced Program

https://github.com/leocjj

Appendix - Activation functions resume.

Some references:

https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

https://towardsdatascience.com/analyzing-different-types-of-activation-functions-in-neural-networks-which-one-to-prefer-e11649256209

11 · 3 Comments

Like Comment Share

Leonardo Calderon J. 6mo
https://aew61.github.io/blog/artificial_neural_networks/1_background/1.b_activation_functions_and_derivatives.html
Like Reply

Leonardo Calderon J. 6mo
https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8
Like Reply

Leonardo Calderon J. 6mo
I tested the Logmoid activation function I created and it works, is five times faster than sigmoid!!! 👍
Like Reply

See more comments

To view or add a comment, **sign in**

More articles by this author

See all

Bitcoin prediction with LSTM

Bitcoin prediction with LSTM
Sep 17, 2021

Regularization in Neural Networks

Regularization in Neural Networks
May 28, 2021

The unwritten laws (part 1d)

The unwritten laws (part 1d)
Jul 18, 2020

© 2021

Accessibility
User Agreement
Privacy Policy
Cookie Policy
Copyright Policy
Brand Policy
Guest Controls
Community Guidelines
Language