# Input Output Scaling

First, some definitions. "Rescaling" a vector means to add or subtract a constant and then multiply or divide by a constant, as you would do to change the units of measurement of the data, for example, to convert a temperature from Celsius to Fahrenheit.

"Normalizing" a vector most often means dividing by a norm of the vector, for example, to make the Euclidean length of the vector equal to one. In the NN literature, "normalizing" also often refers to rescaling by the minimum and range of the vector, to make all the elements lie between 0 and 1.

"Standardizing" a vector most often means subtracting a measure of location and dividing by a measure of scale. For example, if the vector contains random values with a Gaussian distribution, you might subtract the mean and divide by the standard deviation, thereby obtaining a "standard normal" random variable with mean 0 and standard deviation 1.

However, all of the above terms are used more or less interchangeably depending on the customs within various fields. Since the FAQ maintainer is a statistician, he is going to use the term "standardize" because that is what he is accustomed to.

Now the question is, should you do any of these things to your data? The answer is, it depends.

There is a common misconception that the inputs to a multilayer perceptron must be in the interval [0,1]. There is in fact no such requirement, although there often are benefits to standardizing the inputs as discussed below. But it is better to have the input values centered around zero, so scaling the inputs to the interval [0,1] is usually a bad choice.

If your output activation function has a range of [0,1], then obviously you must ensure that the target values lie within that range. But it is generally better to choose an output activation function suited to the distribution of the targets than to force your data to conform to the output activation function.

When using an output activation with a range of [0,1], some people prefer to rescale the targets to a range of [.1,.9]. I suspect that the popularity of this gimmick is due to the slowness of standard backprop. But using a target range of [.1,.9] for a classification task gives you incorrect posterior probability estimates. This gimmick is unnecessary if you use an efficient training algorithm and it is also unnecessary to avoid overflow.

Now for some of the gory details: note that the training data form a matrix. Let's set up this matrix so that each case forms a row, and the inputs and target variables form columns. You could conceivably standardize the rows or the columns or both or various other things, and these different ways of choosing vectors to standardize will have quite different effects on training.

Standardizing either input or target variables tends to make the training process better behaved by improving the numerical condition of the optimization problem and ensuring that various default values involved in initialization and termination are appropriate. Standardizing targets can also affect the objective function.

Standardization of cases should be approached with caution because it discards information. If that information is irrelevant, then standardizing cases can be quite helpful. If that information is important, then standardizing cases can be disastrous.

**Should I standardize the input variables (column vectors)?**

That depends primarily on how the network combines input variables to compute the net input to the next (hidden or output) layer. If the input variables are combined via a distance function (such as Euclidean distance) in an RBF network, standardizing inputs can be crucial. The contribution of an input will depend heavily on its variability relative to other inputs. If one input has a range of 0 to 1, while another input has a range of 0 to 1,000,000, then the contribution of the first input to the distance will be swamped by the second input. So it is essential to rescale the inputs so that their variability reflects their importance, or at least is not in inverse relation to their importance. For lack of better prior information, it is common to standardize each input to the same range or the same standard deviation. If you know that some inputs are more important than others, it may help to scale the inputs such that the more important ones have larger variances and/or ranges.
If the input variables are combined linearly, as in an MLP, then it is rarely strictly necessary to standardize the inputs, at least in theory. The reason is that any rescaling of an input vector can be effectively undone by changing the corresponding weights and biases, leaving you with the exact same outputs as you had before. However, there are a variety of practical reasons why standardizing the inputs can make training faster and reduce the chances of getting stuck in local optima. Also, weight decay and Bayesian estimation can be done more conveniently with standardized inputs.
The main emphasis in the NN literature on initial values has been on the avoidance of saturation, hence the desire to use small random values. How small these random values should be depends on the scale of the inputs as well as the number of inputs and their correlations. Standardizing inputs removes the problem of scale dependence of the initial weights.
But standardizing input variables can have far more important effects on initialization of the weights than simply avoiding saturation. Assume we have an MLP with one hidden layer applied to a classification problem and are therefore interested in the hyperplanes defined by each hidden unit. Each hyperplane is the locus of points where the net-input to the hidden unit is zero and is thus the classification boundary generated by that hidden unit considered in isolation. The connection weights from the inputs to a hidden unit determine the orientation of the hyperplane. The bias determines the distance of the hyperplane from the origin. If the bias terms are all small random numbers, then all the hyperplanes will pass close to the origin. Hence, if the data are not centered at the origin, the hyperplane may fail to pass through the data cloud. If all the inputs have a small coefficient of variation, it is quite possible that all the initial hyperplanes will miss the data entirely. With such a poor initialization, local minima are very likely to occur. It is

therefore important to center the inputs to get good random initializations. In particular, scaling the inputs to [-1,1] will work better than [0,1], although any scaling that sets to zero the mean or median or other measure of central tendency is likely to be as good or better.
Standardizing input variables also has different effects on different training algorithms for MLPs. For example:

- Steepest descent is very sensitive to scaling. The more ill-conditioned the Hessian is, the slower the convergence. Hence, scaling is an important consideration for gradient descent methods such as standard backprop.
- Quasi-Newton and conjugate gradient methods begin with a steepest descent step and therefore are scale sensitive. However, they accumulate second-order information as training proceeds and hence are less scale sensitive than pure gradient descent.
- Newton-Raphson and Gauss-Newton, if implemented correctly, are theoretically invariant under scale changes as long as none of the scaling is so extreme as to produce underflow or overflow.
- Levenberg-Marquardt is scale invariant as long as no ridging is required. There are several different ways to implement ridging; some are scale invariant and some are not. Performance under bad scaling will depend on details of the implementation.

Two of the most useful ways to standardize inputs are:

- Mean 0 and standard deviation 1
- Midrange 0 and range 2 (i.e., minimum -1 and maximum 1)

Formulas are as follows:
Notation:

$X_i$ = value of the raw input variable X for the ith training case

$S_i$ = standardized value corresponding to $X_i$

N = number of training cases

Standardize $X_i$ to mean 0 and standard deviation 1:

$$\text{mean} = \frac{\text{sum } X_i}{N}$$

$$\text{std} = \text{sqrt}\left( \frac{\text{sum}( X_i - \text{mean})^2}{N - 1} \right)$$

$$S_i = \frac{X_i - \text{mean}}{\text{std}}$$

Standardize $X_i$ to midrange 0 and range 2:

$$\text{midrange} = \frac{\max X_i + \min X_i}{2}$$

$$\text{range} = \max X_i - \min X_i$$

$$S_i = \frac{X_i - \text{midrange}}{\text{range} / 2}$$

Various other pairs of location and scale estimators can be used besides the mean and standard deviation, or midrange and range. Robust estimates of location and scale are desirable if the inputs contain outliers. For example, see:

> Iglewicz, B. (1983), "Robust scale estimators and confidence intervals for location", in Hoaglin, D.C., Mosteller, M. and Tukey, J.W., eds., *Understanding Robust and Exploratory Data Analysis*, NY: Wiley.

**Should I standardize the target variables (column vectors)?**

Standardizing target variables is typically more a convenience for getting good initial weights than a necessity. However, if you have two or more target variables and your error function is scale-sensitive like the usual least (mean) squares error function, then the variability of each target relative to the others can effect how well the net learns that target. If one target has a range of 0 to 1, while another target has a range of 0 to 1,000,000, the net will expend most of its effort learning the second target to the possible exclusion of the first. So it is essential to rescale the targets so that their variability reflects their importance, or at least is not in inverse relation to their importance. If the targets are of equal importance, they should typically be standardized to the same range or the same standard deviation.

The scaling of the targets does not affect their importance in training if you use maximum likelihood estimation and estimate a separate scale parameter (such as a standard deviation) for each target variable. In this case, the importance of each target is inversely related to its estimated scale parameter. In other words, noisier targets will be given less importance.

For weight decay and Bayesian estimation, the scaling of the targets affects the decay values and prior distributions. Hence it is usually most convenient to work with standardized targets.

If you are standardizing targets to equalize their importance, then you should probably standardize to mean 0 and standard deviation 1, or use related robust estimators, as discussed under Should I standardize the input variables (column vectors)? If you are

standardizing targets to force the values into the range of the output activation function, it is important to use lower and upper bounds for the values, rather than the minimum and maximum values in the training set. For example, if the output activation function has range [-1,1], you can use the following formulas:

$Y_i$ = value of the raw target variable Y for the ith training case

$Z_i$ = standardized value corresponding to $Y_i$

$$\text{midrange} = \frac{\text{upper bound of Y} + \text{lower bound of Y}}{2}$$

$$\text{range} = \text{upper bound of Y} - \text{lower bound of Y}$$

$$Z_i = \frac{Y_i - \text{midrange}}{\text{range} / 2}$$

For a range of [0,1], you can use the following formula:

$$Z_i = \frac{Y_i - \text{lower bound of Y}}{\text{upper bound of Y} - \text{lower bound of Y}}$$

And of course, you apply the inverse of the standardization formula to the network outputs to restore them to the scale of the original target values.
If the target variable does not have known upper and lower bounds, it is not advisable to use an output activation function with a bounded range. You can use an identity output activation function or other unbounded output activation function instead

**Should I standardize the variables (column vectors) for unsupervised learning?**

The most commonly used methods of unsupervised learning, including various kinds of vector quantization, Kohonen networks, Hebbian learning, etc., depend on Euclidean distances or scalar-product similarity measures. The considerations are therefore the same as for standardizing inputs in RBF networks--see “Should I standardize the input variables (column vectors)?” above. In particular, if one input has a large variance and another a small variance, the latter will have little or no influence on the results.
If you are using unsupervised competitive learning to try to discover natural clusters in the data, rather than for data compression, simply standardizing the variables may be inadequate. For more sophisticated methods of preprocessing, see:

Art, D., Gnanadesikan, R., and Kettenring, R. (1982), "Data-based Metrics for Cluster Analysis," Utilitas Mathematica, 21A, 75-99.

Jannsen, P., Marron, J.S., Veraverbeke, N, and Sarle, W.S. (1995), "Scale measures for bandwidth selection", J. of Nonparametric Statistics, 5, 359-380.

Better yet for finding natural clusters, try mixture models or nonparametric density estimation. For example::

Girman, C.J. (1994), "Cluster Analysis and Classification Tree Methodology as an Aid to Improve Understanding of Benign Prostatic Hyperplasia," Ph.D. thesis, Chapel Hill, NC: Department of Biostatistics, University of North Carolina.

McLachlan, G.J. and Basford, K.E. (1988), Mixture Models, New York: Marcel Dekker, Inc.

SAS Institute Inc. (1993), SAS/STAT Software: The MODECLUS Procedure, SAS Technical Report P-256, Cary, NC: SAS Institute Inc.

Titterington, D.M., Smith, A.F.M., and Makov, U.E. (1985), Statistical Analysis of Finite Mixture Distributions, New York: John Wiley & Sons, Inc.

Wong, M.A. and Lane, T. (1983), "A kth Nearest Neighbor Clustering Procedure," Journal of the Royal Statistical Society, Series B, 45, 362-368.

**Should I standardize the input cases (row vectors)?**

Whereas standardizing variables is usually beneficial, the effect of standardizing cases (row vectors) depends on the particular data. Cases are typically standardized only across the input variables, since including the target variable(s) in the standardization would make prediction impossible.

There are some kinds of networks, such as simple Kohonen nets, where it is necessary to standardize the input cases to a common Euclidean length; this is a side effect of the use of the inner product as a similarity measure. If the network is modified to operate on Euclidean distances instead of inner products, it is no longer necessary to standardize the input cases.

Standardization of cases should be approached with caution because it discards information. If that information is irrelevant, then standardizing cases can be quite helpful. If that information is important, then standardizing cases can be disastrous. Issues regarding the standardization of cases must be carefully evaluated in every application. There are no rules of thumb that apply to all applications.

You may want to standardize each case if there is extraneous variability between cases. Consider the common situation in which each input variable represents a pixel in an image. If the images vary in exposure, and exposure is irrelevant to the target values, then it would usually help to subtract the mean of each case to equate the exposures of different cases. If the images vary in contrast, and contrast is irrelevant to the target values, then it would usually help to divide each case by its standard deviation to equate the contrasts of different cases. Given sufficient data, a NN could learn to ignore exposure and contrast. However, training will be easier and generalization better if you can remove the extraneous exposure and contrast information before training the network.

As another example, suppose you want to classify plant specimens according to species but the specimens are at different stages of growth. You have measurements such as stem length, leaf length, and leaf width. However, the over-all size of the specimen is determined by age or growing conditions, not by species. Given sufficient data, a NN could learn to ignore the size of the specimens and classify them by shape instead.

However, training will be easier and generalization better if you can remove the extraneous size information before training the network. Size in the plant example corresponds to exposure in the image example.

If the input data are measured on an interval scale, you can control for size by subtracting a measure of the over-all size of each case from each datum. For example, if no other direct measure of size is available, you could subtract the mean of each row of the input matrix, producing a row-centered input matrix.

If the data are measured on a ratio scale, you can control for size by dividing each datum by a measure of over-all size. It is common to divide by the sum or by the arithmetic mean. For positive ratio data, however, the geometric mean is often a more natural measure of size than the arithmetic mean. It may also be more meaningful to analyze the logarithms of positive ratio-scaled data, in which case you can subtract the arithmetic mean after taking logarithms. You must also consider the dimensions of measurement. For example, if you have measures of both length and weight, you may need to cube the measures of length or take the cube root of the weights.

In NN aplications with ratio-level data, it is common to divide by the Euclidean length of each row. If the data are positive, dividing by the Euclidean length has properties similar to dividing by the sum or arithmetic mean, since the former projects the data points onto the surface of a hypersphere while the latter projects the points onto a hyperplane. If the dimensionality is not too high, the resulting configurations of points on the hypersphere and hyperplane are usually quite similar. If the data contain negative values, then the hypersphere and hyperplane can diverge widely.