Visual Numerics Technology for Neural Network Analysis

A White Paper

Visual Numerics, Inc.
December 2004

Visual Numerics

Visual Numerics, Inc.
12657 Alcosta Boulevard
Suite 400
San Ramon, CA 94583
USA
www.vni.com

Visual Numerics Technology for Neural Network Analysis

A White Paper

by Edward R. Jones, Ph.D., Visual Numerics, Inc.

Publishing History:

December 2004

TABLE OF CONTENTS

TABLE OF CONTENTS - Continued

Figures

Tables

Neural Network Tools - Background

Many organizations will realize significant time and/or financial benefits by having an accurate forecast or a foreacast that can quickly adapt to the most recent changes in data. The need to generate recurring forecasts arises in many business situations, particularly ones where dependencies occur amoug multiple departaments and decisions are made quickly to keep in step with real-time data and trends. A good example of this is yield analysis in a manufacturing environment or portfolio analysis in a financial institution. The Visual Numerics collection of neural network and data pre- and post-processing algorithms provide a highly configurable neural network algorithm approach, and are provided in a programming language library that facilitates for smooth system integration.

The process of using neural network technology for forecasting requires several key steps, including data preprocessing, training, forecasting, and data postprocessing or validation. Proper preprocessing makes it feasible to train the network, ensures that the training process runs in a reasonable amount of time, and improves the forecasting accuracy of the network. Having appropriate algorithms for preprocessing facilitates the setup process of scaling and converting variables into the appropriate representation to ensure a successful forecasting engine. The availability of these algorithms can avoid the potentially manual and arduous task of preparation, as well as maximize the chance of a successful forecast.

A highly configurable neural net approach creates the ability to tailor it to the time required to train the network as well as the forecasting accuracy requirements of the particular application. The Visual Numerics implementation provides a degree of flexibility that extends beyond what might be found in a more rigid package.

The neural network training process can leverage two types of single-stage trainers or a two-stage epoch trainer. This choice of training options helps reduce training time by allowing the analyst to tailor the process to a particular application. The Visual Numerics offering also includes the ability to provide a user-specified error function, in addition to the typical implementation of sum of squared errors, which provides a very high degree of flexibility to tailor the network to the to improve forecasting accuracy. In addition, Visual Numerics Neural Network solution has the added flexibility of allowing for the creation of network connections that skip layers for a high degree of control over the training time as well as the forecasting accuracy.

Neural Network applications generally fall into the categories of forecasting, classification, and statistical pattern recognition. The Visual Numerics neural network implementation utilizes a multilayer feed-forward neural network, well suited to forecasting as well as classification problems. This neural network algorithm makes an important addition to a large, existing set of algorithms contained within the IMSL Family of Numerical Libraries that are well suited to data mining and predictive modeling.

Visual Numerics' Neural Network Tools

Visual Numerics' neural network routines are divided into two categories: 1) data preprocessing, and 2) neural network routines. The API for these routines is detailed in Visual Numerics Product Development Reference API #02 (2004). Usually, proper data preprocessing is essential to successfully training a neural network. Without proper preprocessing, network training can fail or take an excessively long amount of time.

The objective of data preprocessing is to map the network's input into a format and type required for optimal network training and forecasting. As a minimum, this consists of scaling continuous variables, lagging time series data and mapping nominal and ordinal categorical data into an appropriate numerical representation.

Training a neural network involves estimating the network's optimum weights from a set of training patterns. Training patterns consist of historical data relating network inputs to its outputs or synthetic data constructed to map network inputs to its idealized output. Synthetic training patterns are used to train networks when either historical data does non-existent or unreliable. In some applications, a mixture of both historical and synthetic data might be used for network training. Reliable historical data can be combined with expert opinion or forecasts from well proven theoretical models.

For forecasting applications, the optimum weights are ones that minimize the chosen error calculation. Visual Numerics' network training routine obtains the

optimum weights using either the sum of squared errors or Laplacian error. For classification applications, the cross-entropy error calculation is used to identify the optimum weights, and the number of misclassification errors is used to evaluate the quality of network classifications.

Data Preprocessing Routines

In most real applications, data preprocessing is key to successfully training a neural network. If any of the following conditions exist, data preprocessing should be completed before network training:

1. The input variables include continuous attributes with widely different ranges.
2. A feed-forward neural network is being used to forecast time series values.
3. The network attributes include nominal (classification) data.
4. The network attributes include ordinal data.

Visual Numerics' neural network preprocessing routines consist of the following individual routines designed to make data preprocessing easy.

Table 1. Visual Numerics' Neural Network Preprocessing Routines

Routine	Description
`scale_filter`	Scales continuous input variables so that their values all fall within similar ranges, such as 0-1.
`time_series_filter`	Creates new input variables by lagging time series data
`time_series_class_filter`	Creates new input variables by lagging time series data within classes
`unsupervised_nominal_filter`	Creates new input variables for a nominal variable using 1-to-C encoding
`unsupervised_ordinal_filter`	Creates new input variable for an ordinal variable using cumulative percentages

Preprocessing Continuous Input Attributes

With real word data, continuous attributes often have widely different ranges. One variable might have values running from 0 to 1; while another might have a values from –10,000,000 to +10,000,000. If these attributes are not scaled to have common ranges, then network training may fail to converge or take an excessive amount of time.

As a result, it is common practice to scale all continuous attributes. In some cases, this might even include the network's target outputs. There are several methods of scaling implemented in the routine `scale_filter`.

Table 2. Algorithms for Scaling Continuous Input Variables

Algorithm	Description
1. Bounded Scaling	Scales variable to fall within a fixed range, usually 0-1.
2. Standard unbounded z-score scaling	Standard z-score scaling based upon a mean and standard deviation calculated from the training data
3. Robust unbounded z-score scaling	Robust z-score scaling based upon a median and absolute deviation calculated from the training data
4. Standard bounded z-score scaling	Standard z-score scaling based upon a mean and standard deviation followed by bounded scaling
5. Robust bounded z-score scaling	Robust z-score scaling based upon a median and absolute deviation followed by bounded scaling

The first method, bounded scaling, is probably the most commonly used. Bounded scaling maps the raw values of each input variable, x, into a new variable Z using the following calculation: $Z = r\left(x - \min(x)\right) + a$, where *a* and *b* are the new data limits, $a \le Z \le b$, and $r = \dfrac{b-a}{\max(x) - \min(x)}$. Common choices for the bounds are *a*=0 and *b*=1, or *a* = -1 and *b* = +1.

Method 2, unbounded z-score scaling, is also popular, although it does not guarantee mathematically that Z will fall between definite limits. Z-score scaling consists of the following mapping: $Z = \dfrac{\left(x - m\right)}{s}$,

where m is a measure of the center of the original data and s is a measure of spread for that data. Standard z-score scaling uses $m = \bar{X}$ and $s = \text{standard deviation}$. If the continuous variable is normally distributed, then very few of its z-scores should have values outside the range –6 to +6, i.e., six standard deviations from the mean.

For continuous input attributes that are not normally distributed, some investigators prefer to use robust z-score scaling by using the median and sum of absolute deviations instead of m and s, respectively.

The routine `scale_filter` not only supports these five methods for scaling continuous variables, it also support unscaling for each of these methods. See the Visual Numerics Product Development Reference API #02 (2004) for further details.

Preprocessing Time Series Data

Time series forecasting with neural networks is increasingly popular with investigators looking to improve forecast accuracy, particularly with short series. Time series forecasting can be conducted using either multilayer feed-forward or recurrent neural networks. Of these, the first method is more popular, probably because of the absence of good software for the analysis of recurrent networks. However, this approach requires preprocessing of the time series before passing it to training or forecasting engines.

Even with this requirement, some investigators prefer neural networks to traditional ARIMA approaches. To be effective, ARIMA approaches require a longer stationary time series to produce accurate forecasts. Neural networks using lagged inputs can produce more accurate forecasts from shorter series, and they have the added flexibility fitting non-stationary series without removing non-stationary effects. Accurate forecasts from short time series are possible because neural networks can easily incorporate concomitant variables, including categorical data, in addition to the time series itself.

Today there are two popular methods for time series forecasting using neural networks. One, developed in recent years, is to use a recurrent neural network, see Mandic and Chambers (2001) for a detailed description of recurrent neural network algorithms. The second is to use a multilayered feed-forward neural network with lagged inputs.

Recurrent neural networks do not require lagged inputs. Unscaled or scaled values from the original series are used as input attribute to the series, together with any other exogenous variables. Unlike feed-forward neural networks, recurrent neural networks allow for reverse, or backward flow, in the network computations. The following figure illustrates one example of how an ARMA time series might be modeled using a recurrent neural network.

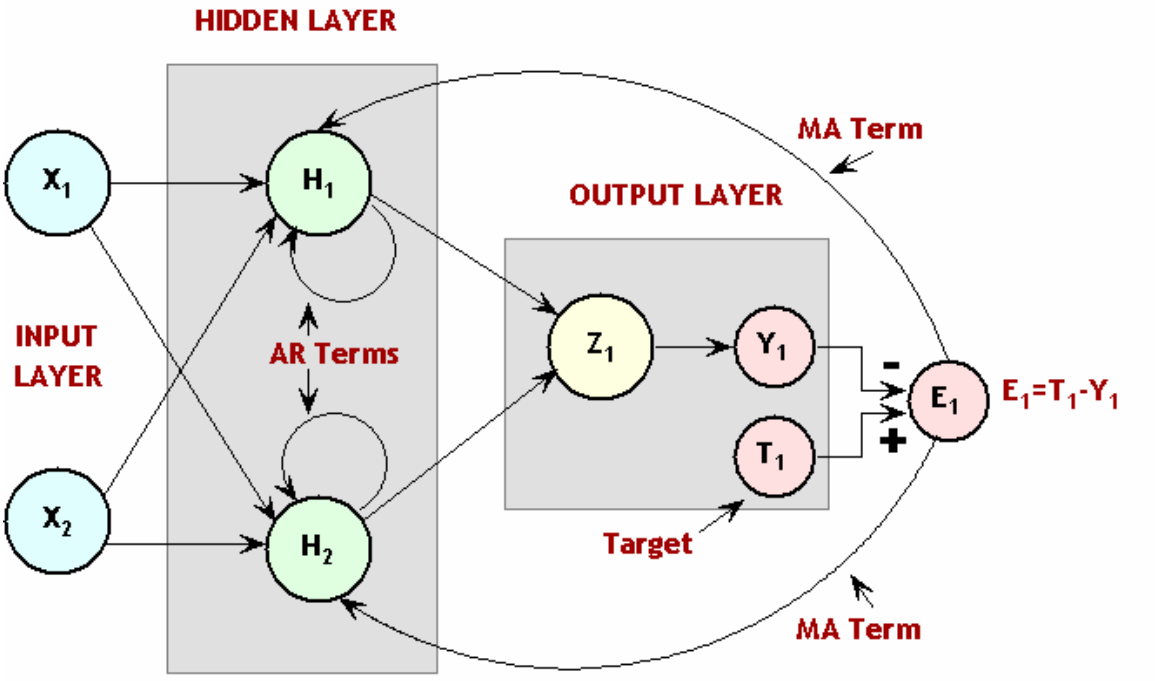


Figure 1. An ARMA Time Series Modeled Using a Recurrent Neural Network

In this recurrent network, there are two input series, represented as X_1 and X_2, and one target variable, represented by T_1 . The network produces Y_1 as an estimate for the target, T_1. Both inputs are transferred into two perceptrons, H_1 and H_2. These two perceptrons have two other inputs. One is from their delayed output, and the second is from an error calculation, represented as $E_1=T_1-Y_1$.

The error calculation is also delayed and estimates the moving average component in the model.

More complex arrangements are possible. This, together with the non-linear activation functions in the perceptrons and the ability to include both continuous and categorical exogenous variables makes recurrent neural networks very flexible and useful for producing accurate time series forecasts.

Another neural network approach to time series forecasting is to adapt a multilayered feed-forward network to the problem. This is done by using lagged values of series as input attributes to the network, along with any exogenous variables. In the following figure, three lags of the series, Y, are used as input attributes.

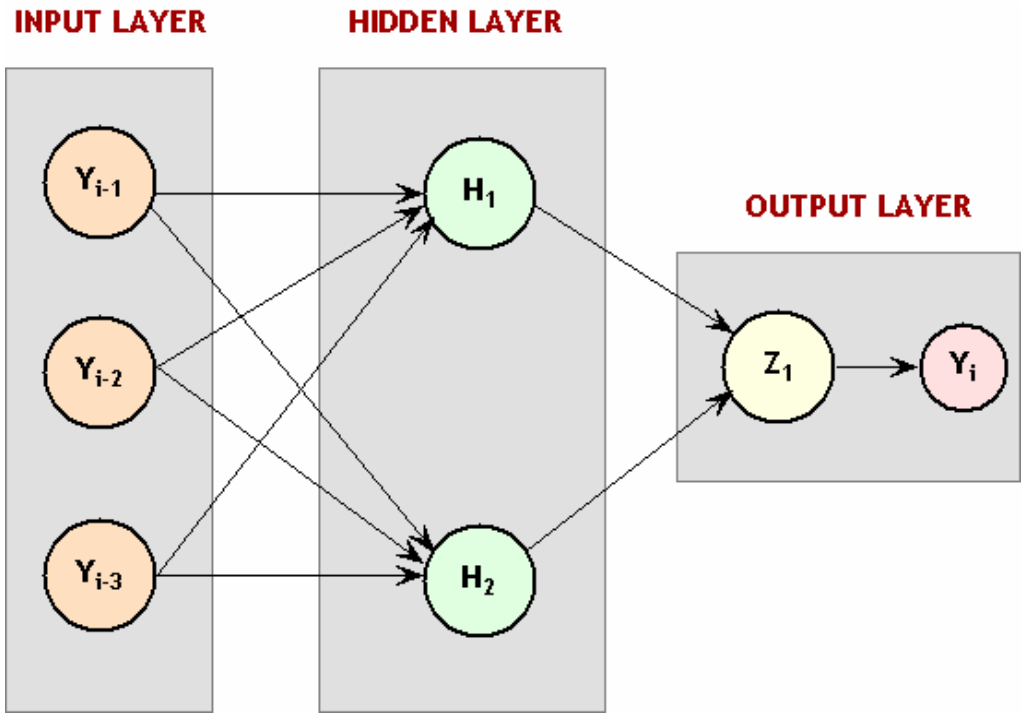


Figure 2. AR(3) Time Series Modeled Using a Neural Network

There are two preprocessing functions that produce lags from a series for input into a multilayer feed-forward network: `time_series_filter` and `time_series_class_filter`. The first routine creates the lags of the series without regard for any sub-classification of the series data. The second creates lagged inputs within designated classes or categories of the series, such as departments, locations or machines.

See the Visual Numerics Product Development Reference API #02 (2004) for further details on these time series preprocessing routines.

Preprocessing Nominal Input Attributes

The last two preprocessing routines, `unsupervised_nominal_filter` and `unsupervised_ordinal_filter`, are for transforming categorical, or classification, variables into numerical attributes. Nominal categorical variables are classifications or categories that have no natural ordering. For example, variables such as gender, color, and season have no natural ordering. Ordinal variables are categorical variable that have a natural ordering such as size, class standing, credit classification, etc.

Nominal variables can also be classified as either binary or multi-class nominal variables. A binary nominal variable is one with only two classifications, such as yes or no. A multi-class nominal variable has three or more classifications. Binary variables can be simply coded as a single column of zeros and ones, with the ones representing one category and the zeros representing the other.

For nominal, multi-class variables, they cannot be represented using a single column of integers. If they were, a neural network would treat the multi-class nominal variable as an ordered continuous variable, which it is not. Instead it is common practice to use 1-of-C encoding for multi-class, nominal variables.

1-of-C encoding maps a single nominal variable with K>2 classifications into K network input attributes. Each of the K classes is represented by its own column of zeros and ones. A value of one represents an occurrence of that particular class, and a zero is indicates that this particular class did not occur.

Consider, for example, a multi-class nominal attribute with 3 classes, denoted as 1, 2 and 3 in the following figure.

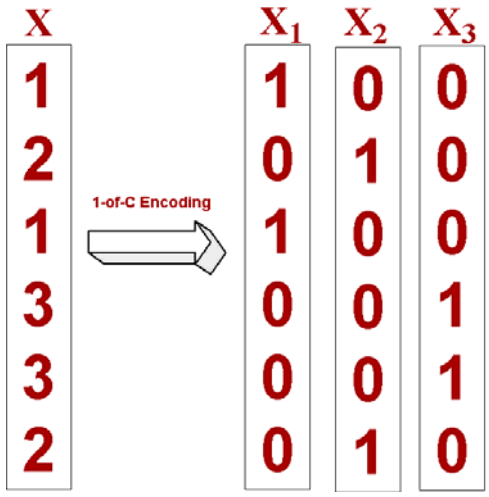


Figure 3. 1-of-C Encoding for a Nominal Variable with 3 Classes

Notice that the classes represented with the values 1, 2 and 3 in the original column for X are converted into three columns, each with zeros and ones. The value one only appears once in each row, in the column used to represent that class. For example, the value 1 in column X_3 appears in the fourth and fifth

positions since the third class value appears in those same positions of the original nominal variable.

The C columns of zeros and ones become input attributes to the network replacing the original column. As a result, when nominal input attributes are used in a network, the number of input attributes can grow quickly.

The columns resulting from 1-of-C encoding are easily produced using the routine `unsupervised_nominal_filter` described in Visual Numerics Product Development Reference API #02 (2004).

Preprocessing Ordinal Input Attributes

Encoding of ordinal classification attributes, such as gender or taste preferences, is done differently. Since ordinal variables are a natural ordering, they can be encoded as a numeric variable, which can then be used as a continuous input attribute to the network.

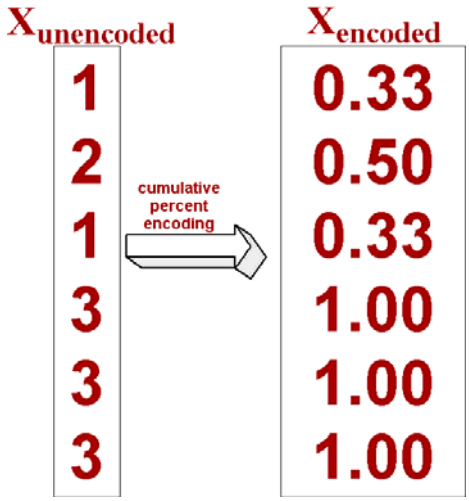


Figure 4. Encoding of Ordinal Variable with 3 Classes into Cumulative Percentages

In this example, the ordinal variable has three ordered categories represented as 1, 2 and 3. In the encoding, each of these values is replaced by the proportion of observations with that value or a lower value. For example, the first category with an original value of 1 is replaced by the proportion of patterns having that same value. Two of the six patterns have this value, 2/6 = 0.33. The second category is replaced by the proportion of patterns with that value or lower, i.e., 2/6+1/6=0.50. The last category is replaced with the proportion of patterns with that value and lower, which is always going to equal to all of the patterns of 6/6=1.0.

Once an ordinal input attribute is encoded to cumulative percentages, the cumulative percentages are used as a continuous input attribute to the network, replacing the original ordinal classifications. Unlike 1 to C encoding for nominal variables, ordinal encoding to cumulative percentages does not increase the number of input attributes to the network.

Cumulative percentage encoding of ordinal variables is implemented in the routine `unsupervised_ordinal_filter`. Details of using this routine and the `unsupervised_nominal_filter` routine are in Visual Numerics Product Development Reference API #02 (2004).

Neural Network Training and Forecasting Engines

Visual Numerics' neural network training and forecasting engines consist of two distinct routines designed for training and forecasting one or more continuous outputs using a multilayer feed-forward neural network with all continuous inputs.

The training engine is designed to estimate the network weights needed by the forecasting engine. Both assume that all input variables are properly scaled or encoded.

Although there is no absolute requirement for scaling continuous inputs to these engines, in most cases, continuous input attributes need to be scaled using the `scale_filter` routine previously described or a similar methodology. The training engine initially places equal weights on all inputs. Unless the continuous variables are scaled over some common range, network training may fail to build a network capable that produces the best forecasts.

Unencoded nominal and ordinal data should not be sent directly to these routines. Nominal and ordinal variables must be encoded before sending them to these engines. The routines `unsupervised_nominal_filter` and `unsupervised_ordinal_filter`, described in the previous section, can be used for this encoding.

Table 3. Visual Numerics' Neural Network Training & Forecasting Routines

Routine	Description
`mlff_training_engine`	Trains a multilayer feed-forward network for forecasting
`mlff_forecasting_engine`	Produces forecasts for a trained multilayer feed-forward network

In general, preprocessing of continuous variables is also necessary before training a neural network. If the network is using time series variables, new continuous variables, created by lagging the time series variables, should be included in the training patterns. This routines `time_series_filter` and

`time_series_class_filter`, described in the previous section, can be used to create these lagged variables.

In addition, continuous variables usually require scaling. Training can be very slow and can fail to produce good forecasts if continuous input variables are not scaled. All continuous variables, including any lagged time series variables, should be scaled before training. Scaling can be done using the `scale_filter` routine described in the previous section.

If lagged time series variables are included in the training patterns, it is easier to scale the original time series variable before creating the lagged columns. There is no restriction on the number of lagged time series columns included in the training patterns. It may be tempting to include a very large number of lagged columns. However, the more columns included in the training patterns, the slower network training. A good rule of thumb is to examine the correlations among the first 20 or more lagged columns. At some point, these correlations normally become very small as the information in the current observation becomes less and less dependent on the past observations. The lag where this occurs should be the highest number of lags.

The training and forecasting engines for multilayer feed-forward neural networks are more complex than the data preprocessing routines because network interconnections must be specified in addition to the training patterns and their corresponding targets. The following table describes the information required to completely describe a neural network's architecture.

Table 4. Neural Network Interconnection Data

Array or Matrix	Description
`Network Counts`	An integer array containing 7 counts: the number of training patterns, the number of input attributes, the number of categorical inputs using 1-of-C encoding, the number of continuous inputs, the number of outputs, the number of network layers, and the total number of perceptrons, including all output perceptrons.
`Perceptron Counts`	An integer array of length M containing the number of perceptrons in each hidden layer, where M=total number of perceptrons including the output perceptrons.
`Perceptron Descriptions`	A two-dimensional integer matrix with M rows and 4 columns. Each row contains the following information for a perceptron: the perceptron identification number (1 through M), its activation function, the number of input and output links for this perceptron.
`Perceptron Input Links`	An integer array containing the perceptron identification numbers (1 through M) associated with each perceptron's input links.
`Perceptron Output Links`	An integer array containing the perceptron identification numbers (1 though M) associated with each perceptron's output links.

These arrays and matrices are always required input to the training and forecasting engines. There are no restrictions on the number of input attributes, number of outputs, number of layers, and number of perceptrons. Unlike most other software, there are also no requirements to use the same activation function for all perceptrons, and to have every perceptron in a layer link to all perceptrons in the following layer. Each perceptron is free to use a different activation function, and can have its own unique set of input and output links.

However, by convention, perceptrons should be numbered sequentially from 1, 2, …, M, where M is the total number of perceptrons, including the output perceptrons.

Figure 5 below is an example of this numbering scheme for a three-layer feed-forward network. Notice that perceptron numbering begins in the first hidden layer and ends with the last perceptron in the output layer.

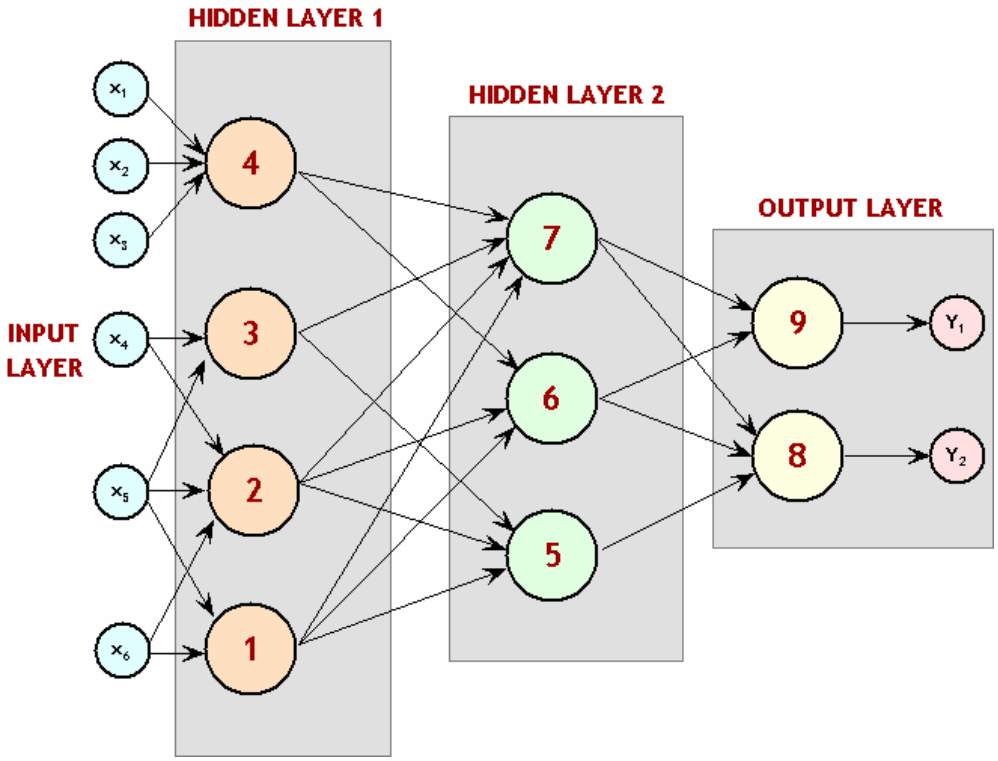


Figure 5. Perceptron Numbering Scheme for a 3-layer Feed-Forward Neural Network

The forecasting routine, `mlff_forecasting_engine`, is designed to work with the `mlff_training_engine`. The forecasting engine uses the same description for describing the interconnections of the trained network. In addition, it accepts, as input, the network weights array returned by the training routine.

However, `mlff_forecasting_engine` also requires an input pattern. This pattern is forecasted by `mlff_forecasting_engine`. The forecasting engine is designed to calculate a single forecast. If multiple forecasts are required, then `mlff_forecasting_engine` must be called once for each input pattern.

It is important that the structure of the input patterns sent to the forecasting engine exactly match the training patterns. The number and order on input variables must be the same used to train the network. In addition, any preprocessing, encoding or scaling done on the training patterns must be applied in an identical fashion to the forecasting input patterns. For example, if a nominal variable with 3 classes is encoded using 1-of-C encoding before training, then that same encoding must be used before calling `mlff_forecasting_engine`. Similarly any lagged time series columns used to train the network, must be included in the forecasting input patterns. Not only must they be included, they must be included in the same order as used during training.

In some cases, the target variable may have been encoded or scaled before network training. This sometimes helps shorten the training time, and in some cases can result in better forecasts. If scaling was applied to the target variable before training, then the inverse of that scaling should be applied to the forecasts produced by `mlff_forecasting_engine` using the `scale_filter` encoding routine.

The Multilayer Feed-Forward Neural Network XML File

The neural network training and forecasting engines described above generally require a good deal of preprocessing training and input patterns, and in some cases post-processing of forecasts. In addition, the network interconnections must be described using the five arrays and matrices described in Table 4.

These follow the specific structure described in Visual Numerics Product Development Reference API #02 (2004). In order to use the neural network training and forecasting engines, an application must be written to assemble the training and forecasting patterns required by these engines. These applications will normally make use of the preprocessing routines listed in Table 1.

Once the data are preprocessed, information about the network structure must be assembled. A preferred method for assembling this information is to place a description of the neural network's structure in an XML file. After training, this file is updated to contain the optimum weights for the network. This produces a single file that contains both a structural description of the network and the results of network training. This also provides a uniform data format that is easily viewed in current web browsers.

Visual Numerics uses an XML schema (.xsd file) to aid in preparing this XML neural network description file that describes both network structural information as well as the results of network training, if completed. This schema is recommended for storing neural network structural information and the results of network training.

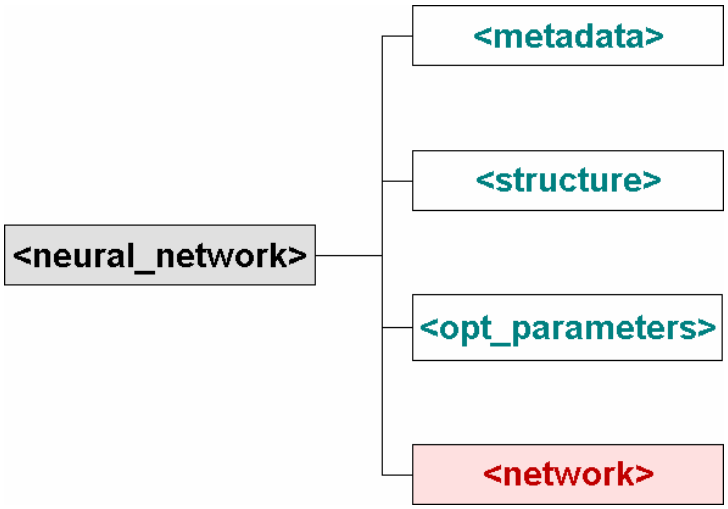
The neural network training and forecasting engines require a description of the network's architecture as well as a description of the training or forecasting patterns. XML is a convenient format for storing this information into a single file for later retrieval. The XML schema described in Appendices A and B,

`NeuralNetSchema.xsd`, is supplied as a suggested format for storing this information.

Keeping the XML description separate from the training patterns makes it convenient to retrain a network without having to re-enter a new description of the neural network structure. Network retraining only requires rerunning the training engine with a new set of training patterns. The structural description of the network is maintained in the XML neural network description file. Retraining updates the weights in this file.

The root element for the XML neural network description file is <neural_network>. The required structure consists of four XML elements, one of which is optional.

Figure 6. Four Primary Neural Network XML Sections



Each of these sections has a specific purpose as described in the following table.

Table 5. Neural Network XML Structure

Section	Description
`<metadata>`	A required section used for documenting the context of this network. This consists of the network name, title, description, trainer and training date.
`<structure>`	A required section that describes the structure of the input layer, the hidden layers and the output layer.
`<opt_parameters>`	A required section with several attributes used to tailor the network training algorithm. There are 8 attributes for controlling training optimization, such as epoch size and whether both Stage I and II training should be used.
`<network>`	This section is optional for network training, but required for forecasting. This contains the network weights obtained during training. This section is completed using the network weights returned from the training engine.

The `<metadata>` Section

The first section, `<metadata>`, gives a minimal description of the neural network application. This consists of the following information:

1. The name given to this neural network (required).
2. The title assigned this network (optional).
3. A brief description for this network (optional).
4. The name of the person training the network (optional).
5. The date the network was trained (optional).

Only the name assigned to this network is a required sub-element within the `<metadata>` section. However, often the researcher will want to include most of this information to assist in distinguishing between one training session and another.

The <structure> Section

The <structure> section describes the network's inputs, hidden layers and outputs. The <structure> element has three attributes:

1. numberOfPerceptrons – the total number of network perceptrons, including the output perceptrons.
2. numberOfAttributes – the total number of network inputs, including both continuous and categorical inputs.
3. numberOfLayers – the number of hidden layers plus one for the output layer.

The first two attributes are required. The last attribute, numberOfLayers, is optional with a default value of 1 (no hidden layers). There are no restrictions on the number of network inputs, number of hidden layers or the number of network outputs.

The <structure> section is further divided into the three sub-elements depicted in Figure 7 and described in Table 6 below.

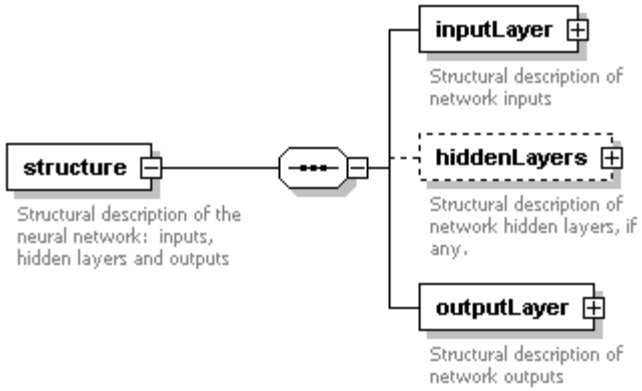


Figure 7. The Three Sub-Elements of the <structure> Section

Table 6. The <structure> Sub-Elements

Element	Sub-Elements	Description
`<inputLayer>`	`<inputNode>`	A required section describing the network inputs and their connections to network perceptrons.
`<hiddenLayers>`	`<layer>`	An optional section that describes the structure of the hidden layer(s), if any. This includes a description of each perceptron by layer, and its input and output connections to other perceptrons or network inputs
`<outputLayer>`	`<output>`	A required section describing the network outputs. Each output is described as a perceptron. Its description includes each perceptrons activation and input connections from other layers or network inputs.

Note that since the number of hidden layers is unrestricted, the number of `<layer>` sub-elements corresponds to the number of hidden layers in the network. Each `<layer>` sub-element describes the perceptrons in that layer, either a hidden layer or the output layer. This description consists of the perceptron ID, activation function, and the perceptron's input and output links.

The <opt_parameters> Section

The third first-level section is an empty element, the `<opt_parameters>` element. This element is required since it is used to customize the training algorithm. There are two major training algorithms that can be used individually or together in network training.

`<opt_parameters>` has no child elements, but it does have the following ten optional attributes.

Table 7. Attributes of the <opt_parameters> Element

Attribute	Default Value	Description
`stageII`	YES	Indicates whether Stage II optimization should be used after Stage I. Allowed values are "YES" and "NO". A value of "NO" causes Stage II optimization to be bypassed.
`maxIterations`	1000	The maximum number of optimization iterations during Stage II optimization.
`maxFunctionEval`	1000	The maximum number of function evaluations allowed during Stage II optimization
`functionTol`	10^{-11}	One of three optimization stopping criteria. If two consecutive optimization iterations have function differences smaller than this value, optimization is halted.
`gradientTol`	10^{-11}	One of three optimization stopping criteria. If two consecutive optimization iterations have a difference between their gradients smaller than this value, optimization is halted.
`accuracy`	10^{-1}	One of three optimization stopping criteria. If the optimization function falls below this value, optimization is halted.
`maxStep`	1	A parameter that controls the largest step size from one iteration to another. A smaller value slows optimization. Too large a value can cause the training algorithm to step over an optimum.
`n_epochs`	1	The number of epochs conducted during Stage I optimization.
`epoch_size`	500	The number of observations randomly selected during each epoch. This value must be greater than zero and less than or equal to 500.
`trace`	NO	This control whether iteration trace information is displayed during optimization. Allowed values are "YES" and "NO".

An epoch is a single Stage I optimization. Stage I optimization is a form of the standard backward propagation optimization algorithm, see Bishop (1995). During Stage I, a steepest descent algorithm uses the gradient calculated by backward propagation to locate the optimum set of network weights. A single optimization using this method is referred to as an epoch, and frequently results in finding a local optimum instead of a global, or near-global, optimum.

To ensure that a global optimum is found, several epochs are normally executed. Before each epoch the initial weights are randomly selected by adding and

subtracting small random values to the initial weights submitted to the training engine.

In addition, during each epoch a random sample of training patterns is randomly selected to find an optimum network. The number of training patterns selected for each training session is controlled by the `epoch_size` attribute of the `<opt_parameters>` element. The number of epochs is equal to the value of the `n_epochs` attribute. The optimum network found in this search is reported as the optimum for Stage I optimization.

Stage II optimization takes the optimum weights from Stage I and tries to improve upon them using the quasi-Newton optimization algorithm. This is slower than backward propagation, but usually improves the solution found during Stage I. As a result, Stage II optimization is sometimes bypassed when the total number of available training patterns is very large. By default, Stage II optimization is automatically invoked. However, it can be avoided by setting the `stageII` attribute of the `<opt_parameters>` element to `"NO"`.

None of the attributes in `<opt_parameters>` are required, but the `<opt_parameters>` element is required. Acceptance of its default values is indicated by including the statement `<opt_parameters />` in the XML file just before the <network> section.

The <network> Section

During network training, the <network> section is optional and any entries in that section are ignored. In fact, the numerical goal of network training is to complete this section. If present, this section contains the values of the network weights estimated during training. Since these weights are required input to the forecasting engine, the <network> section is required for network forecasting applications. See Appendix C for a small example of a neural network XML file that includes a <network> section.

The Neural Network Training Application

The neural network engines described previously require numerical inputs with a specific structure. Training patterns with nominal or ordinal values must be converted to appropriate numerical values before processing by the neural network engines. If they are being used for time series forecasting then the time series data must be lagged before they can be used as input to the neural network engines. In addition, most continuous input attributes should be scaled to ranges such as –1 to 1 or 0 to 1 before using them as input to the neural network engines. These operations are referred to as data preprocessing. Although there are several routines available for data preprocessing, none of this is done automatically by the neural net training and forecasting engines. Likewise, if the network outputs were scaled, the engines will produce forecasts in the scaled output units. These must be un-scaled before they are reported or compared directly to the raw target values.

In addition to data preprocessing, the neural network engines require a structural description of the neural network's architecture as five arrays and matrices, see Table 3 for a description. The neural network engines do not accept an XML neural network description as input. If an XML neural network description is available, it must be parsed and translated into the input formats expected by the neural network engines.

Finally, the quality of the trained network should be evaluated or validated. Again, this is not a direct feature of the training and forecasting engines.

The neural network training application integrates the neural network data-mining process into four steps:

1. data preprocessing,
2. network training,
3. network forecasting, and
4. data post processing.

The following figure illustrates these individual steps. The process begins with an extraction of training patterns from a database. Typically they must undergo preprocessing, the conversion of raw data values into encoded network inputs and scaling. Data preprocessing is conducted automatically by both the training and forecasting applications based upon the description in the XML neural network description file.

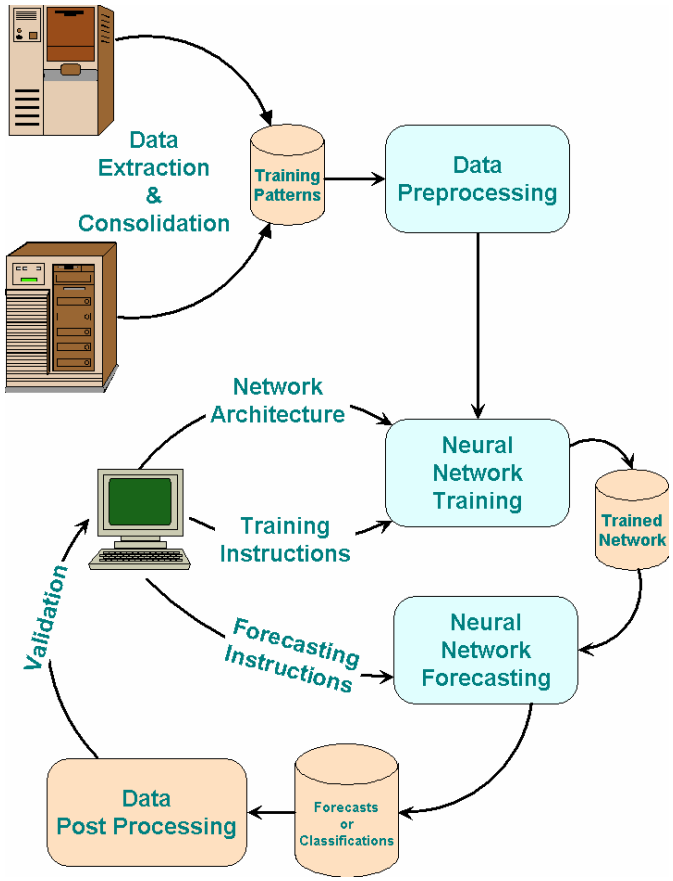


Figure 8. The Neural Network Data Analysis Process

Once the training patterns are extracted and undergo preprocessing then network training can begin. Training a neural network can take hours, depending upon the complexity of the network and number of training patterns. The primary output from the training is a revised XML description file containing the network's weights that were calculated during the training process. Other output includes information on the training process that allows users to monitor the process and determine whether additional training might be necessary.

The neural network training application requires two input files:

1. a text file containing the training patterns, and
2. an XML file following the schema format described previously.

Both of these files can use any name since this application prompts the user for both file names. See Appendix D for a description of the input format for the training patterns.

The training application writes two log files to the `Log` directory created during installation. A processing file with the label `trainingLog.txt` and another containing any XML parse errors, `trainingXmlErrorFile.txt`.

The Neural Network Forecasting Application

The neural network forecasting application operates similar to the training applications. Both applications require an XML neural network description file. However, although the weights described in the `<network>` sections of this file are optional for the training application, they are required for the forecasting applications.

The purpose of the forecasting application is to produce one or more forecasts using the XML neural network description file and a text file delimited with tabs or commas containing forecast patterns. Forecast patterns are similar to the training patterns used by the training application with one exception: each forecast pattern does not require a value for the associated output target. If one

is provided, however, the forecasting application calculates a residual for this forecast pattern.

Similar to the network training application, the neural network forecasting application integrates the forecasting process into four steps:

1. network configuration
2. data preprocessing,
3. forecasting, and
4. forecast un-scaling.

The forecasting application automatically processes all four steps using the information contained in its XML file and forecasting patterns file. The structure of the forecasting patterns file is identical to the training patterns file described in Appendix D. The first line of the file must contain the number of input strings in the forecasting patterns, and the second contains a series of integers, one for each input string. If the integer is a zero, then the forecasting application will ignore the contents of that string. If it is greater than zero then that number should correspond to one of the attribute description identification numbers found in the XML neural network description file.

Any missing values should be indicated using a null string “”. If a forecasting pattern has any missing values for the network’s input attributes then a forecast for that pattern is cannot be calculated.

The forecasting application calculates a forecast for every pattern in the forecast file that does not contain any missing values. These forecasts are written to a text file with the default name `forecasts.txt` in the same directory as the forecasting patterns. It also writes a process log to a file with the name `forecastLog.txt` and writes any errors to `forecastXmlErrorFile.txt` in the `Log` directory created during installation of this application.

Summary

This white paper presented an overview of the Visual Numerics' technology for analysis of neural networks. This technology can be categorized into the following three components:

1. An XML schema for describing a neural network's architecture,
2. A collection of simple routines for building network training and forecasting applications, and
3. A training and forecasting application for forecasting using multilayer feed-forward neural networks.

The XML schema is formally described in Appendices A, B and C. Conceptually this schema is the glue that binds together the neural network data-mining process, depicted in Figure 7. The schema describes the contextual information for the network, the input attributes and their scaling, the network interconnections and the network weights. This schema is required input to any network training or forecasting application.

The purpose of the network training application is to preprocess the training patterns, calculate the weights for the network based upon these patterns and then write those weights into the XML file describing the network. This process can take hours.

The network forecasting applications, on the other hand, usually completes its task in seconds or minutes. It accepts as input the XML created during training as well as a forecasting pattern file. It calculates a forecast for every pattern in this file and writes that information to another text file.

References

Abe, S. (2001) *Pattern Classification: Neuro-Fuzzy Methods and their Comparison,* Springer-Verlag.

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

Bridle, J. S. (1990) Probabilistic Interpretation of Feedforward Classification Network Outputs, with relationships to statistical pattern recognition, in F. Fogelman Soulie and J. Herault (Eds.), *Neuralcomputing: Algorithms, Architectures and Applications*, Springer-Verlag, 227-236.

Bishop, C. M. (1995) *Neural Networks for Pattern Recognition*, Oxford University Press.

Box, G. E. P. and Jenkins, G. M. (1970) Time Series Analysis: Forecasting and Control, Holden-Day, Inc.

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984) *Classification and Regression Trees*, Chapman & Hall. For the latest information on CART visit http://www.salford-systems.com/index.html.

Calvo, R. A. (2001) Classifying Financial News with Neural Networks, *Proceedings of the 6th Australasian Document Computing Symposium*.

Elman, J. L. (1990) Finding Structure in Time, *Cognitive Science,* 14, 179-211.

Giudici, P. (2003) *Applied Data Mining: Statistical Methods for Business and Industry*, John Wiley & Sons, Inc.

Hebb, D. O. (1949) *The Organization of Behaviour: A Neuropsychological Theory*, John Wiley.

Hopfield, J. J. (1987) Learning Algorithms and Probability Distributions in Feed-Forward and Feed-Back Networks, *Proceedings of the National Academy of Sciences*, 84, 8429-8433.

Hutchinson, J. M. (1994) *A Radial Basis Function Approach to Financial Timer Series Analysis*, Ph.D. dissertation, Massachusetts Institute of Technology.

Hwang, J. T. G. and Ding, A. A. (1997) Prediction Intervals for Artificial Neural Networks, *Journal of the American Statistical Society*, 92(438) 748-757.

Jacobs, R. A., Jorday, M. I., Nowlan, S. J., and Hinton, G. E. (1991) Adaptive Mixtures of Local Experts, *Neural Computation*, 3(1), 79-87.

Jones, E. R. (2004) An Introduction to Neural Network Analysis, Visual Numerics, Inc. White Paper available at http://www.vni.com/company/whitepapers/ .

Kohonen, T. (1995) *Self-Organizing Maps*, Springer-Verlag.

Lawrence, S., Giles, C. L, Tsoi, A. C., Back, A. D. (1997) Face Recognition: A Convolutional Neural Network Approach, *IEEE Transactions on Neural Networks, Special Issue on Neural Networks and Pattern Recognition*, 8(1), 98-113.

Li, L. K. (1992) Approximation Theory and Recurrent Networks, Proc. Int. Joint Conf. On Neural Networks, vol. II, 266-271.

Lippmann, R. P. (1989) Review of Neural Networks for Speech Recognition, Neural Computation, I, 1-38.

Loh, W.-Y. and Shih, Y.-S. (1997) Split Selection Methods for Classification Trees, *Statistica Sinica*, 7, 815-840. For information on the latest version of QUEST see http://www.stat.wisc.edu/~loh/quest.html.

Mandic, D. P. and Chambers, J. A. (2001) *Recurrent Neural Networks for Prediction*, John Wiley & Sons, LTD.

Manning, C. D. and Schütze, H. (1999) *Foundations of Statistical Natural Language Processing,* MIT Press.

McCulloch, W. S. and Pitts, W. (1943) A Logical Calculus for Ideas Imminent in Nervous Activity, *Bulletin of Mathematical Biophysics*, 5, 115-133.

Pao, Y. (1989) *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley Publishing.

Poli, I. and Jones, R. D. (1994) A Neural Net Model for Prediction, *Journal of the American Statistical Society*, 89(425) 117-121.

Quinlan, J. R. (1993). C4.5 Programs for Machine Learning, Morgan Kaufmann. For the latest information on Quinlan's algorithms see http://www.rulequest.com/.

Reed, R. D. and Marks, R. J. II (1999) *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, The MIT Press, Cambridge, MA.

Ripley, B. D. (1994) Neural Networks and Related Methods for Classification, *Journal of the Royal Statistical Society B*, 56(3), 409-456.

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

Rosenblatt, F. (1958) The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, *Psychol. Rev.*, 65, 386-408.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986) Learning Representations by Back-Propagating Errors, *Nature*, 323, 533-536.

Rumelhart, D. E. and McClelland, J. L. eds. (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1, 318-362, MIT Press.

Smith, M. (1993) *Neural Networks for Statistical Modeling*, New York: Van Nostrand Reinhold.

Studenmund, A. H. (1992) Using Economics: A Practical Guide, New York: Harper Collins.

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

Tesauro, G. (1990) Neurogammon Wins Computer Olympiad, *Neural Computation*, 1, 321-323.

Warner, B. and Misra, M. (1996) Understanding Neural Networks as Statistical Tools, *The American Statistician*, 50(4) 284-293.

Werbos, P. (1974) *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science*, PhD thesis, Harvard University, Cambridge, MA.

Werbos, P. (1990) Backpropagation Through Time: What It Does and How to do It, *Proc. IEEE*, 78, 1550-1560.

Williams, R. J. and Zipser, D. (1989) A Learning Algorithm for Continuously Running Fully Recurrent Neural Networks, *Neural Computation,* 1, 270-280.

Witten, I. H. and Frank, E. (2000) *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann Publishers.

Wu, S-I (1995) Mirroring Our Thought Processes, *IEEE Potentials*, 14, 36-41.

Visual Numerics Product Development Reference API #02 (2004) Neural Network Analysis: API to neural network analysis.

Appendix A: A Description of the Neural Network Schema

<neural_network>	Type: Root Element	Required: Yes
Children	<metadata> (required one) <structure> (required one) <opt_parameters> (required one) <network> (optional for training, required for forecasting)	
Attributes	None	
Description	Root element of the neural network XML file	

<metadata>	Type: String element neural_network/metadata	Required: Yes
Children	<name> (required) <date> (optional) <trainer> (optional) <title> (optional) <description> (optional)	
Attributes	None	
Description	Information describing the training network	
Example	<metadata> <name>budget_network</name> </metadata>	

<name>	Type: String element neural_network/metadata/name	Required: Yes
Children	None	
Attributes	None	
Description	A name of this network	
Example	<name>budget_forecast</name>	

<title>	Type: String element neural_network/metadata/title	Required: No
Children	None	
Attributes	None	
Description	The title of this network	
Example	<title>Network for Budget Forecasting</title>	

<description>	Type: String element neural_network/metadata/description	Required: No
Children	None	
Attributes	None	
Description	A description of the network	
Example	<description> A 3-layer feed-forward neural network </description>	

<trainer>	Type: String element neural_network/metadata/trainer	Required: No
Children	None	
Attributes	None	
Description	The name of the person responsible for network training	
Example	<trainer>John Doe</trainer>	

<date>	Type: Date element neural_network/metadata/date	Required: No
Children	None	
Attributes	None	
Description	The date the network was trained	
Example	<date>07-16-2004</date>	

<structure>	Type: Element neural_network/structure	Required: Yes
Children	<inputLayer> (require one) <hiddenLayers> (optional) <outputLayer> (require one)	
Attributes	**numberOfLayers**: optional positive integer equal to the number of hidden layers plus one. **numberOfPerceptrons**: required positive integer equal to the total number of perceptron, including the output perceptrons **numberOfAttributes**: required positive integer equal to the number of network input, both continuous and categorical.	
Description	Information describing the inputs, outputs and hidden layer configuration and connections	
Example	<structure numberOfLayers=2 numberOfPerceptrons=4 numberOfAttributes=3> <inputLayer> … </inputLayer> <hiddenLayers> … </hiddenLayers> <outputLayer> … </outputLayer> </structure>	

<inputLayer>	Type: Element neural_network/structure/inputLayer	Required: Yes
Children	<inputNode> (require one)	
Attributes	**n_inputs**: The number of inputs described in the xml file. This is less than or equal to the number of actual network inputs. It is less than the number of actual inputs if some of the inputs are nominal.	
Description	Information describing the inputs, outputs and hidden layer configuration and connections	
Example	<inputLayer n_inputs=1> <inputNode id="2" label="income" type="continuous"> <attributeDescription>Monthly Income</attributeDescription> <perceptronLink hiddenLayer="1" perceptron="1"/> <perceptronLink hiddenLayer="1" perceptron="2"/> <perceptronLink hiddenLayer="1" perceptron="3"/> <continuousAttribute scale="bounded" continuousType="input"> <realLimits lowerRealLimit=0 upperRealLimit=1000000 lowerTargetLimit=0 upperTargetLimit=1 /> </continuousAttribute> </inputNode> </inputLayer>	

<inputNode>	Type: Element neural_network/structure/inputLayer/inputNode	Required: Yes
Children	<attributeDescription> (optional) <perceptronLink> (require one for every link) <ordinalClass> (optional) <nominalClass> (optional) <continuousAttribute> (optional)	
Attributes	**id**: The id number associated with this input attribute. This must be a number from 1 to n_inputs. **label**: The label associated with this input. **type**: The data type associated with this input. Choices are ordinal, nominal, continuous and time.	
Description	Information describing one input attribute, its data type and linkage to perceptrons.	
Example	<inputNode id="2" label="income" type="continuous"> <attributeDescription>Monthly Income</attributeDescription> <perceptronLink hiddenLayer="1" perceptron="1"/> <perceptronLink hiddenLayer="1" perceptron="2"/> <perceptronLink hiddenLayer="1" perceptron="3"/> <continuousAttribute scale="bounded" continuousType="input"> <realLimits lowerRealLimit=0 upperRealLimit=1000000 lowerTargetLimit=0 upperTargetLimit=1 /> </continuousAttribute> </inputNode>	

<hiddenLayers>	Type: Element neural_network/structure/hiddenLayers	Required: No
Children	<layer> (require one or more)	
Attributes	**n_layers**: The number of hidden layers. This must be greater than zero. **n_nodes**: The number of perceptrons across all hidden layers. This does not include the number of output perceptrons.	
Description	This is the Element that groups all hidden layer descriptions, if they are present.	
Example	<hiddenLayers n_layers=1 n_nodes=2> <layer hiddenLayer=1 n_perceptrons=2> <perceptron perceptronID=1 activation="logistic"> <perceptronLinks n_inLinks=3 n_outLinks=2> <inLink inNode=1 nHiddenLayer=0 /> <inLink inNode=2 nHiddenLayer=0 /> <inLink inNode=3 nHiddenLayer=0 /> <outLink outNode=5 nHiddenLayer=2 /> <outLink outNode=6 nHiddenLayer=2 /> </perceptronLinks> </perceptron> <perceptron perceptronID=2 activation="logistic"> <perceptronLinks n_inLinks=3 n_outLinks=2> <inLink inNode=1 nHiddenLayer=0 /> <inLink inNode=2 nHiddenLayer=0 /> <inLink inNode=3 nHiddenLayer=0 /> <outLink outNode=5 nHiddenLayer=2 /> <outLink outNode=6 nHiddenLayer=2 /> </perceptronLinks> </perceptron> </layer> </hiddenLayers>	

<layer>	Type: Element neural_network/structure/hiddenLayers/layer	Required: Yes
Children	<perceptron> (requires one or more)	
Attributes	**hiddenLayer**: The identification number for this hidden layer. Must range from 1 to n_layers. **n_perceptrons**: The number of perceptrons in this hidden layer.	
Description	This Element groups information on one perceptron for this particular hidden layer.	
Example	<layer hiddenLayer=1 n_perceptrons=2> <perceptron perceptronID=1 activation="logistic"> <perceptronLinks n_inLinks=3 n_outLinks=2> <inLink inNode=1 nHiddenLayer=0 /> <inLink inNode=2 nHiddenLayer=0 /> <inLink inNode=3 nHiddenLayer=0 /> <outLink outNode=5 nHiddenLayer=2 /> <outLink outNode=6 nHiddenLayer=2 /> </perceptronLinks> </perceptron> <perceptron perceptronID=2 activation="logistic"> <perceptronLinks n_inLinks=3 n_outLinks=2> <inLink inNode=1 nHiddenLayer=0 /> <inLink inNode=2 nHiddenLayer=0 /> <inLink inNode=3 nHiddenLayer=0 /> <outLink outNode=5 nHiddenLayer=2 /> <outLink outNode=6 nHiddenLayer=2 /> </perceptronLinks> </perceptron> </layer>	

<perceptron>	Type: Element neural network/structure/hiddenLayers/ layer/perceptron	Required: Yes
Children	<perceptronLinks> (required one)	
Attributes	**perceptronID**: The identification number for this perceptron. This must be an integer from 1 to n_nodes **activation**: The activation function for this perceptron	
Description	This Element groups linkage information for one perceptron in this hidden layer.	
Example	<perceptron perceptronID=1 activation="logistic"> <perceptronLinks n_inLinks=3 n_outLinks=2> <inLink inNode=1 nHiddenLayer=0 /> <inLink inNode=2 nHiddenLayer=0 /> <inLink inNode=3 nHiddenLayer=0 /> <outLink outNode=5 nHiddenLayer=2 /> <outLink outNode=6 nHiddenLayer=2 /> </perceptronLinks> </perceptron>	

<perceptronLinks>	Type: Element neural network/structure/hiddenLayers/ layer/perceptron/perceptronLinks	Required: Yes
Children	<inLink> (required one for each input link) <outLink> (required one for each output link)	
Attributes	**n_inLinks**: The total number of input links to this perceptron. **n_outlinks**: The total number of output links to this perceptron.	
Description	This Element groups the linkage information for this perceptron.	
Example	<perceptronLinks n_inLinks=3 n_outLinks=2> <inLink inNode=1 nHiddenLayer=0 /> <inLink inNode=2 nHiddenLayer=0 /> <inLink inNode=3 nHiddenLayer=0 /> <outLink outNode=5 nHiddenLayer=2 /> <outLink outNode=6 nHiddenLayer=2 /> </perceptronLinks>	

<inLink>	Type: Element neural_network/structure/hiddenLayers/ layer/perceptron/perceptronLinks/inLink	Required: Yes
Children	None	
Attributes	**inNode**: The perceptron or input ID for this input. **nHiddenLayer**: The ID associated with the hidden layer that contains this input. If the input is coming from the input layer, set nHiddenLayer=0. Otherwise this will be an integer from 1 to n layers.	
Description	This Element describes one input link to this perceptron	
Example	<inLink inNode=1 nHiddenLayer=0 />	

<outLink>	Type: Element neural_network/structure/hiddenLayers/ layer/perceptron/perceptronLinks/outLink	Required: Yes
Children	None	
Attributes	**outNode**: The perceptron ID for this output. **nHiddenLayer**: The ID associated with the hidden layer for the output from this perceptron. Otherwise this will be an integer from 1 to n layers.	
Description	This Element describes one output link from this perceptron	
Example	<outLink outNode=5 nHiddenLayer=2 />	

<outputLayer>	Type: Element neural_network/structure/outputLayer	Required: Yes
Children	<output> (require one or more)	
Attributes	**n_outputs**: The number of network outputs. This must be one or greater.	
Description	Structural description of the network outputs	
Example	<outputLayer n_outputs=2> <output outputPerceptron=5 outActivation=logistic outVariable=1> <outputDescription>Budget Expenses</outputDescription> </output> <output outputPerceptron=6 outActivation=logistic outVariable=2> <outputDescription>Additional Allocations</outputDescription> </output> </outputLayer>	

<output>	Type: Element neural_network/structure/outputLayer/output	Required: Yes
Children	<outputDescription> (optional)	
Attributes	**outPerceptron**: The perceptron ID number associated with this output. **outActivation**: The activation function associated with this output. **outVariable**: The numerical ID associated with the target variable for this output. For a continuous time variable, this will be the ID associated with that input attribute.	
Description	The network structural information for one network output.	
Example	<output outputPerceptron=5 outActivation=logistic outVariable=1> <outputDescription>Budget Expenses</outputDescription> </output>	

<outputDescription>	Type: String element neural network/structure/outputLayer/ output/outputDescription	Required: No
Children	None	
Attributes	None	
Description	Description of one network output.	
Example	<outputDescription>Budget Expenses</outputDescription>	

<opt_parameters>	Type: Element neural_network/opt_parameters	Required: Yes
Children	None	
Attributes	**stageII**: optional string. Allowed values are "yes" and "no". Default value: "yes" **maxIterations**: (optional integer) the maximum number of iterations. Default value: 1000 **maxFunctionEval**: (optional integer) the maximum number of function evaluations. Default value: 1000 **functionTol**: (optional floating point) a convergence tolerance for the optimization function. If two consecutive function evaluations is less than this value, convergence is assumed. Default value: 10^{-11}. **gradientTol**: (optional floating point) a convergence tolerance for the optimization function. If the difference between two consecutive gradient evaluations is less than this value, convergence is assumed. Default value: 10^{-11}. **maxStep**: (optional floating point) the maximum step size applied to optimization searches. Smaller step size slows convergence, but larger sizes risk the possibility of over-stepping an optimum. Default value: 1.0 **n_epochs**: (optional integer) the number of epochs used during stage I optimization. Default value: 1 **epoch_size**: (optional integer) the number of randomly selected training patterns for each epoch. Default value: 500 **accuracy**: (optional floating point) the accuracy targeted for this training session. If the optimization function falls below this value, training is halted. Default value: 0.1 **trace**: (optional string of "yes" or "no") Indicates whether tracing should occur during optimization. Default value: "no"	
Description	Controls the behavior of the optimization algorithm using ten attributes. Stage I optimization uses a modified steepest descent algorithm with the gradient calculated using backward propagation. This algorithm is fast, depending upon the setting for the epoch size and n_epocs. Stage II optimization uses the quasi-Newton optimization algorithm. This algorithm is slower, but tends to drive the optimization function to smaller values. The optimum weights from Stage I are used as the starting values for Stage II.	
Example	<opt_parameters stageII="no" n_epochs=20 epoch_size=2000 />	

<network>	Type: Element neural_network/network	Required: No
Children	<perceptron> (required)	
Attributes	**errorSS**: (required floating point) value of the optimization function after training, calculated using all training patterns. **perceptronCount**: (required integer) total number of perceptrons in the network, including output perceptrons. **weightCount**: (required integer) total number of weights in the network, including the bias weights.	
Description	Information describing results of network training	
Example	<network errorSS="4.6205442150109" perceptronCount="4" weightCount="10"> <perceptron activation="logistic" layer="1" id="1"> <inputWeights n_inLinks="1"> <link inputNode="0" weight="4.29667440700765" /> <link inputNode="1" weight="-4.7115417573063" /> </inputWeights> </perceptron> <perceptron activation="logistic" layer="1" id="2"> <inputWeights n_inLinks="1"> <link inputNode="0" weight="-1.35081127802218" /> <link inputNode="1" weight="0.493820644432522" /> </inputWeights> </perceptron> <perceptron activation="logistic" layer="1" id="3"> <inputWeights n_inLinks="1"> <link inputNode="0" weight="-4.49737322738591" /> <link inputNode="1" weight="5.81596398453038" /> </inputWeights> </perceptron> <perceptron activation="linear" layer="0" id="4"> <inputWeights n_inLinks="3"> <link inputNode="0" weight="-8.82962452780984" /> <link inputNode="1" weight="5.82969603422833" /> <link inputNode="2" weight="14.7945691833203" /> <link inputNode="3" weight="3.12440712779608" /> </inputWeights> </perceptron> </network>	

<perceptron>	Type: Element neural_network/network/perceptron	Required: Yes
Children	<inputWeights> (one or more required)	
Attributes	**id**: (required positive integer) the numerical ID associated with this perceptron. **layer**: (required non-negative integer) the layer in which this perceptron is located. If it is located in the output layer, then layer=0. **activation**: (optional string) the activation function associated with this perceptron. Default value: "linear". **scale**: (optional integer) the scaling method associated with this perceptron. Default value: 0 (no scaling)	
Description	The Element describes the location and nature of this perceptron.	
Example	<perceptron activation="logistic" layer="1" id="1"> <inputWeights n_inLinks="1"> <link inputNode="0" weight="4.29667440700765" /> <link inputNode="1" weight="-4.7115417573063" /> </inputWeights> </perceptron>	

<inputWeights>	Type: Element neural_network/network/perceptron/inputWeights	Required: Yes
Children	<link> (one or more required)	
Attributes	**n_inLinks**: (required positive integer) the number of input links for this perceptron. This is also equal to the number of weights for this perceptron, excluding the bias weight	
Description	The Element describe groups the weights and describes the number of weights for this perceptron.	
Example	<inputWeights n_inLinks=3> <link inputNode=0 weight=-0.23 /> <link inputNode=1 weight=-1.26 /> <link inputNode=2 weight=2.345 /> <link inputNode=3 weight=3.71 /> </inputWeights>	

<link>	Type: Element neural_network/network/perceptron/inputWeights/link	Required: Yes
Children	None	
Attributes	**inputNode**: (required non-negative integer) the perceptron ID or attribute ID for this input link. If the <perceptron> attribute layer=1 or if layer=0 and there are not hidden layers in the network, then inputNode is the ID associated with an input attribute; otherwise it is a perceptron ID. If inputNode=0, then this is the bias weight for this perceptron. **weight**: (required floating point) this is the weight for this perceptron input determined during network training.	
Description	This element give the value of the weight, either a bias weight or a weight associated with one input link to this perceptron.	
Example	<link inputNode=0 weight=-0.23 />	

Appendix B: NeuralNetSchema.xsd – The Neural Net Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://vni.com/NeuralNetSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0"
xmlns:mstns="http://vni.com/NeuralNetSchema" xmlns="http://vni.com/NeuralNetSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-
msdata">
   <xs:element name="neural_network">
      <xs:annotation>
         <xs:documentation>Schema for Describing a Multilayered, Feed-Forward Neural
Network</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="metadata">
               <xs:annotation>
                  <xs:documentation>Metadata for describing neural
network</xs:documentation>
               </xs:annotation>
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="name" type="xs:string">
                        <xs:annotation>
                           <xs:documentation>Name associated with this neural
network.</xs:documentation>
                        </xs:annotation>
                     </xs:element>
                     <xs:element name="title" type="xs:string" minOccurs="0">
                        <xs:annotation>
                           <xs:documentation>Optional title associated with this neural
network</xs:documentation>
                        </xs:annotation>
                     </xs:element>
                     <xs:element name="description" type="xs:string" minOccurs="0">
                        <xs:annotation>
                           <xs:documentation>Optional description of the neural
network</xs:documentation>
                        </xs:annotation>
                     </xs:element>
                     <xs:element name="trainer" type="xs:string" minOccurs="0">
                        <xs:annotation>
                           <xs:documentation>Optional name of person training this
network</xs:documentation>
                        </xs:annotation>
                     </xs:element>
                     <xs:element name="date" type="xs:string" minOccurs="0">
                        <xs:annotation>
                           <xs:documentation>Optional date of training</xs:documentation>
                        </xs:annotation>
                     </xs:element>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="structure">
               <xs:annotation>
                  <xs:documentation>Structural description of the neural network:  inputs,
hidden layers and outputs</xs:documentation>
               </xs:annotation>
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="inputLayer">
                        <xs:annotation>
                           <xs:documentation>Structural description of network
inputs</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="inputNode" maxOccurs="unbounded">
                                 <xs:annotation>
                                    <xs:documentation>Raw data input node</xs:documentation>
                                 </xs:annotation>
```

```
                                 <xs:complexType>
                                    <xs:sequence>
                                       <xs:element name="attributeDescription"
type="xs:string" minOccurs="0" msdata:Ordinal="4">
                                          <xs:annotation>
                                             <xs:documentation>An optional description of
this input</xs:documentation>
                                          </xs:annotation>
                                       </xs:element>
                                       <xs:element name="perceptronLink" minOccurs="0"
maxOccurs="unbounded">
                                          <xs:annotation>
                                             <xs:documentation>Perceptrons linked to this
input</xs:documentation>
                                          </xs:annotation>
                                          <xs:complexType>
                                             <xs:attribute name="hiddenLayer"
type="xs:positiveInteger" use="optional" form="unqualified">
                                                <xs:annotation>
                                                   <xs:documentation>The ID of the hidden
layer associated with this input link</xs:documentation>
                                                </xs:annotation>
                                             </xs:attribute>
                                             <xs:attribute name="perceptron"
type="xs:positiveInteger" use="required" form="unqualified">
                                                <xs:annotation>
                                                   <xs:documentation>The perceptron ID
associated with this input link.</xs:documentation>
                                                </xs:annotation>
                                             </xs:attribute>
                                          </xs:complexType>
                                       </xs:element>
                                       <xs:choice>
                                          <xs:element name="ordinalClass" minOccurs="0">
                                             <xs:annotation>
                                                <xs:documentation>Ordinal Class
Descriptions</xs:documentation>
                                             </xs:annotation>
                                             <xs:complexType>
                                                <xs:sequence>
                                                   <xs:element name="ordinalValue"
maxOccurs="unbounded">
                                                      <xs:complexType>
                                                         <xs:attribute name="classOrder"
type="xs:positiveInteger" use="required" form="unqualified">
                                                            <xs:annotation>
                                                               <xs:documentation>The order of
this class within the classes for this ordinal attribute</xs:documentation>
                                                            </xs:annotation>
                                                         </xs:attribute>
                                                         <xs:attribute name="value"
type="xs:string" use="required" form="unqualified">
                                                            <xs:annotation>
                                                               <xs:documentation>One of the
possible classes associated with this ordinal attribute.  The order of this class is
found in the classOrder attribute.</xs:documentation>
                                                            </xs:annotation>
                                                         </xs:attribute>
                                                      </xs:complexType>
                                                   </xs:element>
                                                </xs:sequence>
                                                <xs:attribute name="n_ordinalClass"
type="xs:positiveInteger" use="required" form="unqualified">
                                                   <xs:annotation>
                                                      <xs:documentation>Number of classes
associated with this ordinal attribute.  The order of these classes is described in the
ordinalValue elements</xs:documentation>
                                                   </xs:annotation>
                                                </xs:attribute>
                                                <xs:attribute name="m_transform"
use="optional" default="none" form="unqualified">
                                                   <xs:annotation>
                                                      <xs:documentation>Controls whether a
transformation is applied to the cumulative percentage calculated for each ordinal class.
The default value is m_transform=none.  m_transform=sqroot indicates that the square root
of percentages should replace the percentages, and  m_transform=arcsin indicates that the
arcsin square root transformation should be applied to the cumulative
percentages.</xs:documentation>
```

```
                                                </xs:annotation>
                                                <xs:simpleType>
                                                   <xs:restriction base="xs:string">
                                                      <xs:enumeration value="none"/>
                                                      <xs:enumeration value="sqroot"/>
                                                      <xs:enumeration value="arcsin"/>
                                                   </xs:restriction>
                                                </xs:simpleType>
                                             </xs:attribute>
                                          </xs:complexType>
                                       </xs:element>
                                       <xs:element name="nominalClass" minOccurs="0">
                                          <xs:annotation>
                                             <xs:documentation>Nominal Class
Description</xs:documentation>
                                          </xs:annotation>
                                          <xs:complexType>
                                             <xs:attribute name="n_classes"
type="xs:positiveInteger" use="required" form="unqualified">
                                                <xs:annotation>
                                                   <xs:documentation>Number of nominal
classes associated with this variable.</xs:documentation>
                                                </xs:annotation>
                                             </xs:attribute>
                                             <xs:attribute name="transform_method"
use="optional" default="binary" form="unqualified">
                                                <xs:annotation>
                                                   <xs:documentation>The transform method
used to represent this nominal variable as network input.  The only value currently
supported is "binary".  Using this method, a nominal variable will be prepresented as
n_classes separate columns, each containing only two values.</xs:documentation>
                                                </xs:annotation>
                                                <xs:simpleType>
                                                   <xs:restriction base="xs:string">
                                                      <xs:enumeration value="binary"/>
                                                      <xs:enumeration
value="equilateral"/>
                                                   </xs:restriction>
                                                </xs:simpleType>
                                             </xs:attribute>
                                             <xs:attribute name="scale_method"
use="optional" default="1" form="unqualified">
                                                <xs:annotation>
                                                   <xs:documentation>Controls the i_scale
parameter for unsupervised nominal filtering.  Set scale_method=0 if the two values used
to represent the nominal variable are 0.1 and 0.9.  Otherwise the values are unchanged.
They will be 0 and 1 if transform_method=binary.  Otherwise they are -1 and
1.</xs:documentation>
                                                </xs:annotation>
                                                <xs:simpleType>
                                                   <xs:restriction
base="xs:nonNegativeInteger">
                                                      <xs:enumeration value="0"/>
                                                      <xs:enumeration value="1"/>
                                                   </xs:restriction>
                                                </xs:simpleType>
                                             </xs:attribute>
                                          </xs:complexType>
                                       </xs:element>
                                       <xs:element name="continuousAttribute">
                                          <xs:annotation>
                                             <xs:documentation>Continuous Variable
Description</xs:documentation>
                                          </xs:annotation>
                                          <xs:complexType>
                                             <xs:sequence>
                                                <xs:element name="realLimits"
minOccurs="0">
                                                   <xs:annotation>
                                                      <xs:documentation>Optional limits
required for this variable when scale=1, 4 or 5.</xs:documentation>
                                                   </xs:annotation>
                                                   <xs:complexType>
                                                      <xs:attribute name="lowerRealLimit"
type="xs:double" use="required" form="unqualified">
                                                         <xs:annotation>
```

```
                                                <xs:documentation>Absolute
lower limit for this input.  This value should  be below the lowest value expected for
this variable.</xs:documentation>
                                                </xs:annotation>
                                             </xs:attribute>
                                             <xs:attribute name="upperRealLimit"
type="xs:double" use="required" form="unqualified">
                                                <xs:annotation>
                                                   <xs:documentation>Absolute
upper limit for this input.  This value should be greater than the largest value expected
for this variable.</xs:documentation>
                                                </xs:annotation>
                                             </xs:attribute>
                                             <xs:attribute
name="lowerTargetLimit" type="xs:double" use="required" form="unqualified">
                                                <xs:annotation>
                                                   <xs:documentation>The lower
limit targeted for the scaled value of this variable.</xs:documentation>
                                                </xs:annotation>
                                             </xs:attribute>
                                             <xs:attribute
name="upperTargetLimit" type="xs:double" use="required" form="unqualified">
                                                <xs:annotation>
                                                   <xs:documentation>The higest
limit targeted for the scaled version of this variable.</xs:documentation>
                                                </xs:annotation>
                                             </xs:attribute>
                                          </xs:complexType>
                                       </xs:element>
                                       <xs:element name="statLimits"
minOccurs="0">
                                          <xs:annotation>
                                             <xs:documentation>Optional limits
required for this variable when scale=2, 3, 4 or 5.  These are calculated when they are
not supplied.</xs:documentation>
                                          </xs:annotation>
                                          <xs:complexType>
                                             <xs:attribute name="center"
type="xs:double" use="optional" form="unqualified">
                                                <xs:annotation>
                                                   <xs:documentation>The center
value associated with this input.  This is either the average or median, depending upon
whether scale=2 or 3, 4 or 5.</xs:documentation>
                                                </xs:annotation>
                                             </xs:attribute>
                                             <xs:attribute name="spread"
type="xs:double" use="optional" form="unqualified">
                                                <xs:annotation>
                                                   <xs:documentation>Estimate of
the spread associated with this input when scale=2-5. </xs:documentation>
                                                </xs:annotation>
                                             </xs:attribute>
                                          </xs:complexType>
                                       </xs:element>
                                       <xs:element name="lag" minOccurs="0"
maxOccurs="unbounded">
                                          <xs:annotation>
                                             <xs:documentation>A request to
generate an input column derived by lagging this continuous variable.  The number of lags
is equal to the attribute lagvalue.</xs:documentation>
                                          </xs:annotation>
                                          <xs:complexType>
                                             <xs:attribute name="lagvalue"
type="xs:positiveInteger" use="required">
                                                <xs:annotation>
                                                   <xs:documentation>The number
of lags requested for the network input variable generated by this
element.</xs:documentation>
                                                </xs:annotation>
                                             </xs:attribute>
                                          </xs:complexType>
                                       </xs:element>
                                    </xs:sequence>
                                    <xs:attribute name="scale" use="optional"
default="none" form="unqualified">
                                       <xs:annotation>
                                          <xs:documentation>Scale function, if
any, to apply to this input variable.  Choice are none, bounded, z-score, robust z-score,
```

```
bounded z-score and bounded robust z-score.  Bounded scaling uses these limits stated in
the realLimits element to rescale the variable into a limited range, normal 0 to 1.  For
more information, see the description of the method parameter in the API for the
scale_filter.</xs:documentation>
                                                </xs:annotation>
                                                <xs:simpleType>
                                                   <xs:restriction base="xs:string">
                                                      <xs:enumeration value="none"/>
                                                      <xs:enumeration value="bounded"/>
                                                      <xs:enumeration value="z-score"/>
                                                      <xs:enumeration value="robust z-
score"/>
                                                      <xs:enumeration value="bounded z-
score"/>
                                                      <xs:enumeration value="bounded
robust z-score"/>
                                                   </xs:restriction>
                                                </xs:simpleType>
                                             </xs:attribute>
                                             <xs:attribute name="n_lags"
type="xs:nonNegativeInteger" use="optional" default="0" form="unqualified">
                                                <xs:annotation>
                                                   <xs:documentation>If type="time" this
is the number of lags associated with this variable.  Set n_lags=0, the default value, if
lagging is unnecessary.  Setting n_lags to an integer greater than zero generates lagged
columns of the output variable.  These columns are then used as network inputs.  The
number of lag elements in this file must be equal to n_lags.</xs:documentation>
                                                </xs:annotation>
                                             </xs:attribute>
                                             <xs:attribute name="continuous_type"
use="optional" default="input" form="unqualified">
                                                <xs:annotation>
                                                   <xs:documentation>Indicates whether
this is an input, output or time variable.  Set continuous_type="input" or
continuous_type="output" depending upon whether this is an input or output variable.  If
this to have lagged input, set continuous_type="time"</xs:documentation>
                                                </xs:annotation>
                                                <xs:simpleType>
                                                   <xs:restriction base="xs:string">
                                                      <xs:enumeration value="time"/>
                                                      <xs:enumeration value="input"/>
                                                      <xs:enumeration value="output"/>
                                                   </xs:restriction>
                                                </xs:simpleType>
                                             </xs:attribute>
                                          </xs:complexType>
                                       </xs:element>
                                    </xs:choice>
                                 </xs:sequence>
                                 <xs:attribute name="id" type="xs:positiveInteger"
use="required" form="unqualified">
                                    <xs:annotation>
                                       <xs:documentation>The ID associated with this
input.  This is a positive integer from 1 to n_inputs.</xs:documentation>
                                    </xs:annotation>
                                 </xs:attribute>
                                 <xs:attribute name="activation" use="optional"
default="linear" form="unqualified">
                                    <xs:annotation>
                                       <xs:documentation>Any activation function
associated with this input.  Choices are linear, logistic and tanh.</xs:documentation>
                                    </xs:annotation>
                                    <xs:simpleType>
                                       <xs:restriction base="xs:string">
                                          <xs:enumeration value="linear"/>
                                          <xs:enumeration value="logistic"/>
                                          <xs:enumeration value="tanh"/>
                                       </xs:restriction>
                                    </xs:simpleType>
                                 </xs:attribute>
                                 <xs:attribute name="label" type="xs:string"
use="optional" default="Unnamed Input" form="unqualified">
                                    <xs:annotation>
                                       <xs:documentation>An optional label to be
associated with this input.</xs:documentation>
                                    </xs:annotation>
                                 </xs:attribute>
```

```
                                    <xs:attribute name="type" use="optional"
default="continuous" form="unqualified">
                                       <xs:annotation>
                                          <xs:documentation>The input type.  Choices are
nominal, ordinal, continuous, time and date.</xs:documentation>
                                       </xs:annotation>
                                       <xs:simpleType>
                                          <xs:restriction base="xs:string">
                                             <xs:enumeration value="continuous"/>
                                             <xs:enumeration value="nominal"/>
                                             <xs:enumeration value="ordinal"/>
                                          </xs:restriction>
                                       </xs:simpleType>
                                    </xs:attribute>
                                 </xs:complexType>
                              </xs:element>
                           </xs:sequence>
                           <xs:attribute name="n_inputs" type="xs:positiveInteger"
use="required" form="unqualified">
                              <xs:annotation>
                                 <xs:documentation>Number of inputs described in the XML
file.  Must be 1 or greater.  In general, this is less than the number of actual network
inputs.  The number of network inputs depends upon whether the inputs in the XML file
require transforming into multiple columns.</xs:documentation>
                              </xs:annotation>
                           </xs:attribute>
                        </xs:complexType>
                     </xs:element>
                     <xs:element name="hiddenLayers" minOccurs="0">
                        <xs:annotation>
                           <xs:documentation>Structural description of network hidden
layers, if any.</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="layer" minOccurs="0" maxOccurs="unbounded">
                                 <xs:annotation>
                                    <xs:documentation>Structural description of one hidden
layer</xs:documentation>
                                 </xs:annotation>
                                 <xs:complexType>
                                    <xs:sequence>
                                       <xs:element name="perceptron" minOccurs="0"
maxOccurs="unbounded">
                                          <xs:annotation>
                                             <xs:documentation>Description of one perceptron
in this hidden layer</xs:documentation>
                                          </xs:annotation>
                                          <xs:complexType>
                                             <xs:sequence>
                                                <xs:element name="perceptronLinks"
minOccurs="0" maxOccurs="unbounded">
                                                   <xs:annotation>
                                                      <xs:documentation>Description of the
input and output links for a perceptron</xs:documentation>
                                                   </xs:annotation>
                                                   <xs:complexType>
                                                      <xs:sequence>
                                                         <xs:element name="inLink"
maxOccurs="unbounded">
                                                            <xs:annotation>
                                                               <xs:documentation>Perceptron
Input Link</xs:documentation>
                                                            </xs:annotation>
                                                            <xs:complexType>
                                                               <xs:attribute name="inNode"
type="xs:positiveInteger" use="required" form="unqualified">
                                                                  <xs:annotation>
                                                                     <xs:documentation>The
ID associated with this input.  If this is in the first hidden layer, then this is the ID
of one of the network inputs described in this file. If this not the first hidden layer,
this ID is the ID of a perceptron in a previous layer .</xs:documentation>
                                                                  </xs:annotation>
                                                               </xs:attribute>
                                                               <xs:attribute
name="nHiddenLayer" type="xs:nonNegativeInteger" use="optional" form="unqualified">
                                                                  <xs:annotation>
```

```
                                                                  <xs:documentation>The
hidden layer ID associated with this input.  If the input is from a raw input, then this
is "0".  If it is from another perceptron, then this is the hidden layer ID of that
perceptron.</xs:documentation>
                                                               </xs:annotation>
                                                            </xs:attribute>
                                                         </xs:complexType>
                                                      </xs:element>
                                                      <xs:element name="outLink"
maxOccurs="unbounded">
                                                         <xs:annotation>
                                                            <xs:documentation>Perceptron
Output Link</xs:documentation>
                                                         </xs:annotation>
                                                         <xs:complexType>
                                                            <xs:attribute name="outNode"
type="xs:positiveInteger" use="required" form="unqualified">
                                                               <xs:annotation>
                                                                  <xs:documentation>The
ID associated with this output.  If the output is to a network output, then this is the
output's ID.  If this link is to a perceptron in a following layer, then this is the ID
of that perceptron.</xs:documentation>
                                                               </xs:annotation>
                                                            </xs:attribute>
                                                            <xs:attribute
name="nHiddenLayer" type="xs:nonNegativeInteger" use="optional" form="unqualified">
                                                               <xs:annotation>
                                                                  <xs:documentation>The
ID of the hidden layer associated with this output.  If the output is to a network
output, then this is "0".  If this link is to another perceptron, then this is the ID of
the hidden layer associated with that perceptron.</xs:documentation>
                                                               </xs:annotation>
                                                            </xs:attribute>
                                                         </xs:complexType>
                                                      </xs:element>
                                                   </xs:sequence>
                                                   <xs:attribute name="n_inLinks"
type="xs:positiveInteger" use="required" form="unqualified">
                                                      <xs:annotation>
                                                         <xs:documentation>Number of input
links associated with this perceptron.  This must be one or greater.</xs:documentation>
                                                      </xs:annotation>
                                                   </xs:attribute>
                                                   <xs:attribute name="n_outLinks"
type="xs:positiveInteger" use="required" form="unqualified">
                                                      <xs:annotation>
                                                         <xs:documentation>Number of
output links associated with this perceptron.  This must be one or
greater.</xs:documentation>
                                                      </xs:annotation>
                                                   </xs:attribute>
                                                </xs:complexType>
                                             </xs:element>
                                          </xs:sequence>
                                          <xs:attribute name="perceptronID"
type="xs:positiveInteger" use="required" form="unqualified">
                                             <xs:annotation>
                                                <xs:documentation>The perceptron
identification (ID).  Perceptrons are numbered from 1 to n_nodes.  Each perceptron must
have an ID different from all other perceptrons in the network. </xs:documentation>
                                             </xs:annotation>
                                          </xs:attribute>
                                          <xs:attribute name="activation" use="optional"
default="logistic" form="unqualified">
                                             <xs:annotation>
                                                <xs:documentation>Activation function
associated with this perceptron.  Options are logistic, tanh and
linear</xs:documentation>
                                             </xs:annotation>
                                             <xs:simpleType>
                                                <xs:restriction base="xs:string">
                                                   <xs:enumeration value="logistic"/>
                                                   <xs:enumeration value="tanh"/>
                                                   <xs:enumeration value="linear"/>
                                                </xs:restriction>
                                             </xs:simpleType>
                                          </xs:attribute>
                                       </xs:complexType>
```

```
                                    </xs:element>
                                 </xs:sequence>
                                 <xs:attribute name="hiddenLayer"
type="xs:positiveInteger" use="required" form="unqualified">
                                    <xs:annotation>
                                       <xs:documentation>Hidden layer ID.  Layers are
numbered from 1 to n_layers</xs:documentation>
                                    </xs:annotation>
                                 </xs:attribute>
                                 <xs:attribute name="n_perceptrons"
type="xs:positiveInteger" use="required" form="unqualified">
                                    <xs:annotation>
                                       <xs:documentation>Number of perceptrons in this
hidden layer.</xs:documentation>
                                    </xs:annotation>
                                 </xs:attribute>
                              </xs:complexType>
                           </xs:element>
                        </xs:sequence>
                        <xs:attribute name="n_layers" type="xs:positiveInteger"
use="required" form="unqualified">
                           <xs:annotation>
                              <xs:documentation>Number of hidden layers, must be zero or
greater</xs:documentation>
                           </xs:annotation>
                        </xs:attribute>
                        <xs:attribute name="n_nodes" type="xs:positiveInteger"
use="required" form="unqualified">
                           <xs:annotation>
                              <xs:documentation>Total number of perceptrons in all hidden
layers.  This does not include the output perceptrons.</xs:documentation>
                           </xs:annotation>
                        </xs:attribute>
                     </xs:complexType>
                  </xs:element>
                  <xs:element name="outputLayer">
                     <xs:annotation>
                        <xs:documentation>Structural description of network
outputs</xs:documentation>
                     </xs:annotation>
                     <xs:complexType>
                        <xs:sequence>
                           <xs:element name="output" maxOccurs="unbounded">
                              <xs:annotation>
                                 <xs:documentation>An output
perceptron</xs:documentation>
                              </xs:annotation>
                              <xs:complexType>
                                 <xs:sequence>
                                    <xs:element name="outputDescription" type="xs:string"
minOccurs="0" msdata:Ordinal="2">
                                       <xs:annotation>
                                          <xs:documentation>An optional description of
the output expected from this perceptron.</xs:documentation>
                                       </xs:annotation>
                                    </xs:element>
                                 </xs:sequence>
                                 <xs:attribute name="outPerceptron"
type="xs:positiveInteger" use="required" form="unqualified">
                                    <xs:annotation>
                                       <xs:documentation>The ID (number) associated with
this output perceptron.  Output perceptrons are numbered beginning with the next integer
after the last hidden layer perceptron.</xs:documentation>
                                    </xs:annotation>
                                 </xs:attribute>
                                 <xs:attribute name="outActivation" use="optional"
default="linear" form="unqualified">
                                    <xs:annotation>
                                       <xs:documentation>The activation for this output
perceptron.  Valid entries are "linear", "logistic", and "tanh".</xs:documentation>
                                    </xs:annotation>
                                    <xs:simpleType>
                                       <xs:restriction base="xs:string">
                                          <xs:enumeration value="linear"/>
                                          <xs:enumeration value="logistic"/>
                                          <xs:enumeration value="tanh"/>
                                       </xs:restriction>
                                    </xs:simpleType>
```

```
                                    </xs:attribute>
                                    <xs:attribute name="outVariable"
type="xs:positiveInteger" use="required" form="unqualified">
                                       <xs:annotation>
                                          <xs:documentation>The numerical ID associated with
an output or time input variable described in this file that is being predicted by this
output perceptron.  This is information associates the output from this perceptron with
one of the input variables described in the inputLayer element of this
file.</xs:documentation>
                                       </xs:annotation>
                                    </xs:attribute>
                                 </xs:complexType>
                              </xs:element>
                           </xs:sequence>
                           <xs:attribute name="n_outputs" type="xs:positiveInteger"
use="required" form="unqualified">
                              <xs:annotation>
                                 <xs:documentation>Number of network output perceptrons.
This should also be equal to the number of output elements in this
file.</xs:documentation>
                              </xs:annotation>
                           </xs:attribute>
                        </xs:complexType>
                     </xs:element>
                  </xs:sequence>
                  <xs:attribute name="numberOfLayers" type="xs:positiveInteger"
use="optional" default="1" form="unqualified">
                     <xs:annotation>
                        <xs:documentation>Number of network layers:  number of hidden
layers plus 1.</xs:documentation>
                     </xs:annotation>
                  </xs:attribute>
                  <xs:attribute name="numberOfPerceptrons" type="xs:positiveInteger"
use="required" form="unqualified">
                     <xs:annotation>
                        <xs:documentation>The total number of perceptrons, including all
output perceptrons.</xs:documentation>
                     </xs:annotation>
                  </xs:attribute>
                  <xs:attribute name="numberOfAttributes" type="xs:positiveInteger"
use="required" form="unqualified">
                     <xs:annotation>
                        <xs:documentation>The total number of input attributes, both
continuous and categorical.</xs:documentation>
                     </xs:annotation>
                  </xs:attribute>
               </xs:complexType>
            </xs:element>
            <xs:element name="opt_parameters">
               <xs:annotation>
                  <xs:documentation>Optimization parameters for training neural
network.</xs:documentation>
               </xs:annotation>
               <xs:complexType>
                  <xs:attribute name="stageII" use="optional" default="yes"
form="unqualified">
                     <xs:annotation>
                        <xs:documentation>Controls whether Stage II optimization is invoked
after Stage I optimization</xs:documentation>
                     </xs:annotation>
                     <xs:simpleType>
                        <xs:restriction base="xs:string">
                           <xs:whiteSpace value="collapse"/>
                           <xs:enumeration value="yes"/>
                           <xs:enumeration value="no"/>
                           <xs:enumeration value="YES"/>
                           <xs:enumeration value="NO"/>
                           <xs:enumeration value="Yes"/>
                           <xs:enumeration value="No"/>
                        </xs:restriction>
                     </xs:simpleType>
                  </xs:attribute>
                  <xs:attribute name="maxIterations" type="xs:positiveInteger"
use="optional" default="1000" form="unqualified">
                     <xs:annotation>
                        <xs:documentation>Maximum number of optimization iterations -
passed into optimization routine</xs:documentation>
                     </xs:annotation>
```

```
                </xs:attribute>
                <xs:attribute name="maxFunctionEval" type="xs:positiveInteger"
use="optional" default="1000" form="unqualified">
                   <xs:annotation>
                      <xs:documentation>Maximum number of function evaluations - passed
into optimization routine</xs:documentation>
                   </xs:annotation>
                </xs:attribute>
                <xs:attribute name="functionTol" type="xs:double" use="optional"
default="1.0e-11" form="unqualified">
                   <xs:annotation>
                      <xs:documentation>Tolerance applied to the error sum of squares.
Optimization stops when the improvement between two consecutive error sum of squares is
less than this number.</xs:documentation>
                   </xs:annotation>
                </xs:attribute>
                <xs:attribute name="gradientTol" type="xs:double" use="optional"
default="1.0e-11">
                   <xs:annotation>
                      <xs:documentation>Tolerance applied to consecutive gradient
calculations.  Optimization stops when the change between two consecutive gradient
calculations is less than this tolerance.</xs:documentation>
                   </xs:annotation>
                </xs:attribute>
                <xs:attribute name="maxStep" type="xs:double" use="optional"
default="1.0">
                   <xs:annotation>
                      <xs:documentation>The maximum step size applied to optimization
searches.  A small value slows optimization, but a large value can cause the optimization
routine to step over an optimum.</xs:documentation>
                   </xs:annotation>
                </xs:attribute>
                <xs:attribute name="n_epochs" type="xs:positiveInteger" use="optional"
default="1">
                   <xs:annotation>
                      <xs:documentation>The number of epochs in Stage I optimization.
This is the number of optimizations during Stage I, each initiated with new starting
values for the network weights.</xs:documentation>
                   </xs:annotation>
                </xs:attribute>
                <xs:attribute name="epoch_size" type="xs:positiveInteger" use="optional"
default="500">
                   <xs:annotation>
                      <xs:documentation>The number of training cases randomly selected
for training each epoch.  This should be less than the total number of training
cases.</xs:documentation>
                   </xs:annotation>
                </xs:attribute>
                <xs:attribute name="accuracy" type="xs:double" use="optional"
default="0.1">
                   <xs:annotation>
                      <xs:documentation>The absolute accuracy.  If the error sum of
squares falls below this value, optimization is stopped and the current solution is
returned as the optimum.</xs:documentation>
                   </xs:annotation>
                </xs:attribute>
                <xs:attribute name="trace" use="optional" default="no"
form="unqualified">
                   <xs:annotation>
                      <xs:documentation>Controls whether optimization progress is
displayed during network training.</xs:documentation>
                   </xs:annotation>
                   <xs:simpleType>
                      <xs:restriction base="xs:string">
                         <xs:whiteSpace value="collapse"/>
                         <xs:enumeration value="Yes"/>
                         <xs:enumeration value="No"/>
                         <xs:enumeration value="yes"/>
                         <xs:enumeration value="no"/>
                         <xs:enumeration value="YES"/>
                         <xs:enumeration value="NO"/>
                      </xs:restriction>
                   </xs:simpleType>
                </xs:attribute>
             </xs:complexType>
          </xs:element>
          <xs:element name="network" minOccurs="0">
             <xs:annotation>
```

```
                  <xs:documentation>Results of training:  neural network forecasting
weights</xs:documentation>
               </xs:annotation>
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="perceptron" maxOccurs="unbounded">
                        <xs:annotation>
                           <xs:documentation>Network perceptrons, across all hidden layers
and outputs</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="inputWeights">
                                 <xs:annotation>
                                    <xs:documentation>Weights for each input to this
perceptron</xs:documentation>
                                 </xs:annotation>
                                 <xs:complexType>
                                    <xs:sequence>
                                       <xs:element name="link" maxOccurs="unbounded">
                                          <xs:annotation>
                                             <xs:documentation>Weight associated with this
input link.</xs:documentation>
                                          </xs:annotation>
                                          <xs:complexType>
                                             <xs:attribute name="inputNode"
type="xs:nonNegativeInteger">
                                                <xs:annotation>
                                                   <xs:documentation>This is the ID
associated with this input.  If layer="0" then this is the ID associated with this
network input.  Otherwise, this is the ID of a perceptron within the hidden layer
associated with the layer ID.</xs:documentation>
                                                </xs:annotation>
                                             </xs:attribute>
                                             <xs:attribute name="weight" type="xs:double">
                                                <xs:annotation>
                                                   <xs:documentation>The weight obtained for
this input during network training.</xs:documentation>
                                                </xs:annotation>
                                             </xs:attribute>
                                          </xs:complexType>
                                       </xs:element>
                                    </xs:sequence>
                                    <xs:attribute name="n_inLinks"
type="xs:nonNegativeInteger" use="required" form="unqualified">
                                       <xs:annotation>
                                          <xs:documentation>Number of input links to this
perceptron.</xs:documentation>
                                       </xs:annotation>
                                    </xs:attribute>
                                 </xs:complexType>
                              </xs:element>
                           </xs:sequence>
                           <xs:attribute name="id" type="xs:positiveInteger" use="required"
form="unqualified">
                              <xs:annotation>
                                 <xs:documentation>Perceptron ID</xs:documentation>
                              </xs:annotation>
                           </xs:attribute>
                           <xs:attribute name="layer" type="xs:string" use="required"
form="unqualified">
                              <xs:annotation>
                                 <xs:documentation>Layer ID.  This is either the hidden
layer ID or "0" for the output layer.</xs:documentation>
                              </xs:annotation>
                           </xs:attribute>
                           <xs:attribute name="activation" type="xs:string" use="optional"
default="0" form="unqualified">
                              <xs:annotation>
                                 <xs:documentation>Activation function associated with this
perceptron.  This is either linear, logistic or tanh.</xs:documentation>
                              </xs:annotation>
                           </xs:attribute>
                           <xs:attribute name="scale" type="xs:nonNegativeInteger"
use="optional" default="0" form="unqualified">
                              <xs:annotation>
```

```
                                <xs:documentation>Scale function associated with the output
from this perceptron.  Choices are scale=0=linear, 1=bounded, 2=z-score, 3=robust z-
score, 4=bounded, z-score and 5=bounded, robust z-score.</xs:documentation>
                              </xs:annotation>
                            </xs:attribute>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                      <xs:attribute name="errorSS" type="xs:double" use="required"
form="unqualified"/>
                      <xs:attribute name="perceptronCount" type="xs:positiveInteger"
use="required" form="unqualified"/>
                      <xs:attribute name="weightCount" type="xs:positiveInteger" use="required"
form="unqualified"/>
                  </xs:complexType>
              </xs:element>
          </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Appendix C: An Example XML Neural Network Description File

The following XML file describes a 2-layer feed-forward neural network with the following structure:

- Number of Layers: 2 (1 hidden layer and 1 output layer)
- Number of Outputs: 1 (financial obligations, a continuous variable)
- Number of Network Inputs: 1 (the lag 1 value for the target variable, obligations)
- Number of Data Inputs: n_inputs=2
- Number of Perceptrons in the Hidden Layer: 3 (all using logistic activation)
- Output Perceptron uses Linear activation
- Total Number of Network Weights: 10 (2 for each perceptron the hidden layer and 4 for the output perceptron)

```
<?xml version="1.0" encoding="utf-8"?>
<neural_network xmlns="http://vni.com/NeuralNetSchema"
     xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://vni.com/NeuralNetSchema NeuralNetSchema.xsd">
  <metadata>
    <name>The Financial Forecast</name>
    <title>"A Financial Neural Network"</title>
    <description>"This is a 2 layer, feed-forward  neural network trained to
        produce financial forecasts for an uncertain budget"
      </description>
    <trainer>"John Doe"</trainer>
    <date>"03-09-2004"</date>
  </metadata>
  <structure numberOfLayers="2" numberOfPerceptrons="4" numberOfAttributes="1">
    <inputLayer n_inputs="2">
      <inputNode id="1" activation="linear" label="Lagged Obligations" type="continuous">
        <attributeDescription>Obligations</attributeDescription>
        <perceptronLink hiddenLayer="1" perceptron="1" />
        <perceptronLink hiddenLayer="1" perceptron="2" />
        <perceptronLink hiddenLayer="1" perceptron="3" />
        <continuousAttribute continuous_type="input" scale="bounded">
          <realLimits lowerRealLimit="-104978" upperRealLimit="24729475"
                      lowerTargetLimit="0.0" upperTargetLimit="1.0" />
        </continuousAttribute>
      </inputNode>
      <inputNode id="2" activation="linear" label="Obligations" type="continuous">
        <attributeDescription>Obligations</attributeDescription>
        <continuousAttribute continuous_type="output" scale="bounded">
          <realLimits lowerRealLimit="-104978" upperRealLimit="24729475"
                      lowerTargetLimit="0.0"   upperTargetLimit="1.0" />
        </continuousAttribute>
      </inputNode>
    </inputLayer>
    <hiddenLayers n_layers="1" n_nodes="3">
      <layer hiddenLayer="1" n_perceptrons="3">
        <perceptron perceptronID="1" activation="logistic">
          <perceptronLinks n_inLinks="1" n_outLinks="1">
            <inLink inNode="1" />
            <outLink outNode="4" />
          </perceptronLinks>
        </perceptron>
        <perceptron perceptronID="2" activation="logistic">
          <perceptronLinks n_inLinks="1" n_outLinks="1">
            <inLink inNode="1" />
            <outLink outNode="4" />
          </perceptronLinks>
        </perceptron>
        <perceptron perceptronID="3" activation="logistic">
          <perceptronLinks n_inLinks="1" n_outLinks="1">
            <inLink inNode="1" />
            <outLink outNode="4" />
```

```
          </perceptronLinks>
        </perceptron>
      </layer>
    </hiddenLayers>
    <outputLayer n_outputs="1">
      <output outPerceptron="4" outVariable="2" outActivation="linear">
        <outputDescription>"Forecasted Obligations"</outputDescription>
      </output>
    </outputLayer>
  </structure>
  <opt_parameters trace="yes" stageII="yes" maxIterations="1000" maxFunctionEval="1000"
functionTol="1.0e-20"

gradientTol="1.0e-20" maxStep="10" n_epochs="5" epoch_size="1000" accuracy="1.0e-6" />
  <network errorSS="4.6205442150109" perceptronCount="4" weightCount="10">
    <perceptron activation="logistic" layer="1" id="1">
      <inputWeights n_inLinks="1">
        <link inputNode="0" weight="4.29667440700765" />
        <link inputNode="1" weight="-4.7115417573063" />
      </inputWeights>
    </perceptron>
    <perceptron activation="logistic" layer="2" id="2">
      <inputWeights n_inLinks="1">
        <link inputNode="0" weight="-1.35081127802218" />
        <link inputNode="1" weight="0.493820644432522" />
      </inputWeights>
    </perceptron>
    <perceptron activation="logistic" layer="3" id="3">
      <inputWeights n_inLinks="1">
        <link inputNode="0" weight="-4.49737322738591" />
        <link inputNode="1" weight="5.81596398453038" />
      </inputWeights>
    </perceptron>
    <perceptron activation="linear" layer="4" id="4">
      <inputWeights n_inLinks="3">
        <link inputNode="0" weight="-8.82962452780984" />
        <link inputNode="1" weight="5.82969603422833" />
        <link inputNode="2" weight="14.7945691833203" />
        <link inputNode="3" weight="3.12440712779608" />
      </inputWeights>
    </perceptron>
  </network>
</neural_network>
```

Appendix D: Training Pattern File

The neural network training application uses a text file delimited with commas or tabs containing the training patterns as input, together with a corresponding XML neural network description file. The XML file, described in Appendices A and C, contains a description of the architectural description of the network and a description of the input attributes and network output targets.

The training patterns are the network inputs and their output target values for use in training the network. If Stage I training is conducted using an `epoch_size` greater than or equal to the number of training patterns, every training pattern is used during each training iteration. If that is not the case, then the training patterns are obtained by randomly sampling this file. The number of patterns sampled for each epoch is controlled by the `epoch_size` attribute in the network's XML neural network description file.

The first line of the training pattern file must contain the total number of input attributes, including any strings or inputs that are in the file but should be ignored. Whether an attribute or input string is ignored, used as input or treated as a network target, is controlled by the entries in the second line of this file.

The second line must consists of a series of integers each separated by either a space, tab or comma. This line describes the number of attributes to read from each training pattern and their roles. Each entry is associated with the attribute

in that position. For example the first entry refers to the first attribute in each training pattern, the second refers to the second attributes, etc.

If a zero is encountered in this line, then the attribute associated with that entry is ignored. If a value other than zero is encountered, then that value should be one of the input attribute identification numbers given in the XML neural network description file.

The actual training pattern data are given in the lines following these first two lines. The data must be delimited using either a tab, comma or space. String data containing spaces should be enclosed in quotes. Any missing entries should be denoted using the null string: “”.

The following examples will make this clearer.

Example 1: One lagged input and one output target

Assume that the `<inputLayer>` section of the XML file contains the following entries describing two input attributes. The first attribute is a continuous input referred to as *`Lagged Obligations`*. This is probably the first lag of the target attribute referred to as *`Obligations`*.

The `Lagged Obligations` attribute has an identification number of “1” and the target attribute, `Obligations`, has an identification number of “2.” Notice that the input attribute, *`Lagged Obligations`*, is linked to the three perceptrons in the first, and only, hidden layer.

The network input attributes are scaled to a range of 0 – 1. The neural network applications automatically invoke this scaling using the value of the `scale` attribute of the `<continuousAttribute>` element for each continuous input, including the network output target. Note that *bounded* scaling is being requested for both `Lagged Obligations` and `Obligations`.

```
<inputLayer n_inputs="2">
      <inputNode id="1" activation="linear" label="Lagged Obligations" type="continuous">
        <attributeDescription>Obligations</attributeDescription>
        <perceptronLink hiddenLayer="1" perceptron="1" />
        <perceptronLink hiddenLayer="1" perceptron="2" />
        <perceptronLink hiddenLayer="1" perceptron="3" />
        <continuousAttribute continuous_type="input" scale="bounded">
          <realLimits lowerRealLimit="-104978" upperRealLimit="24729475"
                      lowerTargetLimit="0.0" upperTargetLimit="1.0" />
        </continuousAttribute>
      </inputNode>
      <inputNode id="2" activation="linear" label="Obligations" type="continuous">
        <attributeDescription>Obligations</attributeDescription>
        <continuousAttribute continuous_type="output" scale="bounded">
          <realLimits lowerRealLimit="-104978" upperRealLimit="24729475"
                      lowerTargetLimit="0.0"   upperTargetLimit="1.0" />
        </continuousAttribute>
      </inputNode>
</inputLayer>
```

The input format for the neural network applications can specify unused variables or labels that should be skipped by putting zeros in the second input line. A zero entry indicates that the string in this position should be skipped.

In this case, assume that the training patterns file contains not only values for the input attribute and output target, it also contains a date and time label for each training pattern. This is included to help individual training patterns, and it was needed to sort these data into chronologically ascending order.

The following lines are lines 3, 4, … of this file. Each line contains the following input entered in this order: date, first lag, target attribute, time.

```
6/9/2003   1467.89    1295.67    “13:15”
6/8/2003   1372.55    1467.89    “14:54”
6/7/2003   1876.54    1372.55    “15:48”
```

```
.
. etc
.
5/16/2003  1843.23    1745.83    “15:30”
5/15/2003  “”          1843.23    “15:28”
```

Note that missing values are indicated with a null string: “”. Any value can be enclosed in quotes, but any input attribute that is a string containing one or more spaces space, must be enclosed in quotes. Leading and trailing spaces are ignored.

Time series data must be sorted from the most recent data to the oldest. If the data are further divided into nominal categories that are used as input, then the data should be sorted in descending order chronologically and then sorted by category.

The first and second lines in a file containing these patterns would consist of the following lines:

```
2
0     1     2     0
```

The zeros in the first and 4th positions signal to ignore attribute entries in those positions. Their data will not be used for network training.

The “1” in the second position signals that the numbers in the second position are associated with the input attribute with the identification number of “1”. In this example that is *Lagged Obligations*.

The “2” in the third position indicates that entries in that position are associated with the attribute with the identification number of “2”. According to the XML description above, this entry is the target output *Obligations*.

Example 2: A neural network with one continuous input, five categorical inputs and one continuous output.

This is an example of a neural network with a time series target and nominal input attributes. As indicated by the first two lines, there are eleven input strings, or columns. However, there are only 6 that are used for neural network trainng.

```
11
0       0       1       2       3       4       5       0       0       6       0
2004    1530    "0833"  "SQ"    "07030000000"   "1010"  "VTER"  0       0       0       0
2004    1520    "0833"  "SQ"    "07030000000"   "1010"  "VTER"  0       0       0       0
2004    1491    "0833"  "SQ"    "07030000000"   "1010"  "VTER"  0       0       0       0
2003    1460    "0833"  "SQ"    "07030000000"   "1010"  "VTER"  0       0       0       0
2003    1429    "0833"  "SQ"    "07030000000"   "1010"  "VTER"  0       0       0       0
2003    1399    "0833"  "SQ"    "07030000000"   "1010"  "VTER"  0       0       0       0
2003    1368    "0833"  "SQ"    "07030000000"   "1010"  "VTER"  0       0       0       0
2003    1338    "0833"  "SQ"    "07030000000"   "1010"  "VTER"  0       0       0       0
2003    1307    "0833"  "SQ"    "07030000000"   "1010"  "VTER"  0       0       0       0
```

The corresponding `<inputLayer>` element of the XML neural network description file for these data is described here.

```
<inputLayer n_inputs="6">
   <inputNode id="1" activation="linear" label="X1" type="nominal">
      <attributeDescription>Basic Symbol Number</attributeDescription>
      <perceptronLink hiddenLayer="1" perceptron="1"/>
      <perceptronLink hiddenLayer="1" perceptron="2"/>
      <perceptronLink hiddenLayer="1" perceptron="3"/>
      <nominalClass n_classes="3" transform_method="binary" scale_method="1"/>
   </inputNode>
   <inputNode id="2" activation="linear" label="X2" type="nominal">
      <attributeDescription>Program Director</attributeDescription>
      <perceptronLink hiddenLayer="1" perceptron="1"/>
      <perceptronLink hiddenLayer="1" perceptron="2"/>
      <perceptronLink hiddenLayer="1" perceptron="3"/>
      <nominalClass n_classes="19" transform_method="binary" scale_method="1"/>
   </inputNode>
   <inputNode id="3" activation="linear" label="X3" type="nominal">
      <attributeDescription>Army Management Structure</attributeDescription>
      <perceptronLink hiddenLayer="1" perceptron="1"/>
      <perceptronLink hiddenLayer="1" perceptron="2"/>
      <perceptronLink hiddenLayer="1" perceptron="3"/>
      <nominalClass n_classes="50" transform_method="binary" scale_method="1"/>
   </inputNode>
   <inputNode id="4" activation="linear" label="X4" type="nominal">
      <attributeDescription>Element of Resource</attributeDescription>
      <perceptronLink hiddenLayer="1" perceptron="1"/>
      <perceptronLink hiddenLayer="1" perceptron="2"/>
      <perceptronLink hiddenLayer="1" perceptron="3"/>
      <nominalClass n_classes="206" transform_method="binary" scale_method="1"/>
   </inputNode>
   <inputNode id="5" activation="linear" label="X5" type="nominal">
```

```
      <attributeDescription>Management Decision Package</attributeDescription>
      <perceptronLink hiddenLayer="1" perceptron="1"/>
      <perceptronLink hiddenLayer="1" perceptron="2"/>
      <perceptronLink hiddenLayer="1" perceptron="3"/>
      <nominalClass n_classes="25" transform_method="binary" scale_method="1"/>
   </inputNode>
   <inputNode id="6" activation="linear" label="Obligations" type="continuous">
      <attributeDescription>Obligations</attributeDescription>
      <continuousAttribute continuous_type="time" scale="bounded" n_lags="1">
         <realLimits lowerRealLimit="-105000" upperRealLimit="25000000"
lowerTargetLimit="0.0" upperTargetLimit="1.0"/>
         <statLimits center="1905.805" spread="2825.55"/>
         <lag lagvalue="1" />
      </continuousAttribute>
   </inputNode>
</inputLayer>
```

In this example, there are 5 nominal inputs and one variable, variable 6, with `continuous_type=time`. The nominal inputs are labeled: `X1, X2, X3, X4` and `X5`, and the time attribute is labeled as *`Obligations`*. All of the nominal inputs are transformed automatically into encoded columns of zeros and one using the `binary` or `1-in-C encoding` method for each of the five nominal input attributes. Notice that the number of classes or categories for each nominal attribute is also listed using the `n_classes` attribute.

The neural network training and forecasting applications automatically transform these raw input columns into columns of zeros and ones, one for each category, contributing a total of 3+19+50+206+25=303 input attributes. In addition, variable 6 is a time variable with `n_lags=1`. This indicates that the neural network applications should treat this column as both an input and output attribute. The output attribute will be the scaled values of the time series. An additional input column will be automatically constructed using the first lag of the time series.

This results into a total of 303+1=304 input columns that are automatically generated and scaled by the neural network training applications.