Why Not Pi?
A Primer on Neural Networks for Forecasting

by
J. Stuart McMenamin
Regional Economic Research, Inc.
April, 1997

The Question

At a recent conference, a group of electric utility forecasters[1] was discussing the relative merits of econometric models and artificial neural network models for forecasting problems. In the course of this discussion, the topic of functional form surfaced repeatedly. At the heart of most neural network specifications are S-shaped curves using the binary logistic function, which is based on e (2.71828…). This brought up a lively debate. Why not 3? In fact, why not π (3.14159...). After all, it was argued, the 18th century mathematician Euler is credited with assigning both letters (e and π) to their respective tasks. And π, sometimes referred to as Archimedes' number, has been around a lot longer. So, why not π?

In the end, we will put this question to the test based on forecast performance. In the interim, however, we use the opportunity provided by this question to lay out the issues associated with neural networks from a forecaster's perspective. In fact, in large part the question (Why not π?) is just an excuse to think about exactly what a neural network is and how it can be used by forecasters.

Background

A large amount of confusion seems to surround the topic of neural networks. In part, this reflects the fact that a different language is used for neural networks than is used in the more familiar (to forecasters) area of econometrics. Having struggled with this linguistic disconnect for about two years, we can assure you that bridging the gap is difficult without a translation dictionary.

[1] The inspiration for this paper came from a discussion between the author, Frank Monforte, Eric Fox, Joel Gaughan, Frank Ferris, and Brian Harkleroad. The event was the Electric Power Research Conference on New Directions in Forecasting held in Dallas in November 1996, at which much was said about the use of neural networks in forecasting.

To focus on specific elements of the language, the discussion is put in Q & A format. The questions involve issues that we have had to deal with over the last few years in the process of specifying and estimating neural network models, econometric models, and combined models in the context of monthly sales forecasting and hourly load forecasting problems for electric utilities. The answers come from the perspective of forecasters, and our goal is to make the relationship between concepts in neural network modeling and traditional econometric concepts as clear as possible.

So let's start with some basic questions.

Q. What exactly is a neural network?

A. Artificial neural networks, as they are used in forecasting, are flexible nonlinear models that can approximate a wide range of data generating process. In general form, for a single-variable forecasting problem, an artificial neural network looks like this:

$$Y = F(X, B) + u$$

Of course, the X's might be lagged Y's. And there could be several X's. And there could be lots of B's. In this general form, this is nothing new. However, most functions of this general form, including all functions that we normally use in forecasting, do not qualify as neural networks, except as degenerative cases.

Although neural networks can take many forms, the most frequently used form is very specific and can be written as follows:

$$Y^t = B_0 + \sum_{n=1}^{N} \left(B_n \times \frac{1}{1 + e^{-\left(a_{n,0} + \sum_{k=1}^{K} a_{n,k} X_k^t \right)}} \right) + u^t \tag{1}$$

The thing that makes this different is the repetitive nature of the specification. That is, within the summation, the function in parentheses is repeated N times with exactly the same algebraic form.

In network jargon, equation (1) is called a *single-output feedforward artificial neural network, with a single hidden layer, with N nodes in the hidden layer, with logistic activation functions in the hidden layer, and with a linear activation function at the output layer.*

To the forecaster, however, it is best to think about this specification as a flexible form that is nonlinear in the variables and the parameters. The form is flexible in that it allows for a wide variety of nonlinearities and interactions among the explanatory variables, and the repetitive specification is the cornerstone of this claim to flexibility.

Q. How can I get a better understanding of how equation (1) works?

A. It is easiest to understand this nonlinear function by looking at a simple example. If the number of explanatory variables (K) is 3, and the number of nodes (N) is 2, then the network function takes the following form:

$$\begin{aligned} Y^t = B_0 + B_1 \times \frac{1}{1+e^{-\left(a_0+a_1X_1^t+a_2X_2^t+a_3X_3^t\right)}} \\ + B_2 \times \frac{1}{1+e^{-\left(b_0+b_1X_1^t+b_2X_2^t+b_3X_3^t\right)}} + u^t \end{aligned} \tag{2}$$

In this expression, e is the transcendental number, 2.71828… Given values for the explanatory variables and a set of parameter values (the Bs', a's, and b's) we can easily compute the predicted values and residuals for each observation. The estimation problem, then, is to find parameters that make the residuals as small as possible. But more on that later.

Q. The example in equation (2) seems to have some linear parts and some nonlinear parts. What are the names for these parts?

A. Although this function is clearly nonlinear in the X variables and most of the parameters, it has linear components. Let's break this down a bit by rewriting the network function as follows:

$$Y^t = B_0 + B_1 \times H_1^t + B_2 \times H_2^t + u^t \tag{3}$$

In network terms, each H represents a node in the hidden layer. And, for the particular specification we are using here, the output function, which is shown in (3), is linear in these values. (The output function can be specified to be nonlinear in the H's, but for most forecasting problems with continuous dependent variables, there will be no advantage to this additional nonlinearity.)

Q. How does each node work?

A. If we look more closely at the first node (H_1), we see a second linear component in the denominator. Specifically, the exponent is a linear weighted sum of the input variables, and we will rewrite this linear weighted sum as follows:

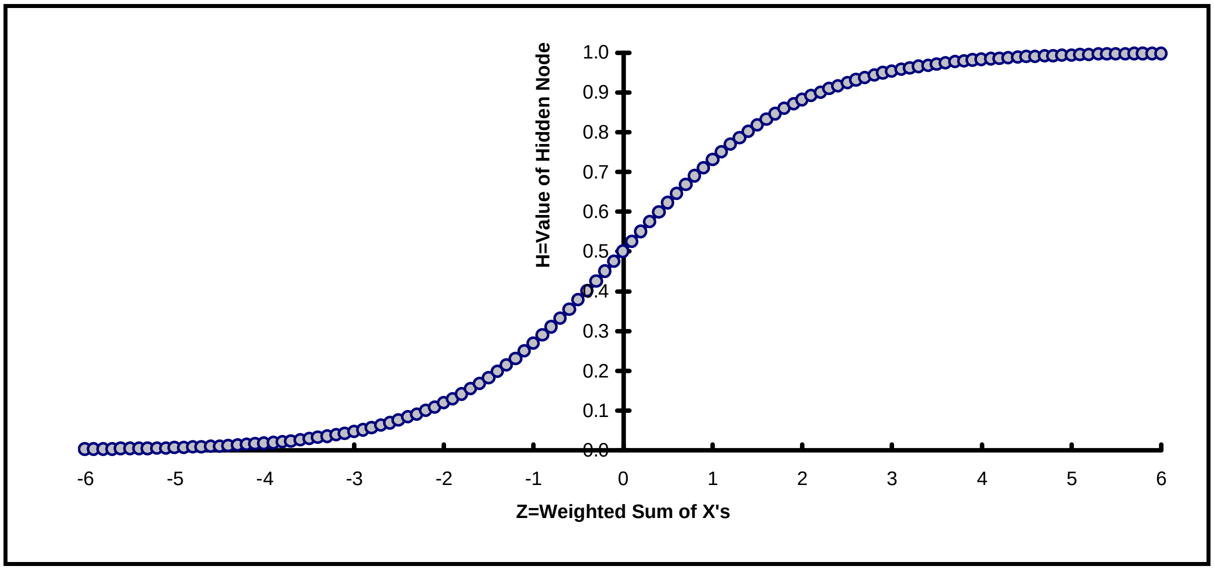
$$Z_1^t = a_0 + a_1X_1^t + a_2X_2^t + a_3X_3^t \tag{4}$$

Then, with a bit of rearrangement, we have the following:

$$H_1^t = \frac{1}{1+e^{-Z_1^t}} = \frac{e^{Z_1^t}}{1+e^{Z_1^t}} \qquad (5)$$

If you are familiar with discrete choice models, you will recognize this as a binary logit, the workhorse of market-share modeling. The linear weighted sum (Z) is the power on e. If Z is a large negative number, H is close to zero. If Z is 0, H is .5. And if Z is a large positive number, H is close to 1.0. In between, it traces out an S-shaped function. If you put this on a spreadsheet and plot H as a function of Z, the result is as depicted in Figure 1.

Figure 1: Binary Logistic Function



This also means that there is an S-shaped relationship between each node value (H) and each explanatory variable (X) in that node. This S curve may be positively or negatively sloped, depending on the sign of the slope coefficient on the X variable ($a_{n,k}$ in equation (1)).

Q. I see this is nonlinear in the X's. What about variable interactions?

A. The specification is automatically interactive, since we can rewrite the exponential as follows:

$$e^{a_0+a_1X_1^t+a_2X_2^t+a_3X_3^t} = e^{a_0}e^{a_1X_1^t}e^{a_2X_2^t}e^{a_3X_3^t}$$

As a result, each X variable interacts with all other X's that do not have zero slopes in the node. This is a strength of the specification if the underlying process has multiplicative interactions.

Q. This is a little strange. Each node has the same functional form. How does this work?

A. It is true that each X variable appears several times and, in the case of equation (1), in exactly the same algebraic form. The econometrician in us is not comfortable with this idea. It looks like an extreme form of multicollinearity. In network language, the repetitive specification is called parallelism or massive parallelism, and it is one of the strengths of the approach. As you might expect, it raises some serious issues for parameter estimation.

But, if you think about it, we have all put X and X squared on the right-hand side of an equation. So suppose that in the first node, variations in X cause movement in the linear part of the logistic equation (Z between -1 and +1) and in the second node, variations in X are operating in the bottom part of the S shape (Z between -4 and -1). This could happen, depending on the values of the other variables and the parameters involved.

With several nodes in the hidden layer, the specification allows for a variety of nonlinearities and for a range of variable interactions. For example, two logistic curves, one positively sloped and one negatively sloped, can be combined to give a U-shaped response over the relevant range of an X variable. Given this flexibility, the estimation problem is to find a set of specific nonlinearities and specific interactions that are useful for explaining history and for forecasting. We will discuss the estimation problem below.

Neural Network Terminology

Well, if you have made it this far we need to get a bit deeper into the linguistics of artificial neural networks before we go back to the original question.

Q. How would the specification above be described in neural network terms.

A. In neural network terms, equation (1) has the following properties.

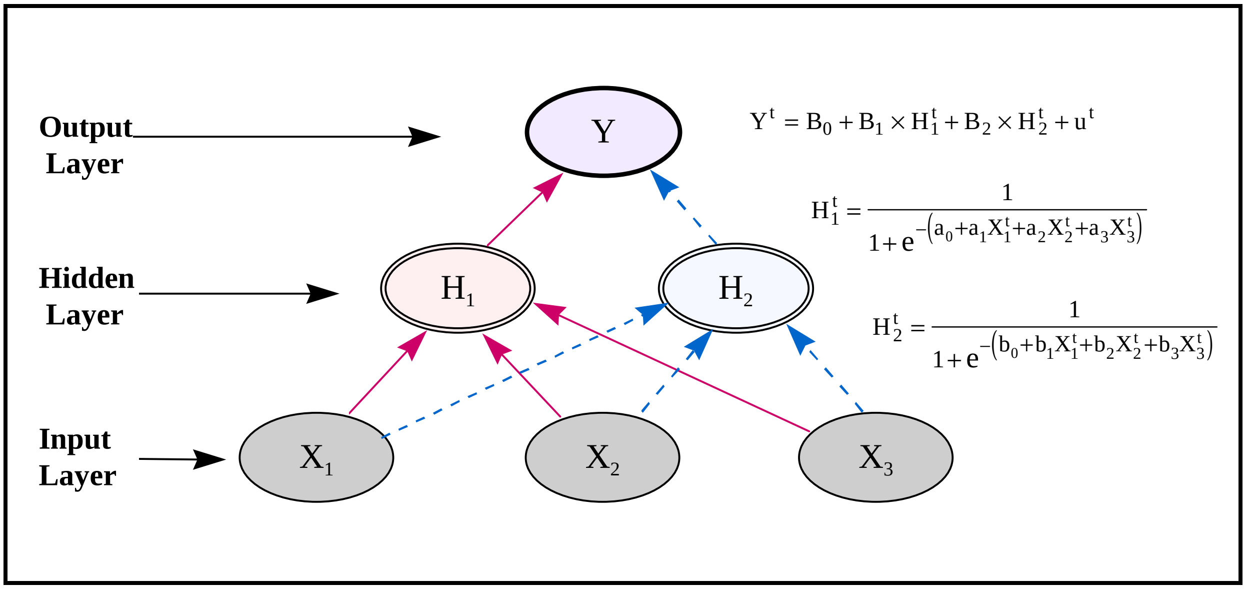
- It is a *feedforward neural net with a single output*
- It has *one hidden layer with one or more nodes*
- It uses *logistic (e-based) activation functions in each hidden layer node*
- It uses a *linear activation function at the output layer*.

Q. Why is it called feedforward neural net?

A. The best way to answer this question is to draw the classic neural net diagram. As shown in Figure 2, the explanatory variables (X) enter at the bottom in the input layer. The logit tranforms appear in the hidden layer. And the result (Y) appears in the output layer.

The idea is that the inputs feed into the nodes in the hidden layer, and there is no feedback. Further, the nodes do not feed sideways into each other. Instead, they feed onward to the output layer. And there is no feedback, delayed or otherwise, from the output layer to the hidden nodes. The absence of feedbacks or node-level interactions makes it a feedforward system. Really, the way it is drawn, it is a feed upward system.

Figure 2: Network Diagram with 3 X's, 2 Nodes


Output Layer
Hidden Layer
Input Layer
Y
H_1
H_2
X_1
X_2
X_3
$Y^t = B_0 + B_1 \times H_1^t + B_2 \times H_2^t + u^t$
$H_1^t = \frac{1}{1+e^{-(a_0+a_1X_1^t+a_2X_2^t+a_3X_3^t)}}$
$H_2^t = \frac{1}{1+e^{-(b_0+b_1X_1^t+b_2X_2^t+b_3X_3^t)}}$

Q. Why are the terms in the middle called the hidden layer?

A. There is an answer to this in the neural sciences (see Rummelhart, Hinton, and Williams, 1986). But the answer is of no interest to us as forecasters. If you look at equation (1), it is clear that nothing is hidden at all. Represented in diagram form, as in Figure 2, there are specific algebraic transformations in the middle layer, but they are hidden only if we decide to hide them somehow. The important and powerful part of the specification is that it is a flexible form capable of approximating a wide range of functions. Analogies to the learning process of the brain do not increase or decrease the power of this approximation. Neither do these analogies help us to understand how these equations work in a forecasting context.

Q. Why is the binary logit in the hidden layer called an activation function?

A. It refers to the S-shaped nature of the function in each node and the fact that the boundary values (0 and 1) can be related to on and off. Again, the history of the term is rooted in neural sciences, and it has to do with the requirement that a signal must reach a certain level before a neuron fires to the next level. This is not a bad description in the forecasting application. For most forecasting problems, if you allow flexibility (more than two nodes), some of the nodes will end up specializing and will activate (take on a value close to 1.0) under specific conditions and take on a value of close to 0 otherwise. To see this for a specific problem, all you need to do is plot out the contribution of each node to the total predicted value, and this pattern will become clear.

For our purposes, the hidden layer functions do not need to be logistic functions. Any other S-shaped function could be used, such as an arc tangent or a cumulative normal, and the model behavior would not be changed significantly. The important thing is to use a smooth differentiable function that will be easy to work with for estimation purposes.

Taking this further, it is not necessary to use S-shaped curves at all. For example, we could use bell shaped functions (like the derivative of a logistic curve) in some nodes rather than S-shaped functions, and this would be quite useful in many forecasting applications. However, as we move away from S-shaped curves, the term “activation” becomes less descriptive of functional performance, since the alternative functions would no longer range between 0 at one extreme and 1 at the other. As a result, the hidden node activation functions are sometimes called neuron transfer functions (see, for example, Azoff, 1994, pp 51-55).

Q. Why use a linear activation function at the output layer?

A. We could include some additional nonlinearity at this level. In fact, we could include about any nonlinearity that can be written down. For Y variables that have a discrete outcome, such as a binary (0/1) variable, a logistic activation function at the output layer would probably be desirable. But for most problems with continuous outcomes, there is no real gain from a further nonlinearity at this level.

Estimation Approaches

You are probably still confused about estimation. How do we estimate the parameters of the network equation? In network terminology, these are called the *connection strengths*. The constant terms are called *bias parameters* and the slope terms are called *tilt parameters*. No matter what we call them, however, they are still parameters. And the goal of estimation is to find a set of parameter values that make the errors small.

Q. Given the repetitive specification, how does estimation work?

A. There is no single answer to this question. There appear to be as many specific estimation algorithms as there are people who have worked on these algorithms. However, many neural network programs use some variant of a method called the *generalized delta rule* or the method of *backpropogation*. Here is how it works in its simplest form.

First, you select a set of random starting values for the coefficients within a range that makes sense for your data. These values probably won't work very well, but they will allow you to compute a residual for each observation and the derivatives of each residual with respect to each of the parameters. Then you can take the following steps:

1. Apply the current parameters to the first observation and compute a residual
2. Compute the derivatives (gradient) of the residual with respect to each parameter
3. Take a small fraction of the computed residual
4. Adjust parameters by the product of this fraction and the negative gradient
5. Take the adjusted parameters to the next observation and repeat steps 1 to 4
6. When you finish working through the data set
   - Check for parameter convergence or other stopping criteria
   - If not done, go back to the beginning of the data and repeat steps 1 to 5.

So the idea is that you use the parameters to compute a residual, and then you use the residual to adjust the parameters (which is the "back" part of backpropogation). For example, if the residual is positive, implying that the actual is above the predicted value, you would adjust the intercept and any slopes that are positively related to the predicted value upward a bit. And you would adjust any slopes that are negatively related to the predicted value downward a bit.

Of course, there is more to it than this, and there are many variations on the backpropogation theme. For example, as you move through the data set, you must reduce the adjustment fraction (called the *learning rate*) downward. Otherwise, the order of the data would strongly impact the parameter values at the end of each iteration, and the algorithm would not converge to a stable parameter point.

Q. Is there any relationship between this type of algorithm and standard nonlinear estimation algorithms?

A. If you are looking for parallels, it turns out that backpropogation is an application of a statistical method called the " method of stochastic approximation" (Robbins and Monro, 1951). In an econometric context, this method has also been called recursive least squares. It has been shown that simple backpropogation with a learning rate that is inversely proportional to the observation number is equivalent to the recursive least squares approach (see White, 1989 for a discussion). Recursive least squares is not widely used in econometrics today due to the widespread availability of fast matrix inversion algorithms and nonlinear optimization algorithms.

Q. Are there alternatives to this type of estimation approach?

A. Yes. The goal of estimation is both familiar and straightforward. The goal is to find a set of parameters that make the in-sample residuals small. If you look around a bit, you will find that there are several ways to do this. Some examples are:

- Recursive algorithms, such as backpropogation.
- Optimization algorithms, such as conjugate gradient, Newton methods, and steepest descent.
- Least squares algorithms, such as Levenberg Marquat, which blends Newton and steepest descent.
- Evolutionary or genetic algorithms.

These algorithms all have their place. But in forecasting problems, the goal is the same: to find parameters that make the errors small. Consider the alternative (algorithms that make the errors large), and the appeal of this goal is pretty obvious.

More specifically, from a forecaster’s perspective, the real goal is to find parameters that generalize to make the out-of-sample forecast errors small. And if flexible nonlinear systems, such as the neural net, allow us to reach this goal, then we should embrace them.

Q. Do backpropagation algorithms minimize the sum of squared errors?

Not necessarily. For a given random starting point, these algorithms are intended to gravitate toward a parameter solution that makes the residuals small. But they do not necessarily find a local minimum to the sum of squared errors, or the average absolute deviation, or any other specific objective function. However, with appropriate treatment of the learning rate, it can be shown that the backpropogation solution will converge to a local solution to the least squares problem (see Kuan and White, 1992 for a discussion).

Estimation Complexities

Q. It still seems that the estimation problem isn’t very well defined. Is it possible that the estimation algorithms will converge to a local solution that doesn’t really minimize the sum of squared errors?

A. This turns out to be a really good question and one that causes great concern to neural network practitioners. After significant analysis of this issue, we have concluded the following. If there is one node in the hidden layer, the objective function is well behaved, much like a linear least-squares problem. Regardless of where you start the initial parameter guesses, you will converge to a comparable parameter solution.

However, as nodes are added, the objective function quickly becomes very complex with an explosive number of solutions that are locally optimal. As a result, to find a good solution it is necessary to look around in parameter space. Merely going downhill from a single random starting point to the nearest local optimum simply will not do the job. And this is true whether you use a backpropogation algorithm or a mathematical optimization algorithm.

The magnitude of this problem was quantified in a recent paper (Goffe, Ferrier, and Rogers, 1994). In this paper, the authors examined several estimation problems, including one that involved a neural network like the one specified above. In addressing this problem, the authors became interested in the shape and properties of the objective function. The problem was a modest one, involving a 5-node network with a total of 35 parameters. As alternative solutions were examined, it became apparent that the objective function had a large number of local optima. By pushing out from each optimum point and seeing if estimation returned back to this point, they were able to quantify the size of the region dominated by each optimum point. Taking this region as a fraction of the total parameter space for their problem, the determined that there were about 10^{19} such points for their problem.

This is a lot of points. As an analogy, consider a two parameter problem (so we can visualize the surface of the SSE function). Think of the objective function as looking like the bottom half of an egg carton, with 1 dozen local minimum values of varying depths in each carton. Now think of a football field covered with such cartons. Now think of the state of Texas covered with such cartons. Now think of the surface of the earth covered with such cartons. Now think of 64 times the surface of the earth covered with such cartons. You now have a big enough surface to contain 10^{19} local optimum points.

So suppose that you pick a random starting point and it turns out to be on the 50 yard line of a high school football field in the middle of Texas. Look around at the footbal field, out toward the state boundaries, and imagine the 64-earth surface from this point and you start to understand the magnitude of the estimation problem. Somewhere on that surface are several mathematically equivalent global optimum points.

Q. Is it possible to find the global optimum?

For their problem, Goffe, Ferrier, and Rogers concluded that “the nature of the (neural network) function makes it virtually impossible for any method to find the global optimum.” They also concluded that “given the relative values of the optima, it would appear that each local optima would need to be examined to find the global optima. Given the number of local optima, this hardly appears likely given current or foreseen computers." And these guys aren't talking about PCs; they were working on a Cray computer at the University of Texas.

Q. This sounds pretty hopeless. Is there any way to guarantee finding a reasonably good solution?

A. Actually, it is not as bad as it sounds. As it turns out, there are a large number of local optima that work pretty well for typical forecasting problems. But we still need to look around a bit to make sure that we settle into one of the better solutions. Several approaches have been examined, including:

- Simulated annealing
- Genetic algorithms
- Multiple seed optimization.

We won't go into each of these here, but the advantage of these approaches is that they allow the parameter terrain to be searched to some degree, reducing the probability of settling into a substandard local optimum. The joint process of searching and optimizing is called training, and once the model is trained, it can be used to forecast.

Q. With econometric models, we often examine different variables, different functional forms, and so on. How does this relate to the training process?

A. There are strong parallels here. Recall that the neural network is a flexible nonlinear model. Each local minimum represents a specific set of useful nonlinearities and variable interactions that combine to make the residuals small for the sample data. So, selecting a specific solution is very similar to selecting a functional form. There is a difference, however. In selecting a final model in econometrics, there are often qualitative criteria involved, such as the signs of parameters and the absolute magnitudes of model elasticities. In training a neural net, the final parameter solution is usually selected on the basis of fit alone.

Q. In the training process, what types of criteria are used to select among competing parameter solutions?

A. Many algorithms estimate parameters on a subset of the data and use the withheld data to test the power of the estimated parameters out of sample. This is like using an in-sample forecast test to choose among competing econometric specifications. So, when comparing solutions, you have a choice between in-sample statistics and out-of-sample forecasting statistics.

Usually, solutions that perform well in sample also perform well out of sample. But this is not always the case. Especially with a large number of nodes, some of the solutions will be more specialized to the specific cases in the sample, and some will be more stable and more useful out of sample. So, when forecasting is the goal, it is probably better to choose a specification that performs well out of sample, based on statistics such as the MAD and MAPE.

Q. What do you do as new data become available?

A. Let's think about this from the perspective of econometric or time-series models first. With these models, you have two options. You can continue to use the parameters that came out of the initial estimation or you can extend the sample period to incorporate the new data and reestimate. Normally, as time moves forward and new data are generated, you just update the estimates using the extended sample period. In absence of weighting, the new data get the same weight in this update process as the earlier data.

It is not that different with neural networks, especially if an optimization algorithm is used. In this case, you start with the parameters from the training (estimation) process and reoptimize with the new data included. Usually, this will result in small and gradual changes in the parameter estimates.

Similarly, using backpropogation algorithms, the learning process is just an extension of the training (estimation) process. That is,

- Start with the parameters from model training
- Compute a residual for the new observation
- Compute the derivatives of the residual with respect to the parameters
- Adjust parameter up if this reduces the residual
- Adjust parameters down if this reduces the residual.

The direction that parameters adjust is determined by the derivatives of the residual with respect to each parameter, and the actual adjustment is determined by these derivatives and the learning rate. For stable problems with large data sets, the learning rate will usually be small, resulting in small and gradual changes in the parameters as additional data are included.

Q. Is there ever a need to retrain the model?

A. Yes. With an econometric model, you can update the parameters on a monthly basis with a minimum of effort. But at some point, you may want to reconsider the specification itself. This is equally true with neural networks. If you think about training as automated selection of a specification, the parallel is clear.

The point is that after significant time has passed, the additional data might be better fit by an alternative set of nonlinearities and interactions. But it is very unlikely that reestimation or continued learning will get to one of these better points starting from the estimated parameters that are in place (think back to the 64-world surface). So, on occasion, it is a good idea to return to the training exercise to insure that you are still working with a relatively good solution.

Summary Statistics for Neural Networks

Thinking of neural networks as flexible nonlinear models, there are a wide variety of standard statistics that can be used to evaluate the performance of the estimation result. Once parameters are estimated, a full set of residuals can be computed for in-sample and out-of-sample observations. In some cases these statistics are not presented by neural network software. Thus, the following questions.

Q. Can you compute the regular test statistics, such as standard errors and t statistics for the estimated coefficients of a neural network?

A. In part, this depends on the method that is used. With backpropogation approaches, there is no natural extension of the calculations that gives such statistics. However, if you view this as a least squares problem and use standard estimation algorithms, you can derive an estimated parameter covariance matrix and standard errors for each estimated coefficient. Given the nature of the objective function, it is not clear exactly how to interpret these statistics. Furthermore, given the strategic redundency in the specification, it is not clear what it means to apply the standard tests to an individual parameter in one of the nodes. Still, these statistics have some local properties and can be used to identify which variables play a statistically important role in each of the nodes.

Q. What are good statistics to evaluate the model fit?

A. Since there are a full set of residuals for the model, all the standard statistics can be computed. These include:

- Sum of squared errors
- R square
- Adjusted R square
- Standard Error
- Mean absolute deviation (MAD)
- Mean absolute percentage error (MAPE).

These statistics are computed from the residuals exactly as they would be for an econometric specification. If there are out-of-sample data points, then forecast MAD and MAPE statistics can also be computed. When comparing alternative parameter solutions in the training process, all of these statistics are relevant. For example, you might decide to select a solution based on its out-of-sample MAPE rather than the in-sample sum of squared errors.

Q. What about properties of the error term?

A. Properties of the error term have not received much attention in the context of neural networks. In part, this reflects the type of problems to which they have most often been applied, many of which fall into the area of pattern recognition.

In a time-series context, however, the error term for a neural network model faces the same potential set of issues as in a linear model. There may be excluded variables, errors in variables, or an unspecified residual process implying that there is still information remaining in the estimated residuals. From a forecasting perspective, this may be important information. For example, in a "next-day" forecasting context, if the residuals reveal a significant pattern of autocorrelation, then yesterday's residual has important information for today's forecast, and a better forecast can be made by accounting for this information.

The standard statistics can be used to evaluate residual behavior, including the following:

- Durbin Watson statistic
- Ljung-Box statistic
- Residual autocorrelations
- Residual partial autocorrelations.

If these statistics indicate that the residuals follow a time-series process, such as an AR process or a short MA process, then the nonlinear estimation algorithm can be extended to include estimation of the parameters of the ARMA process.

Q. How can you evaluate the signs of model sensitivities and elasticities?

A. In a linear model, this is pretty straightforward. Each explanatory variable has a slope parameter, and this slope equals the derivative of the predicted value with respect to that variable. The slopes can be converted to estimated elasticities by multiplying by the ratio of the dependent variable to the explanatory variable, using mean values or end-of-period values, for example.

For neural networks, the result is similar, but significantly more complicated. Because of the nonlinearities and interactions inherent in the specification, the derivative of the predicted value with respect to an explanatory variable can take on a different value for each observation. Fortunately, this slope is easy to compute. Similarly, since the slope is a variable, the elasticity will take on a different value for each period. The best thing to do is to compute these slopes and elasticities and view them as numbers and as graphs. These are the model sensitivities inherent in the estimated network coefficients, and they will need to be looked at and evaluated.

In fact, one of the potential uses of neural networks is to do exploratory analysis. For example, you can place your data in a network with a small number of nodes, do some training, and look at the pattern of the slopes and elasticities over time. Analysis of these results can lead to a better understanding of the factors that determine how model sensitivities vary over time and which variable interactions are important.

Q. Can statistics help you determine the number of nodes?

A. Probably. Although there is no hard and fast rule, for most time-series problems that we have looked at the optimal number of nodes appears to be between two and five. Of course, as you add nodes, the in-sample fit always improves. That is, the sum of squared errors will always decline if you add more parameters. However, beyond a point the coefficients have the freedom to specialize in order to explain specific events in the sample period, and these specialized results do not necessarily generalize to out-of-sample conditions. Flexibility turns out to be a bad thing when taken to excess. And by a bad thing, we mean that it will lead to worse forecasts.

Statistics that are relevant to this issue are:

- Adjusted R Square
- Akaike Information Criterion (AIC)
- Bayesian Information Criterion (BIC).

All of these statistics go down when the sum of squared errors is reduced, but impose a penalty for the increased number of coefficients. In a time-series context, the AIC and BIC are often used as indicators of model power. Coefficients are added as long as these statistics decline. When they start to increase, this is a sign that it is time to stop. These statistics may also be a good guide for neural network modeling.

Back to the Original Question

This brings us back to the original question. As discussed above, the most frequently used neural network specification uses an e-based (binary logit) activation function in the hidden layer. Why not a π-based version? We could then call it a π-gistic activation function or simply a πgit.

Q. So, why not p?

A. As far as we can tell, there are two reasons to use e, both of which are pretty good ones and neither of which involve the theoretical high ground. First, it seems to work pretty well for a broad class of forecasting problems. Second, it is easy. It is easy, because it is really simple to take derivatives when e is involved. Specifically:

$$\frac{\partial e^{x}}{\partial x} = e^{x} \tag{6}$$

That is about as simple as it gets. And it makes computation of the derivatives of (1) with respect to the parameters really easy. But, if we used π, it would not be much harder since:

$$\frac{\partial \pi^{x}}{\partial x} = \ln(\pi) \times \pi^{x} = 1.145 \times \pi^{x} \tag{7}$$

So, given that using π is also easy, it boils down to a fundamental question: Which does the best job of forecasting? We don't propose to answer that question definitively here. In fact, intuition suggests that there is no definitive answer. But we will test the relative merits of e-based and π-based activation functions in terms of forecasting power for a specific problem.

Data

The data used to test the specifications are for monthly residential electricity consumption in the state of California. The data extend from 1977 through 1995, giving a total of 228 observations. Explanatory variables include the following:

- Number of residential electricity customers
- Monthly heating degree days (base 65)
- Monthly cooling degree days (base 65)
- Month (Jan=1, Feb=2, …, Dec=12)
- Winter season binary variable (=1 in Dec, Jan, Feb)
- Spring season variable (=1 in Mar and Apr, =.5 in May)
- Summer season variable (=.5 in Jun, =1.0 in Jul and Aug, =.5 in Sep)
- Average price of electricity.

The variable for heating degree days provides a measure of cold temperatures, and in each month it accumulates the differences between 65 degrees and daily average temperatures, when the daily average temperature is below the 65 degree threshold. Similarly, the variable for cooling degree days provides a measure of hot temperatures, and in each month it accumulates the difference between average temperatures and 65 degrees, when the daily average temperature is above the 65 degree threshold.

Estimation

With these eight variables, each node in the hidden layer adds 10 coefficients, including the constant and eight slopes in the weighted sum for the node and the overall slope coefficient for the node. Model parameters are estimated by picking a random starting point and minimizing the sum of squared errors using a Levenberg-Marquadt algorithm. Full model training involves estimation with a large number of random seeds, and selection of the starting point that fits well in both the estimation period and the test period.

First, the standard neural network equation (1) is estimated with the number of nodes ranging from two to eight. For each specification, 400 alternative sets of random numbers are used. Each estimation includes monthly data from 1997 through 1993, for a total of 204 sample data points. The last 24 observations (for 1994 and 1995) are reserved as a test period for evaluation of forecasting power. In the test period, actual values of the explanatory variables are used as inputs, and residuals are computed as actual consumption less predicted consumption in each month.

Table 1 presents the average results from these training runs in terms of in-sample error statistics and out-of-sample statistics.

Table 1: Training Statistics: Average of 400 Random Starting Points

# of Nodes	# of Coefs	Sample SSE	Sample MAD	Sample MAPE	Forecast MAD	Forecast MAPE	Sample BIC
1	11	6716	145.4	3.13%	147.2	2.70%	10.69
2	21	3689	105.0	2.25%	125.5	2.24%	10.35
3	31	2332	84.0	1.80%	129.7	2.27%	10.15
4	41	1726	72.5	1.55%	137.9	2.41%	10.11
5	51	1367	64.6	1.37%	137.1	2.39%	10.14
6	61	1137	58.6	1.24%	140.2	2.45%	10.22
7	71	973	53.6	1.13%	145.3	2.54%	10.32
8	81	824	49.0	1.03%	149.4	2.62%	10.42

As is clear from the table, the in-sample statistics improve steadily as the number of nodes is increased. However, the forecast statistics from the test period indicate that two or maybe three nodes are optimal from the perspective of this specific forecast period. Also, the Bayesian Information Criterion (BIC) suggests somewhere between three and five nodes.

In addition to the averages across random starting points, it is instructive to look at the distribution of solutions. For the 3-node system, Figure 3 shows the distribution of sample period MAPE values for each of 1,000 random starting points. From each starting point, the optimization algorithm finds a nearby parameter point that provides a local minimum to the sum of squared errors. As seen in Figure 3, the solutions have in-sample MAPE values that range between 1.65% and 2.25%.

Figure 4 shows the comparable distribution for MAPE values over the two-year test period. These values range from 1.5% to over 3.0%. The two spikes in these charts correspond to two types of parameter solutions that were reached from a wide range of starting points. Each of these "popular" solutions are reached from over 100 of the 1,000 starting points.

Figure 3: Sample Period MAPE Values – 3 Nodes, 1,000 Trials

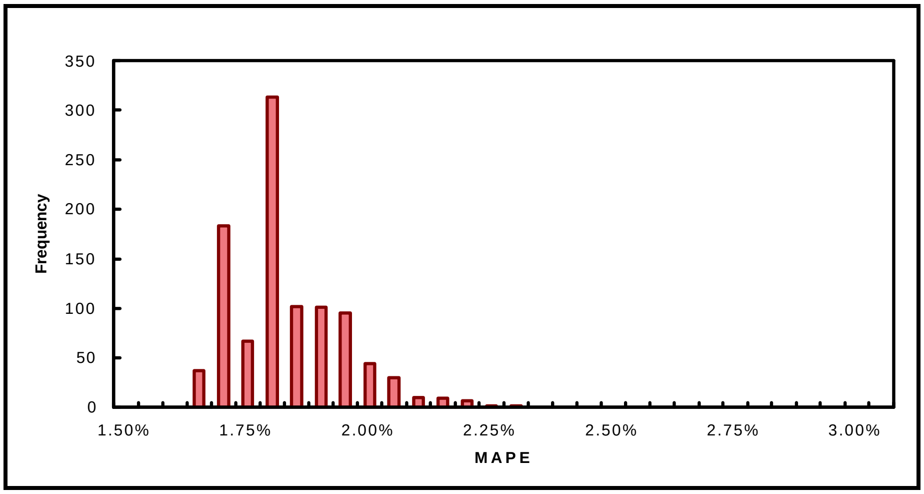


Figure 4: Forecast Period MAPE – 3 Nodes, 1,000 Trials

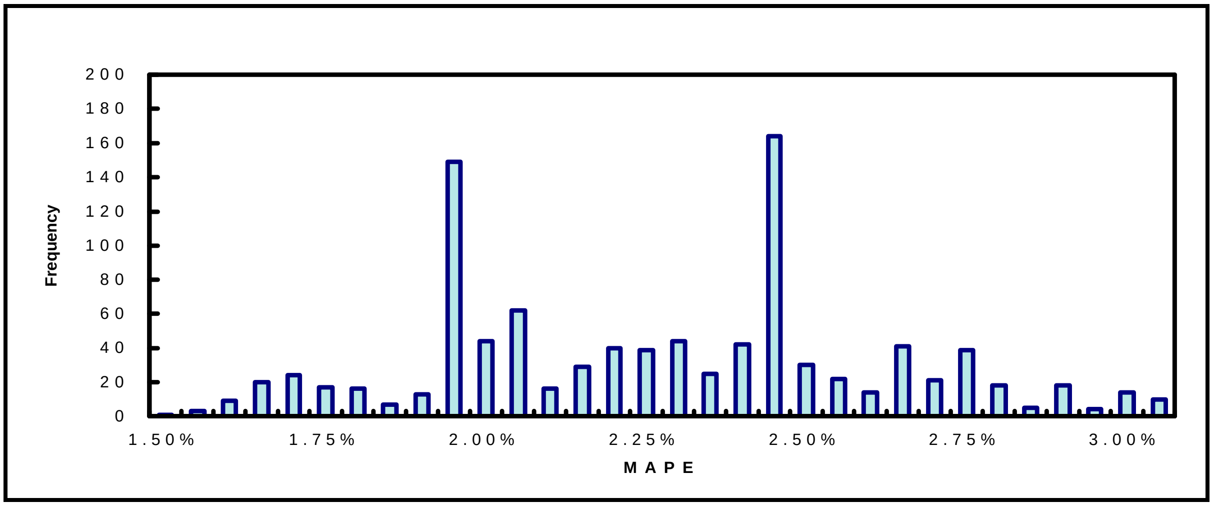


Figure 5 provides a scatter plot of the in-sample MAPE values and the forecast MAPE values. The 3-node models that fit history best appear toward the left of the chart. As is evident, these models do not necessarily provide the best forecasts, which appear toward the bottom part of the chart. For training purposes, it makes sense to select a solution that is in the lower-left-hand portion of the chart, indicating that it fits both history and test-period data well. This type of solution represents a set of nonlinearities and interactions that fit the data well and that also generalize well in terms of performance out of sample.

Figure 5: Sample Period MAPE – 3 Nodes, 1,000 Trials

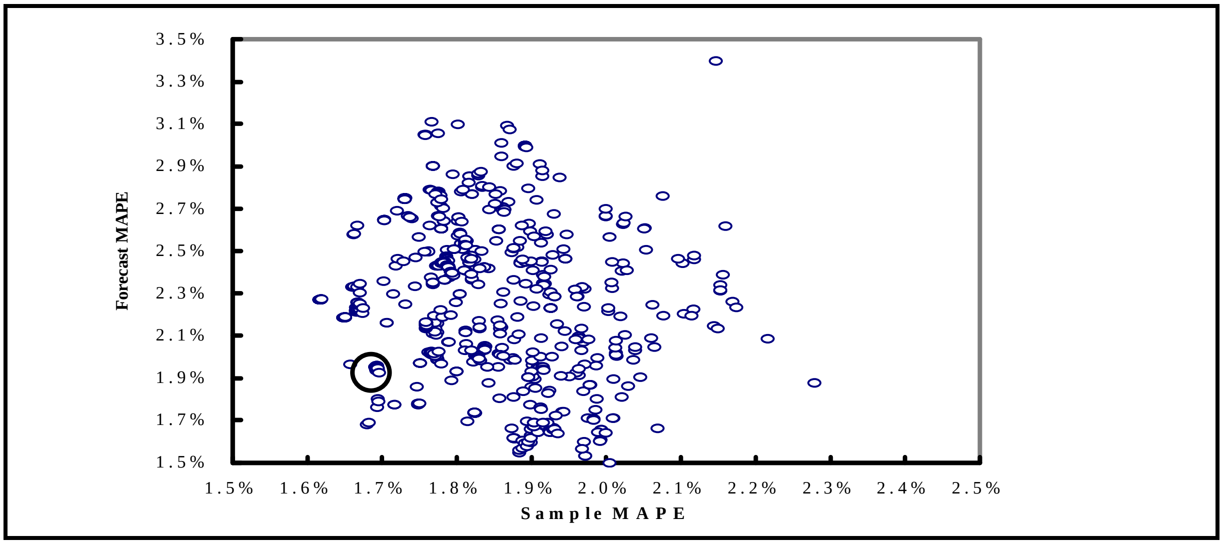
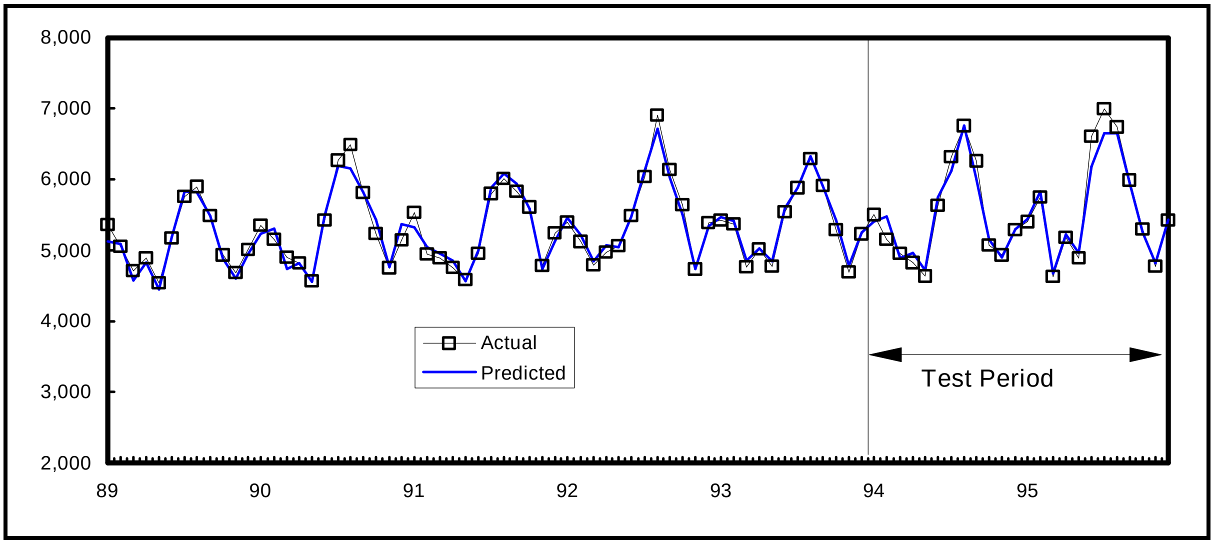


Figure 6 shows model performance over the last five years of the sample period and the two-year test period. The specific model that is depicted corresponds to one of the better fitting of the two "popular" solutions, and the MAPE point for these solutions is circled in Figure 5. With the exception of July and August of 1996, in the test period and August of 1990 in the sample period, the model fit is excellent, and provides better tracking and forecasts than a wide variety of regression models that were examined for these data.

Figure 6: Actual and Predicted Values – 3 Node Network



As a final element, the model elasticities for heating degree days and cooling degree days are shown in Figure 7 and Figure 8. As would be expected, the elasticities of electricity use with respect to cooling degree days are most evident in the summer months. In these months, the typical value is about .2, indicating that a 10% increase in cooling degree days will cause a 2% increase in monthly electricity use. In contrast, the elasticity for heating degree days switches sign. In winter months, cold weather increases heating loads. However, since most heating systems are not electric, the elasticity is modest, at about .10. In the summer months, the elasticity is negative, indicating that cool weather reduces cooling loads.

Figure 7: Elasticity of Consumption with respect to Cooling Degree Days

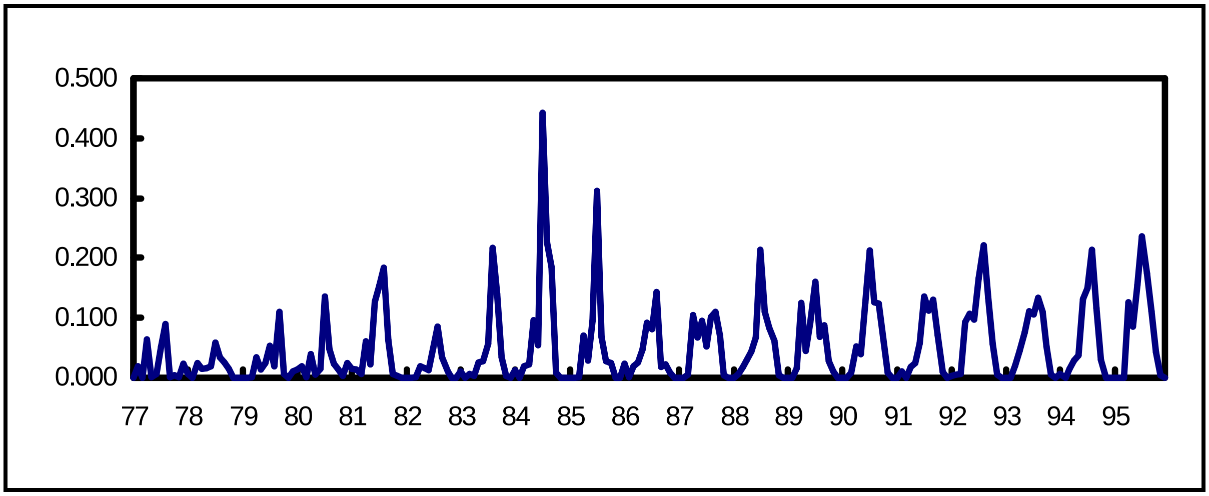
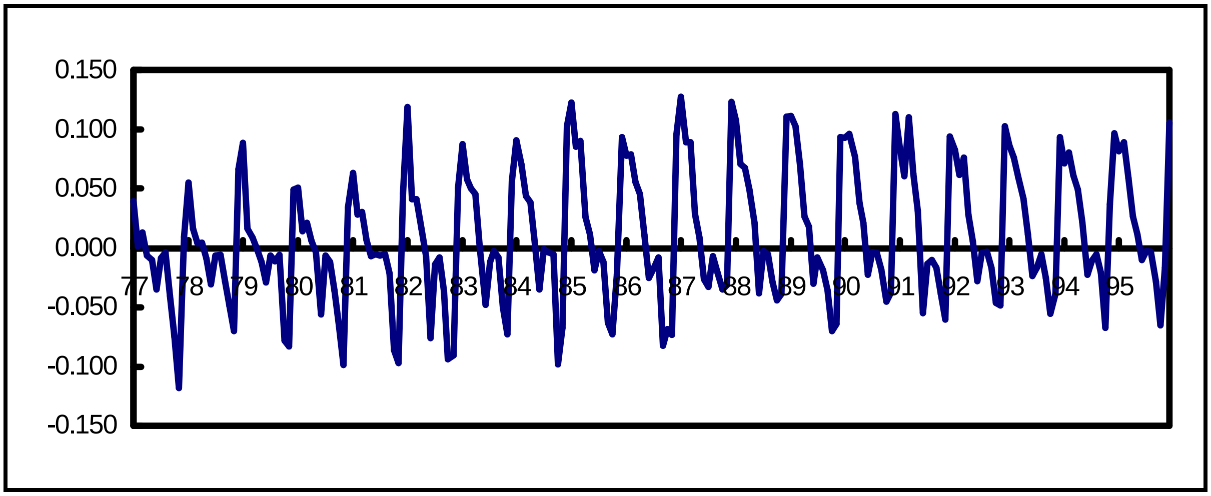


Figure 8: Elasticity of Consumption with respect to Heating Degree Days



e-based Versus p-based Models

Given this background about neural networks applied to these data, the final step is to see how models perform when π-based equations are used instead of e-based equations. Using S-shaped curves that are π-based, the neural network specification in (1) is as follows:

$$Y^t = B_0 + \sum_{n=1}^{N} \left(B_n \times \frac{1}{1+\pi^{-\left(a_{n,0}+\sum_{k=1}^{K} a_{n,k} X_k^t\right)}} \right) + u^t \qquad (8)$$

The relative performance of the two modeling approaches is summarized in Table 2. The model statistics presented in this table are the average sample and forecast statistics based on 1,000 alternative random starting points for each specification. As the statistics indicate, the performance of the two models is virtually identical, with a slight edge in favor of the e-based systems. So, we have an answer to the question, why not π? Using e-based curves is a little easier, and, for these data, at least, the e-based curves work a little bit better.

Table 2: Comparison of Model Statistics (e versus p)

Method	Average Sample MAPE	Average Forecast MAPE	Average Sample MAD	Average Forecast MAD
Logistic (e-based)	1.81	2.28	84.4	130.5
π-based	1.82	2.29	84.8	130.9

Conclusion

Artificial neural networks provide a flexible nonlinear framework with many similarities to structural econometric models. These models are well suited to the forecasting task for data like the monthly sales data analyzed here. However, there is a danger that the approach will be treated as a black-box that is difficult to understand. As we have shown here, these models have much in common with standard econometric approaches, and can be discussed in terms that are familiar to individuals who are comfortable with structural econometric forecasting.

However, there is new ground here as well. The key difference is in the flexibility of these models. Estimation of parameters is a bigger task because the models are nonlinear and because the training exercise amounts to a search for a useful functional form. For a given number of nodes, training involves the search for a set of nonlinearities and interactions that provide the best model fit to the historical data**.** The objective function for error minimization is exceedingly complex, because there are a large number of solutions that are locally optimal and that fit the data well. As a result, it is necessary to search parameter space to find a good solution, one that works well both in sample and in reserved test periods. These solutions can also be evaluated and compared based on model derivatives, elasticities, and a variety of standard model statistics.

As long as the number of nodes is kept to a reasonable level, the result is a forecasting model that is powerful, robust, and sensible. This method is useful for forecasting and also for exploratory analysis that can be used to examine issues related to functional form. The results can be used directly, and they can also be used to strengthen econometric models through the identification of important nonlinearities and interactions.

References

Azoff, E. M. *Neural Network Time Series Forecasting of Financial Markets*. John Wiley & Sons, 1994.

Goffe, William L, Gary D. Ferrier, and John Rogers, "Global Optimization of Statistical Functions with Simulated Annealing." *Journal of Econometrics* 60 (1994), 65-99, North Holland

Kuan, Chung-Ming and Halbert White, "Artificial Neural Networks, An Econometric Perspective." UCSD Discussion Paper, June 1992

Rummelhart, D.E., B.E. Hinton, and R.J. Williams. "Learning Internal Representations by Error Propagation." In D.E. Rumelhart and J.L. McClelland (eds.) *Parallel Distributed Processing: Explorations in the Microstrucutres of Cognition*. MIT Press, Cambridge, Mass, 1986.

Robins., and S. Monro. "A Stochastic Approximation Method." *Annals of Mathematical Statistics* 22 (1951), 400-407.

White, Halbert, "Neural-Network Learning and Statistics." *AI Expert*, December, 1989.