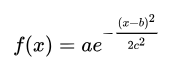
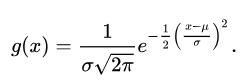
for arbitrary [real](https://en.wikipedia.org/wiki/Real_number) constants *a*, *b* and *c*. It is named after the mathematician [Carl Friedrich Gauss](https://en.wikipedia.org/wiki/Carl_Friedrich_Gauss). The [graph](https://en.wikipedia.org/wiki/Graph_of_a_function) of a Gaussian is a characteristic symmetric "[bell](https://en.wikipedia.org/wiki/Bell_(instrument)) curve" shape. The parameter *a* is the height of the curve's peak, *b* is the position of the center of the peak and *c* (the [standard deviation](https://en.wikipedia.org/wiki/Standard_deviation), sometimes called the Gaussian [RMS](https://en.wikipedia.org/wiki/Root_mean_square) width) controls the width of the "bell".

Gaussian functions are often used to represent the [probability density function](https://en.wikipedia.org/wiki/Probability_density_function) of a [normally distributed](https://en.wikipedia.org/wiki/Normal_distribution) [random variable](https://en.wikipedia.org/wiki/Random_variable) with [expected value](https://en.wikipedia.org/wiki/Expected_value) *μ* = *b* and [variance](https://en.wikipedia.org/wiki/Variance) *σ*2 = *c*2. In this case, the Gaussian is of the form:

{\displaystyle g(x)={\frac {1}{\sigma {\sqrt {2\pi }}}}e^{-{\frac {1}{2}}\left({\frac {x-\mu }{\sigma }}\right)^{2}}.}





***Likelihood Loss***

The [likelihood function](http://www.statisticshowto.com/likelihood-function/) is also relatively simple, and is commonly used in classification problems. The function takes the predicted probability for each input example and multiplies them. And although the output isn’t exactly human interpretable, it’s useful for comparing models.

For example, consider a model that outputs probabilities of [0.4, 0.6, 0.9, 0.1] for the ground truth labels of [0, 1, 1, 0]. The likelihood loss would be computed as (0.6) \* (0.6) \* (0.9) \* (0.9) = 0.2916. Since the model outputs probabilities for TRUE (or 1) only, when the ground truth label is 0 we take (1-p) as the probability. In other words, we multiply the model’s outputted probabilities together for the actual outcomes.

***Log Loss (Cross Entropy Loss)***

[Log Loss](http://wiki.fast.ai/index.php/Log_Loss) is a loss function also used frequently in classification problems, and is one of the most popular measures for [Kaggle](https://www.kaggle.com/) competitions. It’s just a straightforward modification of the likelihood function with logarithms.

https://blog.algorithmia.com/wp-content/uploads/2018/04/word-image-6.png

This is actually exactly the same formula as the regular likelihood function, but with logarithms added in. You can see that when the actual class is 1, the second half of the function disappears, and when the actual class is 0, the first half drops. That way, we just end up multiplying the log of the actual predicted probability for the ground truth class.

The cool thing about the log loss loss function is that is has a kick: it penalizes heavily for being *very confident*and *very wrong*. Predicting high probabilities for the wrong class makes the function go crazy. The graph below is for when the true label =1, and you can see that it skyrockets as the predicted probability for label = 0 approaches 1.

### **Loss Functions and Optimizers**

Loss functions provide more than just a static representation of how your model is performing–they’re how your algorithms fit data in the first place. Most machine learning algorithms use some sort of loss function in the process of optimization, or finding the best parameters (weights) for your data.

For a simple example, consider [linear regression](https://onlinecourses.science.psu.edu/stat501/node/251). In traditional “least squares” regression, the line of best fit is determined through none other than MSE (hence the least squares moniker)! For each set of weights that the model tries, the MSE is calculated across all input examples. The model then optimizes the MSE functions––or in other words, makes it the lowest possible––through the use of an optimizer algorithm like [Gradient Descent](https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0).

Just like there are different flavors of loss functions for unique problems, there is no shortage of different optimizers as well. That’s beyond the scope of this post, but in essence, the loss function and optimizer work in tandem to fit the algorithm to your data in the best way possible.

They usually fall into categories like

1. Gradient based methods using first order information e.g. Batch Gradient Descent and Stochastic gradient descent
2. Gradient based methods using second order information (either by computing the Hessian or approximating it e.g. Newton method, conjugate gradient, scaled conjugate gradient
3. Search based techniques e.g genetic algorithms, simulated annealing e.t.c. These techniques usually don't require the function being optimised to be differentiable, they try to find a solution by sampling from a probability distribution.