Neural Networks and Other Nonparametric Techniques in Economics and Finance

Andrew W. Lo
Harris & Harris Group Professor of Finance
Massachusetts Institute of Technology

Because financial data contain important nonlinearities, neural networks and other nonlinear models can be extremely useful in financial modeling. These models are not always preferable to linear approaches, however, and understanding of their benefits and drawbacks can help the modeler make choices. Moreover, alternative nonparametric techniques may be more useful than neural nets in some cases; a useful technique that closely approximates human estimation, for example, is kernel regression.

Recent advances in the theory and implementation of (artificial) neural networks have captured the imagination and fancy of the financial community.[1] Although they are only one of the many types of statistical tools for modeling nonlinear relationships, neural networks seem to be surrounded by a great deal of mystique and, sometimes, misunderstanding. Because they have their roots in neurophysiology and the cognitive sciences, neural networks are often assumed to have brain-like qualities: learning capacity, problem-solving abilities, and ultimately, cognition and self-awareness. Alternatively, neural networks are often viewed as "black boxes" that can yield accurate predictions with little modeling effort.

In this presentation, I hope to remove some of the mystique and misunderstandings about neural networks by providing some simple examples of what they are, what they can and cannot do, and where neural nets might be profitably applied in financial contexts.[2] I will also discuss several other types of nonparametric techniques to emphasize the connection between neural networks and more standard statistical models of nonlinear relations. As with all emerging technologies, neural networks have been touted by some as a revolution and criticized by others as snake oil. In fact, they are neither: Neural networks are an interesting and potentially powerful way to model complex relations in some, but surely not all, applications.

Motivation for Using Neural Nets

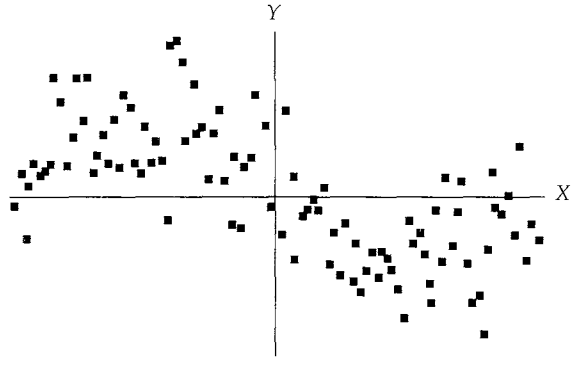
The motivation behind most nonlinear modeling techniques is to determine a useful answer to the following simple question: How are the variables X and Y related? This is a question that can be asked on several different levels. For example, consider **Figure 1**, which depicts a scatter diagram of observations (X_i, Y_i). Is this relationship linear or nonlinear? Despite the pervasiveness of linear models in traditional financial analysis (e.g., the Sharpe–Lintner capital asset pricing model and Ross's arbitrage pricing theory), they are a rather small subset of the set of all possible models. Because, by definition, nonlinear models deal with all relationships that are *not* linear, they are a considerably "larger" set of models than linear models. Only recently have academics and financial analysts begun to realize that financial data contain important nonlinearities.

On another level, we can ask whether the relation depicted in Figure 1 is deterministic or random. Although I myself do not find this distinction very useful, it has attracted considerable attention in some financial circles, particularly in those that apply the mathematics of nonlinear dynamics and "chaos" to

[1] The adjective "artificial" is often used to distinguish mathematical models of neural networks from their biological counterparts. For brevity, I shall omit this qualifier for the remainder of this paper although it will be implicit in all of my references to neural networks.

[2] Since this is meant to be an introductory talk, I will not hesitate to sacrifice rigor for clarity. More mathematically inclined readers are encouraged to consult Hertz, Krogh, and Palmer (1991) for the general theory of neural computation, White (1992) for the statistics of neural networks, and Aït-Sahalia and Lo (1994) and Hutchinson, Lo, and Poggio (1994) for financial applications.

Figure 1. Scatter Plot of a Nonlinear Relationship



Source: Andrew W. Lo.

investment management (see Hsieh 1991 for an excellent overview).

On yet another level, we can ask whether Figure 1 exhibits a predictable relation between X and Y; does observing X help us to forecast Y, or is Y unforecastable? Does having additional X variables—a multifactor model—yield better forecasts of Y? Also, Y can be discrete instead of continuous. For example, Y would be discrete if we were predicting classifications such as whether a security is a good investment or a bad one or whether a counterparty is a good credit risk or a poor one.

You see how difficult it can be to answer the question: How are X and Y related? There are many questions within this seemingly simple question, and without additional structure and information, the question has no single answer. While neural networks can provide a fairly broad set of answers to the question of how X and Y are related (see the section "What Neural Networks Can Do" for an example), they should not be viewed as black boxes; they do "connect" inputs to outputs, but what happens in-between is critical to the interpretation (and exploitation) of the nonlinearities captured by the network.[3]

Now, before I turn to a formal discussion of neural networks, let me provide one simple financial example that motivates our interest in nonlinear models. This example, first suggested by Professor Robert Merton, involves allocating assets each month between Treasury bills and the S&P 500 Index, starting January 1926 and ending December 1993. Now, a $1 investment in Treasury bills, rolled over from month to month, grows to $12 at the end of this 67-year period, whereas the same investment in the S&P 500 grows to $800. What if you had perfect market-timing skills so that, at the start of each month, you knew for sure which asset class would perform better? Starting with a $1 investment in January 1926 and switching your assets wholly into Treasury bills or the S&P 500 each month depending on which will have the higher return, what would your investment become at the end of 1993? The answer: $1,038,317,644! No, this is not a typographical error; with perfect asset allocation skills, a $1 investment in 1926 would have grown to over one *billion* dollars in 1993.

Now, of course, no one has perfect asset allocation skills; therefore, in practice, the returns to asset allocation may be only a small fraction of $1,038,317,644. However, it does not take a very large fraction of $1,038,317,644 to exceed the $800 return on the S&P 500! And this is perhaps the most tantalizing aspect of investment management: Even a very slight advantage in a very competitive market can translate into handsome returns over time. Detecting and modeling nonlinearities might just provide such a slight advantage.

[3]To see why, consider the nonlinear stock selection system Θ-94 in the presentation "Data-Snooping Biases in Financial Analysis" contained in this proceedings.

What Are Neural Networks?

To develop some basic intuition for neural networks, consider a typical nerve cell, or "neuron." A neuron has dendrites (receptors) at different sites that react to stimuli. These stimuli are transmitted along the axon by an electrical pulse. If the electrical pulse exceeds some threshold level when it hits the nucleus, this triggers the nucleus to react—for example, to make a particular muscle contract. This basic biological unit is what mathematicians attempt to capture in a neural network model.

Now, the human brain has approximately 100 billion neurons. Each neuron is relatively simple, having the basic structure I just described. But when such large numbers of these relatively simple nerve cells are combined in parallel and allowed to interact in nonlinear ways, these interactions can produce very complex behavior, perhaps even human thought. This is probably the single most compelling feature of neural network models: the ability to capture complex phenomena through the combination of many units (hence, the term "network") of a simple mathematical model of a neuron.

Perhaps the first, and simplest, mathematical model of the neuron—the binary threshold model—was introduced by McCulloch and Pitts in 1943. According to their model,

$$Y_t = \Theta\left(\sum_{j=1}^{J} \gamma_j X_{jt} - \mu\right) \qquad (1)$$

and

$$\Theta(x) = \begin{Bmatrix} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{Bmatrix}, \quad (2)$$

where X_{jt} represents the stimulus received at the jth dendrite or receptor site at time t, and Y_t represents the neuron's response at time t to all the stimuli. In Equation 1, each stimulus X_{jt} is multiplied by a factor γ_j called the "connection strength." The connection strength γ_j indicates how sensitive the neuron is to stimulus from that receptor site. The larger the γ_j, the more sensitive the neuron to that particular input.[4] After multiplication by the connection strengths, the inputs are summed, and if this weighted sum exceeds the "threshold level" μ, then according to Equation 2, the neuron is "activated," yielding a value of 1 for Y_t. If the weighted sum does not exceed the threshold level, then the neuron remains "dormant," yielding a value of 0 for Y_t.

This model of the neuron is the simplest of its kind, with only two output values, 0 or 1, but it does capture the basic features of a nerve cell. However, neurobiologists have come to realize that nerve cells are not so simple, exhibiting considerably more complex and subtle behavior than the binary threshold model can capture. Nevertheless, as a purely statistical tool, the binary threshold model may still be useful for modeling nonlinear relations between a set of inputs and an output. In fact, Equations 1–2 are already a nonlinear model because the mapping Θ between the (possibly) continuous variables $\{X_{jt}\}$ and the discrete Y_t is indeed nonlinear.

In the 1940s, there was a tremendous amount of interest and research in these kinds of systems, and we are seeing a resurgence of this enthusiasm today. Interestingly, one of the earliest motivations for developing the modern digital computer had nothing to do with computing applications; it was to construct a digital representation of the human brain! No less a mathematician than John von Neumann, often called the father of computer science, was deeply interested in mathematical models of biological systems. His ultimate goal was to construct a machine that would mimic human thought, and it is unfortunate that his untimely death interrupted this particular line of research. Who knows what he would have created had he been able to complete his ambitious studies?[5]

With respect to current (nonbiological) applications, neural networks are used to give structure to data. Viewed as a statistical estimation technique, neural networks are a flexible model of nonlinearities. In this sense, they are just one example of a nonparametric nonlinear estimation technique. "Nonparametric" means that no parametric assumptions are required, such as the assumption of normally distributed errors for the typical linear regression model.

In addition to being nonparametric, neural network models also have a very useful feature known as the "universal approximation property." This property means that, with enough nodes or enough hidden layers (to be defined shortly) and under certain conditions, a neural network can approximate *any* nonlinear relationship, no matter how strange or nonlinear.

Many nonparametric techniques also share this universal approximation property (see the section "Other Nonparametric Techniques," for example), so neural networks should not be preferred on these grounds. Instead, it is often argued that neural networks are computationally more efficient in some specific applications. One feature rarely emphasized about neural networks in financial applications is that, because they are fairly simple, many of them can be "strung together" in parallel, yielding extraordinary gains in computational speed. However, the most commonly used computers today are not yet parallel. Thinking Machines Corporation does market its Connection Machine,[6] and some other massively parallel processors are available at the high end of the market, but the typical computer is not able to take advantage of this property. In fact, parallel computation is one of the main advantages of neural networks.

Examples of other nonlinear estimation techniques include splines, wavelets, kernel regression, projection pursuit, radial basis functions, nearest-neighbor estimators, and perhaps the most accurate of all, human intuition. The question is not so much whether one technique is preferred over another but, rather, how they compare overall to linear alternatives, and in which particular application is one more efficient and accurate than another? Most of them have the universal approximation property, so what is more relevant is which is right for a *particular* application.

A Neural Network Primer

To set up a neural network, we first define the target variable Y (or dependent variable, in the terminology

[4] If $\gamma_j > 0$, the input X_{jt} is said to be "excitatory." If $\gamma_j < 0$, the input is said to be "inhibitory."

[5] Some thought-provoking philosophical musings and preliminary research findings are contained in von Neumann's (1958) incomplete monograph *The Computer and the Brain*, which was published posthumously.

[6] Unfortunately, Thinking Machines Corporation has just filed for bankruptcy, perhaps an indication that the market is not yet ready to take full advantage of the computational efficiency that neural networks can provide.

of linear regression) and the set of inputs $\{X_j\}$ (or explanatory variables) that are used to forecast or explain Y. The weighted sum of the inputs, weighted by the connection strengths $\{\gamma_j\}$ (or coefficients), is then compared against the threshold level μ. This comparison requires an activation function $\Theta(\cdot)$ that determines the state of the neuron. However, for most financial applications, a 0–1 response is not appropriate. For example, if we are attempting to forecast equity returns, the response should be continuous since returns are continuous. In this case, we want a continuous activation function, not Equation 2. **Figure 2** provides an example of a continuous activation function, often called the "logistic" or "sigmoid" function, which is graphed alongside the 0–1 activation function, Equation 2, often called the "Heaviside" function.

We are now ready to define the most basic neural network model on which most current applications are built: the "single-layer feedforward perceptron" with a continuous (logistic) activation function:

$$Y_t = \Theta\left(\sum_{j=1}^{J} \gamma_j X_{jt}\right), \tag{3}$$

where

$$\Theta(x) = \frac{1}{1+\exp(-x)}. \tag{4}$$

As before, the inputs are multiplied by the connection strengths and summed. However, observe that the threshold level μ is no longer subtracted from the sum in Equation 3. Without loss of generality, we can let one of the inputs, say X_{1t}, take on the value –1 for every t; hence, the connection strength γ_1 of X_{1t} plays the same role as μ in Equation 1. Such networks are called "feedforward" because there is no feedback from the output layer back to the input layer (see Hertz, Krogh, and Palmer 1991, Chapter 7, for a discussion of "recurrent" networks in which feedback is allowed). Despite the simplicity of Equation 3, it is the basis of even the most complex neural networks used in practice.

Now, in general, it is almost impossible to obtain a perfect fit or forecast of the output Y_t given the inputs $\{X_{jt}\}$; hence, there will almost always be some error ε_t. Therefore, we can recast the neural network model in a more general statistical framework in which the target Y_t is related to an unknown nonlinear function $m(X_{1t}, \ldots, X_{Jt})$ plus an error term:

$$Y_t = m(X_{1t}, \ldots, X_{Jt}) + \varepsilon_t. \tag{5}$$

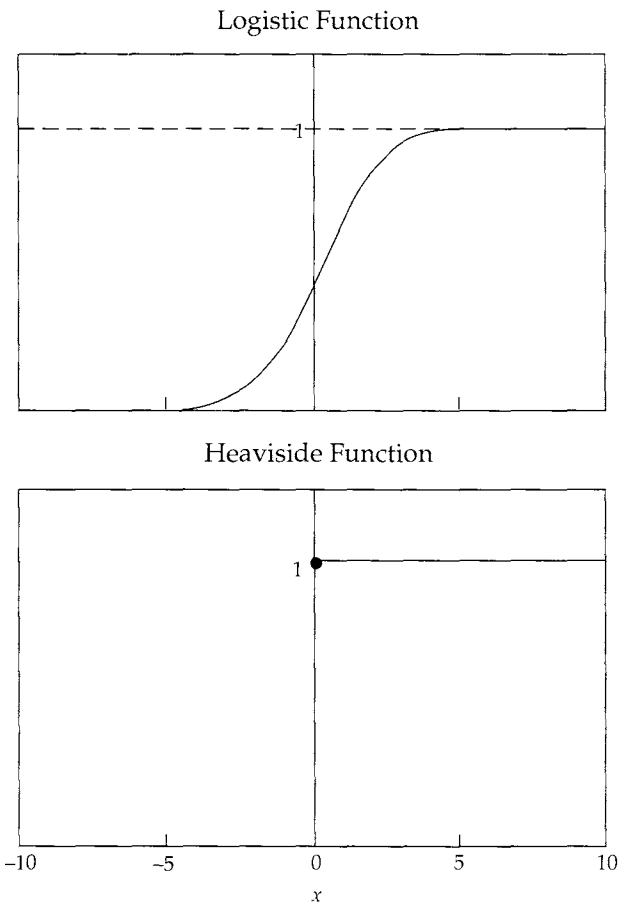
The goal is to find the function $m(\cdot)$ that works "best." If $m(\cdot)$ were restricted to be linear and "best" meant minimum mean squared error, ordinary least squares or linear regression would be the solution to this problem. But because we wish to allow m to be nonlinear, the problem is considerably more complicated, and this is where nonparametric nonlinear techniques such as neural networks come into play. The neural network model can be viewed as an estimate $\hat{m}(\cdot)$ of the function $m(\cdot)$; hence,

$$\hat{Y}_t = \hat{m}(X_{1t}, \ldots, X_{Jt}) = \Theta\left(\sum_{j=1}^{J} \gamma_j X_{jt}\right). \tag{6}$$

Back-Propagation versus Nonlinear Least Squares

How do we specify the connection strengths $\{\gamma_j\}$? They must be estimated. The popular notion that neural networks can "learn" is based on the process of successive estimates of $\{\gamma_j\}$ as new data are acquired; in neural network parlance, the network is

Figure 2. Activation Functions



Source: Andrew W. Lo.

"trained" on the data. In particular, we might consider selecting $\{\gamma_j\}$ to minimize the sum of squared "errors" or differences between the target and the network output:

$$\underset{\gamma_j}{\text{Min}} \sum_{\tau=1}^{t} \left[Y_\tau - \Theta\left(\sum_{j=1}^{J} \gamma_j X_{j\tau} \right) \right]^2 . \tag{7}$$

If $\Theta(\cdot)$ were linear, the solution to Equation 7 would be linear regression. But since $\Theta(\cdot)$ is assumed to be nonlinear, the solution to Equation 7 must be obtained by another method: nonlinear least squares. Nonlinear least squares is a well-known technique for optimizing nonlinear functions and is available in many computer packages.

An alternative to nonlinear least squares that is commonly used is an algorithm called "back-propagation," in which the connection strengths are updated recursively in the following manner:

$$\gamma_t = \gamma_{t-1} + \eta \times \nabla\Theta_t \times (Y_t - \Theta_t),$$

$$\Theta_t \equiv \Theta\left(\sum_{j=1}^{J} \gamma_{jt} X_{jt} \right), \tag{8}$$

where $\gamma_{t-1} \equiv [\gamma_{1t-1}, \gamma_{2t-1}, \ldots, \gamma_{Jt-1}]$ is the vector of connection strengths based on time $t-1$ information, $\nabla\Theta_t$ is the vector of first derivatives of Θ_t with respect to the γ_{jt}'s, γ_t is the updated vector that takes into account the information contained in the forecast error $Y_t - \Theta_t$, and η is a parameter that determines the sensitivity of the updating process to the forecast error. As this algorithm is given more data, it will update the model to determine a "better" set of γ_j's. Although at any given time t, γ_t generally does not solve the minimization problem (Equation 7), it can be shown that, eventually, γ_t will converge to the nonlinear least squares solution.

But if the back-propagation estimate of γ is only guaranteed to converge to the solution of Equation 7 *eventually* (and it is virtually impossible to say how long this will take in general), why not simply use the nonlinear least squares estimate, which does solve Equation 7? The typical justification for back-propagation is the fact that it is an "on-line" method: It is a rule for updating the connection strengths given only the current connection strengths and the most recent observations of the inputs and target.[7] In other words, we do not have to reprocess the entire historical data set in order to estimate a new set of connection strengths; today's observation and today's $\{\gamma_j\}$ are sufficient to arrive at the new $\{\gamma_j\}$.

On-line methods are most useful when speed is relatively more important than accuracy. For example, while a guided missile is tracking its target in flight, it cannot reprocess all of the accumulated historical tracking data each time it receives new coordinates for its target; otherwise, it would never reach its target in real time. Instead, a guided missile uses on-line methods for tracking its target, which may not be as accurate as off-line methods but are accurate enough for all practical purposes.

For financial applications, the distinction between on-line and off-line algorithms is not as critical since off-line processing is, for most financial data sets, almost instantaneous anyway. Perhaps for very short term real-time trading systems, those involving tick data, for example, an on-line algorithm may be preferable to an off-line algorithm. But, in general, on-line algorithms tend to be numerically unstable and very slow to converge to the solution of Equation 7 in financial applications. If instantaneous processing is not a critical factor, nonlinear least squares is probably preferable. Recall that nonlinear least squares does solve Equation 7 exactly for a given sample, whereas back-propagation does so only approximately, as the sample size grows.

Hidden Layers

So far, we have considered the simplest neural network model, a single-layer feedforward perceptron in which inputs are connected to an output through a nonlinear activation function. Although this specification does indeed capture some kinds of nonlinearities, it does *not* possess the universal approximation property, a point forcefully made by a simple counterexample in Minsky and Papert (1969). In that example, Minsky and Papert constructed a particularly simple nonlinear function, called the "exclusive-OR," or XOR, function, which no single-layer feedforward perceptron can approximate.

The solution to the XOR problem was discovered relatively recently by Rumelhart, Hinton, and Williams (1986), and it consists of adding a "hidden layer" in between the network's input and output layer. In particular, instead of feeding inputs directly to the output, let the inputs be fed to several activation functions:

[7] This terminology comes from the signal-processing literature, in which an on-line algorithm does not require switching off the signal generator while it processes the most recent data, whereas an off-line algorithm processes the most recent data together with all of the accumulated data from the past and usually does require switching off the generator until its processing is complete.

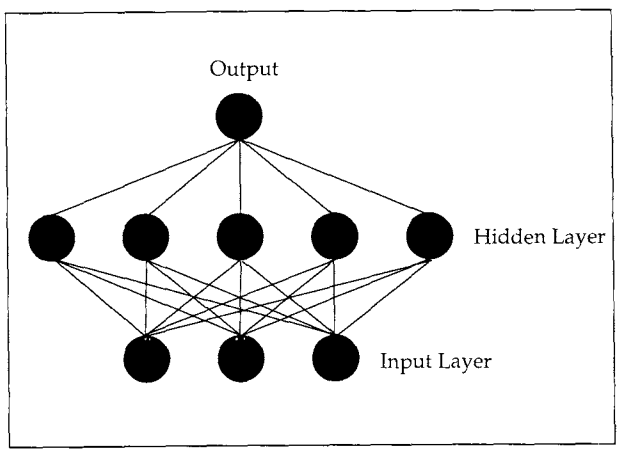
$$\Theta\left(\sum_{j=1}^{J}\gamma_{1j}X_{jt}\right), \Theta\left(\sum_{j=1}^{J}\gamma_{2j}X_{jt}\right), \ldots, \Theta\left(\sum_{j=1}^{J}\gamma_{Kj}X_{jt}\right), \quad (9)$$

where the activation functions $\Theta(\cdot)$ are all the same but there are now K distinct sets of connection strengths, one for each "node" of the hidden layer (**Figure 3**). The output from each node k is multiplied by a coefficient δ_k, summed over all K nodes, and then fed into another activation function $F(\cdot)$. The final output of this network is then

$$\hat{Y}_t = F\left[\sum_{k=1}^{K}\delta_k\Theta\left(\sum_{j=1}^{J}\gamma_{kj}X_{jt}\right)\right]. \quad (10)$$

With one hidden layer, the perceptron is now called a "multilayer perceptron" (MLP), and it possesses the universal approximation property. A single-layer perceptron cannot capture all forms of nonlinearity; a multilayer perceptron can. MLPs are the most common type of neural networks currently used. Most commercially available neural network software packages estimate MLPs, and they allow for many hidden layers. In this way, by taking functions of functions of functions, etc., you can quickly get extraordinarily complex behavior from rather simple building blocks.

Figure 3. Feedforward Perceptron with a Single Hidden Layer



Source: Andrew W. Lo.

What Neural Networks Can Do

Even simple neural networks with only one hidden layer can capture a variety of nonlinearities. Consider, for example, the sine function plus a random error term:

$$Y_t = \sin(X_t) + 0.5\varepsilon_t, \quad (11)$$

where ε_t is a standard normal random variable. Can a neural network extract the sine function from observations (X_t, Y_t)?

To answer this question, 500 (X_t, Y_t) pairs were randomly generated subject to the nonlinear relationship (Equation 11) and a neural network model was estimated using these artificial data (or in the jargon of this literature, a neural network was trained on this data set). The particular neural network used has one hidden layer and five hidden nodes.[8] The following equation is the result of training the neural network on the data using nonlinear least squares:

$$\begin{aligned}\hat{Y}_t = {} & 5.282 - 14.576 \times \Theta(-1.472 + 1.869\,X_t) \\ & - 5.411 \times \Theta(-2.628 + 0.641\,X_t) \\ & - 3.071 \times \Theta(13.288 - 2.347\,X_t) \\ & + 6.320 \times \Theta(-2.009 + 4.009\,X_t) \\ & + 7.892 \times \Theta(-3.316 + 2.484 X_t), \end{aligned} \quad (12)$$

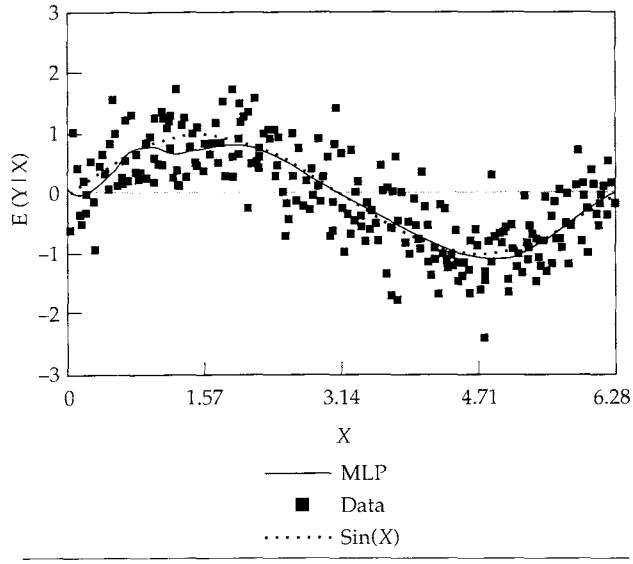
where $\Theta(x)$ is the usual logistic function $1/(1 + \exp[-x])$. This network has five identical activation functions $\Theta(\cdot)$ (corresponding to the five nodes in the hidden layer) and a constant term, and the activation function $F(\cdot)$ for the output layer is assumed to be the identity function $F(x) = x$. There are only two inputs, X_t and 1.

Now, Equation 12 looks nothing like the sine function, so in what sense has the neural network "approximated" the nonlinear relation, Equation 11? In **Figure 4**, the data points (X_t, Y_t) are plotted as squares, the dotted line is the theoretical relation to be estimated (the sine function), and the solid line is the relation as estimated by the neural network, Equation 12. The solid line is impressively close to the dotted line, despite the noise that the data points clearly contain. Therefore, although the functional form of Equation 12 does not resemble any trigonometric function, its numerical values do.

Figure 4 also shows that the neural network estimate deviates from the sine curve more for some values of X_t than for other values of X_t. For example, when X_t is near 1.57, the solid line seems to dip below the dotted line, deviating to a greater extent than for other values of X_t. Why is this? The answer lies in the *absence* of data points above the dotted line and the presence of data points below it when X_t is near 1.57. The MLP model can only reveal what is in the data, and since there are more points below the

[8]There are no formal rules yet for optimally selecting the number of hidden layers or nodes per layer, and this is one of the primary drawbacks of neural network models. Currently, experience and heuristics are the only guides we have for specifying the network "topology."

Figure 4. Multilayer Perceptron Estimator of $Y_t = \sin(X_t) + 0.5\,\varepsilon_t$



Source: Andrew W. Lo.

dotted line than above it when X_t is near 1.57, the estimated curve will also inherit this bias. Like all statistical models, neural networks are subject to estimation error. Nevertheless, this example demonstrates that, if there is genuine nonlinearity in the data, neural networks can capture it reasonably well.

What Neural Networks Cannot Do

Having seen what neural networks can do, it is time to consider what they cannot do. Consider the well-known law of supply and demand that is taught in every introductory economics course: The price P and quantity Q that clear a market are determined by the intersection of the market's supply and demand curves. Let the (linear) demand curve at time t be given by

$$Q_t^d = d_0 + d_1 P_t + d_2 I_t + \varepsilon_t^d, \tag{13}$$

where P_t is the purchase/selling price, I_t is household income, and ε_t^d is a random demand shock, and let the (linear) supply curve at time t be given by

$$Q_t^s = s_0 + s_1 P_t + s_2 C_t + \varepsilon_t^s, \tag{14}$$

where C_t is production costs and ε_t^s is a random supply shock. An economic equilibrium, or market clearing, occurs when supply equals demand; hence,

$$Q_t^s = Q_t^d = Q_t, \tag{15}$$

where Q_t is the equilibrium quantity produced and consumed at time t.

Suppose that the target variable to be forecasted is the quantity demanded Q_t^d as given by Equation 13,[9] and we train an MLP with a single hidden layer on all of the available inputs: equilibrium price P_t, production costs C_t, household income I_t, and the constant 1. Moreover, let us assume that this MLP is given an *infinite* number of data points on which to train! It can easily be shown that this single hidden-layer MLP can *never* recover the demand curve (Equation 13), despite the fact that the demand curve is linear and despite the fact that the MLP is given an infinite amount of data on all the relevant variables.

In fact, under the stated assumptions, the demand curve can be readily estimated by a variant of linear regression: two-stage least squares (also known as instrumental variables estimation). This linear procedure exploits the fact that what we observe in the data is neither the demand curve nor the supply curve but, rather, the intersection of the two (see Equation 15). Without this crucial piece of information, neural networks and any other nonparametric, nonlinear estimation technique will never be able to recover the demand curve.

Now, of course, one may cry "foul play" at my example because I have purposely omitted Equation 15 from the network specification. But this is precisely the point. The network cannot "learn" Equation 15 by itself, contrary to popular belief, but must have this structure built into its design to be able to recover the demand curve. Of course, if I structured the network topology to account for Equation 15, the resulting network would perform as well as (if not better than) two-stage least squares. But there will always be a need for structure and information obtained from outside the network because there are simply too many nonlinear models to choose from, and even an infinite amount of data is sometimes not enough. Neural networks are not black boxes.

Practical Considerations

Although MLP neural networks do have their roots in biology, their current relation to biological phenomena is metaphoric at best. When popular ac-

[9]This is not as frivolous an example as it may seem. In particular, a significant portion of consumer marketing research is devoted to estimating demand curves. Public policies such as gasoline, alcohol, and cigarette taxes, energy tax credits, and investment incentives also often hinge on price-elasticity-of-demand estimates. Finally, economic analysis of demand is frequently the basis of damage awards in a growing number of legal disputes involving securities fraud, employment discrimination, and antitrust violations.

counts of neural networks claim that they "learn," what they mean is that estimates of connection strengths and other model parameters change when confronted with new data. But virtually all statistical models "learn" in exactly the same sense; for example, linear regression parameters generally change when confronted with additional data. Whether or not neural networks learn in the same ways that humans learn is an open question, primarily because we do not yet have a satisfactory theory of how humans learn. A neural network is one way to estimate a nonlinear relation, but it is certainly not the only way (as discussed in the next section).

In particular, neural networks have some advantages and disadvantages. One advantage is the simplicity with which neural networks can be implemented on digital computers. They are especially simple to code in almost any higher-level programming language such as C or Fortran; hence, development time is reduced. This is often at the expense of computer time, but given the continuing decline in computing costs and increase in computing speed, this trade-off may be economically optimal.

The main disadvantages of neural networks are the need for theoretical restrictions in designing the network topology and the tendency towards "overfitting," in which the network yields an almost perfect fit in-sample by "memorizing" the data but breaks down out-of-sample. Overfitting is more likely to be a problem for nonlinear estimation techniques because such techniques typically have many more degrees of freedom than linear models (an arbitrary number of nodes in each hidden layer, an arbitrary number of hidden layers, etc.). Therefore, nonlinear techniques can fit almost any set of data to almost any degree of accuracy by simply "connecting the dots." But this implies nothing about the model's performance on new data; hence, a nonlinear model with near-perfect fit may have almost no predictive power.

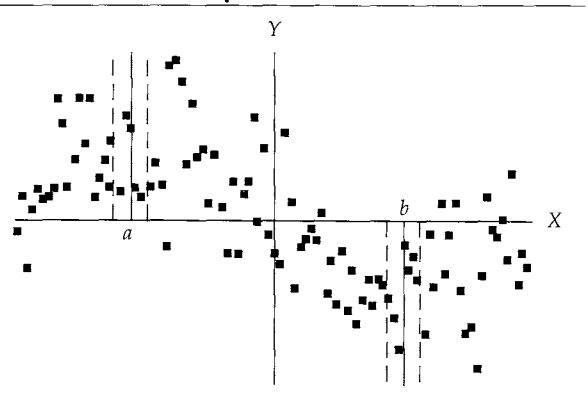
Much of the success or failure of networks depends on how much clean data you have, what the "signal-to-noise" ratio is, and how good your theoretical restrictions are. For relatively small data sets, nonlinear techniques will not work very well unless there is a great deal of theoretical structure and information available. Loosely speaking, there is a trade-off between structure and sample size: With larger samples, some structure may be recovered from the data directly; with smaller samples, that structure must be supplied. Of course, as the example in the preceding section demonstrated, there are cases where even an infinite amount of data cannot generate the structure needed to yield an accurate estimate.

Another disadvantage of neural networks is the difficulty in performing standard statistical inference for estimates of the model's parameters. This is a result of the network's layered structure. For example, a simple *t*-test to see whether the connection strength of one hidden node is statistically significant requires many other assumptions about whether the various connections above it will be significant. Because of the recursive nature of the networks, such *t*-tests and related goodness-of-fit measures are almost impossible to interpret.

Other Nonparametric Techniques

To emphasize the fact that neural networks are not the only way to capture nonlinear relations, I want to present a popular alternative for estimating nonlinear relations nonparametrically: kernel regression. This is a very intuitive approach to estimating nonlinearities based on taking "local averages." To see how it works, try to draw a continuous curve free-hand through the scatter of points in Figure 1 so as to obtain the "best" fit. Your drawing probably looks something like **Figure 5**, a curve that passes through the middle of the scatter plot.

Figure 5. Free-Hand Estimation of a Nonlinear Relationship



Source: Andrew W. Lo.

What is the biological neural network (i.e., in you) doing to arrive at this free-hand curve? It is computing local averages, visually. For example, to estimate the value of the nonlinear function for values of X near a, consider the group of points in between the dashed lines surrounding the vertical line centered at a. Common intuition suggests picking a point somewhere in the middle of the vertical range. This is your visual estimate of the value of the function when X is near a, and it is a local average of the vertical distances of the points in between the dashed lines around a. Similarly, in estimating the value of the function for values of X near b, you also pick a point somewhere in the middle of the group

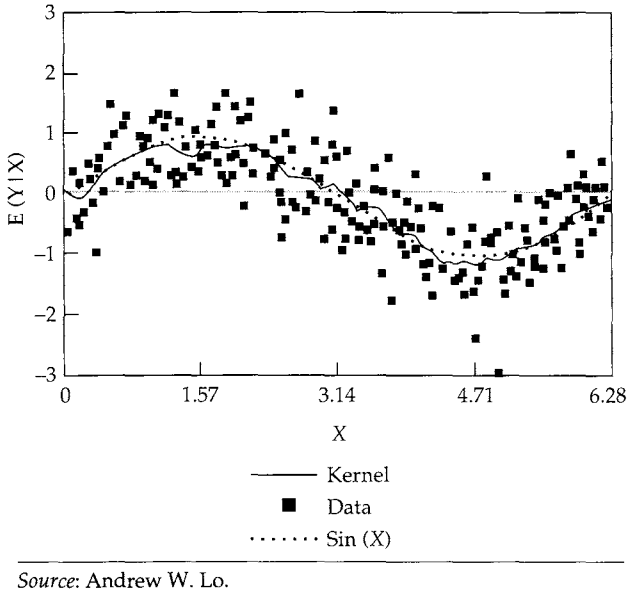
of points in between the dashed lines around b. If you continue to do this—taking vertical averages of points around a small neighborhood of each value of X—you will arrive at the free-hand curve drawn in Figure 5.

This process of taking local averages can be formalized mathematically and yields the nonparametric, nonlinear estimation technique known as kernel regression. The main subtlety involved in kernel regression is how to choose the width of the neighborhood over which the local averaging is done, often called the "bandwidth." For example, if the local averaging is over the entire range (corresponding to dashed lines that enclose *all* the points in the data set), the result would be a flat line. If the local averaging is over very tiny intervals (corresponding to dashed lines that contain a very small number of points), the result would not be averaging but, rather, connecting the points. Too wide a bandwidth produces an estimated curve that is too smooth, and too narrow a bandwidth produces an estimated curve that is too choppy. The ideal bandwidth is something in between these two extremes.

To see kernel regression in action, **Figure 6**, **Figure 7**, and **Figure 8** display kernel regression estimates on the simulated sine-curve-plus-noise data of Figure 4 for several bandwidths. If the averaging is done over very small neighborhoods, as in Figure 6, the kernel estimator is fitting noise as well as the genuine sine curve; hence, the estimated curve is too choppy. If the bandwidth is increased slightly, as in Figure 7, the estimated curve looks about as good as the MLP estimator in Figure 4. If the bandwidth is increased beyond this, as in Figure 8, the kernel is averaging over too wide a region, and the result is a nearly flat line.

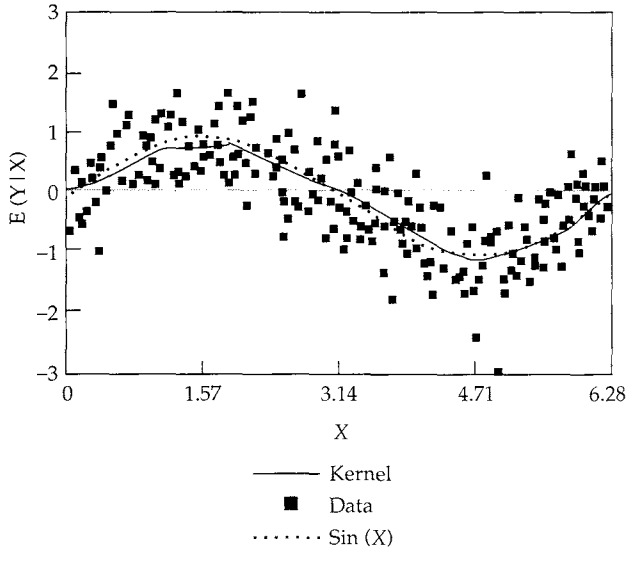
This nonparametric technique works just as well as neural networks in some contexts, better in others, and worse in yet others. The success of either of these approaches depends intimately on the particular application at hand.

Figure 6. Kernel Regression Estimator of $Y_t = \sin(X_t) + 0.5\varepsilon_t$ (bandwidth $h = 0.1\sigma$)



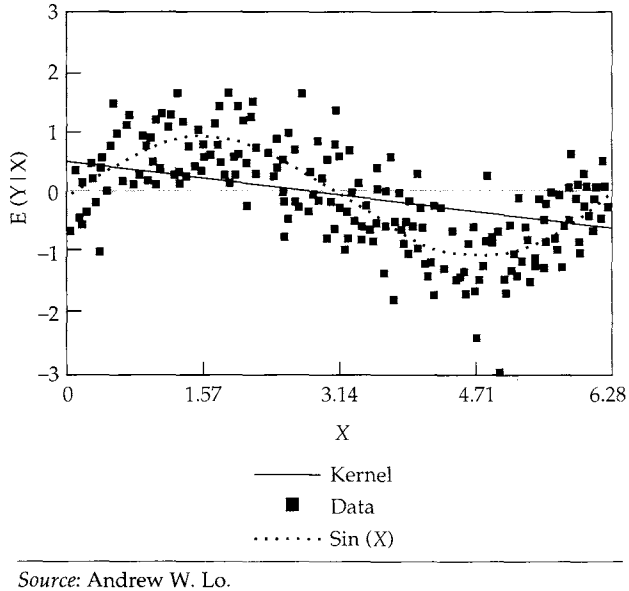
Source: Andrew W. Lo.

Figure 7. Kernel Regression Estimator of $Y_t = \sin(X_t) + 0.5\,\varepsilon_t$ (bandwidth $h = 0.3\sigma$)



Source: Andrew W. Lo.

Figure 8. Kernel Regression Estimator of $Y_t = \sin(X_t) + 0.5\varepsilon_t$ (bandwidth $h = 2.0\sigma$)



Source: Andrew W. Lo.

Conclusion

All quantitative models are approximations to reality. The issue is not so much whether a model is true or false but, rather, how good an approximation it is to the reality we wish to capture. Nonlinearities are important in the data, so using a nonlinear model has advantages, but a variety of trade-offs must be made. If you are mindful of these trade-offs, you will find that nonlinear techniques can often shed a great deal more light on the data than linear techniques.

Question and Answer Session

Andrew W. Lo

Question: Couldn't the quantitative techniques you discussed just be considered sophisticated pattern-recognition techniques (i.e., technical analysis)?

Lo: There is a very important difference between the nonlinear techniques I discussed and pure pattern recognition: In the approach I discussed, economic and financial theories impose some stability and discipline on the modeling process. One exercise I give to my investments class is to have them look at six plots of data, of which one is historical stock returns and the other five are randomly generated numbers. Their task is to pick out the plot with "real" data. On average, only one-sixth of the class picks the right one, about what you would expect from pure chance. I invited a couple of technical analysts to my class to do the same thing. Of course, they picked the wrong ones.

Now, this is not to say that technical analysis is useless; on the contrary, some of my earlier research suggests that technical analysis can add value to investment strategies. The point of my exercise was to show that we can all see patterns in *random* data. Relying solely on statistical techniques, solely on pattern recognition, can be very misleading. By themselves, statistics and pattern recognition techniques cannot tell you what is going on in the data. You must have some structure, which can take the form of a fancy economic model, 20 years of trading experience, or your own pet theories about what human psychology is doing over the course of the business cycle. But in the end, the best models are those that bring some additional structure and information to the process of statistical analysis and pattern recognition.

Question: For nonparametric techniques, what is the best approach to take in identifying and constructing input variables?

Lo: There are a variety of techniques, but the most reliable is "wetware," the stuff between your ears, the human brain. For input variables, nothing can replace basic human judgment, intuition, and whatever economic models are available. For example, you might have an economic theory that says dividend yield affects stock returns differently in business cycle upturns than in downturns. That theory might be based on 40 years of living through upturns and downturns and reading the newspaper, or it might be based on some fundamental insight about human psychology. Using biological neural networks is probably the most powerful method for selecting inputs.

Inputs can be selected in other ways too. Reading the research of others is perhaps the most common method; when you read about an interesting anomaly or a clever relation that someone else has uncovered, you might want to try it too. On a more mechanical level, you could use a higher-level statistical analysis package and evaluate all possible combinations of input variables to determine the best fit. That is not a bad way of choosing inputs as long as you are aware that, even if no relationship exists, you will find some very impressive answers. Data snooping and biases pose real problems for mechanical techniques of input-variable selection, and you have to take these risks into account.

Question: When you described how you approached the sine-wave net, you said you selected one hidden layer and five nodes through experience. Isn't most of the knowledge of a neural net embedded in the particular structure of the net? For instance, I would assume that, when we learn, we do so by building links between specific individual neurons in our brain. If so, isn't a detailed design of all the hidden layers and nodes and their meanings required to build a truly successful net?

Lo: Yes, such a design would be required to build a truly successful model of human thought or biological systems. For example, neurophysiologists have already compiled a great deal of evidence that demonstrates the importance of network topology for biological functions. However, as a statistical model of economic phenomena, a single hidden layer can match virtually any kind of nonlinear phenomenon as long as you have a sufficient number of nodes in the hidden layer. Nevertheless, there can be great increases in network efficiency and accuracy by carefully designing the network topology. Unfortunately, this is still more of an art than a science, and we have to fall back on human intuition and experience to guide our design process.

Question: Do techniques such as Box–Jenkins ARIMA, Fourier transforms, etc., have value in solving the same problems as neural networks? If so, what are the relevant efficiencies,

pros/cons, and so forth?

Lo: Fourier analysis, which is another example of a nonlinear nonparametric estimation technique, is quite similar to neural networks in how it captures nonlinear phenomena. A neural network with a single hidden layer and many hidden nodes can be viewed as a "series expansion" of a nonlinear function. The Fourier series is also a series expansion of a nonlinear function but in terms of sines and cosines. You can also construct an expansion in terms of exponentials (wavelets). Therefore, there is a close connection between neural networks, Fourier analysis, and other series expansion techniques.

However, Box–Jenkins ARIMA models are *linear* models; the inputs are mapped linearly to an output. Although these models are useful, often providing good approximations of what you are trying to capture, they are still linear. So, if you are trying to capture a nonlinear relationship, you will not be able to do it by ARIMA models.

To emphasize the importance of nonlinearities in financial applications, I recently completed a study in which I used neural network models to price and hedge options (see Hutchinson, Lo, and Poggio 1994). I chose option pricing as the test bed for this neural network analysis because options are inherently nonlinear instruments. The nonlinearity is clearly defined in the payoff of the option. For comparison, I used a linear regression to estimate the option price as a function of strike price, current stock price, and various other inputs. Not surprisingly, the linear model did poorly, whereas several different nonlinear techniques did considerably better.

As a general rule of thumb, you should first have a good idea of the phenomena you are trying to capture with your quantitative model *before* you begin your statistical analysis. If the phenomena are truly nonlinear, then there is a good chance that nonlinear techniques can add value to your analysis. If, however, the nonlinearities are rather unimportant, then you may be much better off sticking with more standard techniques like linear regression and ARIMA models.

Question: Is a back-propagation network different from multilayer perceptrons? Can back-propagation have multiple hidden layers too?

Lo: The term "back-propagation network" is quite misleading because it associates a technique for estimating connection strengths with the network. Back-propagation is one way to estimate the connection strengths of a multilayer perceptron; nonlinear least squares is another way. Therefore, the two concepts are really distinct. They tend to be so closely associated because back-propagation was originally introduced as a model of learning behavior, but it is now well known that it is a rather inefficient means of training a network.

Question: In a neural net, how does the number of observations required relate to the number of hidden layers, the number of nodes, and the number of inputs?

Lo: This is an excellent question, but one for which I don't have a satisfactory answer. As I mentioned before, the specification of network topology is still more of an art than a science, and no one has yet developed an optimal algorithm for choosing the number of hidden layers, nodes per layer, inputs, etc. These choices must rely almost completely on experience and intuition, and undoubtedly will depend heavily on the specific application at hand. What works for foreign exchange rates may be completely inappropriate for mortgage-backed securities. Therefore, a great deal more research needs to be done before your question can be answered satisfactorily.

Question: One of the most important features of neural networks is the ability to deal with "holes" in the data. From a conceptual viewpoint, how do the nets accomplish this?

Lo: Like any other nonparametric nonlinear technique, neural networks deal with holes in the data by a kind of fancy interpolation. For example, if observation 1 of a variable is 5.0 and observation 3 is 7.0, how would you interpolate the value of a missing observation 2? A linear interpolation would simply take the midpoint: 6.0. A nonlinear interpolation is similar but takes into account the curvature of the data around the missing observation. In essence, neural networks fill in holes by the kind of local averaging on which kernel regression is based.