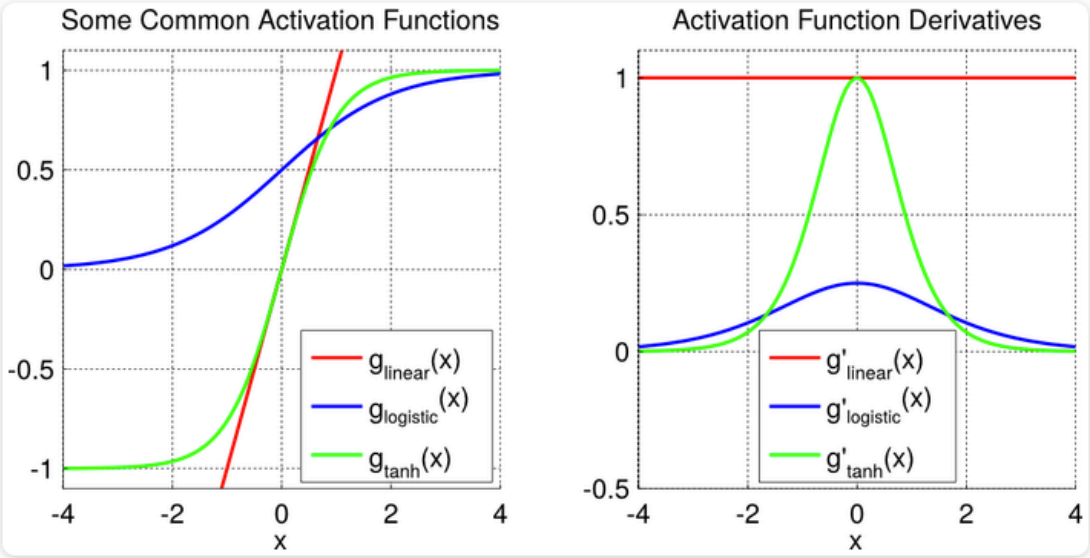
Activation Function and its Derivatives :
==========

https://theclevermachine.wordpress.com/2014/09/08/derivation-derivatives-for-common-neural-network-activation-functions/



Common activation functions functions used in artificial neural, along with their derivatives

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 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 $g'_{logistic}(z)$ evaluated at z is simply $g_{logistic}(z)$ weighted by 1-minus-$g_{logistic}(z)$. 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 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.