Side exploration

- To summarize, for a neural network classifier,
  - during training you can use mean squared error or average cross-entropy error, and average cross-entropy error is considered slightly better.
    - If you use MSE, the weight adjustment factor (the gradient) contains a term of (output) * (1 - output). As the adjustment factor (the diff between y and t) gets smaller and smaller, the change in weights gets smaller and smaller and training can stall out
    - But if you use cross-entropy error, the (output) * (1 - output) term goes away. So, the weight changes don't get smaller and smaller and so training isn't s likely to stall out. Note that this argument assumes you're doing neural network classification with softmax output node activation.
  - If you are using back-propagation, the choice of MSE or ACE affects the computation of the gradient.
  - After training, to estimate the effectiveness of the neural network it's better to use classification error.
- mean squared error
  - (0.1 - 0)^2 + (0.3 - 0)^2 + (0.6 - 1)^2
- classification error
  - (y' - y) ^ 2
- cross-entropy
  - y log(y')
  - below, y' is the target/ground truth output, while y is the estimated output.

One way to interpret cross-entropy is to see it as a (minus) log-likelihood for the data y'_i, under a model y_i.

Namely, suppose that you have some fixed model (a.k.a. "hypothesis"), which predicts for n classes $\{1, 2, \ldots, n\}$ their hypothetical occurrence probabilities $y_1, y_2, \ldots, y_n$. Suppose that you now observe (in reality) k_1 instances of class 1, k_2 instances of class 2, k_n instances of class n, etc. According to your model the likelihood of this happening is:

$$P[data|model] := y_1^{k_1} y_2^{k_2} \ldots y_n^{k_n}.$$

Taking the logarithm and changing the sign:

$$-\log P[data|model] = -k_1 \log y_1 - k_2 \log y_2 - \cdots - k_n \log y_n = -\sum_i k_i \log y_i$$

If you now divide the right-hand sum by the number of observations $N = k_1 + k_2 + \cdots + k_n$, and denote the empirical probabilities as $y'_i = k_i/N$, you'll get the cross-entropy:

$$-\frac{1}{N}\log P[data|model] = -\frac{1}{N}\sum_i k_i \log y_i = -\sum_i y'_i \log y_i =: H(y', y)$$

Furthermore, a log-likelihood of a dataset given a model can be interpreted as a measure of "encoding length" - the number of bits you expect to spend to encode this information if your encoding scheme would be based on your hypothesis.

This follows from the observation that an independent event with probability y_i requires at least $-\log_2 y_i$ bits to encode it (assuming efficient coding), and consequently the expression

$$-\sum_i y'_i \log_2 y_i,$$

is literally the expected length of the encoding, where the encoding lengths for the events are computed using the "hypothesized" distribution, while the expectation is taken over the actual one.