Fixed Income Attribution with Minimum Raw Material

This paper proposes the use of machine learning algorithms to model price/yield relationships in fixed income attribution. The advantage of the described technique is that it produces highly accurate and detailed analyses, but without the substantial data and pricing requirements needed by other approaches.

Andrew Colin, Ph.D.
is Director of Fixed Income Research for the StatPro Group plc. He holds a Ph.D. in Mathematics from the University of St. Andrews and has worked or consulted for a wide range of banks and other companies. He also is Adjunct Professor in the Faculty of Business at Queensland University of Technology, Brisbane, where he heads a team researching risk in financial and engineering asset management. He is the author of Fixed Income Attribution *(John Wiley & Sons, 2005).*

OVERVIEW

A central task in fixed income attribution is the calculation of how a change in an instrument's yield affects its performance. Yield changes are driven by movements in the market's yield curve, credit spreads, and other sources of risk, such as liquidity and optionality. As an instrument's yield changes due to changes in market conditions, its price will change and so it will show additional return to that generated by the payment of regular coupons. For instance, if credit spreads contract, the overall yield of a corporate bond will fall and its price will rise, and so extra performance will be generated by the bond's exposure to credit spreads.

In an attribution analysis, changes in yield are typically broken down into changes arising from movements in the reference yield curve (or term structure), changes in credit spreads, and others such as those from liquidity or the presence of embedded options. Changes in the yield curve may further be broken down into shift, twist, and curvature motions, or into other types of movement generated by a principal component analysis; see, for example, Phoa (1998) or Colin (2005).

By measuring yield changes from various sources, the performance due to each source of risk may be measured. These returns may then be aggregated over the entire portfolio and compared to the corresponding returns from the benchmark. In this way, the skills of the portfolio manager at predicting and managing various types of exogenous risk can be measured and assessed.

On a security-level basis, the performance generated by these yield changes can be calculated by pricing the security with a range of intermediate yields. Suppose that the yields at the start and end of the calculation interval are y_0 and y_1, respectively. The change in yield of a security $\delta y = y_1 - y_0$ is given by the sum of the various changes due to the above effects. If we assume for simplicity that all yield changes are due to movements in curve shift δy_{SHIFT}, curve twist, δy_{TWIST}, curve curvature $\delta y_{CURVATURE}$, and credit spread δy_{CREDIT}, then the overall change in yield δy is given by

$$\delta y = \delta y_{SHIFT} + \delta y_{TWIST} + \delta y_{CURVATURE} + \delta y_{CREDIT} \quad (1)$$

To calculate a security's performance due to the changes in yield, we can successively reprice the security by adding in each yield in turn. Denote the time and price of the security at the start and end of the interval as t_0 and t_1, and P_0 and P_1, respectively, and suppose that the price P of the security is a function of t and y. Assuming there are no cash flows or coupons, the overall return of the security over the interval $[t_0, t_1]$ is then given by

$$r_{[t_0,t_1]} = \frac{P(t_1, y_1) - P(t_0, y_0)}{P(t_0, y_0)} \quad (2)$$

This expression can be broken down into a sum of returns arising from various sources. The return generated by accrued interest over time is calculated by examining how the price of the security varies when the yield of the security does not change over the interval; in other words, we use the starting yield at both the start and end of the interval:

$$r^{COUPON}_{[t_0,t_1]} = \frac{P(t_1, y_0) - P(t_0, y_0)}{P(t_0, y_0)} \quad (3)$$

The return due to curve shift can now be calculated by adding the change in yield from this source into the above equation, and subtracting the coupon return out from the expression. This gives

$$r_{[t_0,t_1]}^{SHIFT} = \frac{P\left(t_1, y_0 + \delta y_{SHIFT}\right) - P\left(t_0, y_0\right)}{P\left(t_0, y_0\right)} - r_{[t_0,t_1]}^{COUPON} \quad (4)$$

The returns due to twist, curvature, and credit are calculated in a similar way:

$$r_{[t_0,t_1]}^{TWIST} = \frac{P\left(t_1, y_0 + \delta y_{SHIFT} + \delta y_{TWIST}\right) - P\left(t_0, y_0\right)}{P\left(t_0, y_0\right)} - \left(r_{[t_0,t_1]}^{COUPON} + r_{[t_0,t_1]}^{SHIFT}\right) \quad (5)$$

$$r_{[t_0,t_1]}^{CURVATURE} = \frac{P\left(t_1, y_0 + \delta y_{SHIFT} + \delta y_{TWIST} + \delta y_{CURVATURE}\right) - P\left(t_0, y_0\right)}{P\left(t_0, y_0\right)} - \left(r_{[t_0,t_1]}^{COUPON} + r_{[t_0,t_1]}^{SHIFT} + r_{[t_0,t_1]}^{TWIST}\right) \quad (6)$$

$$r_{[t_0,t_1]}^{CREDIT} = \frac{P\left(t_1, y_0 + \delta y_{SHIFT} + \delta y_{TWIST} + \delta y_{CURVATURE} + \delta y_{CREDIT}\right) - P\left(t_0, y_0\right)}{P\left(t_0, y_0\right)} - \left(r_{[t_0,t_1]}^{COUPON} + r_{[t_0,t_1]}^{SHIFT} + r_{[t_0,t_1]}^{TWIST} + r_{[t_0,t_1]}^{CURVATURE}\right) \quad (7)$$

Equation 7 may be written in the alternative form

$$r_{[t_0,t_1]}^{CREDIT} = \frac{P\left(t_1, y_1\right) - P\left(t_0, y_0\right)}{P\left(t_0, y_0\right)} - \left(r_{[t_0,t_1]}^{COUPON} + r_{[t_0,t_1]}^{SHIFT} + r_{[t_0,t_1]}^{TWIST} + r_{[t_0,t_1]}^{CURVATURE}\right) = r_{[t_0,t_1]} - \left(r_{[t_0,t_1]}^{COUPON} + r_{[t_0,t_1]}^{SHIFT} + r_{[t_0,t_1]}^{TWIST} + r_{[t_0,t_1]}^{CURVATURE}\right) \quad (8)$$

Since we assumed that the yield changes from each of these four sources of risk add up to the total yield change, it follows that

$$r_{[t_0,t_1]} = r_{[t_0,t_1]}^{COUPON} + r_{[t_0,t_1]}^{SHIFT} + r_{[t_0,t_1]}^{TWIST} + r_{[t_0,t_1]}^{CURVATURE} + r_{[t_0,t_1]}^{CREDIT} \quad (9)$$

Performance attribution on this security therefore requires six evaluations of price, one at the start and end of the interval, and four intermediate prices.

Linking Risk to Return

To measure the effect of a change in yield on a bond's price, and hence its performance, we can either price the bond using a standard pricing formula or we can use some measure of price / interest rate sensitivity such as modified duration.

These two approaches typically give results that are very close, but they both have well-documented data issues.

FIRST-PRINCIPLE PRICING

Pricing of a security from first principles via a pricing formula requires that we actually know both a suitable pricing algorithm and the market's pricing conventions. A closed-form pricing formula may not always be available, particularly for instruments with stochastic cash flows such as mortgage-backed securities, or bonds with embedded options.

In addition, first-principle pricing requires that we have available the coupon, maturity date, coupon frequency, and other security-specific information for the bond, such as whether it has a long or short first coupon, pricing conventions, day-count conventions, and local market holidays, as well as relevant information on any embedded options. The costs and effort required to obtain this data are not trivial, particularly where large benchmarks with many thousands of bonds are to be modeled, or for large portfolios of floating-rate notes where coupons may be reset on a monthly basis.

Perturbational Pricing

An alternative approach is to form a Taylor expansion for the price of the security. If the return of the security is already known, we may suppose that a bond's price is a function of time t and yield y so that $P \equiv P\left(y,t\right)$. In this case, we can write

$$\delta P = \frac{\partial P}{\partial t}\delta t + \frac{\partial P}{\partial y}\delta y + \frac{1}{2}\frac{\partial^2 P}{\partial y^2}\delta y^2 + \varepsilon \quad (10)$$

where

$$\varepsilon = O\left(\delta t^2, \delta y^3\right) \approx 0 \qquad (11)$$

Dividing Equation 10 through by P, we now have an expression for the return r of the security. If we identify $\frac{1}{P}\frac{\partial P}{\partial t}$ as yield y $-\frac{1}{P}\frac{\partial P}{\partial y}$, as modified duration MD, and $\frac{1}{P}\frac{\partial^2 P}{\partial y^2}$ as convexity C, then expression Equation 10 reduces to the compact form

$$r = y \cdot \delta t - MD \cdot \delta y + \frac{1}{2} C \cdot \delta y^2 \qquad (12)$$

These quantities y, MD, C are collectively known as the security's risk numbers. If these are available on a daily basis, the perturbational attribution equation, Equation 12, provides a convenient and rapid means to calculate the effect of various changes in yield on performance. No pricing machinery is involved, but a source of risk data is required, which has to be sourced externally.

To summarize, security-level approaches to fixed income attribution currently either require all the information necessary for independent pricing of all securities held from first principles or an external source of risk sensitivity measures. Both techniques require a supply of data for which a performance calculation-oriented back office will probably not have access. For many organizations, the acquisition and integration of this new data is one of the principal obstacles to implementing a detailed fixed income attribution capability.

MODEL-FREE ATTRIBUTION

Reduced to its simplest form, the common feature of both the above approaches - and the main requirement of an attribution analysis - is simply a function that maps changes in yield (and other market-imposed quantities) to performance. This leads us to a third attribution approach, and to the central idea of the paper:

By using machine intelligence algorithms, we can train a universal function approximator to learn the relationship between yield and price, without having to specify the form of this relationship.

In this paper, the algorithm considered is a neural network.

NEURAL NETWORKS

For the purposes of this paper, we define a neural network as an array of connected nodes. Nodes are grouped into separate layers, and a network is organized as an input layer, one or more hidden layers, and an output layer. Here we assume that the output layer is composed of a single node. Information passes one way though the network, from the input layer via the hidden layers to the output layer.

Each node in a given layer is connected to every other node in the next layer via a link, and each link has an associated weight (see Figure 1). The information in each layer can be mapped to an *(m,n)* and *n* the number in the ending layer. In principle a neural network can have many hidden layers, but one is usually sufficient to model most problems.

Every node contains a simple activation function that performs simple calculations based on (i) the strength of the links between that node and other upstream nodes, and (ii) the degree of activation of the upstream nodes (or the network's inputs for nodes lying in the input layer), as shown below, a matrix where m is the number of nodes in the starting layer.

Figure 1 is a diagrammatic chart of a simple neural network with two hidden layers, five input nodes, three hidden nodes, and one output node. There are 15 links between the input and the hidden layer, and three links between the hidden and the output layer.

FEED FORWARD

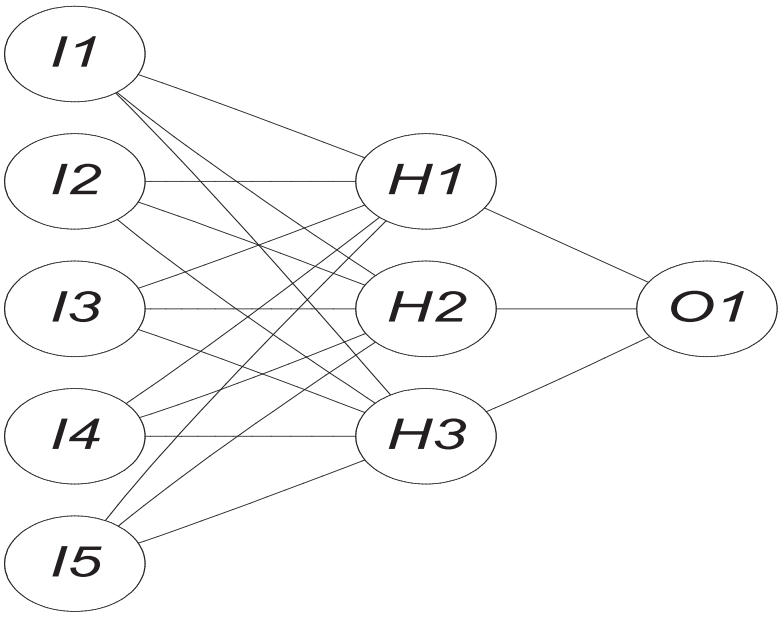
Suppose that each node in the input layer is activated to an externally set value. For instance, we might set the first and last nodes in the input layer to a level of 1, and all the other nodes to a level of -1. The signal is propagated through the network as follows:

First, the activation level of each node in the hidden layer is set to the following expression:

$$a_j^{Hidden} = \sum_i a_i^{Input} w_{i,j} \qquad (13)$$

where

Figure 1: A Diagrammatic Chart of Simple Neural Network



a_j^{Hidden} is the activation level of node *j* in the hidden layer

a_i^{Input} is the activation level of node *i* in the input layer

$w_{i,j}$ is the strength of the link between node *i* in the input layer and node *j* in the hidden layer

At each hidden node, we then transform the activation level using the logistic, or sigmoid function,

$$f(x) = \frac{1}{1+e^{-x}} \tag{14}$$

so that the activation level of node *j* is mapped from the range $(-\infty, \infty)$ to (-1,1).

We then repeat this process in exactly the same way, feeding the resulting signal from the hidden layer to the output layer, so that the initial signal cascades through the network. This is an example of a feed-forward network.

At the end of the feed-forward process, the output node has an activation level that represents the network's response to the initial activation vector.

TRAINING

Various robust algorithms exist to train a neural network to derive a set of weights $\{w_{i,j}^0, w_{i,j}^1, \ldots\}$ to match a set of given input and output vectors. During training, the various network weights are adjusted until the network generates a predefined output level for each input vector as closely as possible, in which case we say the network has learned the training data.

The best-known training algorithm is the back propagation of errors technique, popularized by Rumelhart (1986). Other techniques include simulated annealing (Aarts and Korst (1989)), genetic algorithms (Yao (1999)), and particle swarm optimization (Kennedy (2001)). The results in this paper were derived using a standard back propagation implementation in C.

For simplicity, we will assume just one hidden layer. A network with no hidden layers will be unable to learn training data that is nonlinearly separable, which in practice means that such a network will fail when trained on any but the most trivial of data sets (Minsky and Papert (1969)).

Whichever algorithm is used, the ideal outcome is for

the network to learn all of the data with which it is presented, so that if it is subsequently presented with one of the training samples after training is complete it will produce the same output vector that was originally presented in training. We say that a network has converged if the values of the output vectors, produced by some training algorithm, are close to the corresponding true values. The level to which a network has converged is often shown by the RMS (root mean square) level, which is a measure of the difference between the simulated and the true outputs.

Convergence may not be achieved if;

- there is little or no structure in the training data, in which case there is no implicit model to learn;
- there is a great deal of noise in the training data;
- the network topology has insufficient flexibility to learn the details of a complex relationship.

The first two points are unlikely to be the case when training a network to price a fixed income security, and the last point can be addressed by adding more capacity to the network in the form of extra nodes in the hidden layer.

A network with one single hidden node is implicitly performing linear regression on the model data, and we return to this point in more detail below.

Neural Nets for Attribution

How can such a system help in attribution analysis? A very powerful feature of a feed-forward network is that it can approximate virtually any continuous mathematical function (Hornik *et.al.,* (1989)), as long as a suitable means to derive the network's internal weights exists. The various training algorithms mentioned above do just this.

For attribution, a neural network has two particularly important and desirable features:

1. The first is that neural networks act as model-free function estimators. In other words, given a set of input and output data, it is not necessary to know very much about the internal function being modeled in order to train the network and to model the relationship implied by the supplied data. This makes the use of neural networks particularly suitable for approximate security pricing, where we know the values of the pricing inputs (such as yield, repayment rates, or interest rate volatilities) and the outputs (the market price and returns for the security), but the relationship between the two may not be fully specified analytically.

2. The second is that a feed-forward network will also interpolate data in a sensible way. Unlike a high-degree polynomial, which may model a data set exactly at a set of samples but whose interpolated values may vary widely from the underlying data set, the way that a feed-forward network is designed means that input vectors that do not precisely match an existing input will still result in sensible outputs. The network will use the knowledge abstracted from the various training cases and provide a weighted average of the various cases that is the closest match to the new data.

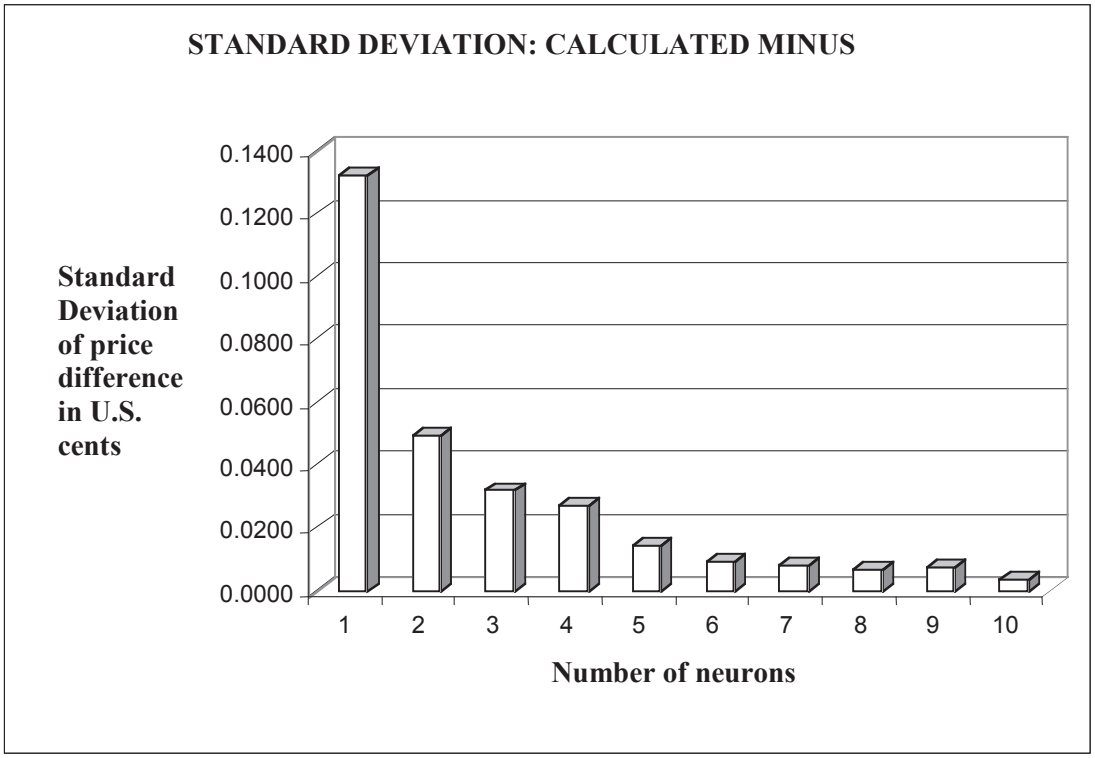
Learning to Price a U.S. Treasury Bond

To demonstrate the technique, we calculated the capital price of a U.S. Treasury bond with coupon and yield varying randomly between 2% and 7%, a maturity date of January 1, 2050, and settlement dates varying randomly between January 1, 2000 and January 1, 2050. The long maturity date was chosen to ensure that the bond showed high convexity, and hence that the yield / price relationship would be nonlinear.

Five hundred prices were calculated and used as training data for a (4, N, 1) neural network where N is the number of hidden neurons, and a further five hundred prices were then calculated by the network on previously unseen data samples. The selected inputs were settlement date, coupon, yield, maturity date and the output was the bond's capital price, quoted using the standard convention per $100 face value, just as for a standard pricing formula. Figure 2 shows the distribution of forecast prices against true prices, and Table 1 presents the same information in tabular form.

The trained network has learned the properties of the pricing equation, for quite general inputs, without any input from the user (apart from selection of input data

Figure 2: Standard Deviation of Calculated Minus True Prices on Unseen Data Sample for Varying Number of Hidden Neurons



and numbers of hidden neurons). In principle, this network could be used to price any security that is priced in the same way as a U.S. Treasury bond.

The case N=1 corresponds to ordinary linear regression. Since the yield / price relationship is nonlinear, we expect this model to produce the least accurate results, and this indeed proves to be the case. As the number of neurons increases, the accuracy of the forecast prices rises rapidly until at N=10 neurons 95% of the predic-

Table 1: Pricing Network

Number of hidden neurons	Standard deviation of difference between true and forecast price, in US cents
1	0.1322
2	0.0498
3	0.0322
4	0.0274
5	0.0148
6	0.0098
7	0.0083
8	0.0073
9	0.0080
10	0.0038

tions are within 0.01 percent of the true price.

There are two points to note from these results:

1. First, the accuracy of the learned results is very high, even on previously unseen data. It therefore appears that the neural network can indeed learn the underlying form of the equation that relates yield to price, without the user having to specify the form of this relation in any way.

2. Second, the accuracy of the results is not high enough for exact pricing. For instance, one could not use neural-network based pricing to produce deal tickets, where exact agreement on prices between counterparties is vital. For this reason, deal prices are usually calculated using standard valuation formulae with highly detailed documentation on usage and rounding conventions.

We contend that this is unimportant for attribution. In attribution analysis we are primarily concerned with achieving a qualitative understanding of the relationships between risk and rewards in a managed portfolio, and for this purpose a discrepancy of a fraction of a percent between risk sources is unlikely to be important, particularly when there is wide variation in the techniques used to decompose yield / curve movements.

Applying These Results in Practice

There are several ways one could apply this algorithm.

Suppose that we have a portfolio of vanilla U.S.D. Treasury and corporate bonds, and a suitable record of past price movements.

(i) If one knows the maturity, coupon, and yield for each security, one could construct a standard bond model using a single network for all securities.

(ii) Suppose now that we only know the security's yield, and that maturity and coupon are not supplied. In this case, we can still use similar machine intelligence techniques to construct a similar model of the relationship between price and yield, but a separate model will have to be constructed and trained for each security in the portfolio.

DISCUSSION

The main benefit of using neural network based attribution is that it is possible to dispense with much of the data and security pricing specification required by both the first-principle and perturbational approaches.

Pricing Formulae

The precise form of the pricing equation need not be specified. From the network's point of view, the relationship is already present in the data. The situation is analogous to teaching a child to catch a ball; the relationships involving perspective, physics, and gravity are learned directly by experience, without any knowledge of Newtonian physics or perspective being required.

We stress that this model-free part of the calculation only applies to the portions of the calculation for which there is a continuous relationship between yield and price. Thus, we calculate capital price rather than market price of the sample bond, since market price includes accrued interest, the value of which is not continuous on coupon payment dates.

Data and Computation

Secondly, we do not need to supply risk numbers in order to perform attribution. By using machine intelligence techniques, we can exchange many of the data maintenance requirements of more traditional approaches for number-crunching power.

CONCLUSIONS

Neural network-based pricing potentially offers a powerful tool for fixed income attribution. Its ability to model pricing relationships without any user-supplied intervention solves one of the major operational difficulties in the implementation of an attribution system – that of accurately modeling the relationship between risk and performance at a security-level basis.

The application of this approach to securities with multiple sources of pricing risk, such as mortgage-backed securities, will form the subject of a separate paper.

REFERENCES

Aarts, Emile, Jan Korst, Simulated Annealing, and Boltzman Machines, Wiley, 1989.

Bollobas, Bela (ed.), Littlewood's Miscellany, Cambridge University Press, 1997.

Colin, Andrew, Fixed Income Attribution, Wiley, 2005.

Hornik, K., Stinchcombe, M., White, H., "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks,* 1989, Vol. 2, No. 5, pp. 359-366.

Kennedy, James, Russell C. Eberhart, Yuhui Shi, Swarm Intelligence, Morgan Kaufmann/Academic Press, 2001.

Minsky, Marvin, Seymour Papert, Perceptrons, MIT Press, 1969.

Phoa, Wesley, Advanced Fixed Income Analytics, Frank J. Fabozzi Associates, 1998.

Rumelhart, David E., McClelland, James L., Parallel Distributed Computing, Volume 1, MIT Press, 1986.

Yao, Xin, "Evolving Artificial Neural Networks, Proceedings of the IEEE," 1999, Vol. 87 No. 9, pp. 1423-1447.

Andrew can be contacted at
andrew.colin@statpro.com.

ENDNOTES

[1] The title of this paper was suggested by the essay, "Mathematics with Minimum Raw Material" by E. J. Littlewood (Bollobas (1997)).

[2] For other instruments such as mortgage-backed and asset-backed securities, sources of exogenous risk include the level of the entire Treasury curve, repayment rates, current-coupon spreads, option-adjusted spreads, and market volatilities. In this case, attribution analysis requires a functional link to be devised between changes in these exogenous variables and the return of the security.