The best number of hidden units depends in a complex way on:

 o the numbers of input and output units
 o the number of training cases
 o the amount of noise in the targets
 o the complexity of the function or classification to be learned
 o the architecture
 o the type of hidden unit activation function
 o the training algorithm
 o regularization

In most situations, there is no way to determine the best number of hidden units without training several networks and estimating the generalization error of each. If you have too few hidden units, you will get high training error and high generalization error due to underfitting and high statistical bias. If you have too many hidden units, you may get low training error but still have high generalization error due to overfitting and high variance. Geman, Bienenstock, and Doursat (1992) discuss how the number of hidden units affects the bias/variance trade-off.

Some books and articles offer "rules of thumb" for choosing an architecture; for example:

 o "A rule of thumb is for the size of this [hidden] layer to be somewhere
   between the input layer size ... and the output layer size ..." (Blum,
   1992, p. 60).
 o "To calculate the number of hidden nodes we use a general rule of:
   (Number of inputs + outputs) * (2/3)" (from the FAQ for a commercial
   neural network software company).
 o "you will never require more than twice the number of hidden units as you
   have inputs' in an MLP with one hidden layer (Swingler, 1996, p. 53). See
   the section in Part 4 of the FAQ on The Worst books for the source of
   this myth.)
 o "How large should the hidden layer be? One rule of thumb is that it
   should never be more than twice as large as the input layer." (Berry and
   Linoff, 1997, p. 323).
 o "Typically, we specify as many hidden nodes as dimensions [principal
   components] needed to capture 70-90% of the variance of the input data
   set." (Boger and Guterman, 1997)

These rules of thumb are nonsense because they ignore the number of training cases, the amount of noise in the targets, and the complexity of the function. Even if you restrict consideration to minimizing training error on data with lots of training cases and no noise, it is easy to construct counterexamples that disprove these rules of thumb. For example:

 o There are 100 Boolean inputs and 100 Boolean targets. Each target is a
   conjunction of some subset of inputs. No hidden units are needed.

 o There are two continuous inputs X and Y which take values uniformly
   distributed on a square [0,8] by [0,8]. Think of the input space as a
   chessboard, and number the squares 1 to 64. The categorical target
   variable C is the square number, so there are 64 output units. For
   example, you could generate the data as follows (this is the SAS
   programming language, but it should be easy to translate into any other
   language):

   data chess;
      step = 1/4;
      do x = step/2 to 8-step/2 by step;
         do y = step/2 to 8-step/2 by step;
            c = 8*floor(x) + floor(y) + 1;
            output;
         end;
      end;
   run;

   No hidden units are needed.

 o The classic two-spirals problem has two continuous inputs and a Boolean
   classification target. The data can be generated as follows:

   data spirals;
      pi = arcos(-1);
      do i = 0 to 96;
         angle = i*pi/16.0;
         radius = 6.5*(104-i)/104;
         x = radius*cos(angle);
         y = radius*sin(angle);
         c = 1;
         output;
         x = -x;
         y = -y;
         c = 0;
         output;
      end;
   run;

   With one hidden layer, about 50 tanh hidden units are needed. Many NN
   programs may actually need closer to 100 hidden units to get zero
   training error.

 o There is one continuous input X that takes values on [0,100]. There is
   one continuous target Y = sin(X). Getting a good approximation to Y
   requires about 20 to 25 tanh hidden units. Of course, 1 sine hidden unit
   would do the job.

Some rules of thumb relate the total number of trainable weights in the network to the number of training cases. A typical recommendation is that the number of weights should be no more than 1/30 of the number of training cases. Such rules are only concerned with overfitting and are at best crude approximations. Also, these rules do not apply when regularization is used. It is true that without regularization, if the number of training cases is much larger (but no one knows exactly how much larger) than the number of weights, you are unlikely to get overfitting, but you may suffer from underfitting. For a noise-free quantitative target variable, twice as many training cases as weights may be more than enough to avoid overfitting. For a very noisy categorical target variable, 30 times as many training cases as weights may not be enough to avoid overfitting.

An intelligent choice of the number of hidden units depends on whether you are using early stopping or some other form of regularization. If not, you must simply try many networks with different numbers of hidden units, estimate the generalization error for each one, and choose the network with the minimum estimated generalization error. For examples using statistical criteria to choose the number of hidden units, see
ftp://ftp.sas.com/pub/neural/dojo/dojo.html.

Using conventional optimization algorithms (see "What are conjugate gradients, Levenberg-Marquardt, etc.?"), there is little point in trying a network with more weights than training cases, since such a large network is likely to overfit.

Using standard online backprop, however, Lawrence, Giles, and Tsoi (1996, 1997) have shown that it can be difficult to reduce training error to a level near the globally optimal value, even when using more weights than training cases. But increasing the number of weights makes it easier for standard backprop to find a good local optimum, so using "oversize" networks can reduce both training error and generalization error.

If you are using early stopping, it is essential to use lots of hidden units to avoid bad local optima (Sarle 1995). There seems to be no upper limit on the number of hidden units, other than that imposed by computer time and memory requirements. Weigend (1994) makes this assertion, but provides only one example as evidence. Tetko, Livingstone, and Luik (1995) provide simulation studies that are more convincing. Similar results were obtained in conjunction with the simulations in Sarle (1995), but those results are not reported in the paper for lack of space. On the other hand, there seems to be no advantage to using more hidden units than you have training cases, since bad local minima do not occur with so many hidden units.

If you are using weight decay or Bayesian estimation, you can also use lots of hidden units (Neal 1996). However, it is not strictly necessary to do so, because other methods are available to avoid local minima, such as multiple random starts and simulated annealing (such methods are not safe to use with early stopping). You can use one network with lots of hidden units, or you can try different networks with different numbers of hidden units, and choose on the basis of estimated generalization error. With weight decay or MAP Bayesian estimation, it is prudent to keep the number of weights less than half the number of training cases.

Bear in mind that with two or more inputs, an MLP with one hidden layer containing only a few units can fit only a limited variety of target functions. Even simple, smooth surfaces such as a Gaussian bump in two dimensions may require 20 to 50 hidden units for a close approximation. Networks with a smaller number of hidden units often produce spurious ridges and valleys in the output surface (see Chester 1990 and "How do MLPs compare with RBFs?") Training a network with 20 hidden units will typically require anywhere from 150 to 2500 training cases if you do not use early stopping or regularization. Hence, if you have a smaller training set than that, it is usually advisable to use early stopping or regularization rather than to restrict the net to a small number of hidden units.

Ordinary RBF networks containing only a few hidden units also produce peculiar, bumpy output functions. Normalized RBF networks are better at approximating simple smooth surfaces with a small number of hidden units (see How do MLPs compare with RBFs?).

There are various theoretical results on how fast approximation error decreases as the number of hidden units increases, but the conclusions are quite sensitive to the assumptions regarding the function you are trying to approximate. See p. 178 in Ripley (1996) for a summary. According to a well-known result by Barron (1993), in a network with I inputs and H units in a single hidden layer, the root integrated squared error (RISE) will decrease at least as fast as H^(-1/2) under some quite complicated smoothness assumptions. Ripley cites another paper by DeVore et al. (1989) that says if the function has only R bounded derivatives, RISE may decrease as slowly as H^(-R/I). For some examples with from 1 to 4 hidden layers see How many hidden layers should I use?" and "How do MLPs compare with RBFs?"

For learning a finite training set exactly, bounds for the number of hidden units required are provided by Elisseeff and Paugam-Moisy (1997).

References:

Barron, A.R. (1993), "Universal approximation bounds for superpositions of a sigmoid function," IEEE Transactions on Information Theory, 39, 930-945.

Berry, M.J.A., and Linoff, G. (1997), Data Mining Techniques, NY: John Wiley & Sons.

Blum, A. (1992), Neural Networks in C++, NY: Wiley.

Boger, Z., and Guterman, H. (1997), "Knowledge extraction from artificial neural network models," IEEE Systems, Man, and Cybernetics Conference, Orlando, FL.

Chester, D.L. (1990), "Why Two Hidden Layers are Better than One," IJCNN-90-WASH-DC, Lawrence Erlbaum, 1990, volume 1, 265-268.

DeVore, R.A., Howard, R., and Micchelli, C.A. (1989), "Optimal nonlinear approximation," Manuscripta Mathematica, 63, 469-478.

Elisseeff, A., and Paugam-Moisy, H. (1997), "Size of multilayer networks for exact learning: analytic approach," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) Advances in Neural Information Processing Systems 9, Cambridge, MA: The MIT Press, pp.162-168.

Geman, S., Bienenstock, E. and Doursat, R. (1992), "Neural Networks and the Bias/Variance Dilemma", Neural Computation, 4, 1-58.

Lawrence, S., Giles, C.L., and Tsoi, A.C. (1996), "What size neural network gives optimal generalization? Convergence properties of backpropagation," Technical Report UMIACS-TR-96-22 and CS-TR-3617, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742,
http://www.neci.nj.nec.com/homepages/lawrence/papers/minima-tr96/minima-tr96.html

Lawrence, S., Giles, C.L., and Tsoi, A.C. (1997), "Lessons in Neural Network Training: Overfitting May be Harder than Expected," Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97, AAAI Press, Menlo Park, California, pp. 540-545,
http://www.neci.nj.nec.com/homepages/lawrence/papers/overfitting-aaai97/overfitting-aaai97-bib.html

Neal, R. M. (1996) Bayesian Learning for Neural Networks, New York: Springer-Verlag, ISBN 0-387-94724-8.

Ripley, B.D. (1996) Pattern Recognition and Neural Networks, Cambridge: Cambridge University Press,

Sarle, W.S. (1995), "Stopped Training and Other Remedies for Overfitting," Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics, 352-360,
ftp://ftp.sas.com/pub/neural/inter95.ps.Z (this is a very large compressed postscript file, 747K, 10 pages)

Swingler, K. (1996), Applying Neural Networks: A Practical Guide, London: Academic Press,

Tetko, I.V., Livingstone, D.J., and Luik, A.I. (1995), "Neural Network Studies. 1. Comparison of Overfitting and Overtraining," J. Chem. Info. Comp. Sci., 35, 826-833.

Weigend, A. (1994), "On overfitting and the effective number of hidden units," Proceedings of the 1993 Connectionist Models Summer School, 335-342.

Send corrections/additions to the FAQ Maintainer:
saswss@unx.sas.com (Warren Sarle)