Preparing for Basel II

Common Problems, Practical Solutions

Part 5: Artificial Neural Networks

by Jeffrey S. Morrison

This article continues the series dealing with modeling requirements for Basel II. Previous articles have focused on the problems of missing data, model-building strategies, special challenges in model validation, and time to default. This article looks at a completely different approach to modeling PD (probability of default) and LGD (loss given default)—artificial neural networks.

When credit and repayment relationships are extraordinarily complex, it becomes far more difficult to arrive at the probability of default and loss given default. Previous articles in this series offered a variety of strategies, such as transforming the predictor variables, discretizing and binning the data, constructing dummy variables, including interaction terms, and developing segmentation schemes using CART. [1]

However, it is possible that even these procedures do not go far enough. Developing *interaction variables*, for example, usually requires prior knowledge for the specification to be useful. Techniques such as logistic regression may prove less accurate than methods designed to handle nonlinear data.

An artificial neural network (ANN), often just referred to as a neural net, uses a mathematical approach to represent the processing of information in a fashion similar to the human brain.[2] Originally designed for pattern recognition and classification, ANN also can be used for clustering and prediction applications.

The terminology associated with designing ANNs can be so different from anything we're used to that even professional model-building practitioners can get lost in the nomenclature. In a 1997 paper[3], J. Stuart McMenamin attempts to bridge the gap between traditional econometric methods, such as regression analysis, and the concepts surrounding neural networks. He explains that neural networks are essentially nonlinear models that can approximate a wide variety of data-generating processes. Furthermore, at the heart of most ANNs is a recommended procedure for modeling probability of default—the logistic function.

So let's define some ANN terms by comparing them with

© 2004 by RMA. Jeff Morrison is vice president, Credit Metrics—PRISM Team, at SunTrust Banks, Inc., Atlanta, Georgia.

terms used earlier in this series:

- In regression terminology, we estimate *parameter weights* or *coefficients*. In ANN terminology, these are called *connection strengths*.
- In regression, the parameter estimate for a particular variable is called its *slope coefficient*. In neural network terms, we call it the *tilt* parameter.
- In regression, we have a term for constants—the y-intercept. In neural network terminology, these are called *bias* parameters.

The objective for both processes is to find a set of parameter weights (or connection strengths) that will make the errors as small as possible. In regression, this process tends to be rather direct. In neural networks, however, the solution can be much more iterative and sometimes quite a bit slower.

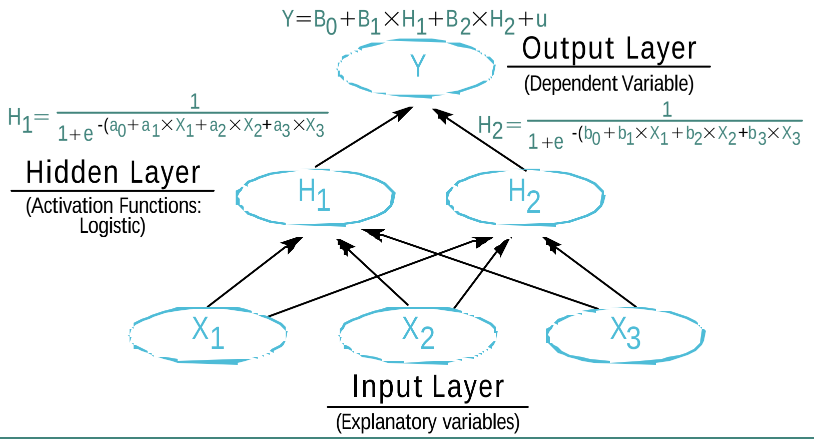
Back-propagation. Although a variety of parameter estimation techniques abound in the industry, *back-propagation* may be most popular. Basically, this is a repetitive process of using the estimated parameters to compute the forecasting errors where an adjustment is made so the next pass of errors is smaller. Back-propagation is an example of *supervised learning* because it requires knowledge of the "correct answers" as it learns the data and finds acceptable parameter estimates.

The Neural Network Structure

Figure 1 shows the most basic type of structure you will find in neural network software packages.

Figure 1

Common Neural Network Structure



This is called a *feed-forward* architecture. In this type of network, information is passed from one layer to the next in a one-directional fashion. No information is channeled laterally or backwards in the network. At the bottom of Figure 1, we see the process begins with the predictive variables at the *input layer*. These variables would be data such as LTV, loan type, debt to income, credit score, and so forth.

Next, we see this information is passed to the *hidden layer* containing the primary workhorses of the neural network—the *activation functions*. This is where the logistic functions or other nonlinear mechanisms reside. For a logistic function, the value is bounded between 0 and 1. Under some conditions, a node will take on a value close to 1, implying a firing or activation condition in a human neuron. Under other conditions, the firing or activation condition is not implied, as the node takes on a value closer to 0.

The more nodes you specify in the hidden layer, the more complex the network. It's typically up to the user to decide how many nodes to include in the hidden layer—that is, how much complexity the network will need to produce the best forecast possible. Usually, two to five nodes in the hidden layer will provide the best results. If too many nodes are included in this layer, the network can "overfit" the data, making it look predictive on training data but result in poor performance when new information is presented.

Finally, we see the process ends at the *output layer* where the information from the previous layer is put together often in a linear fashion—not unlike what we might see in a regression equation.

In addition to the terminology already presented, other terms may be encountered when dealing with neural networks. The term *epoch* represents a network's pass through the data. Many neural networks are designed to make numerous passes through the data to find the most appropriate parameter values. This is called *training* the network. Each time the data is presented, the information may be reordered or randomized. The *learning rate* of a neural net reflects the step size when the weights are adjusted, as might be the case in back propagation. Often it is up to the user to provide input into these parameters—parameters that can dramatically affect the accuracy of the network.

Essential Key Features

Neural network software comes in a variety of flavors, each with its own unique user interface to make things simple while providing the flexibility needed to build good models. Regardless of which software you choose, look carefully at how the package does four things.

1. Data interface. The software you select should have an excellent interface to allow you to look at your data, perform preliminary analysis on it, and get it in the right form for model building.

2. Classification and/or prediction. The software needs to have the proper routines available for building PD and LGD models. If you are building a probability of default model for Basel requirements, then you may be interested in a prediction tool rather than a pure classification tool. Be sure to determine whether the software offers the capability of extracting probability estimates. For LGD modeling, you might want a *general regression neural network* (GRNN). GRNN often is quick to estimate but sometimes a little slow to make predictions on new data. However, a good neural net package should be flexible enough to include all these capabilities.

3. Sensitivity analysis. This feature indicates the model's most important predictor variables. Sensitivity analysis can be done in a variety of ways[4], but because nonlinearity issues surround a neural net model, take care to evaluate each variable. In other words, the impact of each variable may depend not only on its value, but also on the interaction of other variables. So perform this analysis by using graphical plots of the data across the range of input values for each predictor. Sometimes the software automatically performs this type of analysis by removing the variable from the model and examining the impact on the output value. It would be wise to have looked into the sensitivity issue before deployment, as regulators expect you to know what variables play the most important role in your model.

4. Implementation code. Implementing a PD model derived from a logistic regression is relatively simple; the most difficult part is performing a basic exponential function. However, implementing a neural network model can be a complex and difficult task. A good neural net software package should have a deployment engine that automatically generates the code necessary to score new data in languages like C (a programming language. similar to Visual Basic—VB—but much quicker). Deployment software should be able to handle data cleansing, transformations and to perform the appropriate methods to account for missing data. The code also should be written in a manner efficient enough to score the entire portfolio in a timely manner.

Modeling Challenges

Building regression models is both art and science. Designing an effective neural network poses similar challenges. In building a PD or LGD model using a regression technique, you must decide at the very least which variables to include and which to eliminate, and whether to transform the variables or break them up into more discrete predictors. In building a PD or LGD model using a neural net, you still have to decide which input predictors to use, how many nodes should be present in the hidden layer, the number of epochs to run, which learning rate to specify, and sometimes which training routine is appropriate.

It occasionally is possible that the estimation algorithm in a logistic regression model will not "converge" because of one problem or another. However, lack of convergence is obvious because warning or error messages are usually produced. In neural networks, it is possible that the network will converge on a solution that is more local than global—meaning that the process does not find the overall best solution. This is part of the trial-and-error methodology associated with neural nets. Some schools of thought say that rather than a single model, you should build a variety of models that vary the number of nodes in the hidden layer, learning rates, and other parameters needed by the neural net software.

Summary

Neural networks can offer the

analyst a means to build flexible nonlinear models for portfolios with credit behavior that are potentially too complex for traditional techniques such as regression analysis.[5] One advantage is that the network can automatically determine what these nonlinearities may look like—a big plus if the relationships within the data are not well understood beforehand. These networks are generally designed around a three-layer architecture—the input layer, the hidden layer, and the output layer. The primary engine in these networks resides in the hidden layer through activation functions. Although neural networks are by design able to capture nonlinearities in the data, their use is not without human judgment. Since there are many different neural net software packages commercially available—each with its own variety of estimation algorithms, optimization routines, input parameters, and graphical interfaces, software-specific experience is needed to build an effective predictive model.

Once the neural network has been trained and deployed, there remains the problem of explaining its design to both the regulators and the line of business. You must be prepared for such questions as why one account receives a high score while others that appear very similar receive a much lower score. Although a good sensitivity analysis can help answer these questions, you still face the formidable task of explaining a possibly complex model that will inevitably be seen by some as a "black box." Given that modelers using regression approaches are equipped with strategies to approximate nonlinear relationships to some extent, it may be argued that the use of neural networks for PD and LGD estimates is worthwhile only when their incremental predictive value is substantial.[6] Therefore, I recommend using neural networks in the modeling process as part of a champion-challenger strategy where the winner, if it is found to be a neural network, is only deployed given a substantial increase in predictive power. ❐

Contact Morrison by e-mail at Jeff.Morrison@suntrust.com.

Notes

1 Morrison, Jeffrey S., "Preparing for Modeling Requirements in Basel II, Part 2: Modeling Strategies. The RMA Journal, May 2004.

2 ANN was used early on in the defense industry to correctly recognize and classify enemy submarines. The initial concepts behind the approach began to surface in the 1940s, but the exponential growth in computer technology has allowed neural networks to be applied in almost every industry imaginable. In the credit world, perhaps one of the most prominent successes includes fraud prediction using transactional data. Because of the highly technical nature inherent in any discussion of the subject, we will provide only a high level overview of the topic and focus our attention on its value in modeling and implementing requirements for Basel II.

3 McMenamin, J. Stuart, "Why not Phi? A Primer for Neural Networks for Forecasting," Regional Economic Research, Inc., April 1997.

4 Yao, J.T., "Sensitivity Analysis for Data Mining," Department of Computer Science, University of Regina, Canada. Research publication.

5 Atyia, Amir F., "Bankruptcy Prediction for Credit Risk Using Neural Networks: A Survey and New Results," IEEE Transactions of Neural Networks, Vol. 12, No. 4, July 2001.

6 Fedenczuk, Leon L., "To Neural or Not to Neural? SUGI27—Data Mining Techniques," SAS User's Group publication.