AUTOMATIC NEURAL-NET MODEL GENERATION AND MAINTENANCE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the priority of U.S. application no. 10/374,406, filed February 26, 2003 and entitled "AUTOMATIC NEURAL-NET MODEL GENERATION AND MAINTENANCE", and the benefit of the following commonly assigned, co-pending U.S. provisional applications:

(a) Serial No. 60/374,064, filed April 19, 2002 and entitled "PROCESSING MIXED NUMERIC AND/OR NON-NUMERIC DATA";

(b) Serial No. 60/374,020, filed April 19, 2002 and entitled "AUTOMATIC NEURAL-NET MODEL GENERATION AND MAINTENANCE";

(c) Serial No. 60/374,024, filed April 19, 2002 and entitled "VIEWING MULTI-DIMENSIONAL DATA THROUGH HIERARCHICAL VISUALIZATION";

(d) Serial No. 60/374,041, filed April 19, 2002 and entitled "METHOD AND APPARATUS FOR DISCOVERING EVOLUTIONARY CHANGES WITHIN A SYSTEM";

(e) Serial No. 60/373,977, filed April 19, 2002 and entitled "AUTOMATIC MODEL MAINTENANCE THROUGH LOCAL NETS"; and

(f) Serial No. 60/373,780, filed April 19, 2002 and entitled "USING NEURAL NETWORKS FOR DATA MINING".

TECHNICAL FIELD

This application relates to neural nets. In particular, the application relates to neural net model building and maintenance.

DESCRIPTION OF RELATED ART

Humans use their abilities of pattern recognition in many things they do, and particularly to solve problems. The following are some examples of how people use pattern recognition to anticipate and/or detect problems and find solutions to the problems:

(a) an experienced manager, based on her experience and knowledge of customers' buying patterns and her observations of current conditions (for example, weather, day of the week, date, local economy, etc.), predicts the number of units of a merchandise that should be ordered for the upcoming month; and

(b) a brewmaster samples his product over time and his intuition and experience suggests to him changes to the ingredients or process he should make to improve the product.

Artificial neural network ("neural net") techniques provide an information processing tool, with similar pattern recognition capabilities, which may be trained to provide an input-output model for assorted applications.

A typical neural net comprises a number of interconnected neuron-like processing elements (or nodes) that send data to each other along connections. A processing element receives a number of inputs, either from other processing elements or directly from inputs of the network, and multiplies each of the inputs by a corresponding weight and adds the results together to form a weighted sum. It then applies a transfer function (also referred to herein as "activation function" and "basis function") to the sum to obtain a value known as

the state of the element. The state is then either passed on to one or more other processing elements along weighted connections, or provided as an output of the network. Collectively, states are used to represent information in the short term, while weights represent long-term information or learning.

Processing elements in a neural net may be organized into layers. For example, a multi-layer hidden-layer net has an input layer, an output layer and one or more hidden layers between the input layer and output layer. The outputs of the input layer are passed to one of the hidden layers. Generally, hidden layer processing elements allow the network to build intermediate representations which combine input data in ways that help the neural net model to learn the desired input-output mapping with greater accuracy through training. Outputs of the hidden layers are passed to the output layer, and the output layer produces one or more outputs.

Training is a process through which neural nets learn an input-output model through exposure to data and adjustment of the weights associated with connections between processing nodes. A training process may involve the following steps:

1) Repeatedly presenting examples of a particular input/output task to the neural net model;
2) Comparing the model output and a desired output to measure error; and
3) Modifying model weights to reduce the error.

This process is repeated until further iteration fails to decrease the error (or the error falls below a predetermined minimum). The network then is said to be

"trained".

Through training with, for example, training sets of sample data, neural nets can learn to extract relationships from the data, similar to the way that humans learn from experience, and, when in operation (often also called "consultation"), recall the learned relationships in order to extrapolate suitable solutions in new situations (for example, not expressly represented in the training data). Training a neural net by applying sets of specific, selected samples helps the network to develop a general input-output model. The trained model is expected to output for each input pattern one or more output values associated with the input pattern, while maintaining the appearance of a blackbox (i.e. the details or inner workings, such as weights and nodes, within the trained model are not readily apparent to a user or observer).

Selecting an appropriate net structure plays a substantial role in building a neural net computational model of a functional relationship or system. If it is assumed that no prior knowledge of the problem is known and therefore only commonly-used node activation functions are used, the issues in the neural net generation process include the following. First, a net type (for example, layers in the net) are selected. Once the net type is selected, one determines an appropriate number and connectivity of nodes in the net as well as node parameters.

FIG. 1 shows a plot of training error versus number of nodes (ranging from one to twenty) in a hidden layer of a conventional single hidden-layer net, for a sample

data set of seven inputs and one output. As exemplified in FIG. 1, there is often no observable relationship between the training error and the number of hidden-layer nodes. The number of nodes used in a conventional neural net model typically is determined by experience and trial-and-error, which of course is not suitable for automatic model building.

In addition, conventional neural nets are not conducive to incremental and/or adaptive learning. The term "incremental learning", as used herein, means that (a) the net can be expanded with new nodes added and (b) computation of a new set of weights for the expanded net utilizes the weights from before the expansion as a starting point rather than starting from scratch. The term "adaptive learning", as used herein, means that after a neural net model has been established, additional data can be used to update the model to achieve better overall results. For example, neither incremental learning nor adaptive learning can be achieved efficiently by a hidden-layer net because the nonlinear processing by a hidden-layer net is widely distributed and interconnected across the nodes, and therefore any adjustments to the weights based on determined error also must be nonlinearly distributed.

Therefore, neural net model generation and maintenance methodologies which facilitate incremental and adaptive learning are needed.

SUMMARY

The disclosure provides a method of incrementally forming and adaptively updating a neural net model. According to one embodiment, the method includes (a) incrementally adding to the neural net model a function approximation node, and (b) determining function parameters for the function approximation node and updating function parameters of other nodes in the neural network model, by using the function parameters of the other nodes prior to addition of the function approximation node to the neural network model. Steps (a) and (b) may be repeated, if a model accuracy of the neural net model with the function approximation node added thereto is below a predetermined accuracy level.

According to one embodiment, a set of sample data patterns is used to form a list of function approximation node candidates, and the function approximation node is selected from the list of function approximation node candidates. The list of function approximation node candidates may be formed by splitting the set of sample data patterns into a plurality of clusters in a first level of a cluster hierarchy, determining that a selected cluster in the first level has a population exceeding a predetermined size, and splitting the selected cluster into two or more clusters and replacing the selected cluster with the two or more clusters in a next level of the cluster hierarchy. The clusters on each level of the cluster hierarchy based on cluster size, to form a sorted list of function approximation node candidates.

The function parameters for the nodes in the neural

net model may be determined by applying a hierarchical k-means clustering methodology to a set of sample data patterns. For example, a function approximation node may be a radial basis node, and a center and radius of the radial basis node are determined through a hierarchical k-means clustering methodology.

The method may further include monitoring a model accuracy of the neural net model while the neural net model is used on-line, and adaptively updating the neural net model, if the model accuracy of the neural net model is below a predetermined threshold. The adaptive update may include incrementally adding one or more additional nodes to the neural net model, to represent new data. The new data may correspond to a data range not represented in the set of sample data patterns and/or to a change in system dynamics. The adaptive update may include updating the function parameters of the nodes in the neural net model. If the adaptive updating reaches a limit, a full retrain of the neural net model may be performed.

The additional nodes, according to one embodiment, may be formed by applying a clustering methodology to new data patterns. The clustering methodology may include, for example, clustering the new data patterns into a number of clusters which is approximately a number of the nodes in the neural net model, determining that a selected cluster is far away from positions associated with the respective nodes in the neural net model, and adding to the neural net model an additional node associated with the selected cluster and a center of the selected cluster.

An orthogonal least squares methodology may be applied to determine a set of weights for the neural net model. The set of weights may be adaptively updated by using new data patterns and/or to compensate for system drift. The weights of the nodes in the neural net model prior to the adaptive update may be combined with a set of new weights based on a forgetting factor. The forgetting factor may be determined based on a cause of model degradation.

The present disclosure also provides a method of incrementally forming a neural net model. In one embodiment, the method includes applying a hierarchical clustering methodology to a set of sample data patterns to form a list of function approximation node candidates, and incrementally applying function approximation nodes from the list of function approximation node candidates to form a model with an accuracy at or above a selected accuracy level.

According to another embodiment, the method includes forming a plurality of function approximation nodes for the neural net model by applying a hierarchical clustering methodology to a set of sample data patterns, and applying an orthogonal least squares methodology to determine a set of weights associated with the function approximation nodes.

BRIEF DESCRIPTION OF THE DRAWINGS

The features of the present disclosure can be more readily understood from the following detailed description with reference to the accompanying drawings wherein:

FIG. 1 shows a graphical representation of a plot of training error versus number of nodes in a hidden layer of a single-layer hidden-layer net;

FIG. 2A shows a flow chart for a method of incrementally forming a neural net model, in accordance with one embodiment of the present disclosure;

FIG. 2B shows a flow chart for a hierarchical clustering methodology, in accordance with one embodiment;

FIG. 2C shows a flow chart for a method of generating an ordered list of candidate node functions, according to one embodiment, using the hierarchical clustering methodology shown in FIG. 2B;

FIG. 2D shows a flow chart for a method of incrementally constructing a model, according to one embodiment, using a list of candidate node functions;

FIG. 3A shows a flow chart for a method of incrementally forming and adaptively maintaining a neural net model at an adequate accuracy level, according to one embodiment of the present disclosure;

FIG. 3B shows a flow chart for a method of adaptively updating a model to maintain accuracy of the model, according to one embodiment, using new data;

FIG. 4 shows a schematic view of a functional link net structure;

FIG. 5 shows a plot of data patterns and outputs produced by a model generated by applying methodologies provided by the present disclosure, for a non-linear time series example;

FIG. 6 shows a plot of data patterns and outputs produced by a model, generated and updated by applying

methodologies provided by the present disclosure, for a non-linear time series (with drift) example.

DETAILED DESCRIPTION

This disclosure provides tools (in the form of methodologies and systems) for neural net model generation and maintenance. Novel incremental and/or adaptive methodologies for efficiently building adequately accurate neural net models of mappings learned through training and for maintaining the accuracy of the models are provided. The incremental methodologies described herein provide efficiency, as compared with other approaches, and the adaptive methodologies are exercised in response to changes in a nature of the data or in the system response.

In accordance with one embodiment, efficient incremental improvement of a model encompasses addition of nodes in an appropriate manner and computation of improved system model parameters recursively from the previously learned model, which is thus improved incrementally.

In accordance with another embodiment, incremental learning includes adding nodes from an ordered candidate list sequentially based on guidance provided by hierarchical clustering. The architecture of the hierarchical clustering may be binary beyond the first level and may have a maximum number of levels. The candidate list may be generated by sorting the clusters first by level and then by cluster size. Nodes are selected from the sorted list starting with functions from top-level clusters. If accuracy of the model is not

adequate, more nodes are added sequentially until all nodes in the candidate list are exhausted or until a desired accuracy is obtained. This methodology may be applied to obtain a model with adequate accuracy and having a moderate size.

Adaptive learning, in response to changes either in system dynamics or to range of data or both, includes, in accordance with one embodiment, a simple methodology for adjusting the network parameters and/or structure, without having to undergo a complete retrain. Under this methodology, clustering of new data is used to determine if new nodes should be added, and the top most linear weights for the existing nodes and for any new nodes generated are computed by using only the new data in combination with existing weights in the net, according to a "forgetting factor" based on a determination of a cause of degradation in performance of the original model and also on number of patterns used for training.

An embodiment of the present disclosure may use a combination of hierarchical clustering, radial basis function and linear orthogonal least squares methodologies to provide incremental model building and adaptive maintenance.

An exemplary method for incrementally forming a neural net model, in accordance with one embodiment of the present disclosure, is described with reference to FIG. 2A. A hierarchical clustering methodology is applied to a set of sample data patterns to form a list of function approximation node candidates (step S21). Function approximation nodes selected from the list of function approximation node candidates are incrementally

added to the neural net model to form a model with an accuracy at or above a selected accuracy level (step S22).

The method of incrementally forming a neural net model, according to another embodiment, includes applying a hierarchical clustering methodology to a set of sample data patterns to form a list of function approximation node candidates, and incrementally adding to the neural net model one or more function approximation nodes selected from the list of function approximation node candidates, until a model with an accuracy at or above a selected accuracy level is obtained. The function approximation node candidates may include Gaussian nodes, sigmoidal basis nodes, wavelet basis nodes, etc. The nodes may be non-linear.

A hierarchical clustering methodology, according to one exemplary embodiment, is described with reference to FIG. 2B. The set of sample data patterns is split into a plurality of clusters in a first level of cluster hierarchy (step S211). Select a cluster in current (first) level and compare the population of the cluster to a predetermined size threshold (step S212). If the population of the cluster exceeds the threshold (step S213), the cluster is split into and replaced with two or more clusters in a next level of the cluster hierarchy (step S214). The process is repeated until no clusters exceeding the size threshold remain (step S215).

A method of generating an ordered list of candidate node functions, according to one exemplary embodiment (FIG. 2C), may use the methodology of FIG. 2B to develop a cluster hierarchy. When all remaining clusters are

moderate-sized (e.g., population is below threshold) [steps S212-S215], a list of candidate node functions may be generated by sorting the clusters on each level of the cluster hierarchy, based on cluster size (step S216).

A method of incrementally constructing a moderately-sized model with adequate accuracy, in accordance with one embodiment (FIG. 2D), may use a list of candidate node functions generated by applying, for example, the methodology shown in FIG. 2C. If there are any node functions on the candidate list (step S221), the first node function on the list is selected and added to the model (step S222). New weights are computed, along with a system error (or other accuracy indicator (step S223). If the accuracy of the model is not adequate (step S224), the process returns to step S221 to process any candidate node functions remaining on the list.

A method, in accordance with another embodiment, for incrementally forming a neural net model off-line and adaptively maintaining the neural net model at an adequate accuracy level and a moderate size is described with reference to FIG. 3A. A function approximation node is incrementally added to the neural net model (step S31). Function parameters are determined for the function approximation node and function parameters of other nodes in the neural net model are updated, by using new data and the existing function parameters of the other nodes prior to addition of the function approximation node (step S32). Function approximation nodes may be added to the neural net model (steps S31 and S32) until the model has an adequate accuracy (see, for example, FIG. 2D). The model formed off-line may be

deployed and used on-line (step S33). Accuracy of the model is monitored as new data is fed as input to the model (step S34). If the model is not adequately accurate (for example, meet a minimum level of accuracy) [step S35], model parameters and/or structure may be updated adaptively (step S36). The adaptive update may be necessitated, for example, by a change in system dynamics or by drift in the system. The neural net model may have poor accuracy for new data that is not represented by (for example, far away from) any of the clusters of sample data patterns which correspond to the function approximation nodes. In this latter instance, the adaptive update may use the new data to add additional nodes to the model. See, for example, FIG. 3B and corresponding discussion below.

If the limit for adaptive updating is reached (for example, accuracy is not improved through adaptive update) [step S37], preparation for full off-line retrain is performed (step S38), and then the process restarts at step S21. On the other hand, if the adaptive update improves the accuracy of the model (step S37), the adaptively updated model is redeployed and used on-line (step S33) and accuracy of the updated model is monitored (step S34).

A method of adaptively updating a model to maintain accuracy of the model (for example, step S36), according to one embodiment of the present disclosure (FIG. 3B), may be applied in response to, for example, new data corresponding to change in system dynamics and/or in range of data. The new data is clustered into roughly the same number of clusters as the number of current

nodes in the model (step S361). For clusters that are far away from the current nodes in the model, corresponding new nodes are added at the positions of the cluster centers (and weights on output links of the new nodes may be initialized to zero) [step S362]. A set of new weights is computed for all (current and new) nodes based on the new data (step S363). The new weights are combined with the existing weights of the existing nodes by applying a forgetting factor (step S364). The forgetting factor may be selected based on, for example, a cause of degradation in model performance and/or on a number of training patterns.

Function parameters associated with the nodes in the neural net model may be generated by using a hierarchical k-means clustering methodology. For example, the nodes may include radial basis nodes, and the centers and radii of the radial basis nodes are determined through the hierarchical k-means clustering methodology.

Some neural nets may be trained to model or approximate a functional relationship between input and output, without requiring training with every possible input pattern. A relationship between input and output is described as "functional" to signify that the input-output relationship can be approximated by a mathematical function, for example, for each input pattern the input pattern has only one associated output value (e.g., if inputs x and y are 4 and 5, output z is always 9; if x and y are 94 and 73, z is always 26; etc.).

The functional relationship may be linear or non-linear. Linearity is a mathematical property (applicable, for example, to a relationship between input

and output of a function) that output (y) is proportionally related to input (x) [for example, y=2x, y=5-4x, etc.], such that a small/large change in input produces a corresponding small/large change in output. Once it is known that a problem embodies a linear relationship between variables, the linear factor may be determined numerically, empirically and/or methodically through well-known methods. However, in real life, the relationship between independent variables in a mathematical problem is typically non-linear (i.e. there is not such a fixed ratio). Therefore, in order for a neural net to be a universal approximator (which means that a network can be used to approximate any function to arbitrary precision when enough nodes are provided in the net), the neural net should be capable of modeling non-linear input-output relationships. Some neural nets may be trained to capture nonlinearity and interactions among independent variables automatically without pre-specification.

A functional-link net ("FLN") is one type of neural net which can be used to model a functional relationship between input and output. A FLN may be used to approximate any scalar function with a vector of inputs, $\boldsymbol{x}$, and an output y. The structure of a FLN with non-linearity fully contained in a functional-link layer is illustrated in FIG. 4. The nodes in the functional-link layer have associated non-linear basis functions. Examples of FLNs are described in commonly owned United States Patents Nos. 4,979,126, 5,734,796, 6,134,537 and 6,212,509 which are incorporated herein in their entirety by reference. Since a vector function may be decomposed

into scalar dimensions, and therefore may be approximated with multiple output nodes or multiple nets, the discussion of FLN in this disclosure focuses on the case of one output node, such as shown in FIG. 4, without loss of generality to cover the typical circumstance in which there are multiple output nodes.

Hidden-layer nets and FLNs are two types of neural nets that can serve as universal approximators. However, a location of the non-linearity is different for the two types of nets. For a multi-layer hidden-layer net, the activation function for the hidden-layer nodes typically is non-linear. Although all the weights are linear weights, any methodology used to train such a net has to be non-linear. On the other hand, when a FLN is used, non-linearity may be fully contained in the functional-link layer, and the rest of the net may be linear. Therefore, linear training techniques such as regression-based training may be used with a FLN structure. Linear training refers to techniques that solves the parameters in the net through linear algebra techniques.

Although both types of nets (i.e., multi-layer hidden-layer net and functional-link net) may be universal approximators, how many nodes is enough varies according to the data and/or the problem. Further, since the initial set of weights and/or function parameters in the net are usually randomly generated, the resulting trained net is usually just a local minimum in the error space (e.g., the associated error appears to be at a minimum). A result of a trained net being at a local minimum is that adding one more node may not reduce the model error at all, if one does not try a significant

number of different initial weights and/or sets of parameter values. This is less of a problem for the linearly trainable FLN, unless the randomly generated parameter values cause the linear problem to be close to singular (e.g., with an undefined mathematical derivative at some point in the problem space). Therefore, the problem of training error settling to a local minimum is much more noticeable and likely to occur with hidden-layer nets.

The neural net model generation and maintenance methodologies of the present disclosure (referred collectively herein as "orthogonal functional-link net methodologies" or "OFLN methodologies") may be applied to generate, in accordance with a preferred embodiment, an efficient, high-performance function approximation neural net. The OFLN methodologies also include provisions for maintenance of the subject net so that the net can be automatically updated in accordance with data obtained from the system being modeled.

A FLN with linear output nodes is selected as the net type under the OFLN methodologies. An advantage of a FLN is that linear regression training techniques, such as an orthogonal least squares (OLS) learning methodology (discussed below), can be used to achieve incremental and adaptive learning.

Under the OFLN methodologies, the appropriate number of function approximation nodes are considered together with the parameters of the nodes. For radial-basis function approximation nodes (discussed below), the parameters include the location of the center and for some cases the effective radius. One may use a heuristic

radius in combination with random centers, or place centers on randomly selected existing patterns. A drawback, however, is that it may be difficult to determine the quality of the parameters in terms of their ability to represent the available data. Therefore, multiple trials and/or user experience, i.e. heuristics specific to a problem, may be necessary to arrive at a good model. Under the OFLN methodologies, the data speak for themselves, that is, the candidate center and radii values are generated through hierarchical clustering (discussed below).

Combining the advantages of radial-basis FLN, hierarchical clustering and the orthogonal least squares methodology, the OFLN methodologies may be applied to provide automatic generation of suitable models of a system if a set of training data is available. The model created by the OFLN methodologies may also be updated adaptively. The combination of effective model building, which may be performed periodically off-line as new data accumulate, and adaptive model updating, which may be carried out on-line as new data are available, provide the tools for maintaining optimal performance by the model.

The OFLN methodologies provide a number of features, including the following, which make the methodologies particularly suited for solving new and complex real-world problems:

(a) Learning: the subject methodologies develop solutions by extracting relationships from data, in a manner analogous to the way we learn from experience;

(b) Multi-dimensional: under the subject

methodologies all the features of a problem may be considered at once, in parallel, in contrast to the human ability to analyze a finite number of data streams at once, and conventional sequential algorithms, which may require a great deal of complex programming to solve a problem that has many features;

(c) Non-linear: the subject methodologies are not bound to conventional modeling or problem solving techniques which require that some knowledge of the underlying nature of the problem is known or assumed in advance and the solution limited to that form; and

(d) Adaptive: the model may readily be updated to accommodate new data, continuously improving its knowledge of the problem.

Orthogonal Least Squares Methodology

The orthogonal least squares (OLS) learning methodology is an incremental learning methodology.

A FLN may be trained to approximate the following scalar function, if a set of observed associated input-output pattern pairs $\{(\boldsymbol{x}_p, y_p)\}$ is provided, where p = 1, ..., P (P being the number of pattern pairs):

$$y = y(\boldsymbol{x}) \qquad (1)$$

Using the net illustrated in FIG. 4, the following linear sum of a set of non-linear basis functions, $f_j(\boldsymbol{x})$ where j = 1, ..., J (J being the number of nodes), may be used to represent the approximation for the function in Equation (1):

$$y(\boldsymbol{x}) = \sum w_j f_j(\boldsymbol{x}) \qquad (2)$$

Since Equation (2) is an approximation, there may be in addition an error term on the right hand side of the equation to make it a true equality. However, the error term is dropped in Equation (2) in the interest of clarity. Although radial basis functions such as Gaussians are frequently selected as $f_j(x)$ in Equation (2), other functions, such as sigmoids or wavelets, may also be used. Substituting the known patterns into Equation (2), P simultaneous equations are obtained. Since P (i.e., the number of pattern pairs) is usually much larger than J, (i.e., the number of function approximation nodes), the problem is over-specified and a solution can only be obtained in the sense of least sum of squares-of-error, or least squares.

If a training methodology based on least squares is used to train the net, the larger the number of basis functions used, the smaller the training error results, assuming the selected basis functions do not cause the resulting linear system to be nearly singular. However, the goal of course is to obtain an approximation that is truly representative of the implicit functional relationship. The trained net may be tested with the aid of a validation set (for example, a test set distinct from the training set) of patterns. Small errors for the training set patterns alone are not a satisfactory result of the learning process if a test with the validation set reveals that the learned relationship is not widely valid across the validation set (e.g., the learned relationship does not yield an appropriate output when an input

pattern that is not in the training set is applied). Under such circumstances, incremental learning and/or adaptive learning techniques may be applied to refine the model.

The set of simultaneous equations obtained from Equation (2) may be written in matrix form, as follows:

$$\begin{bmatrix} f_1(x_1) & f_2(x_1) & \dots & f_J(x_1) \\ f_1(x_2) & f_2(x_2) & \dots & f_J(x_2) \\ . & . & & . \\ f_1(x_p) & f_2(x_p) & \dots & f_J(x_p) \\ : & : & & : \\ f_1(x_P) & f_2(x_P) & \dots & f_J(x_P) \end{bmatrix} \begin{pmatrix} w_1 \\ w_2 \\ : \\ w_J \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ . \\ y_p \\ : \\ y_P \end{pmatrix} \tag{3}$$

or

$$\mathbf{Fw} = \mathbf{y} \tag{4}$$

Each of the non-linear functions $f_j(x)$ are described in terms of parameters. Though these parameters may also be varied in the training process, they usually are pre-selected and remain constant, while only the linear weights are adjusted, during the training process for such a net structure.

Using a linear least squares technique, a solution for Equation (4) may be expressed as follows:

$$\mathbf{w} = \left(\mathbf{F}'\mathbf{F}\right)^{-1}\mathbf{F}'\mathbf{y} \tag{5}$$

However, in actual computation, the weight vector **w** is usually computed directly using singular value

decomposition (SVD) or LU decomposition of $\mathbf{F}^t\mathbf{F}$ rather than the pseudo-inverse technique corresponding to Equation (5). The computation is straightforward if **F** is fixed. A problem arises when **F** is augmented during training as in the case of adding function approximation nodes dynamically. Direct computing of **w** may require a whole new SVD or LU decomposition of the augmented $\mathbf{F}^t\mathbf{F}$, although the only change in **F** is an added column.

In order to handle augmenting of **F** during training, the OLS methodology provides an extra step of orthogonalization. The extra step allows most of the results from a computation before **F** is augmented to be reused after **F** is augmented in, for example, a recursive fashion.

In Equation (4), each row of the matrix **F** is a representation of a single pattern vector in terms of the J basis functions. Accordingly, the matrix **F** is a representation of an entire set of P pattern vectors in terms of the J basis functions. The matrix **F** can also be considered as a row of column vectors $\mathbf{f}_j$, as follows:

$$\mathbf{F} = [\mathbf{f}_1 \ \mathbf{f}_2 \ldots \mathbf{f}_J] \tag{6}$$

In the OLS methodology, a set of orthogonal vectors **h** may be built from the **f** vectors in the Gram-Schmidt manner as follows:

$$\mathbf{h}_1 = \mathbf{f}_1$$

$$\mathbf{h}_2 = \mathbf{f}_2 - C_{21}\mathbf{h}_1 \tag{7}$$

$$\mathbf{h}_k = \mathbf{f}_k - \sum_{i=1}^{k-1} C_{ki}\mathbf{h}_i$$

The coefficients C_{ki} are determined by the following orthogonalization condition:

$$<\mathbf{h}_i\mathbf{h}_k> = <\mathbf{h}_i\mathbf{f}_k> - C_{ki} <\mathbf{h}_i\mathbf{h}_i> = 0$$

so that

$$\mathbf{C}_{ki} = <\mathbf{h}_i\mathbf{f}_k> / <\mathbf{h}_i\mathbf{h}_i> \qquad (8)$$

With this notation, the matrix **F** may be rewritten as follow:

$$\mathbf{F} = \mathbf{HA} \qquad (9)$$

where **A** is an upper triangular matrix with the coefficients as its elements and ones on its diagonal.

Now Equation (4) may be rewritten as follows:

$$\mathbf{HAw} = \mathbf{y} \qquad (10)$$

By denoting

$$\mathbf{g} = \mathbf{Aw} \qquad (11)$$

the set of simultaneous equations is transformed to the following:

$$\mathbf{Hg} = \mathbf{y} \qquad (12)$$

The least squares solution for **g** is as follows:

$$\mathbf{g} = \left(\mathbf{H}'\mathbf{H}\right)^{-1}\mathbf{H}'\mathbf{y} \tag{13}$$

where $\mathbf{H}^t\mathbf{H}$ is the following diagonal matrix:

$$\mathbf{H}'\mathbf{H} = \begin{pmatrix} \mathbf{h}_1'\mathbf{h}_1 & 0 & \dots & 0 \\ 0 & \mathbf{h}_2'\mathbf{h}_2 & \dots & 0 \\ : & : & : & : \\ 0 & 0 & \dots & \mathbf{h}_k'\mathbf{h}_k \end{pmatrix} \tag{14}$$

and therefore $(\mathbf{H}^t\mathbf{H})^{-1}$ is as follows:

$$\left(\mathbf{H}'\mathbf{H}\right)^{-1} = \begin{pmatrix} \frac{1}{\mathbf{h}_1'\mathbf{h}_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\mathbf{h}_2'\mathbf{h}_2} & \dots & 0 \\ : & : & : & : \\ 0 & 0 & \dots & \frac{1}{\mathbf{h}_k'\mathbf{h}_k} \end{pmatrix} \tag{15}$$

If another **h** vector is added to the representation, the new **g** vector may be evaluated recursively as follows, which may be shown with straightforward linear algebra:

$$\mathbf{g}_{k+1} = \left[\mathbf{g}_k \frac{\mathbf{h}_{k+1}'\mathbf{y}_{k+1}}{\mathbf{h}_{k+1}'\mathbf{h}_{k+1}}\right]' \tag{16}$$

The solution for **w** is then as follows:

$$\mathbf{w} = \mathbf{A}^{-1}\mathbf{g} \tag{17}$$

Since **A** is an upper triangular matrix, the inverse matrix $\mathbf{A}^{-1}$ may also be computed recursively as follows:

$$\mathbf{A}_1^{-1} = [1]$$

$$\mathbf{A}_1 = [1]$$

$$\mathbf{A}_2 = \begin{pmatrix} 1 & C_{21} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{A}_1 & C_{21} \\ 0 & 1 \end{pmatrix}$$

$$\mathbf{A}_2^{-1} = \begin{pmatrix} \mathbf{A}_1^{-1} & -\mathbf{A}_1^{-1}C_{21} \\ 0 & 1 \end{pmatrix}$$

...

$$\mathbf{A}_k = \begin{pmatrix} \mathbf{A}_{k-1} & \mathbf{c}_k \\ 0 & 1 \end{pmatrix} \quad \text{where } \mathbf{c}_k = (C_{k1}, C_{k2}, ..., C_{k,k-1})$$

$$\mathbf{A}_k^{-1} = \begin{pmatrix} \mathbf{A}_{k-1}^{-1} & -\mathbf{A}_{k-1}^{-1}\mathbf{c}_k \\ 0 & 1 \end{pmatrix} \tag{18}$$

Using the new **g** vector and the $\mathbf{A}^{-1}$ matrix, the new weight vector may be obtained using Equation (17).

Thus, a FLN with non-linearity fully contained in a functional-link layer may be constructed incrementally by applying the OLS methodology. The OLS methodology provides a natural control on the number of nodes. As nodes are added to the net, the error of training generally reduces. The adding of nodes stops when the target of training is reached or when signs of over-

training are evident.

Hierarchical Clustering

The OLS methodology allows for easy addition of new function approximation nodes during the time of training. The next question is: what new node should be added, that is, what parameters should be used in the new node function.

Random choice is one technique. The random vector FLN uses randomly selected node parameters and applying the OLS methodology to it may be a natural extension of this type of net. However, randomly selected parameters may not provide adequate coverage of the data distribution, and a number of (random selection) trials may be desired to obtain a good model such as described in the ensemble net technique.

The OFLN methodologies, in accordance with one embodiment, uses radial basis function approximation nodes in the functional-link layer. An advantage of using radial basis functional-link layer nodes is that clustering methodologies may be used to generate the centers and radii.

A k-means clustering methodology may be used for determining the centers and radii. Since the number of clusters typically is unknown before training, hierarchical k-means clustering (discussed below) may be used to generate node parameters, in accordance with one embodiment. The hierarchical k-means clustering methodology is a divisive technique. The whole sample data set is first clustered into a small number of clusters. Depending on the population of the resulting

clusters, large ones may be split further into smaller clusters until the populations of the clusters are moderate or the number of levels in the cluster hierarchy exceeds a selected maximum. The limit on the number of levels is used to control the maximum complexity of the resulting model. The k-means methodology is used in each level of clustering.

There are several advantages to using hierarchical clustering versus single level clustering. For example, there is no need to guess the number of clusters to be generated or the radii of the clusters, as are required up front by some other clustering methodologies. In addition, different levels in the cluster hierarchy represent descriptions of data at different levels of detail. The different levels is quite important in building a model since, in order for the model to generalize well, it is better to have a small number of nodes in the net as long as the training error is acceptable. With different levels of detail, the net may start with a coarse description on the first level, and if the training error is not satisfactory, additional detail may be added using results from lower level clusters until the model is acceptable. This automatically results in models of modest complexity.

Under many other methodologies, the number of nodes is often chosen to be larger than necessary, in the interest of obtaining sufficient detail. Either forward selection or backward elimination, however, may be desired to control complexity. With hierarchical clustering, the process is comparable to forward selection. However, the number of candidates is much

smaller at any stage since only clusters in the same or next level may be considered. For a small value of k (for example, two) in generating next-level clusters, the whole selection process may be eliminated since simply adding the candidate which corresponds to the cluster with the largest population among the remaining same-level or next-level clusters achieves performance similar to that obtained with forward selection. This simplification often may significantly reduce the training time.

Adaptive Model Update

Even under the best circumstances, a model is only an approximation of the underlying functional relationship or system during a period of time when the data with which the model was trained was collected. There are many factors which contribute to a motivation to update the model.

If the same inputs are provided at different times, a model should return the same computed output values. However, the value is not necessarily representative of the functional relationship or system being modeled. Noise in observation typically causes the old computed value to be somewhat different from the newly observed one.

The model also may be obsolete. In the use of a mathematical model of a system, the focus is on the inputs, and system parameters are considered to be constant. However, after the model is established, the system parameters may drift and cause the underlying system behavior to shift. For example, the further a

driver steps on a brake pedal of a car, the faster the car stops, and a model may be built to predict how fast the car stops for a particular distance the brake pedal moves. However, as the brake pads wear thin and brake fluid ages over time, the same amount of travel in the brake pedal results in the car stopping slower than before.

It is possible to compensate for minor changes due to noise or system drift, by updating some appropriate parameters, for example, the weights in the FLN model. Newly available data may help to cancel noise in previous training data or to bring the model more up to date. In practice, it is difficult to determine whether the cause is noise or drift since normally both exist and it is likely that noise effects are dominant in the short term while drift might be responsible for secular effects and is dominant for a longer term.

Another situation in which updating the model is desired is when novel cases are identified in newly available data. For cases of novel data, changing weights associated with existing nodes may not be enough to represent the effects of the novel data. Adding new nodes associated with data regions in which the novel cases exist may be used to address this problem. Again, clustering may be used to determine the position and spread of the new node function. With centers from clustering results, it is easy to identify patterns that fall outside of known regions.

While it is difficult to perform adaptive learning for the conventional multi-layer hidden-layer net structure due to the non-linearity within the training

methodology, the adaptive update for a FLN with non-linearity contained in the functional-link layer, such as the net generated by the OFLN methodologies, may be carried out as follows.

If a set of newly obtained associated pattern pairs $\{(\mathbf{x}'_p, y'_p)\}$ is provided, in which $p = 1, \ldots, P'$, and P' is much smaller than the original number P of patterns in the training set, a new F' matrix may be obtained using the same radial-basis function approximation nodes supplemented with additional nodes as warranted. A least squares solution **w'** equivalent to Equation (5) may be obtained. Assuming **w** represents the current weights in the model, with zeros for links from the additional nodes, the new weights w_{new} may be obtained using the following equation:

$$\mathbf{w}_{new} = (1-\alpha)\mathbf{w} + \alpha\mathbf{w}' \tag{19}$$

where **α** is a parameter between 0 and 1. Determination of **α** is based on several parameters, such as the number of patterns in the new set compared with the previous training set, confidence level in the newly available set versus the previous one, the estimated rate of change in the underlying system and the time elapsed since the model was first established. One way to compute the parameter **α** is to use the following equation:

$$\alpha = \frac{P'}{(1-\lambda)P + P'} \tag{20}$$

where P and P' are the numbers of patterns in the

training set used to train the current model and in the newly available data set, respectively. The parameter λ is a forgetting factor. Its value is also within [0, 1]. The higher the value of λ, the less effect the existing weights $\mathbf{w}$ have on the new weights $\mathbf{w}_{new}$. In other words, the current weights are forgotten to a greater extent. The advantage of introducing λ is that it separates the tangible part (i.e., the number of patterns) from the intangible parts of the determination process for the parameter α.

Although adaptive learning may help to reduce differences caused by both noise and drift, there is a distinction between the two causes. To obtain a desired response, the input parameters may still be in the same neighborhood for differences caused by zero-mean noise but may be progressively different for differences caused by drift. In addition, the desired operations to update the model are also different. For the noise case, the parameters in the existing model are treated with equal importance. In contrast, for drift, they need to be forgotten. The parameter λ is provided for this purpose in Equation (20).

For adaptive updating of model parameters to work, the general underlying principle in the functional relationship or model may still be valid and the amount of change may be small. If these conditions are not true, a completely new model may be established. Even if the change is gradual, the amount of change may become significant over time. Therefore, adaptive updating is best for a short-term solution. As new data accumulates, a full retrain may be conducted, periodically or when $\mathbf{w}_{new}$

is sufficiently different from the original **w**. A difference between adaptive update and full retrain is that adaptive update keeps all existing nodes and only adjusts the weights for them, while for full retrain all function approximation nodes (and weights) are newly generated. Full retrain may provide better performance but is also much more time consuming. Performing an adaptive update to an on-line model frequently coupled with installing a fully retrained model from off-line periodically may be an effective technique to ensure that the model is always up to date.

Whether previous training data may be archived is a trade-off issue. The adaptive updating does not require any of them. Whether a full retrain uses them depends on whether the retrain is called for by large amount of new data or by large difference in weights. For the latter case (i.e. large difference in weights) with a small amount of new data, retrain with all or some of the previous training data may be desired. However, for fast changing situation, dropping old data may yield better results. One technique that may work sufficiently well for most situations is to keep a fixed amount of data, dropping old ones as new ones are available.

Examples

A simple non-linear time series example is discussed below to illustrate a process of automatic model generation and updating by using OFLN methodologies. The example may apply exemplarily to network performance and positioning of network centers.

A non-linear times series is simulated by the

following equation:

$$y(t) = [0.8 - 0.5\exp(-y^2(t-1))]y(t-1) - [0.3 + 0.9\exp(-y^2(t-1))]y(t-2) + 0.1\sin(\pi y(t-1)) + e(t) \quad (21)$$

where e(t) is a zero mean noise sequence with variance 0.01. The initial conditions were set as $y(0) = 0.1$ and $y(-1) = 0$. The previous two points are used to predict the value of the current point.

Two thousand samples of the time series were generated. The first 1000 patterns were used as the training set and the remaining 1000 were used as the validation set. Gaussians were used as the function approximation nodes. The centers were determined by using cluster centers and the spreads were determined by using the radii of the clusters. Hierarchical k-means clustering up to 3 levels with 3 clusters at the first level and binary split at the next two levels were used. The hierarchical clustering resulted in a total of 21 clusters. The clusters were sorted based on ascending level and descending population. The net started with 3 nodes corresponding to the top-level clusters, and nodes were added from the list of clusters sequentially. For an error target of 5e-4, a total of 12 nodes were selected. An error for the validation set was also less than 5e-4, indicating good generalization capability of the resulting model.

FIG. 5 shows the training patterns, a noise-free limit cycle, a limit cycle produced by the model when the

model output was fed back to the input, and positions of cluster centers from different levels of clustering, for a simulated non-linear time series and results from the FLN model. The small dots correspond to training patterns. The gray loop corresponds to the noise-free limit cycle. The black loop corresponds to the limit cycle produced by the FLN model when the output was fed back to the input. The large black circles correspond to the positions of the centers of the first level clusters. The triangles correspond to the second level clusters. The diamonds correspond to selected third level clusters. The selected centers appear to be at strategic positions and the limit cycle produced by the model agrees well with the noise-free system limit cycle.

To illustrate the process of maintaining optimal model performance through a combination of adaptive model update and periodic retraining of the model, a constant drift term is added to the non-linear time series system corresponding to Equation (21), as follows for $t > 1000$:

$$\begin{aligned} y(t) = & [0.8 - 0.5\exp(-y^2(t-1))]y(t-1) \\ & - [0.3 + 0.9\exp(-y^2(t-1))]y(t-2) \\ & + 0.1\sin(\pi y(t-1)) + 0.25 + e(t) \end{aligned} \qquad (22)$$

Two additional training data sets containing 500 patterns in each set were generated. The error of consulting these patterns using the original model was 0.00326. Adaptive learning was applied with each training

set sequentially. The forgetting factor used was 0.5 in both cases. The resulting errors of the modified models corresponding to the training data sets were 0.00114 and 0.000713, respectively. A new model was also built by using both new training data sets to simulate the periodic retrain. Again, for an error target of 5e-4, only 12 new nodes were used.

FIG. 6 shows limit cycles produced by the models at different stages when the output was fed back to the input as compared to the noise-free limit cycle for the drifted system corresponding to Equation (22). The gray dot loop corresponds to the noise-free limit cycle. The solid gray loop corresponds to the limit cycle produced by the original model. The solid black loop corresponds to the limit cycle produced after the first adaptive learning. The asterisk dot loop corresponds to the limit cycle produced after the second adaptive learning. The plus dot loop corresponds to the new model after full retrain with new data.

From FIG. 6, it is evident that adaptive learning may be used to correct the model parameters so as to bring the model outputs progressively closer to the target. However, as new patterns accumulate to warrant a retrain, the new model performs better than the adaptively updated model since the retrained model is not affected by the old parameters in the original model. In addition, centers may be tuned to the new data set.

Therefore, an advantage of the OFLN methodologies over other existing methodologies, such as backpropagation, is that OFLN methodologies may be used to generate candidate processing nodes automatically and

efficiently by utilizing knowledge of where the data points are situated. The OFLN methodologies may also be used to dynamically adjust the number of processing nodes to maintain or improve the fidelity of the function approximation, without user intervention. The functional-link net structure facilitates adaptive learning, through which the FLN model may successfully perform the tasks of learning with noisy training data, predicting a value of a current observation based on previous ones, and maintaining accuracy in a presence of drift in the underlying functional relationship or system (e.g., signal generation mechanism).

Applications of the OFLN Methodologies

Neural net models formed and maintained through the OFLN methodologies may be applied to provide computer application software with abilities similar to human pattern recognition and predictive skills. The methodologies may be incorporated in a computer program or software module stored in a computing system's memory, on a computer readable medium and/or transmitted via a computer network and/or other transmission media in one or more segments, which is executable on the computing system.

The following are just a few examples of such skills which application software may be adapted to have.

Application software may be adapted to make predictions based on the current state and on the historical trend, such as, for example, predicting an amount of merchandise to order from a supplier to avoid running out of inventory in the upcoming month, while not

keeping too much inventory (e.g., above what is needed for the month). Over time, the model may be updated, by applying adaptive update methodologies, to account for changes in fashion trends, economic conditions, etc.

Application software also may be adapted to emulate other intelligent behavior, such as the following: (a) predictive modeling: developing cause-and-effect models of systems from data that describes the systems behavior, and predicting the behavior of the system based on new "cause" data; and (b) optimization: improving the performance of a system, or solving a problem. If an operating point of the system has drifted (e.g., caused by wear and tear) or system requirements have changed (e.g., because of increased demand), the system model may be adaptively updated.

The OFLN methodologies may be applied to, for example, profiling (which is known in the information technology art as "data mining"), to look for interesting patterns in data and trying to explain them. The model is typically updated incrementally as new data is gathered, because at least some of the new data may not be represented in the existing model.

The OFLN methodologies may be applied to value prediction. For example, an input to a FLN model may be a recipe containing a list of ingredients and processing conditions for producing rubber, polymers, glass, metals, petrochemicals, food, etc., and resulting properties of the product. The FLN model may be trained to model the production process. The prediction model may be trained from historical product data in a database corresponding to product properties for each recipe. For example, a

model trained to predict the properties of bread recipes may receive the amounts of various ingredients and the baking conditions as inputs, and predict the measurable qualities of the bread product. Alternatively, the model may be trained to specify an appropriate recipe based on input of the desired properties. The model may be adaptively updated to account for drift (e.g., equipment wear and tear) or new data (e.g., as predicted recipes are tested). Additional recipes independently discovered may require updating the model incrementally.

The OFLN methodologies may be adapted for business intelligence. For example, a local utility may be interested in improving the way that they forecast the price of electric power. Traditionally, managers decide on a daily basis which plants are run in production, and how much power to buy or sell on the spot market, based on forecasts of the next day's demand and price. These decisions also may be made on an hour-by-hour basis for the following day, and so forecasts are desired for each hour of the following day. A model may be trained to predict the next day's hourly demand for electric power based on the previous 24-hours of outdoor temperature and actual demand. The trained model may be adaptively updated to account for social trends (for example, change from five-day to four-day work week, which affects demand for each day of the week).

Additional variations may be apparent to one of ordinary skill in the art from reading the following U.S. applications, which are incorporated herein by reference:

(a) Serial No. 60/374,064, filed April 19, 2002 and entitled "PROCESSING MIXED NUMERIC AND/OR NON-NUMERIC

DATA";

(b) Serial No. 60/374,020, filed April 19, 2002 and entitled "AUTOMATIC NEURAL-NET MODEL GENERATION AND MAINTENANCE";

(c) Serial No. 60/374,024, filed April 19, 2002 and entitled "VIEWING MULTI-DIMENSIONAL DATA THROUGH HIERARCHICAL VISUALIZATION";

(d) Serial No. 60/374,041, filed April 19, 2002 and entitled "METHOD AND APPARATUS FOR DISCOVERING EVOLUTIONARY CHANGES WITHIN A SYSTEM";

(e) Serial No. 60/373,977, filed April 19, 2002 and entitled "AUTOMATIC MODEL MAINTENANCE THROUGH LOCAL NETS";

(f) Serial No. 60/373,780, filed April 19, 2002 and entitled "USING NEURAL NETWORKS FOR DATA MINING"; and

(g) Serial No. 10/374,406, filed February 26, 2003 and entitled "AUTOMATIC NEURAL-NET MODEL GENERATION AND MAINTENANCE".

What is claimed is:

1. A method of incrementally forming and adaptively updating a neural net model, comprising:

(a) incrementally adding to the neural net model a function approximation node; and

(b) determining function parameters for the function approximation node and updating function parameters of other nodes in the neural network model, by using the function parameters of the other nodes prior to addition of the function approximation node to the neural network model.

2. The method of claim 1, wherein if a model accuracy of the neural net model with the function approximation node added thereto is below a predetermined accuracy level, steps (a) and (b) are repeated.

3. The method of claim 1, wherein a set of sample data patterns is used to form a list of function approximation node candidates, and the function approximation node is selected from the list of function approximation node candidates.

4. The method of claim 3, wherein the list of function approximation node candidates is formed by

splitting the set of sample data patterns into a plurality of clusters in a first level of a cluster hierarchy,

determining that a selected cluster in the first level has a population exceeding a predetermined size, and

splitting the selected cluster into two or more clusters and replacing the selected cluster with the two or more clusters in a next level of the cluster hierarchy.

5. The method of claim 4 further comprising sorting the clusters on each level of the cluster hierarchy based on cluster size, to form a sorted list of function approximation node candidates.

6. The method of claim 3, wherein the neural network model is adaptively updated by incrementally adding one or more additional nodes to the neural net model, to represent new data corresponding to a data range not represented in the set of sample data patterns.

7. The method of claim 1 further comprising:

monitoring a model accuracy of the neural net model while the neural net model is used on-line; and

adaptively updating the neural net model, if the model accuracy of the neural net model is below a predetermined threshold.

8. The method of claim 7, wherein the adaptive update includes incrementally adding one or more additional nodes to the neural net model, to represent new data.

9. The method of claim 8, wherein the new data corresponds to a change in system dynamics.

10. The method of claim 7, wherein the adaptive update includes updating the function parameters of the nodes in the neural net model.

11. The method of claim 7, wherein if the adaptive updating reaches a limit, a full retrain of the neural net model is performed.

12. The method of claim 1 further comprising adaptively updating the neural net model by adding one or more additional nodes to the neural net model, based on new data patterns.

13. The method of claim 12, wherein the additional nodes are formed by applying a clustering methodology to the new data patterns.

14. The method of claim 13, wherein the clustering methodology includes

clustering the new data patterns into a number of clusters which is approximately a number of the nodes in the neural net model;

determining that a selected cluster is far away from positions associated with the respective nodes in the neural net model; and

adding to the neural net model an additional node associated with the selected cluster and a center of the selected cluster.

15. The method of claim 12, wherein

a set of initial weights is determined for the nodes

in the neural net model when the neural net model is formed, and

when the additional nodes are added during adaptive update, a set of new weights for the nodes in the neural net model is computed, and the initial weights are combined with the new weights for the nodes based on a forgetting factor.

16. The method of claim 15, wherein the forgetting factor is determined based on a cause of model degradation.

17. The method of claim 1 further comprising applying an orthogonal least squares methodology to determine a set of weights for the neural net model.

18. The method of claim 17, wherein the set of weights are adaptively updated by using new data patterns.

19. The method of claim 17, wherein the set of weights are updated to compensate for system drift.

20. The method of claim 1, wherein the function parameters for the nodes in the neural net model are determined by applying a hierarchical k-means clustering methodology to a set of sample data patterns.

21. The method of claim 1, wherein the function approximation node is a radial basis node, and a center and radius of the radial basis node are determined

through a hierarchical k-means clustering methodology.

22. The method of claim 1, wherein the function approximation node is a Gaussian node.

23. The method of claim 1, wherein the function approximation node is a sigmoidal basis node.

24. The method of claim 1, wherein the function approximation node is a wavelet basis node.

25. The method of claim 1, wherein the function approximation node is non-linear.

26. A method of incrementally forming a neural net model, comprising:

applying a hierarchical clustering methodology to a set of sample data patterns to form a list of function approximation node candidates; and

incrementally adding one or more function approximation nodes to the neural net model until a model with an accuracy at or above a predetermined accuracy level is formed, wherein the function approximation nodes are selected from the list of function approximation node candidates.

27. A computer system, comprising:

a processor; and

a program storage device readable by the computer system, tangibly embodying a program of instructions executable by the processor to perform the method of

claim 26.

28. A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform the method of claim 26.

29. A computer data signal embodied in a transmission medium which embodies instructions executable by a computer to perform the method of claim 26.