

The Clever Machine

Topics in Computational Neuroscience & Machine Learning

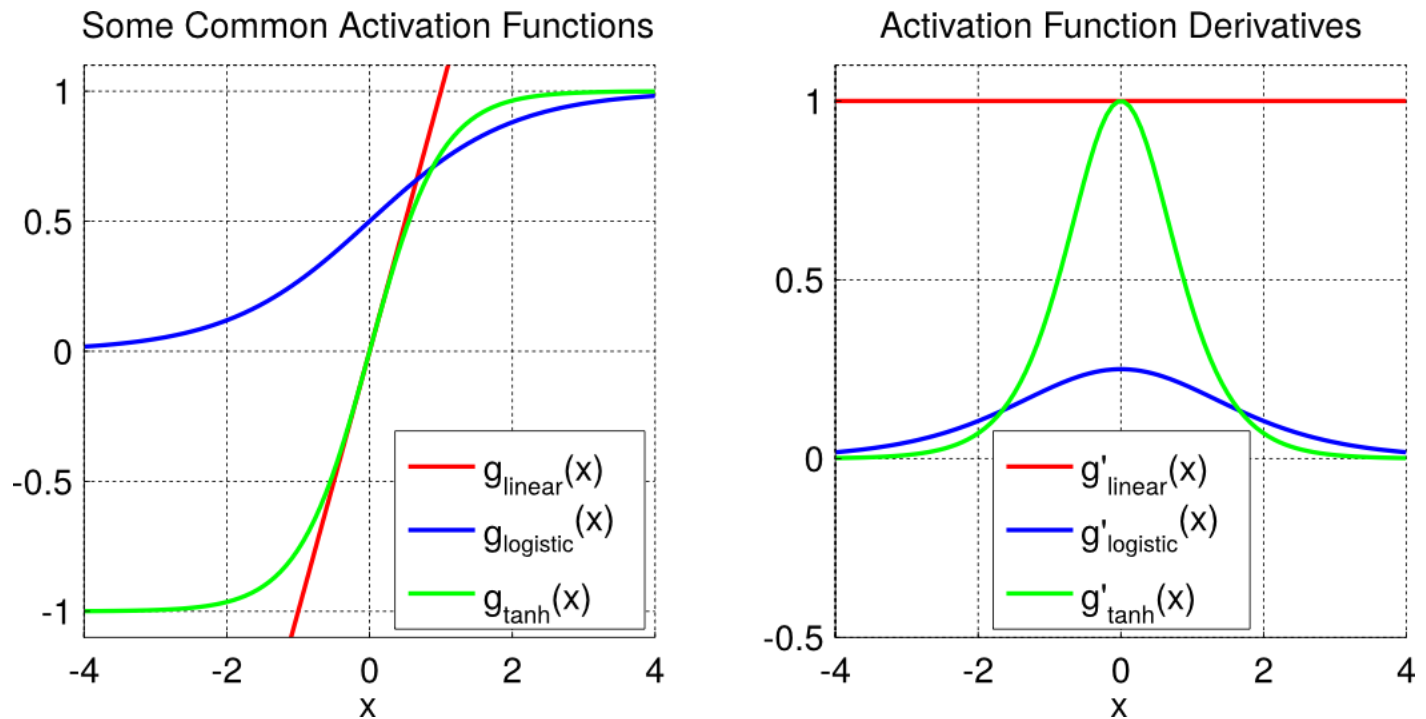
Derivation: Derivatives for Common Neural Network Activation Functions

SEP 8

Posted by dustinstansbury.

Introduction

When constructing Artificial Neural Network (ANN) models, one of the primary considerations is choosing activation functions for hidden and output layers that are differentiable. This is because calculating the backpropagated error signal that is used to determine ANN parameter updates requires the gradient of the activation function gradient. Three of the most commonly-used activation functions used in ANNs are the identity function, the logistic sigmoid function, and the hyperbolic tangent function. Examples of these functions and their associated gradients (derivatives in 1D) are plotted in Figure 1.



(<https://theclevermachine.files.wordpress.com/2014/09/nnet-error-functions2.png>).
Common activation functions functions used in artificial neural, along with their derivatives

In the remainder of this post, we derive the derivatives/ gradients for each of these common activation functions.

The Identity Activation Function

The simplest activation function, one that is commonly used for the output layer activation function in regression problems, is the identity/linear activation function:

$$g_{\text{linear}}(z) = z$$

(Figure 1, red curves). This activation function simply maps the pre-activation to itself and can output values that range $(-\infty, \infty)$. Why would one want to do use an identity activation function? After all, a multi-layered network with linear activations at each layer can be equally-formulated as a single-layered linear network. It turns out that the identity activation function is surprisingly useful. For example, a multi-layer network that has nonlinear activation functions amongst the hidden units and an output layer that uses the identity activation function implements a powerful form of nonlinear regression. Specifically, the network can predict continuous target values using a linear combination of signals that arise from one or more layers of nonlinear transformations of the input.

The derivative of g_{linear} , g'_{linear} , is simply 1, in the case of 1D inputs. For vector inputs of length D the gradient is $\vec{1}^{1 \times D}$, a vector of ones of length D .

The Logistic Sigmoid Activation Function

Another function that is often used as the output activation function for binary classification problems (i.e. outputs values that range $(0, 1)$), is the logistic sigmoid. The logistic sigmoid has the following form:

$$g_{\text{logistic}}(z) = \frac{1}{1+e^{-z}}$$

(Figure 1, blue curves) and outputs values that range $(0, 1)$. The logistic sigmoid is motivated somewhat by biological neurons and can be interpreted as the probability of an artificial neuron “firing” given its inputs. (It turns out that the logistic sigmoid can also be derived as the maximum likelihood solution to for logistic regression (http://en.wikipedia.org/wiki/Logistic_regression) in statistics). Calculating the derivative of the logistic sigmoid function makes use of the quotient rule and a clever trick that both adds and subtracts a one from the numerator:

$$\begin{aligned} g'_{\text{logistic}}(z) &= \frac{\partial}{\partial z} \left(\frac{1}{1+e^{-z}} \right) \\ &= \frac{e^{-z}}{(1+e^{-z})^2} (\text{chain rule}) \\ &= \frac{1+e^{-z}-1}{(1+e^{-z})^2} \\ &= \frac{1+e^{-z}}{(1+e^{-z})^2} - \left(\frac{1}{1+e^{-z}} \right)^2 \\ &= \frac{1}{(1+e^{-z})} - \left(\frac{1}{1+e^{-z}} \right)^2 \\ &= g_{\text{logistic}}(z) - g_{\text{logistic}}(z)^2 \\ &= g_{\text{logistic}}(z)(1 - g_{\text{logistic}}(z)) \end{aligned}$$

Here we see that $g'_{\text{logistic}}(z)$ evaluated at z is simply $g_{\text{logistic}}(z)$ weighted by $1 - g_{\text{logistic}}(z)$. This turns out to be a convenient form for efficiently calculating gradients used in neural networks: if one keeps in memory the feed-forward activations of the logistic function for a given layer, the gradients for that layer can be evaluated using simple multiplication and subtraction rather than performing any re-evaluating the sigmoid function, which requires extra exponentiation.

The Hyperbolic Tangent Activation Function

Though the logistic sigmoid has a nice biological interpretation, it turns out that the logistic sigmoid can cause a neural network to get “stuck” during training. This is due in part to the fact that if a strongly-negative input is provided to the logistic sigmoid, it outputs values very near zero. Since neural networks use the feed-forward activations to calculate parameter gradients (again, see this [previous post](#)

(<https://theclevermachine.wordpress.com/2014/09/06/derivation-error-backpropagation-gradient-descent-for-neural-networks/>) for details), this can result in model parameters that are updated less regularly than we would like, and are thus “stuck” in their current state.

An alternative to the logistic sigmoid is the hyperbolic tangent, or tanh function (Figure 1, green curves):

$$\begin{aligned} g_{\tanh}(z) &= \frac{\sinh(z)}{\cosh(z)} \\ &= \frac{e^z - e^{-z}}{e^z + e^{-z}} \end{aligned}$$

Like the logistic sigmoid, the tanh function is also sigmoidal (“s”-shaped), but instead outputs values that range $(-1, 1)$. Thus strongly negative inputs to the tanh will map to negative outputs. Additionally, only zero-valued inputs are mapped to near-zero outputs. These properties make the network less likely to get “stuck” during training. Calculating the gradient for the tanh function also uses the quotient rule:

$$\begin{aligned} g'_{\tanh}(z) &= \frac{\partial}{\partial z} \frac{\sinh(z)}{\cosh(z)} \\ &= \frac{\frac{\partial}{\partial z} \sinh(z) \times \cosh(z) - \frac{\partial}{\partial z} \cosh(z) \times \sinh(z)}{\cosh^2(z)} \\ &= \frac{\cosh^2(z) - \sinh^2(z)}{\cosh^2(z)} \\ &= 1 - \frac{\sinh^2(z)}{\cosh^2(z)} \\ &= 1 - \tanh^2(z) \end{aligned}$$


Similar to the derivative for the logistic sigmoid, the derivative of $g_{\tanh}(z)$ is a function of feed-forward activation evaluated at z , namely $(1 - g_{\tanh}(z)^2)$. Thus the same caching trick can be used for layers that implement tanh activation functions.


Wrapping Up

In this post we reviewed a few commonly-used activation functions in neural network literature and their derivative calculations. These activation functions are motivated by biology and/or provide some handy implementation tricks like calculating derivatives using cached feed-forward activation values. Note that there are also many other options for activation functions not covered here: e.g. rectification, soft rectification, polynomial kernels, etc. Indeed, finding and evaluating novel activation functions is an active subfield of machine learning research. However, the three basic activations covered here can be used to solve a majority of the machine learning problems one will likely face.

ADVERTISEMENT

Advertisements



 **Yoga on Your Vacation**
No matter how many miles your travel takes you,
it is possible to bring yoga along for the ride.

[Report this ad](#)



 **Massachusetts:**
New Rule in Somerville, MA Leaves Drivers Fuming

[Report this ad](#)



About dustinstansbury

I recently received my PhD from UC Berkeley where I studied computational neuroscience and machine learning.

[View all posts by dustinstansbury »](#)

Posted on September 8, 2014, in [Classification](#), [Derivations](#), [Machine Learning](#), [Neural Networks](#), [Regression](#) and tagged [Backpropagation](#), [backpropagation algorithm](#), [Logistic Sigmoid](#), [Neural Networks](#), [Quotient Rule](#), [Tanh Function](#). Bookmark the [permalink](#). [4 Comments](#).

- **Leave a comment**

- **Trackbacks 1**

- **Comments 3**

James | June 7, 2015 at 5:16 am

It was very helpful! Thanks!

What about softmax function?

James | February 4, 2016 at 12:54 am

Related: <https://brenocon.com/blog/2013/10/tanh-is-a-rescaled-logistic-sigmoid-function/>

Khang Pham | March 6, 2017 at 7:13 pm

Thank you for this great explanation!

1. Pingback: **A Gentle Introduction to Artificial Neural Networks | The Clever Machine**

[Create a free website or blog at WordPress.com.](#)