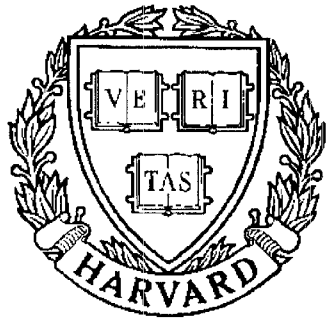
TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Hierarchical Neural Networks

by M. L. Mavrovouniotis

TR 91-4

HIERARCHICAL NEURAL NETWORKS

Submitted for publication
Computers and Chemical Engineering, special issue on Neural Networks

Michael L. Mavrovouniotis
Department of Chemical Engineering and Systems Research Center
A.V.Williams Bldg.
University of Maryland, College Park, MD 20742

ABSTRACT

With the common three-layer neural network architectures, networks lack internal structure; as a consequence, it is very difficult to discern characteristics of the knowledge acquired by a network, in order to evaluate its reliability and applicability. An alternative neural-network architecture is presented, based on a hierarchical organization. Hierarchical networks consist of a number of loosely-coupled subnets, arranged in layers. Each subnet is intended to capture specific aspects of the input data. A subnet models a particular subset of the input variables, but the exact patterns and relationships among variables are determined by training the network as a whole. However, the hierarchy of subnets gives the network hints to look for patterns in the most promising directions. Their modular organization makes hierarchical neural networks easier to analyze, because one can focus on the analysis of one subnet at a time, rather than attempt to decipher the whole network at once.

1. INTRODUCTION

A neural network is a computational structure which consists of a number of interconnected *processing elements*, also called *neurons* or *nodes*. Each neuron carries out a very simple operation on its inputs and, possibly, internal parameters. The network as a whole, however, can exhibit complex and useful input/output behavior, which can be modified through adjustment of the internal parameters of the neurons. From the engineering viewpoint, this simply means that, with appropriate values for the internal parameters, a network of neurons may be able to perform a non-trivial mapping from inputs to outputs and solve a non-trivial problem, even though each node has fairly trivial behavior.

Neural networks have been used in problems as diverse as sonar target recognition (e.g., Gorman and Sejnowski, 1988), sound recognition (e.g., Kammerer and Kupper, 1989, Matsuoka *et al.*, 1989, Sawai *et al.*, 1989, Waibel, 1989), text pronunciation (e.g., Sejnowski and Rosenberg, 1987), natural language understanding (e.g., Selman, 1989), backgammon playing (e.g., Tesauro and Sejnowski, 1987), visual pattern recognition (e.g., Fukushima, 1988), forecasting (e.g., Foster, Callope, and Ungar, 1990) and a number of chemical engineering problems (e.g., Bhat and McAvoy, 1988, Hoskins and Himmelblau, 1988, Vaidyanathan and Venkatasubramanian, 1990, Whitely and Davis, 1990).

We will discuss below the most widespread neural network architecture for engineering applications, because it is the starting point of the approach proposed in this article. We will not endeavor to

describe all the interesting alternative architectures that have been recently explored; many of these are no doubt discussed in other contributions in this journal issue.

The neural network architecture which we will call *traditional* involves nodes organized in *layers*. There are often three layers of nodes, commonly referred to as the input layer, the hidden layer, and the output layer (Figure 1). Each output node receives the outputs of all hidden nodes and produces an output which is (a portion of) the final result of the computation. Each hidden node receives inputs from all nodes in the input layer; the node sends its output to all nodes in the output layer, as shown in Figure 1. The role of the input nodes is merely to distribute the external inputs to all the hidden nodes; usually, input nodes do not perform any other transformation. No communication is allowed between nodes in the same layer or nodes that are separated by one or more intervening layers. For example, the input nodes in the three-layer network of Figure 1 cannot communicate directly with the output nodes.

The number of input and output nodes is determined by the nature of the problem and the particular representation of the data (inputs) being utilized and results (outputs) being sought. The number of nodes in the hidden layer can be as large as required and is related to the difficulty of the task that the network is carrying out. One can increase the number of layers, by adding more intermediate hidden layers, while maintaining the full connectivity between successive layers: Each node of a particular hidden layer is connected to all nodes in the next layer.

The computation carried out by each node involves taking a weighted summation (linear combination) of the inputs, adding another internal parameter (often called *bias*), and finally applying a sigmoid function to the result (Figure 2). We will restrict ourselves to one common sigmoid function, $f(x)=(1+e^{-x})^{-1}$, whose range is 0 to 1. The weights used in the linear combination and the bias are internal parameters of the node and can be adjusted to obtain the behavior desired of the neural network as a whole.

Other interesting alternatives recently proposed involve nodes that perform multiplication rather than addition (Durbin and Rumelhart, 1988), or use bell-shaped rather than sigmoid curves (e.g., Moody and Darken, 1988, Yao and Zafiriou, 1990). It will be seen that the approach proposed in this paper does not depend on the kind of internal processing carried out by each node; it is concerned mainly with the topology or connectivity of the network.

Training or learning is the process of adjusting the internal parameters of a network. Usually, one has a number of examples, for which the inputs as well as the correct (desired) outputs are known. The internal parameters of the network are usually initialized to small random values and then iteratively adjusted so that the network's outputs match, as closely as possible, the desired outputs for the

available examples. Then the network is *tested* using other examples; if the network can produce the correct outputs for these *testing examples* which were not used in the training process, it is presumed that the network has been successfully trained. It should be emphasized that the set of examples used in testing a network must be distinct from the set used in training the network.

A common training procedure is backpropagation (e.g., Lippmann, 1987), which is essentially a steepest-descent minimization of the total error. This total error is defined as the sum of the squares of the errors, the latter being merely the differences between the desired outputs and the observed outputs of the network. Steepest descent requires the partial derivatives of the total error with respect to each of the adjustable parameters (weights and biases); the partial derivatives are computed starting from the output nodes and proceeding backwards, in a manner that involves only local computations around (the inputs and outputs of) each node. There are many embellishments and improvements possible for backpropagation (e.g., Leonard and Kramer, 1990).

Difficulties with traditional neural networks. Inspection of the structure of a traditional neural network does not reveal anything about the task that the network is meant to carry out, save for the number of inputs and outputs. A traditional neural network that is meant to detect a particular kind of fault in a distillation column from 25 measurements will have the same general structure as a network detecting a fault in a reactor from 25 measurements; the networks may have different numbers of hidden nodes, but this difference is not a distinguishing feature, in the sense that it does not bear a direct, predictable relationship to the nature of the physical system in question.

The internal structure of a traditional neural network is thus too general, and in many respects unrelated to the physical system and task at hand. As a consequence, the network is not predisposed to any meaningful kinds of patterns, until it is actually trained. Before the training reaches some reasonable performance threshold, all patterns and relationships appear equivalent to the network. The network has no a priori orientation about the likely kinds of relationships and features of its inputs. Such a network that does not have an initial orientation or predisposition is clearly difficult to train. When the set of examples that is used in the training happens to be incomplete, or contains extraneous (coincidental) correlations, the network may simply pick up on the wrong features and head off in the wrong direction.

Even after (and *if*) the network has been successfully trained, its structure cannot be easily be correlated to the task it is carrying out. The network, as a whole, may be successful, but one cannot easily point out particular nodes, connections, or portions of the network to which specific aspects or portions of the network's performance can be attributed. Thus, one cannot decipher the features and

knowledge captured by the network. The root of these difficulties is the absence of internal structure in traditional neural networks.

It is in fact often mentioned that neural networks are *robust*, because removing only a small number of connections or nodes from a trained network does not significantly alter its performance. This necessarily implies that what the network has learned is spread over most of its internal weights and biases; thus, the skills acquired by the network cannot be easily mapped to portions of its structure. Since it is very difficult to discern what the network has learned, it is also difficult to evaluate the network's range of applicability and its limitations.

The analysis and *explanation* of the predictions of a traditional network is quite difficult, because a large number of weights must be dealt with all at once; it is not possible to isolate portions of the network and examine them separately, because of the dense interconnection of nodes. Traditional neural networks do not have any internal structure or features that can be *a priori* related to the structure of the problem, i.e., one cannot *a priori* relate particular parts of the network to parts or attributes of the problem.

These difficulties are exacerbated when large problems are to be tackled. Use of neural networks for large problems with many inputs and complex patterns necessitates a large number of nodes in the input and hidden layers, resulting in a number of weights that may be proportional to the *square* of the number of inputs. As the size of the network increases, it becomes progressively harder to select a sufficiently large and diverse set of training examples, perform a sufficient number of iterations (since each iteration takes much longer), and avoid inappropriate weight patterns that hinder convergence. If the task at hand involves some kind of analysis of the dynamics of a large system, one may use as inputs the values of 200 system variables at 20 successive time-points. Thus, one would have to construct a network with 4,000 inputs. Even with a mere 250 hidden nodes (which may very well be inadequate to capture the features of 4,000 inputs), the number of connections will reach one million. A network of this size may be very slow to train and require an enormous number of examples. The complexity of the system may even necessitate a network with several hidden layers; the presence of more hidden layers would make the network much harder to decipher, and further exacerbate all the difficulties that have been discussed here.

Detailed mathematical models represent the exact opposite of neural networks, as they are fully structured and decipherable. One should, naturally, make use of mathematical models whenever they are available and reliable. It is assumed, then, that one resorts to neural networks precisely because such detailed models are not available. The question is, can we introduce some kind of internal structure in neural networks, without resorting to detailed models?

2. HIERARCHICAL ARCHITECTURE

Overview. It is proposed in this article that the structure of a neural network can be a priori arranged to reflect prior knowledge we have about the structure of the physical system, the task at hand, and the kinds of patterns that are most important or most likely. Before we ask the network to learn, we can, through a meaningful internal structure, give the network hints about what kinds of patterns and relationships to look for. This internal structure is derived from knowledge and intuitions in each specific case, and it leads to a network that is specialized to the application at hand, even before the training commences. Although detailed quantitative models are presumably not available, there is a lot of qualitative information about the structure of the physical system, and the potential relationships among the inputs and outputs; it is this type of qualitative information that we would like to use in constructing neural networks with an internal structure. This does not mean that the network would be restricted to learn only certain kinds of patterns; its structure would merely make some patterns easier to learn (and other patterns harder to learn).

The internal structure proposed here is based on the decomposition of the set of inputs into subsets of related inputs. This decomposition can be done hierarchically, beginning with the smallest possible sets of related inputs. These can be, for example, inputs that refer to a specific small portion of the physical system, a particular physical phenomenon or constraint, or a single time instant within the time interval; in other words, such a small subset would be localized spatially, temporally, or conceptually. The hierarchy is created by combining these subsets into larger sets, which are still localized but cover a greater portion of the overall input set; for example, a subset localized around a specific processing unit may be formed by combining the smaller subsets that refer to input and output streams of the unit.

The hierarchy is reflected in the structure of the neural network, which starts with small subnets that focus on the smallest subsets of inputs. These subnets feed their outputs into subnets that focus on larger subsets of the input set. The set-subset relation in the hierarchical decomposition/reconstruction of the inputs is thus mirrored by connections in the construction of the overall network from subnets.

Illustration. We will use a distillation column as a typical chemical engineering system within which we can sketch out the concept of hierarchical neural networks, although numerical demonstrations will be carried out only for other, simpler examples. Consider the distillation column, shown in Figure 3. The process fluid contains three components, A, B, and C. The variables associated with each stream i are its flowrate F_i, its pressure P_i, its temperature T_i, and the concentrations of the three components A_i, B_i, and C_i. To avoid additional variables we assume that

the control system's setpoints are fixed, and its controlled and manipulated variables are within the system examined.

To focus our attention on the set of inputs, we will assume here that the task we are carrying out requires only one output. For example, for detection of a particular kind of fault, the neural network needs only one output node, which produces 1 if there is a fault, and 0 if there is no fault; the hierarchical approach we will discuss, however, is not specific to this task and could also be used, for example, within networks that carry out estimation of values for unmeasured variables or prediction of future values of variables. The inputs to the network are all the measured stream variables, for successive points within a time-window. A moving window is a simple way to isolate subsets of a long string of measurements (Figure 4). If t is the present time, the series of n+1 time points [t, t-Δt, t-2Δt, t-3Δt, ..., t-nΔt] covers a window with total duration nΔt. The value of a variable V_i at time t-kΔt is denoted as $V_i(k)$.

The processing of a large number of signals for this application would ordinarily require an enormous neural network. Such a network would be very difficult to train and might be too slow for real-time applications. It would also be very difficult to analyze the weights of the network, discern characteristics of the knowledge acquired, and evaluate the reliability and range of applicability of the network. If there are 10 points in the time-window (i.e., n=9), the network has 540 input nodes, whose values are quantitative (rather than boolean). The number of hidden nodes is generally comparable to the number of input nodes, to ensure that important patterns are recognized. If we use, for example, 180 hidden nodes, the number of connections between the input layer and the hidden layer is approximately 10^5. This number increases as the square of the number of points in the time-window or the number of measured variables. Even if the number of hidden nodes is reduced, this network is large and difficult to train, decipher, and use. In addition to being hard to train and use, the network is also hard to analyze, partly because of its large number of weights and nodes, but mainly because of its *lack of internal structure.* It is not possible to discern any manageable subnetworks, study and analyze those separately, and then build an explanation of the behavior of the whole network.

How can this enormous complexity be tamed? The key lies in reducing the number of weights and organizing a network as a hierarchy of subnets. The network can extend to several hidden layers and have a large number of hidden nodes, but its organization is structured in a way that limits the number of connections. The goal of this organization is to take advantage of prior knowledge about what *kinds of patterns* are likely in the input data – instead of searching for patterns blindly. We return to the distillation column with 36 variables and 10 time-points. The value of variable V for stream i at time t-kΔt is denoted as $V_i(k)$ and stands for a particular input of the overall neural network. The domains

of V, i, and k are: $V \in \{F, P, T, A, B, C\}$, $i \in \{1, 2, ..., 9\}$, $k \in \{0, 2, ..., 9\}$. We want to arrange the structure of a neural network to reflect prior knowledge we have about the physical and conceptual structure of the system, as a way of giving the network some hints about what kinds of patterns and relationships to look for.

The proposed hierarchically structured neural network consists of a number of subnets, which are arranged in layers, much the same way that nodes are arranged in layers within a simple neural network. At the bottom layer, the subnets receive directly the input variables, and at the very top a subnet provides the final output. In the intermediate layers, each subnet receives inputs from subnets in the previous layer and send its outputs to subnets of the next layer up. If we trace the inputs of a subnet all the way back to the original input variables, we can identify the ultimate set of input variables that could possibly affect the output of the subnet; the subnet is expected to model and summarize in its output the important characteristics of this ultimate set of input variables. No two subnets can have identical set of variables; however, each variable may participate in many subnets of each layer.

In order to construct the subnets we select all those clusters of *related* inputs, based on our *a priori* knowledge of the structure and behavior of the system being modelled. In general, the clusters will overlap, and there will even be clusters that are fully contained in other clusters. This defines the hierarchy that will be used in the construction of the network: Whenever a larger cluster is equal to the union of smaller clusters, the subnet that corresponds to the larger cluster will not receive its inputs directly from the input set, but rather from the outputs of the subnets that correspond to the smaller clusters. This should only take place if the rationale for establishing the cluster can be viewed as a combination of the rationales of the smaller cluster, i.e., if the set-subset relation of the clusters is not coincidental. In general, a subnet of a particular cluster may receive some of its inputs from other subnets (in any of the previous layers) and some from the original input set. The smallest clusters receive all their inputs directly from the input set.

Selection of Clusters of Inputs. In this example, we will only form the most obvious clusters:

- *For any given process variable, its measurements at different time points* form an important cluster, because they uncover the dynamics of the variable. Taken together, the measurements can indicate the average value of the variable and its time-derivative, the presence of abrupt spikes or step-changes, the extent of fluctuation of the measurements, etc.

- *For any given process stream and any fixed time point, the measurements for all the variables* of the stream are related. The measurement describe the state of the process fluid and are

related to derivative quantities, such as the enthalpy and heat-capacity of the stream, the molar flowrate of each component, etc.

- *For any specific kind of variable (such as flowrate or temperature) and a fixed time point, the measurements of that kind of variable for all the streams* form a cluster. For example, measurements of the flowrates of the streams, taken together, are interrelated through a mass balance around the processing unit, while measurements of temperatures provide information on heat transfer and energy balances.

In the symbol $V_i(k)$, we can substitute X for one or more of V, i, and k to denote a subnet that ultimately receives and models the set of inputs derived by letting each X take all its allowable values. Consider two of the subnets shown in Figure 5:

- $F_x(0)$ denotes a subnet with direct inputs $F_1(0)$, $F_2(0)$, $F_3(0)$, $F_4(0)$, $F_5(0)$, $F_6(0)$, $F_7(0)$, $F_8(0)$, $F_9(0)$; similarly, $F_x(1)$ denotes a subnet with direct inputs $F_1(1)$, $F_2(1)$, $F_3(1)$, $F_4(1)$, $F_5(1)$, $F_6(1)$, $F_7(1)$, $F_8(1)$, $F_9(1)$; to generalize, for any k, $F_x(k)$ denotes a subnet with direct inputs $F_1(k)$, $F_2(k)$, $F_3(k)$, $F_4(k)$, $F_5(k)$, $F_6(k)$, $F_7(k)$, $F_8(k)$, $F_9(k)$. Each of these subnets belongs to the input layer of subnets and describes the state of the flowrates of the process, for a particular time-point.

- $F_1(X)$ denotes a subnet with direct inputs $F_1(0)$, $F_1(1)$, $F_1(2)$, $F_1(3)$, $F_1(4)$, $F_1(5)$, $F_1(6)$, $F_1(7)$, $F_1(8)$, $F_1(9)$; generalizing, $F_i(X)$ denotes a subnet with direct inputs $F_i(0)$, $F_i(i)$, $F_i(2)$, $F_i(3)$, $F_i(4)$, $F_i(5)$, $F_i(6)$, $F_i(7)$, $F_i(8)$, $F_i(9)$. Each of these subnets belongs to the input layer of subnets and describes the state and dynamics of a particular process variable (F_i) over the whole time-window.

Completion of the Hierarchical Architecture. The outputs of the bottom subnets are fed to higher-level subnets, which correspond to larger clusters of variables. For example, we denote as $F_x(X)$ a subnet which corresponds to the cluster of all flowrate measurements (for all streams and all time points). This subnet belongs to the intermediate layer of subnets. Its inputs are the outputs of all subnets of the form $F_i(X)$ and $F_x(k)$, which were described above. The role of $F_x(X)$ is to receive information about the dynamics of individual flowrates – from $F_i(X)$ – and information about the state of all flowrates together at particular time points – from $F_x(k)$ – and produce an overall picture of the behavior of all flowrates over all the time points. Thus, the output of $F_x(X)$ ultimately depends on the values of all the flowrates over the time window. The hierarchy of subnets is a *multiple* hierarchy, because each subnet (and each input variable) can be used by several subnets at the next layer (Figure 6).

Beyond the specific examples of clusters mentioned above, other process variables are used by subnets in the same way. There is a subnets describing the state of each particular stream; for example, $X_1(0)$ takes as direct inputs the parameters of stream 1 at time t (i.e., t-0Δt). Its output is used by subnet $X_x(0)$ which describes the state of the whole process at time t; it is also used by subnet $X_1(X)$ which describes the state and dynamics of stream 1 over the whole time-window, using all its parameters. The output of the last subnet, $X_1(X)$, ultimately depends on all the values of the parameters of stream 1.

For this particular system, we structure the subnets in three layers, according to the number of X's in their symbol. In the first layer, we have 186 subnets that have only one X in their symbol and receive their inputs directly from the input variables. In the second layer, we have 25 subnets with two X's in their symbols; each of those receives its inputs from several first-layer subnets. In the third layer, we have only one subnet, namely $X_x(x)$, which receives inputs from all second-layer subnets. The complexity of individual subnets will depend on the character of the application and the cluster of inputs on which the subnet is focused. A subnet may consist of just *one layer of nodes; i*n a formal sense, this subnet has no hidden layers. If there is only one layer of such subnets, the resulting overall network will look like a sparsely-connected network with one hidden layer. In the simplest possible case, a subnet could even consist of just one node; this will often be the case in the examples of Section 3.

In the above example, we only formed sets of inputs that are comprised of either measurements of a given process variable (at different time points), measurements that refer to a given process stream, or measurements of the same kind of variable in different streams. In general, the formation of good sets of related input variables depends on the depth of prior knowledge about the system that is modelled. For example, if there is a small set of variables that participate in a constraint (such as a mass or energy balance) or relate to a particular physical phenomenon (such as diffusion of a component) then it is appropriate to use a subnet at the bottom layer to capture characteristics of the phenomenon. If there is a set of constraints which are used together, or a set of related phenomena, then a subnet at the next layer, receiving inputs from the subnets of the individual constraints is appropriate.

Subnets should also be used to model distinct *loosely-interacting subsystems*, since most real-life complex systems can be decomposed into loosely-coupled subsystems. This decomposition is recursive, as large subsystems can be decomposed into smaller subsystems, and those into yet smaller ones. Each subsystem can be the focus of a subnet. This subnet will receive inputs from the subnets corresponding to smaller subsystems which are contained in the original subsystem; the subnet will similarly provide its outputs to subnets corresponding to larger subsystems.

Each of the subnets of a hierarchical network is expected to summarize in its outputs the important features of the selected subset of the inputs. The outputs of the first layer of subnets form an internal representation, used by subnets at the next level, modelling larger sets of inputs at a higher level. The output of a subnet constitutes an *internal* representation which is not prespecified or hard-coded into the network. **Only the overall network as a whole is trained, based on its final outputs.** The outputs of internal subnets are not directly trained. Thus, the hierarchical structure of the neural network has the effect of decomposing a large, complex task into a network of interconnected subtasks, but the nature of the subtasks is prespecified only qualitatively. The inputs that may be involved in the task are preset, but precisely which inputs will actually exert an influence and how, and what outputs will be produced, is not a priori specified. These details are left to the network as a whole to adjust, as it strives for good performance in terms of its final outputs.

Nevertheless, the organization of the network into subnets of good sets of related variables provides hints that may help the network learn in the right direction. The organization makes some kinds of patterns easier than others to learn. Specifically, patterns of variables that belong to the same small subnet are easier for the network to pick up, because a small subnet that is not influenced by a large number of inputs unrelated to the pattern in question is examining these variables. Thus, by providing the subnet $F_1(X)$ which focuses on the measurements of flowrate F_1 over the whole time-window, we are making it much easier for the network to pick up the dynamic trends of F_1.

The hierarchical structure does not preclude the network from learning other non-intuitive patterns which do not conform to the decomposition of the set of inputs. The final output of the network is influenced by all the inputs and all internal representations. Some non-intuitive patterns simply become more difficult for the network to discern, because some combinations of variables have to traverse longer paths within the network before they can interact. The paths are usually not very long, though, because each input can participate in many subnets (Figure 5). As we already mentioned, even the smallest subsets of the input set can overlap, and the larger subsets most certainly will.

One obvious advantage of the hierarchical architecture is that it can lead to networks with more nodes but much fewer connections. If we assume that hidden nodes are intended to capture features of the inputs, it is in fact advantageous to use a larger number of hidden nodes, to ensure that all features are captured. With traditional neural networks, the computational burden forces one to restrict the number of hidden nodes. In reality, however, it is the number of connections that defines the computational complexity, and there are always many more connections than nodes. Because it results in a sparse network, the hierarchical architecture allows more nodes without a large number of connections. The

smaller number of connections and their organization may reduce the number of required examples and the training time.

For the example of Figure 3, we assume that each subnet has three layers of nodes, with two nodes in the output layer, and the number hidden nodes equal to one half of the number of the input nodes, the total number of connections in the whole hierarchical network is less than 7,000 (or at least 14 times fewer than the connections of an unstructured three-layer network). As a simple generalization, consider a system with n streams, n process variables per stream, and n time-points to examine, there are n^3 inputs to be processed. The common three-layer network architecture would require $O(n^6)$ connections, but the proposed architecture requires only $O(n^4)$ connections. This architecture actually has a total of 7 layers of nodes, with O(n) nodes per layer, but it does not require many connections, because it is organized into subnets in a hierarchical fashion. The network should be easier to train and use because of the small number of connections and its logical structure.

The most important advantage of the hierarchical structure is that they allow one to analyze the behavior of the network and determine the knowledge acquired by the network. This can be accomplished because one can examine each subnet separately; each subnet is connected to the rest of the network only in a loose and organized way, and has the specific role to summarize a specified set of inputs into a small number of outputs. One can begin with the smallest subnets, i.e., the bottom layer of the hierarchy of Figure 5. Since the subnet has only a small number of inputs and weights, one can either examine the weights directly or feed test patterns and observe the outputs of the subnet, in order to draw conclusions about what attributes and patterns in the input are recognized by the subnet.

Consider, for example, the subnet $F_1(X)$ which has the direct inputs $F_1(0)$, $F_1(1)$, $F_1(2)$, $F_1(3)$, $F_1(4)$, $F_1(5)$, $F_1(6)$, $F_1(7)$, $F_1(8)$, $F_1(9)$ and describes the behavior of F_1 over the whole time-window. Once the network converges, we can try several test-inputs for the subnet, such as: a flat profile, i.e., $F_1(k)=a$ for all k; a step change, i.e., $F_1(k)=a$ for $k\leq m$, $F_1(k)=b$ for $k>m$; a pulse, i.e., $F_1(k)=a$ for $k=m$, $F_1(k)=b$ for $k\neq m$;and a linear profile, i.e., $F_1(k)=a+bk$. For each kind of profile, different values of the parameters (a, b, m) can be tested. The dependence of the output values on the nature and parameters of the profile allows one to determine how the subnet is coding its inputs, i.e., what important attributes of the input the subnet has learned. For example, it may be found that a subnet's two outputs provide roughly: the average value and standard deviation of the inputs; the average value and the average slope; or the presence of abrupt changes in the inputs.

Having analyzed the behavior of the subnets in the bottom layer (which receive their inputs directly from the measured parameters), one can proceed to the next layer. Here, each subnet receives its

inputs from subnets of the previous layer, which have already been deciphered. Thus, test inputs can be generated and the behavior of each subnet can be analyzed. Layer by layer, one can accomplish an analysis of the behavior and knowledge of the whole network. There is no harm (other than the computational cost) in introducing subnets of uncertain value. It is in fact expected that proper analysis of the subnets, as described above, will show that for any given example only some subnets play an important role.

3. NUMERICAL EXAMPLES

In this section, we provide some simple numerical examples to show the promise of the hierarchical neural networks. It should be emphasized that the primary purpose of this article is the introduction of the concept of hierarchical neural networks. The examples in this section are only small-scale experiments with this new concept; the examples should certainly not be regarded as full-fledged applications with value of their own.

The software we use in these examples to train neural networks is "Parallel Distributed Processing" or PDP (McClelland and Rumelhart, 1989), running on Macintosh II computers. The software allows the construction networks in any topology and training using the backpropagation method. It also provides a number of other neural-network paradigms which were not used in this work.

The main example that will be demonstrated here is based on a recognition of a simple pattern that does not originate from models or industrial data. The example involves four signals, named F_1, F_2, T_1, T_2, with values scaled in the range 0-1. A window of four time-points is used, bringing the total number of inputs to 16. There is only one output and it signifies a fault condition. The output is expected to be 1 (fault) if, and only if, there is a sharp drop for signal F_1 at the most recent time point and there is no corresponding sharp drop in signal F_2 (Figure 7); a sharp drop is defined as a difference of 0.5 or higher in the scaled values of the signal at t–1 and t. The signals T_1 and T_2 do not influence the characterization of the pattern. One can easily define simple processing systems, for which this is a qualitatively correct example; F_1 and F_2 can be thought of as flowrates of an outlet stream and a feed stream of a system, and the fault condition as a leak in the system.

We used 36 training examples and 24 testing examples for this problem, each training/testing example containing 16 inputs and one output. The examples included some marginal cases, in which there existed drops of 0.48 or 0.49, which are smaller than 0.50 and therefore not "sharp" by the definition given previously. The pattern to be recognized is sufficiently simple to permit any reasonable network to operate correctly, except for these marginal patterns. The criterion used for terminating the training is that the total sum of squared errors on the training patterns be less than 0.1; this assures that the training examples are correctly classified.

Network Structures. The abbreviations HNN for hierarchical neural network and TNN traditional neural network will be used in the rest of this section. To name specific networks, the number of nodes in each hidden layer will precede this abbreviation. For hierarchical networks, the small letters p and/or t are also attached to the name, signifying which of the two obvious ways were used in clustering the inputs; for p (parameter clusters) the four measurements of each variable form a cluster, while for t (time-point clusters) the measurements of all four variables at each fixed time point form a cluster. Thus, 8-TNN (Figure 8) denotes a traditional neural network with one hidden layer which contains 8 nodes; similarly, 8-2-HNN-p (Figure 10) denotes a hierarchical neural network with two hidden layers, the first hidden layer containing 8 nodes and the second 2 nodes, with parameter-based clusters.

Following the conventions of the PDP software (McClellan and Rumelhart, 1989), if a network has n nodes, they are numbered, as 0,1, ..., n–1, beginning with the input nodes. For example, 8-TNN which has a total of 25 nodes; its input nodes are numbered as 0-15, its hidden nodes as 16-23, and its output node as 24.

Seven different neural network structures were considered (Figures 8-14), and the results for all the neural networks are listed in Table 1. The networks were:

- Traditional neural networks with one hidden layer possessing either 8 hidden nodes (8-TNN, Figure 8) or 3 hidden nodes (3-TNN, Figure 9).

- Hierarchical neural networks in which a cluster consists of the measurements of one parameter; this clustering is the best for this example. The simplest of these networks had only one hidden layer with 4 nodes, i.e., one node devoted to each cluster (4-HNN-p, Figure 10). A network with two hidden layers was also considered (8-2-HNN-p, Figure 11).

- Hierarchical neural networks in which a cluster consists of the measurements of all variables at a fixed time-point. This is an *a priori* plausible clustering, but we know from the definition of the example that this clustering is inappropriate. Since in a realistic application one would not know the best clustering of the inputs, this clustering was introduced so that we could examine the effects of inappropriate clustering. The networks had only one hidden layer with either 4 nodes, i.e., one node devoted to each cluster (4-HNN-t, Figure 12), or 8 hidden nodes, i.e., two nodes devoted to each cluster (8-HNN-t, Figure 13).

- Finally, a hierarchical network in which both types of clusters exist, with one hidden node per cluster (8-HNN-pt, Figure 14).

Results and comparisons. The examples included some marginal cases. As indicated in Table 1, only these marginal cases were misclassified by any network; all networks classified the clear-cut cases correctly. The results in Table 1 show that the best behavior with respect to these marginal cases is exhibited by the hierarchical networks which include parameter-based clusters, namely 4-HNN-p (Figure 10), 8-2-HNN-p (Figure 11), and 8-HNN-pt (Figure 14). The performance of TNNs, as well as HNNs that contain only time-point clusters, is not as good. Thus, the presence of the parameter-based clusters, which are the natural clusters for this example, is essential for the fine distinctions required in the difficult, marginal cases.

With respect to the size of the networks, as measured by the number of connections, we should point out that hierarchical networks are quite compact, even when they are unnecessarily complex. For example the 8-2-HNN-p, compared to the 4-HNN-p which performs just as well, has an unnecessary second hidden layer and extra hidden nodes; however, it ends up with only 50 connections. In contrast, 8-TNN, with only one hidden layer, already has 136 connections.

The third column in Table 1 shows the training iterations and the number of connection updates required for training. The latter is equal to the number of connections times the number of iterations; for example, in the first line, 7,208= 53×136. The number of connection updates can be considered a hardware-independent measure of the training time. Under this measure, the network 3-TNN trains the fastest – but does not perform very well. The two hierarchical networks that include *only* parameter-based clusters, 4-HNN-p and 8-2-HNN-p, are next, and they are also good performers. The 8-HNN-pt which contains both time-point and parameter clusters has one of the longest training times. The presence of different types of clusters apparently confuses the network in the initial stages of training, even though the correct clusters are present and lead to eventual good performance.

Analysis of a Simple Hierarchical Network. The susceptibility of the networks to analysis of their behavior (last column of Table 1) is the last but most important element of the comparison. In order to extract the internal representations captured by the trained networks, we examine their weights and biases. Table 2 shows the weights (also called *connection strengths*) for the 4-HNN-p (Figure 10), along with the biases of the hidden and output nodes.

Node 19 has positive weights and bias. Since the parameters are scaled to the range 0-1, this node will always be active, i.e., will always have an output approaching 1; thus, it cannot affect the discrimination of the fault and no-fault behaviors. The weight from this node to the output node (equal to –5.0) can be treated as an extra bias, which is added to the existing bias of –6.4, bringing the effective bias to -11.4. In order to counteract this negative bias and produce a final output close to 1, at least two of the remaining three hidden nodes (16, 17, and 18) must be active. Node 18 has a

positive bias but negative weights. If its inputs (the measurements of T_2) are large, the node will be inactive, i.e., it will produce a 0 output. If all its inputs are very small, the output of the node will be far from 0, but in no case can it exceed 0.75. Returning to the weights of the last column of Table 2, we can now see that the contribution of T_2 hidden node 18 is quite small, and the the only way the output node 20 can be active is if both nodes 16 and 17 are active.

Consider now node 16. The node is active if and only if:

$$2\,F_1(t-3) + 1.9\,F_1(t-2) + 1.9\,F_1(t-1) - 7.8\,2\,F_1(t) - 1.4 \gg 1$$

Since the analysis is only meant to be qualitative, we round the coefficients to the nearest integer:

$$2\,F_1(t-3) + 2\,F_1(t-2) + 2\,F_1(t-1) - 8\,F_1(t) - 1.4 \gg 1$$

or:

$$\left[\frac{F_1(t-3) + F_1(t-2) + F_1(t-1)}{3} - F_1(t)\right] - 0.3\,F_1(t) - 0.2 \gg 0.16$$

The bracketed expression is the difference between the average of the first three measurements of F_1 and the last (most recent) measurement. This difference is high enough to satisfy this relationship precisely when there is a sharp drop in F_1.

Node 17 exhibits exactly the reverse behavior, i.e., it detects the *absence* of a sharp drop; note that its coefficients and magnitude are *almost the exact opposites* of those of node 16. Since we argued that the output node is active when nodes 16 and 17 are active, we conclude that the node is active (with output approaching 1) if there is a sharp drop in F_1 but no sharp drop in F_2.

One last worthwhile observation is that, although in the construction of the example we considered only the drop of the signals between t–1 and t, the network compares the value at t to the average of the values at t-3, t–2, and t–1. In a realistic application, the latter approach, essentially discovered by the network, is likely to be much more sensible.

Analysis of Other Networks. We have argued that traditional neural networks are hard to analyze and decipher. This point can be illustrated within Table 3, which contains the weights and biases for the 8-TNN (Figure 8). With so many weights entering each hidden node, it is not possible to isolate the role of the node. All the hidden nodes have both positive and negative weights, hence they cannot be pruned as indiscriminate. All but a couple of the hidden nodes also have comparable weights in their connections to the output. The 3-TNN presents similar difficulties; even though it has much fewer weights, it still lacks internal structure which was essential in our interpretation of the 4-HNN-p.

The 8-2-HNN-p (Figure 11) has a second hidden layer as well as extra hidden nodes in the first hidden layer. Despite its unnecessary complexity, the 8-2-HNN-p can be analyzed almost as easily as 4-HNN-p. The relevant weights and biases are given in Table 4. The weights and biases of nodes 16-19 indicate that each one of them is detecting a sharp drop in the signal it is examining. The connection weights from nodes 16-19 to nodes 24-25 advance the analysis: Node 24 is active when nodes 16 and 17 are active and nodes 18 and 19 inactive; node 25 is *inactive* under precisely the same conditions. At the output end of the network, the weights allow one to reason that an output of 1 results when node 24 is active and node 25 inactive. Clearly, there is quite a bit of redundancy in the network, but its hierarchical organization allowed the analysis of its behavior.

Hierarchy alone is not enough, however. We must have hierarchy that corresponds to the structure of the system and task at hand. Thus, if we form clusters that include all the measurements for a fixed time-point, as in the network 4-HNN-t (Figure 12), we obtain neither good performance (Table 1) nor ability to decipher. The weights and biases of this network are given in Table 5. One may infer that the values of F_1 and F_2 are more important than those of T_1 and T_2, because of the observed differences in the magnitudes of the weights. It also appears that the hidden nodes 16, 18, and 19 are examining the difference between the value of F_1 and the value of F_2 at their respective time points; this difference is not a bad indirect way to detect sharp drops. However, it is not easy to come up with a more complete explanation of the network's behavior.

Adding more hidden nodes based on time-point clusters (8-HNN-t in Figure 13) improves the performance and, better yet, adding hidden nodes based on parameter clusters (8-HNN-pt in Figure 14) brings it up to the level of the best networks (Table 1). However, in the *analysis* of network behavior no improvement is observed. In fact, mixing both types of cluster (in 8-HNN-pt) appears to make the network harder to analyze.

An Example using Plant Measurements. A second example is presented here, entailing dynamic modelling using actual plant data. The system under investigation is the crude oil heater shown in Figure 15. In addition to the temperatures, flowrates, and pressures (shown in the figure as T, F, and P), the specific gravity of the fuel gas (ρ) is an important parameter.

The data contain successive measurements of fuel-gas flowrate setpoint (SP), measured fuel-gas flow-rate (FR), mixed outlet temperature (T_O), and the fuel-gas specific gravity (SG). The training set covered 155 time-points and the testing set 119 time-points, for the same heater unit but a different day. With the specification of the time window that is made below, our raw data translate into 146 training patterns and 110 testing patterns. All the data were scaled to the range 0-1.

We will use neural networks for dynamic modelling, estimating the outlet temperature of the heater (Figure 16). A window of 5 past measurements will be used for SP, FR, T_O, and SG, along with 5 future values for SP. The 5 corresponding future values of T_O will be the outputs of the network. Thus, the networks will have 25 inputs and 5 outputs. Essentially, we are carrying out dynamic modelling for T_O, with the flowrate control-loop closed, but without control of the temperature T_O. This would be useful, for example, in designing a control loop for T_O. We are considering here only *some* of the parameters that affect the behavior of this system, because the remaining parameters were not present in the set of plant measurements we obtained. This limits the accuracy with which the system can be modelled; it is nevertheless interesting to examine the effect of the available measurements on T_O.

In training networks for this example, we found out that the lowest total error for the training set does not give good predictions, i.e., good performance for the testing set. The networks have a strong tendency to overtrain; in an overtrained state, the networks can recall patterns from the training set but they loose their ability to generalize. To overcome the problem of overtraining, we evaluate each network's performance on the testing set, as a function of the total training error, during the training process (Figures 17-19). The best state of a network is that which gives the best performance (lowest error) on the testing set.

Two neural networks are used in this example. The traditional neural network (5-TNN) has 5 hidden nodes arranged in one hidden layer, leading to a total of 150 connections; the number of hidden nodes of the 5-TNN was chosen to yield a number of connections identical to that of the hierarchical network described below.

The hierarchical neural network (18-HNN) has one hidden layer with 18 hidden nodes which can be viewed as one-node subnets, as follows. For each of the variables SP, FR, T_O, and SP there are two hidden nodes modelling their past measurements. For each of the past 5 time points, there is one hidden node receiving the measurements of SP, FR, T_O, and SP for that time point. Each of the preceding 13 hidden nodes sends its output to all output nodes. For the future SP, however, we have five hidden nodes which are selective not only in the inputs they receive but also in the output nodes to which they send their results. There is a separate hidden node for each of the time points t+1 to t+5; the node for t+i receives the SP values SP(t+1) to SP(t+i) and sends its output only to the output node for T_O(t+i). This arrangement ensures that the setpoint SP(t+i+1) cannot affect temperatures T_O(t+1) to T_O(t+i) which precede it temporally. The hierarchical architecture permits us to enforce unequivocally this type of commonsense constraint.

The 18-HNN has a total of 145 connections. Because its size and complex internal structure do not permit a clear sketch of the network, it is presented in the form of two incidence matrices in Tables 6 and 7. These tables show the connections between the input and hidden layers (Table 6) and the connections between the hidden and output layers (Table 7) in a concise way and make the internal structure of 18-HNN apparent.

Results for Networks using Plant Measurements. Both the hierarchical and traditional neural networks are trained three times, each run starting with a different set of initial random weights, to assure the reproducibility of the results. Training is terminated when the lowest total sum of squared errors for the testing patterns has been found. In Figures 17-19, the total sum of squared errors on the testing patterns (or testing error) is plotted against the total sum of squared errors on the training patterns (or training error). The total squared errors reported involve a summation over all the respective examples and over all 5 output nodes.

Figures 17 and 18 show that the results are not strongly dependent on the initial random weights. Beyond error criterion 15, the neural networks are untrained, and the performance of the networks is not significant. Below error criterion 8, all networks are overtrained (the testing error is significantly larger than the training error); the networks are memorizing the training patterns and cannot extrapolate to the testing patterns. In the intermediate region of interest, the hierarchical neural network outperforms the traditional neural network (Figure 19). The testing error achieved by the best hierarchical neural network is 8.1, which is lower than the error of 9.5 achieved by the best traditional neural network. Since total squared errors reported involve a summation over all the testing examples and over all 5 outputs, the total error of 8.1 corresponds to a squared error of $\frac{8.1}{110\times5}=0.0147$ for a single output and a single example; hence a standard deviation of $(0.0147)^{1/2}=0.12$ in the scaled temperature prediction is achieved by the 18-HNN. The 5-TNN gives a slightly worse standard deviation of 0.13. In terms of unscaled temperatures, this corresponds to a standard deviation of 2.4°F for the predictions of the hierarchical neural network and a standard deviation of 2.6°F for the traditional neural network.

These predictions would not be sufficient for quantitative analysis. An error of 0.12 denotes uncertainty equal to ±12% of the range of variation of the temperature; in effect, 24% of the whole range is included in the rough region of uncertainty. This should not be surprising, given that the available data covered only *some* of the important parameters of the system. The predictions are, however, suitable for a qualitative analysis of the influence of the parameters at hand on the temperature.

In order to investigate the knowledge acquired by the networks, we only aim to assess the influence of different input parameters, and we follow an approach different from that of the previous example. We examine the activations (output signals) produced by the hidden nodes on each of the training and testing patterns, for the best network in each architecture. Consider the standard deviation of the activation for each hidden node in the network. When a hidden node has a very low standard deviation, then it contributes an approximately constant amount to the output nodes; it acts effectively as a redundant bias term. A hidden node that has a high standard deviation, on the other hand, is discriminant and plays an important role in the performance of the network. To obtain a more accurate view, we also multiply the standard deviation of each hidden node by each weight leading from that hidden node to an output node. These parameters will be called *discriminant parameters* and they represent a qualitative measure of the discriminant effect of each hidden node on each output node. These values are given for the 18-HNN in Tables 8 and 9. For example, the value –0.01 in the first cell of Table 9 represents the standard deviation of the output (activation) of hidden node 25 over the testing set, multiplied by the weight connecting node 25 to the output $T_O(t+1)$. The analysis of Tables 8 and 9 will be concerned only with the absolute values of these discriminant parameters; it is not meaningful to take the sign into account, because the relation of the hidden nodes to the input data is not considered here.

A comparison of the values in Tables 8 and 9 reveals that the standard deviations of the hidden nodes for the training set are in general higher than those for the testing set. This indicates that there is some memorization taking place: There are some discriminating features which the network found in the training set but not the testing set. However, there is a general consistency between the training and testing sets as far as the relative values of the discriminant parameters are concerned. We generally find the highest (and lowest) values of Table 8 in the same cells as those of Table 9 . This consistency shows that a solid relationship does exist between the training and testing patterns. Because of this consistency, the rest of the analysis will refer to Table 9, with the understanding that similar conclusions can be reached within Table 8.

Large values of the discriminant parameters show that past temperature is a more important variable than past flowrate setpoint, past flowrate, and past specific gravity. This conclusion is drawn because hidden nodes 29-30 have generally higher discriminant parameters than nodes 25-28 and 31-32. This is especially true for the prediction of $T_O(t+1)$ and $T_O(t+2)$. As we move to $T_O(t+4)$ and $T_O(t+5)$, the effect of the past temperature (nodes 29-30) drops and the effect of the specific gravity SG (nodes 31-32) rises. Since the temperature does not vary sharply among successive time-points, one expects the network to use the available measurements $T_O(t)$ to $T_O(t-4)$ for its predictions in the first few minutes. It is in fact observed (from detailed results not shown here) that for the *output* nodes, in the order t+1

to t+5, there is a decline in the standard deviation and an increase in the error. For a longer time-horizon, the other parameters come into play, and it appears that the specific gravity SG is more important than other past measurements for this example. For a long horizon, even more important than SG are the future setpoints (nodes 33-37). The influence of the future setpoint is nil on $T_0(t+1)$, but it increases gradually, giving rise to the largest of the discriminant parameters in the last column of the table for $T_0(t+4)$ and $T_0(t+5)$.

Among the five time-point subnets (t-4 to t; nodes 38 to 42), as one would expect, the measurements at t are the most important in predicting the future temperature. A more interesting result is that the decline in the discriminant parameters as we move from node 42 to 38 is not monotonic. The measurements at t-3 (node 39) are more important than the measurements at t-2 and t-1. This would be consistent with a time-delay of approximately 4 minutes from the measurements to the temperature T_0.

The above are but a few of the observations one can make from the discriminant parameters. Based on these observations, one might construct other hierarchical networks, with fewer hidden nodes and/or fewer inputs. One could also devise other ways to analyze the networks to further validate the results of the analysis carried out here. The key point in the above analysis is that the *internal structure* of the hierarchical network allows such comparisons to be made.

The discriminant parameters for the traditional neural network are shown in Tables 10 and 11. It is clear that the values in Table 10, obtained from the training patterns, are in general larger than those in Table 11, obtained from the testing patterns. However, there is no apparent way to decipher the 5-TNN in the same way that we analyzed the 18-HNN; since all the hidden nodes are connected to all the input nodes, it is difficult to discern what the trained network considers more important in predicting the future temperature of the heater. In fact, a comparison of Table 10 and Table 11 reveals several inconsistencies, further complicating any attempted analysis. For example, in Table 10 (training set), node 26 appears more important than node 27, but this is not confirmed by Table 11 (testing set). Thus, the traditional neural network, in addition to having slightly worse performance than the hierarchical network, is essentially a black box, and it is very hard to analyze and decipher whatever knowledge the network acquired.

4. SUMMARY AND DISCUSSION

With the common three-layer neural network architectures, the processing of a large number of signals requires a large neural network. Such a network is very difficult to train and may not have sufficient speed for real-time applications. The computational complexity also prevents the use additional hidden layers, potentially leading to total inability of the network to capture essential complex patterns in the signals. Furthermore, the lack of internal structure in a large network makes it very difficult to

discern characteristics of the knowledge acquired by the network, in order to evaluate its reliability and applicability. The difficulties are particularly severe for large networks, but they are also present in small networks. For a problem of any size, the smaller and internally more structured network will generally train faster, achieve better performance, and be easier to analyze.

The alternative neural-network structures presented here contain much fewer connections and are organized in a hierarchical fashion. Our hierarchical networks consist of a number of loosely-coupled subnets, arranged in layers; each subnet is intended to capture specific aspects of the input data. At the bottom layer, each subnet operates directly on a particular subset of the input variables. In the intermediate layers, each subnet receives its inputs from subnets of the previous layer and sends its outputs to subnets in the next higher layer. Each subnet is expected to model and summarize in its output the important characteristics of a particular set of related input variables.

In order to construct the subnets, we start from the set of inputs and identify all its subsets which, based on our a priori knowledge of the structure and behavior of the system being modelled, consist of related inputs. We call these subsets input clusters. In general, the clusters will overlap, and there will even be clusters that are fully contained in (i.e., are subsets of) other clusters. Whenever a larger cluster is equal to the union of smaller clusters, the subnet that corresponds to the larger cluster will not receive its inputs directly from the input set, but rather from the outputs of other subnets that correspond to the smaller clusters. The complexity of each subnet (i.e., its number of hidden and output nodes) can be adjusted based on the complexity of the relationship that led to the establishment of the cluster. In simple cases, a subnet could consist of just one or two nodes.

We are not hardwiring exact knowledge in a hierarchical neural network, because the exact patterns and relationships among variables are still determined by the network during training. In essence, by organizing the variables into related sets and structuring the neural network as a multiple hierarchy of subnets, we are giving the network hints to look for patterns in the most promising directions, based on our prior knowledge about the structure and behavior of the system being modelled. The fact that hierarchical networks can contain several layers of subnets may allow these neural networks to capture patterns more complex than those captured by a three-layer network. However, due to their small number of connections, hierarchical neural-network structures can be trained faster. Their modular organization also makes them easier to analyze, because one can focus on the analysis of one subnet at a time, rather than attempt to decipher the whole network at once.

Picking appropriate clusters of inputs is crucial for the success of hierarchical networks. If the clusters that are selected are inappropriate, then the network becomes considerably more difficult to train and analyze. Consequently, if many outputs are present in a single network, then they should be require

same clusters; if this is not the case they should be modelled by separate networks, each clustered in a different way.

In this article, we have introduced the concept of hierarchical neural networks and demonstrated the advantages of the architecture for two examples. In the first example, involving a simple pattern and only 16 inputs, we examined a variety of traditional and hierarchical structures. We showed that hierarchical networks with the appropriate input clusters are more accurate, faster to train, and easier to analyze than traditional networks. The analysis of the networks in this example was carried out by inspection of the weights, and allowed full determination of the networks' function. In the second example, we considered dynamic modelling using incomplete measurements (from a plant), and examined only one traditional and one hierarchical architecture. The hierarchical network performed better, and allowed us to distinguish the qualitative significance of some of the input parameters; the traditional network's behavior could not be analyzed even qualitatively.

Our long-term goals include further formalization of this approach in a mathematical framework and theoretical investigation of its properties. The theoretical investigation will facilitate the application of the techniques to a variety of systems.

ACKNOWLEDGEMENTS

Financial support for this work was provided by the Electric Power Research Institute and by the Systems Research Center, which is supported by the National Science Foundation (CDR 8803012). The computations were carried out using the public-domain software "Explorations in Parallel Distributed Processing" (McClelland and Rumelhart, 1989). The author acknowledges the assistance of Ms. Shirley Chang in applying the PDP software on the examples in Section 4, which were designed and analyzed by the author.

REFERENCES

Bhat, N. and McAvoy, T. J.: *Use of Neural Nets for Dynamic Modeling and Control of Chemical Process Systems. Computers and Chemical Engineering*, **14**, 561-572 (1990)

Durbin, R., and Rumelhart, D.E. "Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks". *Neural Computation,* **1**, 133-142 (1989).

Foster, B., Collope, F., and Ungar, L.H. Forecasting Using Neural Networks. *AIChE Annual Meeting*, Chicago, 1990.

Fukushima, K. "A Neural Network for Visual Pattern Recognition" *Computer*, March 1988, 65-76 (1988).

Gorman R., and Sejnowski T., Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets, *Neural Networks*, Vol. 1, 75-89 (1988)

Hopfield, J. J. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. Proceedings of the National Academy of Science 79, 2554-2558 (1982).

Hoskins, J. C. and D. M. Himmelblau. Artificial Neural Network Models of Knowledge Representation in Chemical Engineering. Computers & Chemical Engineering 12:9/10, 881-890 (1988).

Kammerer, B. R. and Kupper, W. A.: *Design of Hierarchical Perceptron Structures and Their Application to the Task of Isolated-Word Recognition.* International Conference on Neural Networks, 6(1989).

Leonard, J., and Kramer, M.A. Improvement of the backpropagation algorithm for training neural networks, *Computers and Chemical Engineering*, **14**, 337-342 (1990).

Lippmann R., An Introduction to Computing with Neural Nets, IEEE ASSP Magazine, 4-22 (April 1987)

Matsuoka, T., Hamada, H. and Nakatsu, R.: *Syllable Recognition Using Integrated Neural Networks.* International Conference on Neural Networks, 1(1989).

Miikkulainen, R. and Dyer, M.G.: *A Modular Neural Network Architecture for Sequential Paraphrasing of Script-Based Stories.* International Conference on Neural Networks, 2(1989).

Moody, J. and Darken, C.: Learning with Localized Receptive Fields. *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann Publishers, 1988.

Rumelhart, D. E. and J. L. McClelland. (Eds.). Parallel Distributed Processing: Explorations in the Microstructure of Cognition. The MIT Press, Cambridge (1986).

McClelland, J. L., and Rumelhart, D. E. Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises. The MIT Press, Cambridge (1989).

Sawai, H., Waibel, A., Haffner, P., Miyatake, M., and Shikano, K.: *Parallelism Hierarchy, Scaling in Time-Delay Neural Networks for Spotting Japanese phonemes CV-Syllables.* International Conference on Neural Networks, 6(1989).

Selman, B. "Connectionist systems for natural language understanding" *Artificial Intelligence Review*, **3,** 23-31 (1989).

Sejnowski T., and Rosenberg C., "Parallel networks that learn to pronounce English text". *Journal of Complex Systems*, **1**(1):145-168, 1987.

Tesauro, G. and Sejnowski, T. J.: *A Neural Network that Learns to Play Backgammon.* (1987)

Tou, J. T. and R. C. Gonzalez. Pattern Recognition Principles. Addison-Wesley, Reading (1977).

Vaidyanathan, R., and Venkatasubramanian, V. Representing and Diagnosing Process Trends using Neural Networks. *AIChE Annual Meeting*, Chicago, 1990.

Vogl, T. P., Mangis, J. K., Rigler, A. K., Zink, W.T. and Alkon, D. L.: *Accelerating the Convergence of the Back-Propagation Method.* Biological Cybernetics, 59(1988).

Waibel, A.: *Modular Construction of Time-Delay Neural Networks for Speech Recognition.* Neural Computation, 1(1989).

Whitely, J.R., and Davis, J.F. Backpropagation Neural Networks for Qualitative Interpretation of Process Data. *AIChE Annual Meeting*, Chicago, 1990.

Widrow, B. and S. D. Sterns. Adaptive Signal Processing, Prentice-Hall, Englewood Cliffs (1985).

Yao, S. C., and Zafiriou, E. Control System Sensor Failure Detection via Networks of Localized Receptive Fields. Proceedings of the American Control Conference, 1990.

Table 1. Results for various neural networks used in the sharp-drop example.

Network	Number of connections	Training iterations / connection updates	Misclassified testing patterns*	Decipherable?
8-TNN (Figure 8)	136	53 / 7,208	2*	No
3-TNN (Figure 9)	51	63 / 3,213	2-3*	No
4-HNN-p (Figure 10) parameter clusters	36	139 / 5,004	1*	Yes
8-2-HNN-p (Figure 11) parameter clusters	50	116 / 5,800	1*	Yes
4-HNN-t (Figure 12) time-point clusters	36	279 / 10,044	3*	Partially
8-HNN-t (Figure 13) time-point clusters	72	196 / 14,112	2*	Partially
8-HNN-pt (Figure 14) mixed clusters	72	172 / 12,384	1*	No

* As mentioned in the definition of this example, only marginal cases are ever misclassified by the networks.

Table 2. Weights and biases for the 4-HNN-p. The rows correspond to the sending units (input and hidden nodes) while the columns indicate receiving units (hidden and output nodes). Biases of the hidden and output nodes are given in the last row.

		to hidden nodes				to output node
	from node	to #16	to #17	to #18	to #19	to #20
F1 inputs	0	2.0				
	1	1.9				
	2	1.9				
	3	-7.8				
F2 inputs	4		-2.2			
	5		-2.1			
	6		-1.8			
	7		7.7			
T1 inputs	8			-1.3		
	9			-2.0		
	10			-0.9		
	11			-0.4		
T2 inputs	12				1.3	
	13				1.3	
	14				0.6	
	15				0.9	
F1 hidden	16					8.0
F2 hidden	17					8.1
T1 hidden	18					3.1
T2 hidden	19					-5.0
	biases	-1.4	1.6	1.1	0.5	-6.4

Table 3. Connection weights and biases for the trained 8-TNN on the sharp drop problem. The rows correspond to the sending units (input and hidden nodes) while the columns indicate receiving units (hidden and output nodes). Biases of the hidden and output nodes are given in the last row.

from node	to #16	to #17	to #18	to #19	to #20	to #21	to #22	to #23	to #24
#0	-0.5	1.2	-0.1	-0.2	0.1	0.1	0.4	0	
#1	-0.8	0.8	0.3	-1	-0.6	0.2	0.4	0.8	
#2	-0.7	0.5	0.3	-0.7	0.2	0.4	0.8	0.6	
#3	1.8	-3.4	-0.5	3.1	0.9	-0.4	-1.5	-2.2	
#4	0.7	-0.7	-0.2	1	0.4	0.4	-0.4	-1.1	
#5	-0.1	-1.3	0	0.7	0.5	0.1	-0.5	-0.9	
#6	0.6	-0.8	-0.5	0.6	0.5	0.2	-0.2	-0.2	
#7	-1	3.8	-0.2	-3.4	-0.6	0.4	1.4	2.6	
#8	-0.3	-0.1	-0.4	0.2	0.5	-0.2	-0.1	0.2	
#9	0.5	-1.4	0.1	0.7	0.1	-0.3	-0.3	-0.6	
#10	0.5	0.1	0.3	0.1	-0.4	-0.3	0.2	0.1	
#11	0.3	-0.1	0	-0.1	-0.5	-0.2	-0.6	0	
#12	0.1	-0.8	-0.5	0.4	0.5	-0.4	-0.4	0.1	
#13	0.4	-0.1	-0.2	-0.2	-0.4	0.2	-0.1	0.3	
#14	0	0.3	0.2	-0.4	-0.2	-0.3	0.3	0	
#15	0	-0.2	0	0.2	0	0	-0.3	-0.4	
#16									-2.6
#17									5.7
#18									0.3
#19									-5.3
#20									-1.7
#21									0.4
#22									2.2
#23									3.6
bias	-0.1	1	-0.6	0	0.1	0.1	-0.3	-0.2	-0.4

Table 4. Connection weights and biases for the trained 8-2-HNN-p.

		F_1 hidden		F_2 hidden		T_1 hidden		T_2 hidden		2nd hidden		Out
from nodes		16	17	18	19	20	21	22	23	24	25	26
F_1 inputs	0	1.4	0.3									
	1	1.1	0.7									
	2	0.6	0.9									
	3	-4.3	-2.7									
F_2 inputs	4			1	0.8							
	5			1.4	1							
	6			0.8	0.9							
	7			-4.4	-3.4							
T_1 inputs	8					0.7	-0.7					
	9					0.8	-1.7					
	10					0	-0.9					
	11					0.5	-0.5					
T_2 inputs	12							0	-0.2			
	13							-0.5	-0.1			
	14							0	-0.4			
	15							-1.4	-0.8			
F_1 hidden	16									3.3	-3.1	
	17									2.2	-1.5	
F_2 hidden	18									-4	3.1	
	19									-3	2.7	
T_1 hidden	20									-1.5	0.8	
	21									2.2	-1.1	
T_2 hidden	22									0.9	-0.7	
	23									0.4	-0.2	
Second hidden layer	24											6.8
	25											-5.9
BIAS	26	-0.9	-0.7	-1.1	-1	0	0.6	0.1	0	-0.1	0.3	-0.9

Table 5. Weights and biases for the 4-HNN-t.

		to hidden nodes				to output node
	from node	to #16	to #17	to #18	to #19	to #20
F1 inputs	0	2.7				
	1		-5.1			
	2			0.8		
	3				-6.3	
F2 inputs	4	-2.7				
	5		4.6			
	6			-1.0		
	7				5.7	
T1 inputs	8	1.3				
	9		2.1			
	10			-0.1		
	11				-0.6	
T2 inputs	12	-3.1				
	13		0.6			
	14			-0.2		
	15				-0.5	
F1 hidden	16					4.3
F2 hidden	17					-8.4
T1 hidden	18					0.3
T2 hidden	19					13.2
	biases	-0.5	-0.4	-0.3	0.4	-3.3

Table 6. Connections between the input nodes and the hidden nodes of the 18-HNN. The input nodes are numbered as 0-24 and represented by rows; the hidden nodes are numbered as 25-42 and represented by columns.

		SP	net	FR	net	To	net	SG	net	future SP					t-4	t-3	t-2	t-1	t
nature of input	input node	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
past SP	0	X	X																X
	1	X	X															X	
	2	X	X														X		
	3	X	X													X			
	4	X	X												X				
past FR	5			X	X														X
	6			X	X													X	
	7			X	X												X		
	8			X	X											X			
	9			X	X										X				
past To	10					X	X												X
	11					X	X											X	
	12					X	X										X		
	13					X	X									X			
	14					X	X								X				
past SG	15							X	X										X
	16							X	X									X	
	17							X	X								X		
	18							X	X							X			
	19							X	X						X				
future SP	20									X	X	X	X	X					
	21										X	X	X	X					
	22											X	X	X					
	23												X	X					
	24													X					

Table 7. Connections between the hidden nodes and the output nodes of the 18-HNN. The hidden nodes, numbered 25-42, are represented by rows; the output nodes are represented by columns.

subnets	node number	To (t+1)	To (t+2)	To (t+3)	To (t+4)	To (t+5)
SP subnet	25	X	X	X	X	X
	26	X	X	X	X	X
FR subnet	27	X	X	X	X	X
	28	X	X	X	X	X
To subnet	29	X	X	X	X	X
	30	X	X	X	X	X
SG subnet	31	X	X	X	X	X
	32	X	X	X	X	X
Future SP	33	X				
	34		X			
	35			X		
	36				X	
	37					X
t-5 subnet	38	X	X	X	X	X
t-4 subnet	39	X	X	X	X	X
t-3 subnet	40	X	X	X	X	X
t-2 subnet	41	X	X	X	X	X
t-1 subnet	42	X	X	X	X	X

Table 8. Discriminant parameters, based on the training patterns, for the best 18-HNN. A discriminant parameter for a hidden node and an output node is defined as the standard deviation of the activation of the hidden node multiplied by the weight connecting the hidden node to the output node.

subnet	hidden node	To(t+1)	To(t+2)	To(t+3)	To(t+4)	To(t+5)
SP subnet	node 25	-0.03	-0.05	-0.06	-0.06	-0.02
	node 26	-0.02	-0.02	-0.06	-0.01	-0.02
FR subnet	node 27	-0.02	-0.02	-0.04	-0.02	-0.05
	node 28	-0.02	-0.04	-0.01	-0.03	-0.03
To subnet	node 29	-0.17	-0.16	-0.08	0.00	0.04
	node 30	-0.18	-0.05	0.03	0.02	0.07
SG subnet	node 31	-0.01	0.02	0.04	0.06	0.03
	node 32	-0.01	0.03	0.06	0.03	0.07
Future SP	node 33	0.01				
	node 34		0.07			
	node 35			0.10		
	node 36				0.25	
	node 37					0.25
t-4 subnet	node 38	0.00	0.00	0.00	0.00	0.00
t-3 subnet	node 39	-0.03	0.07	0.12	0.07	0.05
t-2 subnet	node 40	-0.01	-0.01	0.01	0.00	-0.01
t-1 subnet	node 41	-0.01	-0.01	0.00	0.01	0.00
t subnet	node 42	0.29	0.21	0.17	0.11	0.09

Table 9. Discriminant parameters for the best 18-HNN on the testing patterns. A discriminant parameter for a hidden node and an output node is defined as the standard deviation of the activation of the hidden node multiplied by the weight connecting the hidden node to the output node.

subnet	hidden node	To(t+1)	To(t+2)	To(t+3)	To(t+4)	To(t+5)
SP subnet	node 25	-0.01	-0.02	-0.02	-0.02	-0.01
	node 26	-0.01	-0.01	-0.02	-0.01	-0.01
FR subnet	node 27	-0.01	-0.01	-0.02	-0.01	-0.02
	node 28	-0.01	-0.02	0.00	-0.01	-0.02
To subnet	node 29	-0.13	-0.12	-0.06	0.00	0.03
	node 30	-0.14	-0.04	0.03	0.02	0.05
SG subnet	node 31	-0.01	0.03	0.05	0.07	0.04
	node 32	-0.01	0.04	0.07	0.04	0.08
Future SP	node 33	0.00				
	node 34		0.03			
	node 35			0.04		
	node 36				0.11	
	node 37					0.10
t-4 subnet	node 38	0.00	0.00	0.00	-0.01	0.01
t-3 subnet	node 39	-0.02	0.04	0.08	0.04	0.03
t-2 subnet	node 40	-0.01	-0.01	0.01	0.00	-0.01
t-1 subnet	node 41	-0.02	-0.01	0.00	0.01	-0.01
t subnet	node 42	0.25	0.18	0.15	0.09	0.07

Table 10. Discriminant parameters based on the training patterns, for the 5-TNN with the best performance.

hidden node	To(t+1)	To(t+2)	To(t+3)	To(t+4)	To(t+5)
node 25	-0.03	0.04	0.12	0.20	0.25
node 26	-0.08	0.12	0.23	0.27	0.25
node 27	0.09	0.08	0.08	0.05	0.03
node 28	0.46	0.33	0.15	-0.03	-0.17
node 29	-0.01	-0.01	-0.02	-0.02	-0.01

Table 11. Discriminant parameters based on the testing patterns, for the 5-TNN with the best performance.

hidden node	To(t+1)	To(t+2)	To(t+3)	To(t+4)	To(t+5)
node 25	-0.01	0.02	0.05	0.09	0.11
node 26	-0.01	0.01	0.03	0.03	0.03
node 27	0.05	0.04	0.04	0.03	0.02
node 28	0.37	0.27	0.12	-0.03	-0.13
node 29	-0.03	-0.04	-0.06	-0.07	-0.06

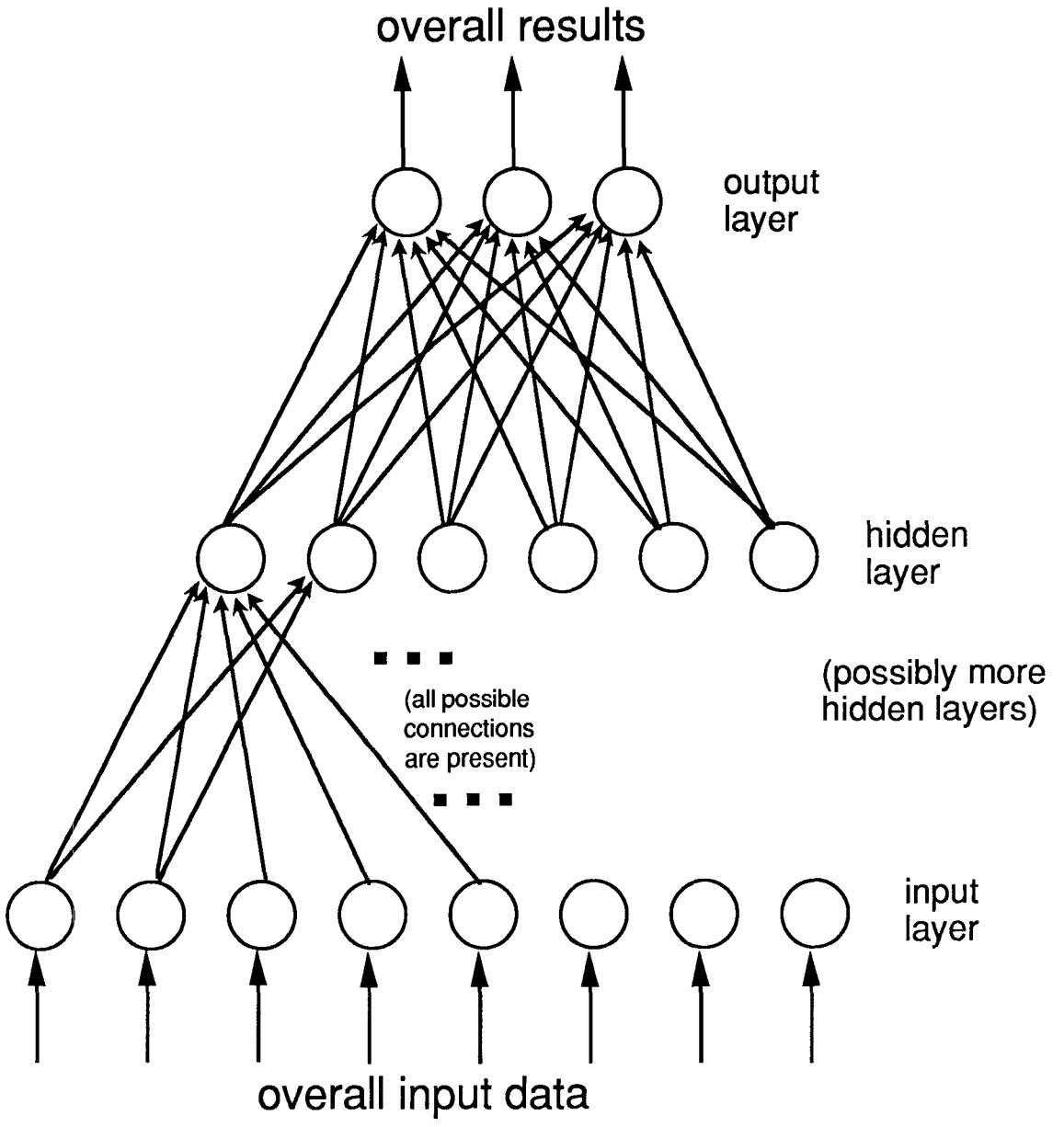


Figure 1. A sketch for a three layer traditional neural network. A connection signifies transfer of information in the direction of the arrow. The number of nodes in each layer can be as large as required; there can also be more than one hidden layers. All possible connections from one layer to the next are present; here, only the connections from the hidden layer to the output layer are drawn, to avoid cluttering the figure.

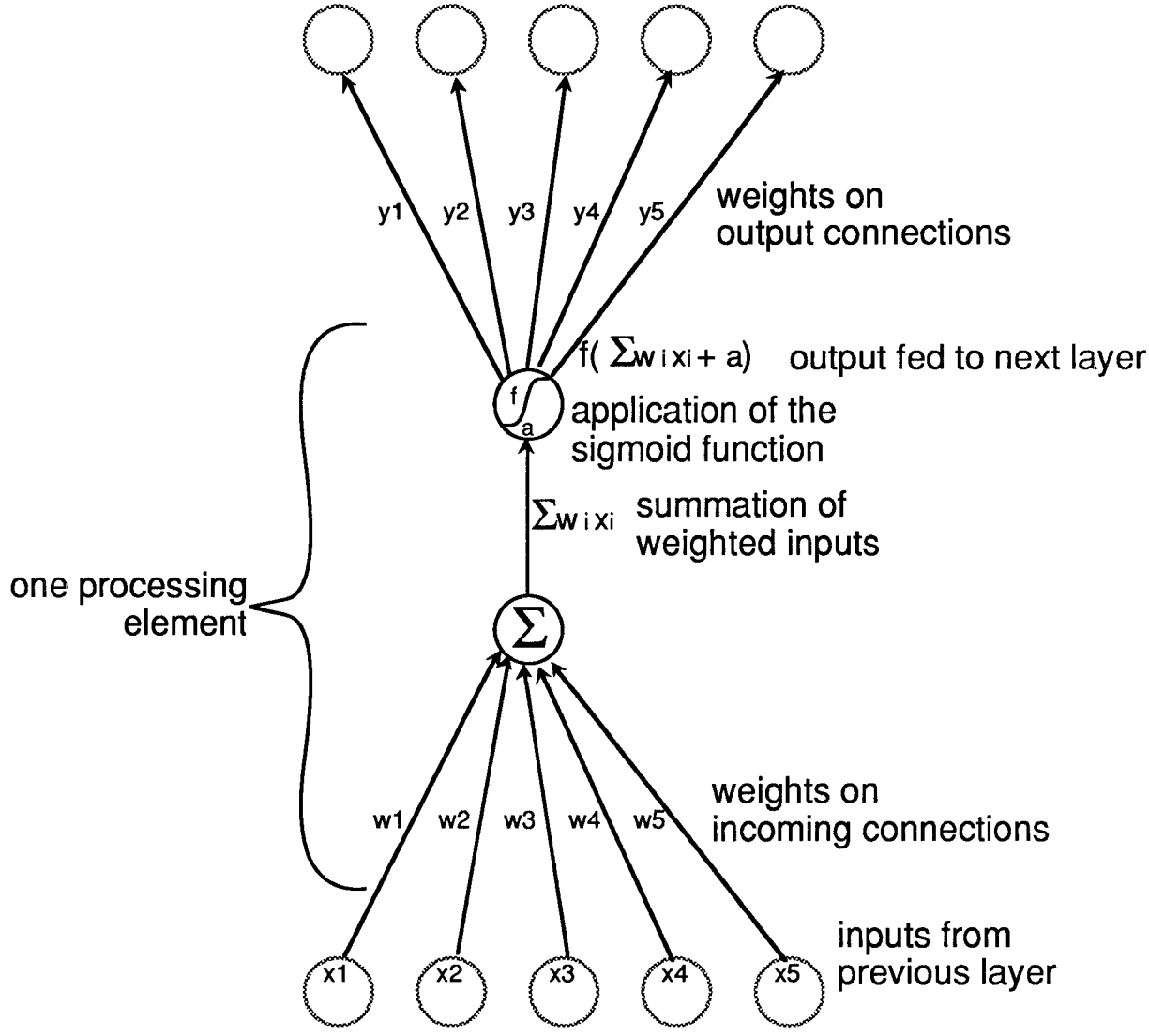


Figure 2. Processing carried out by a single node.

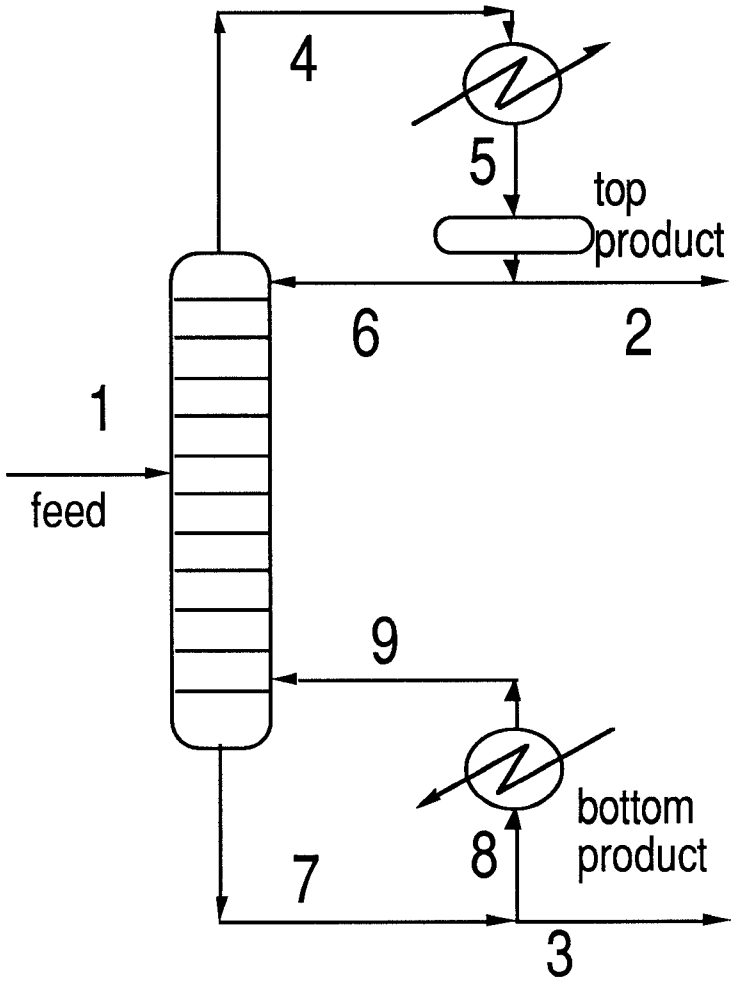


Figure 3: A distillation column, as an example of a complex dynamic system.

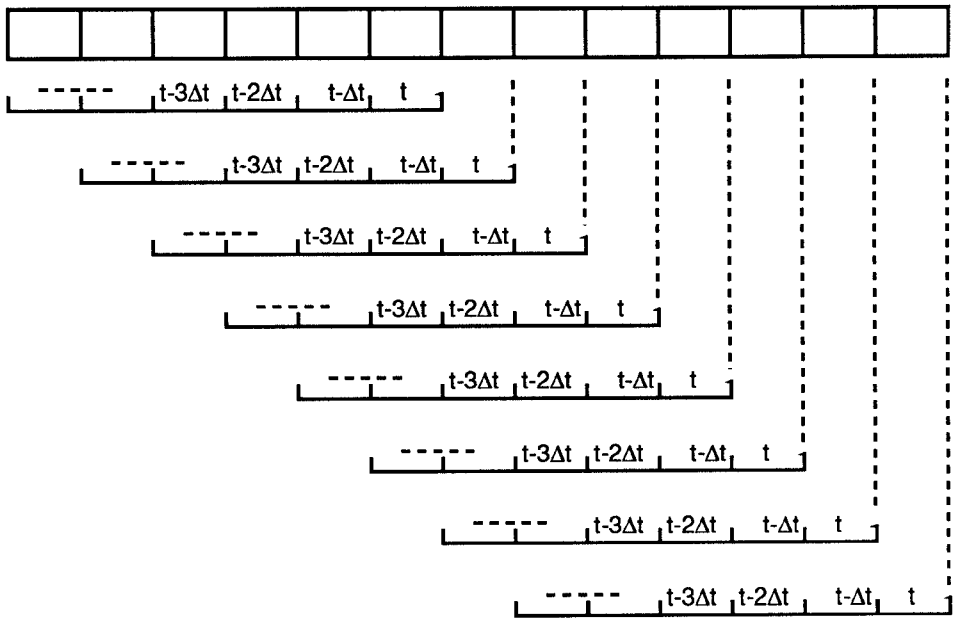


Figure 4: A moving window is a simple way to isolate subsets of a long string of measurements. For training, a moving window allows the creation of several training examples from raw data.

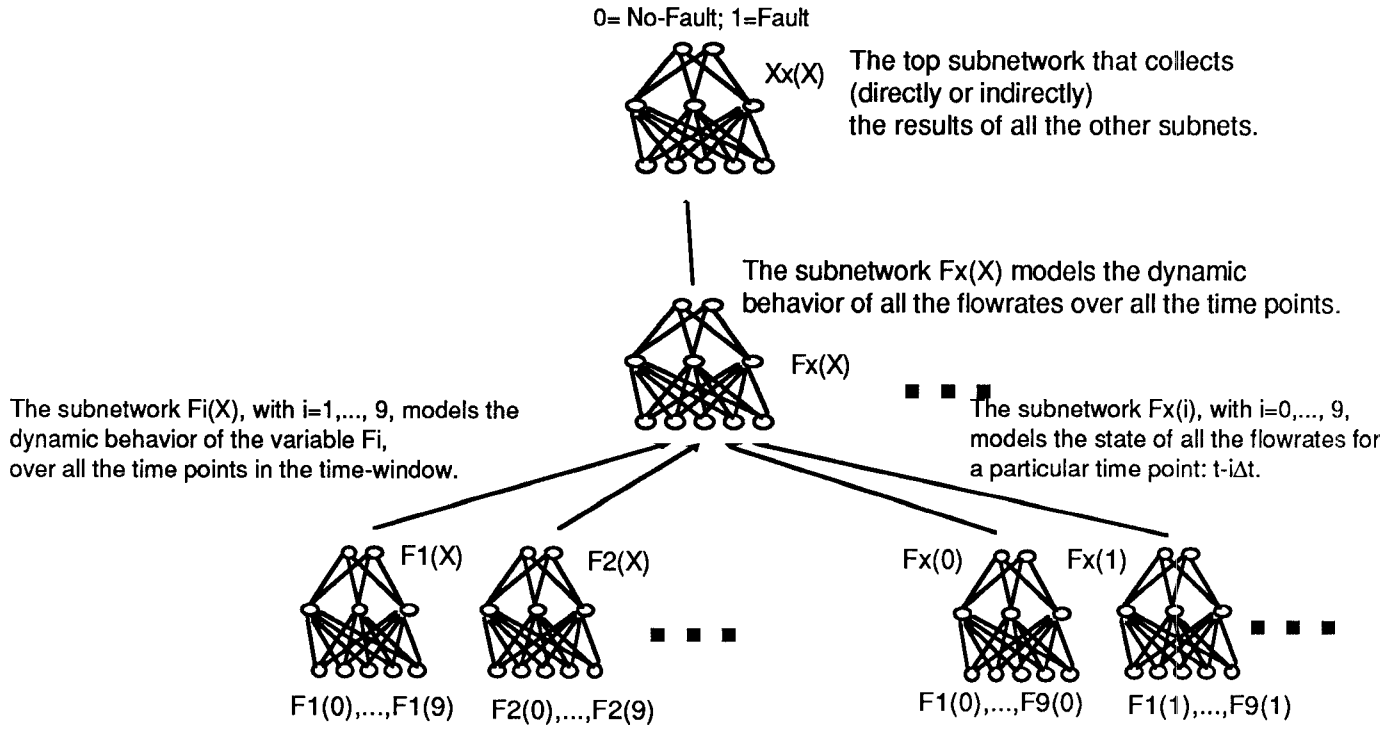


Figure 5: Part of the structure of a hierarchical neural network for monitoring the distillation column of Figure 3.

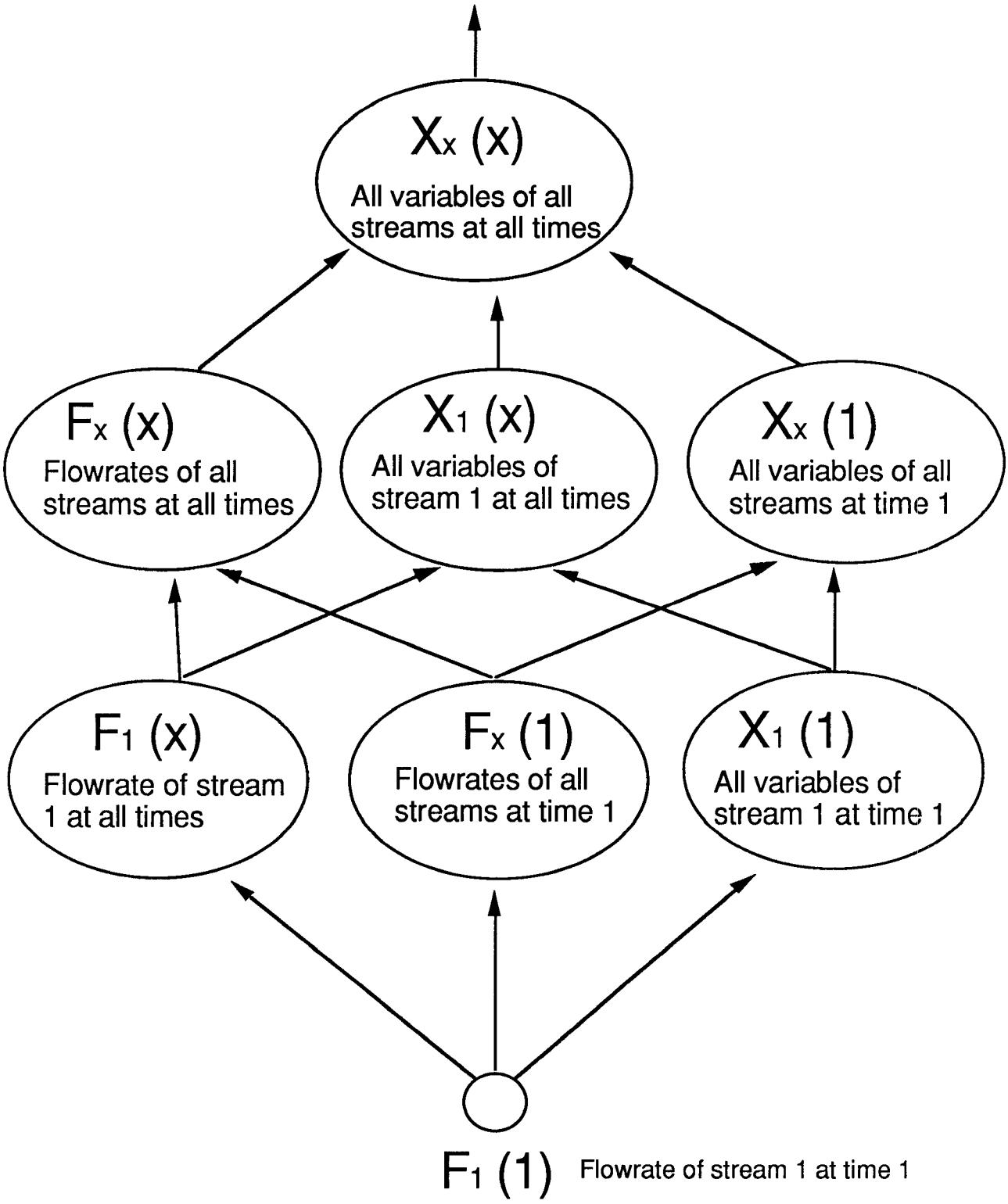


Figure 6. Each variable is propagated through many subnets

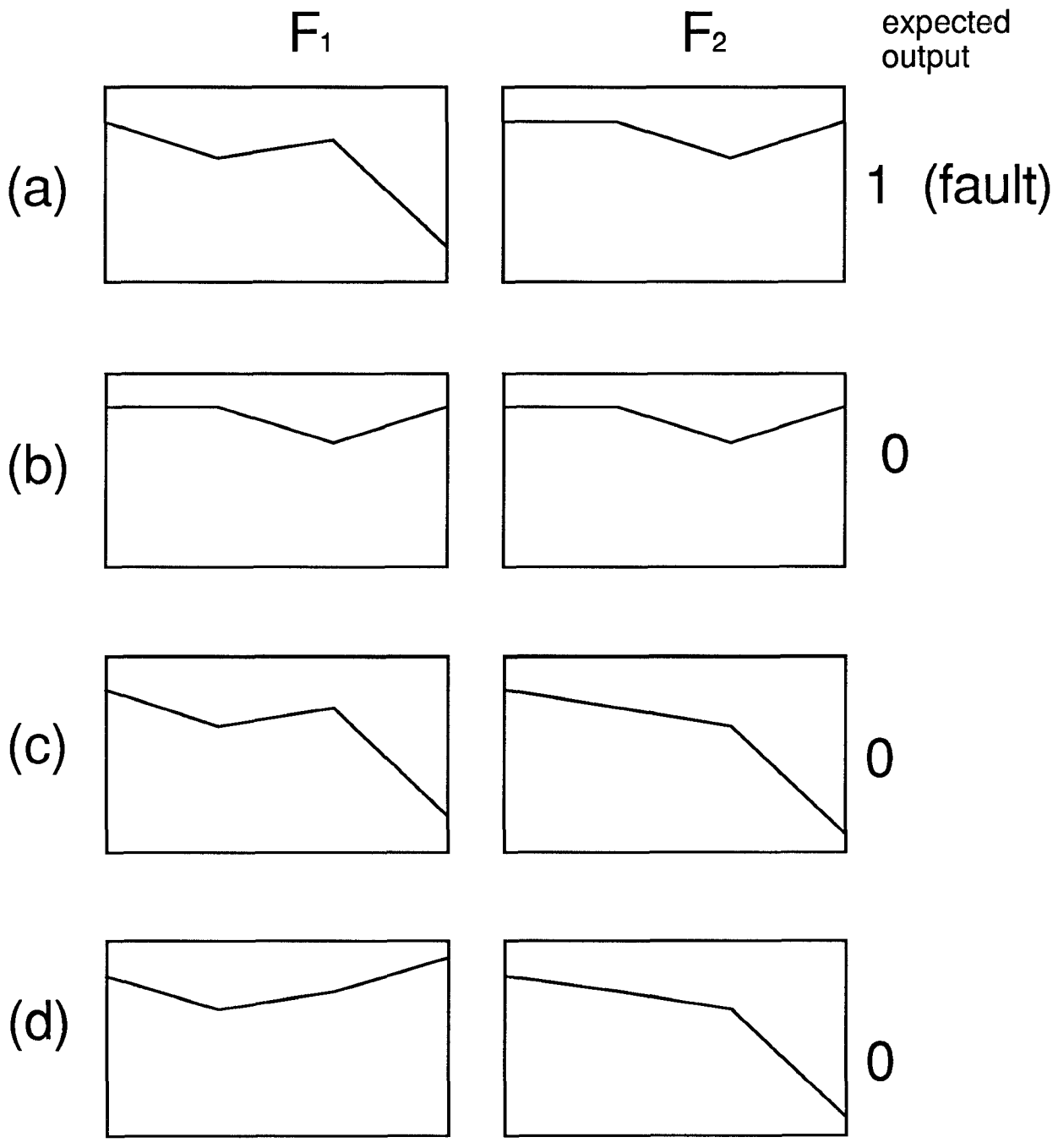


Figure 7. Definition of the output for a simple example. When there is a sharp drop in the signal F_1, but no similar drop in F_2, there is a fault and the output of the net is expected to be 1. In all other cases, the output should be 0.

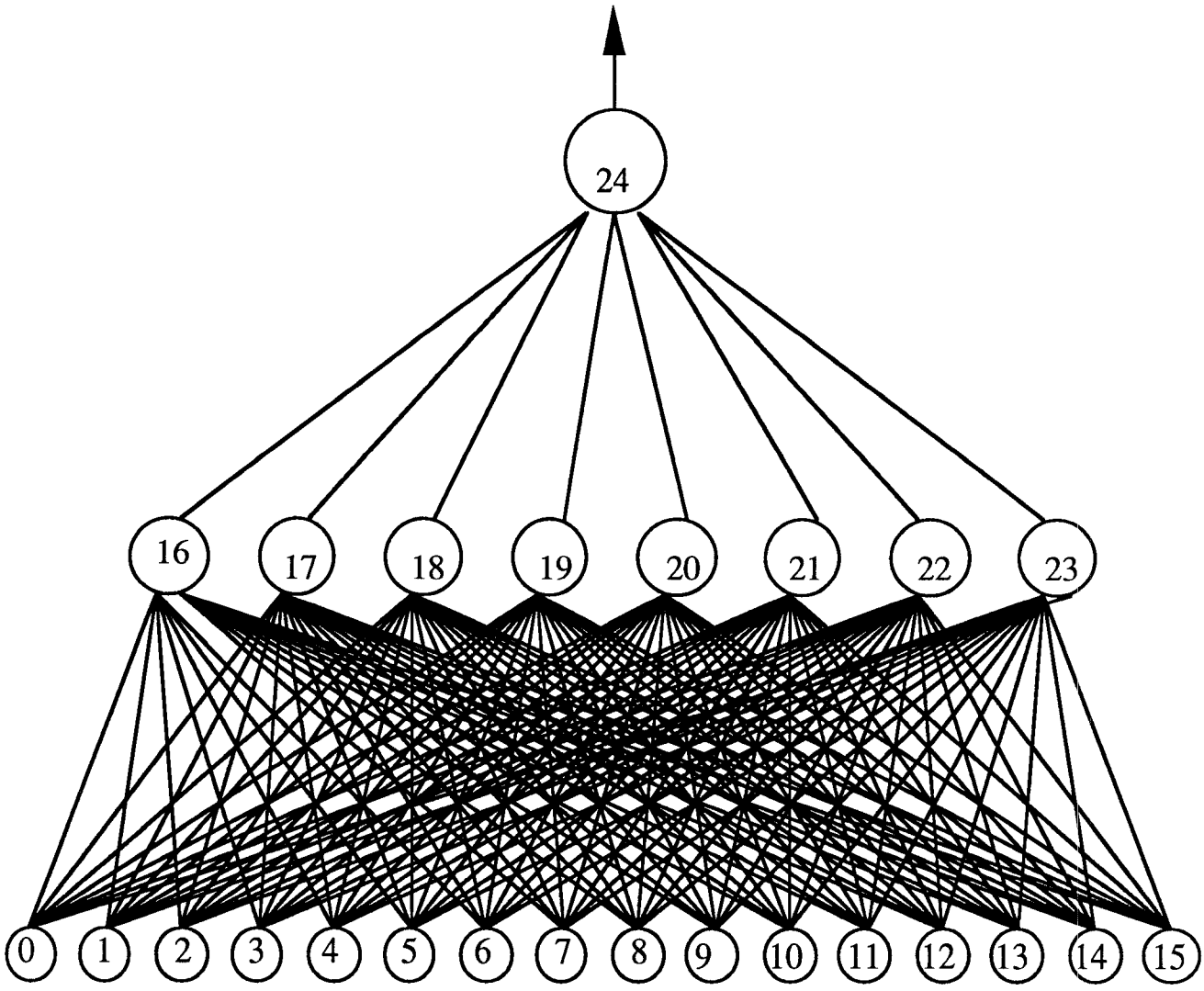


Figure 8. Structure of 8-TNN which is a traditional neural network with one hidden layer containing 8 hidden nodes . The 25 nodes (input, hidden, and output) are numbered from 0 to 24.

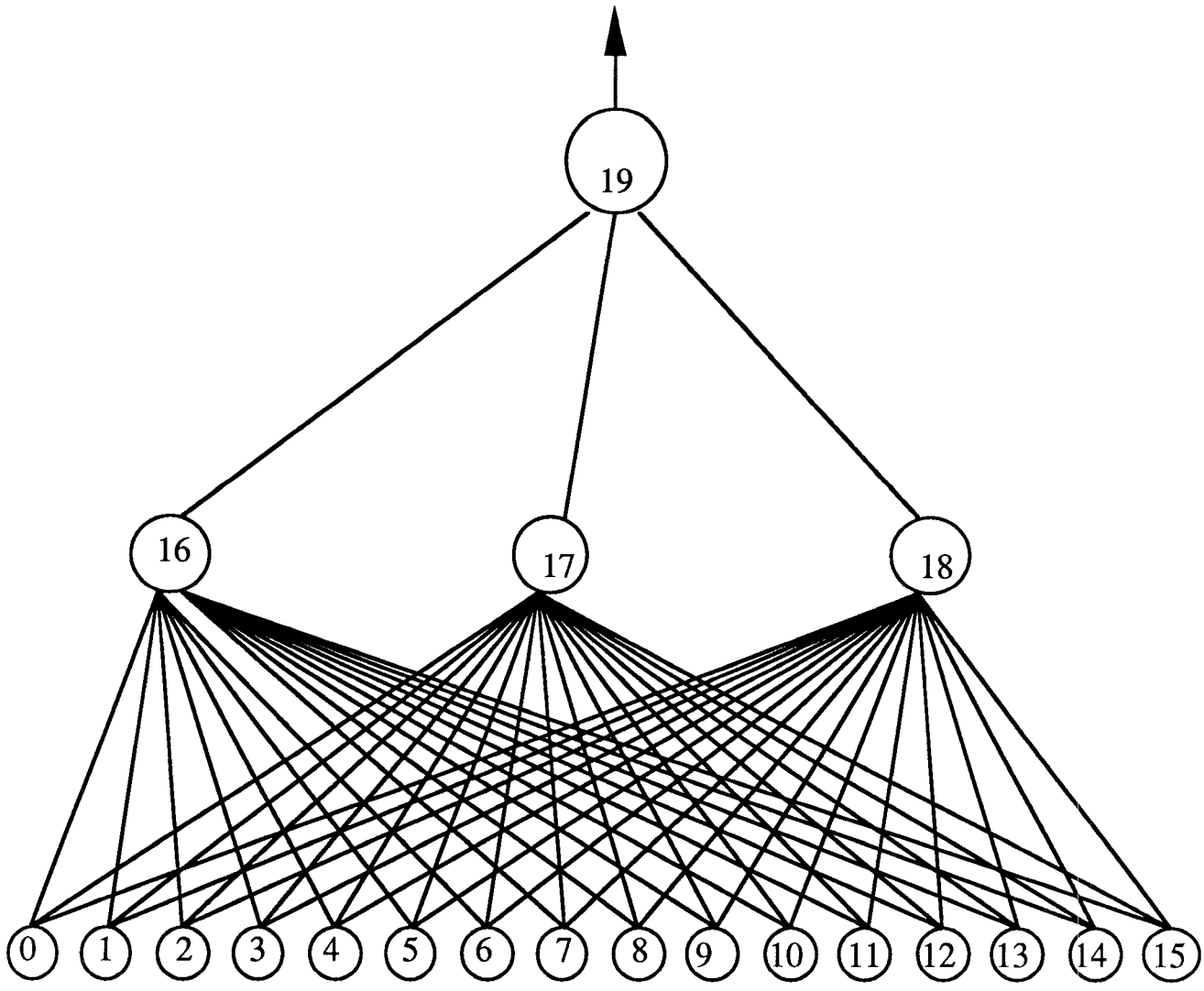


Figure 9. Structure of 3-TNN which is a traditional neural network with 3 hidden nodes.

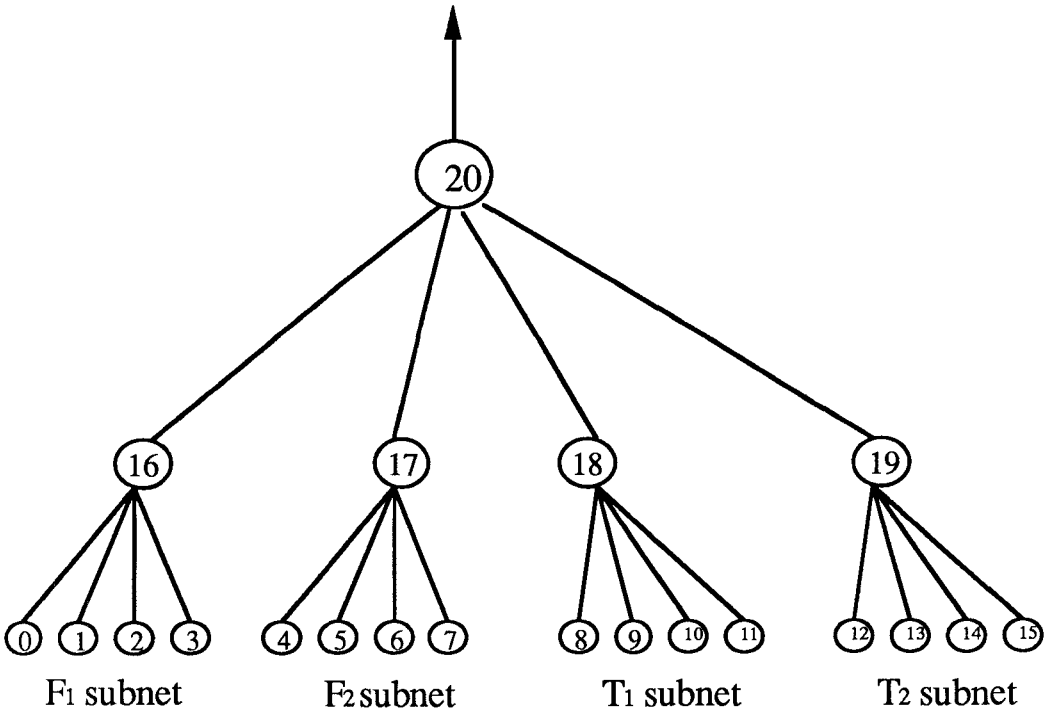


Figure 10. The network 4-HNN-p used in the sharp drop example. There are 4 subnets in this network structure. Each subnet is responsible for examining the dynamics of one system variable only. Each subnet consists of just one hidden node.

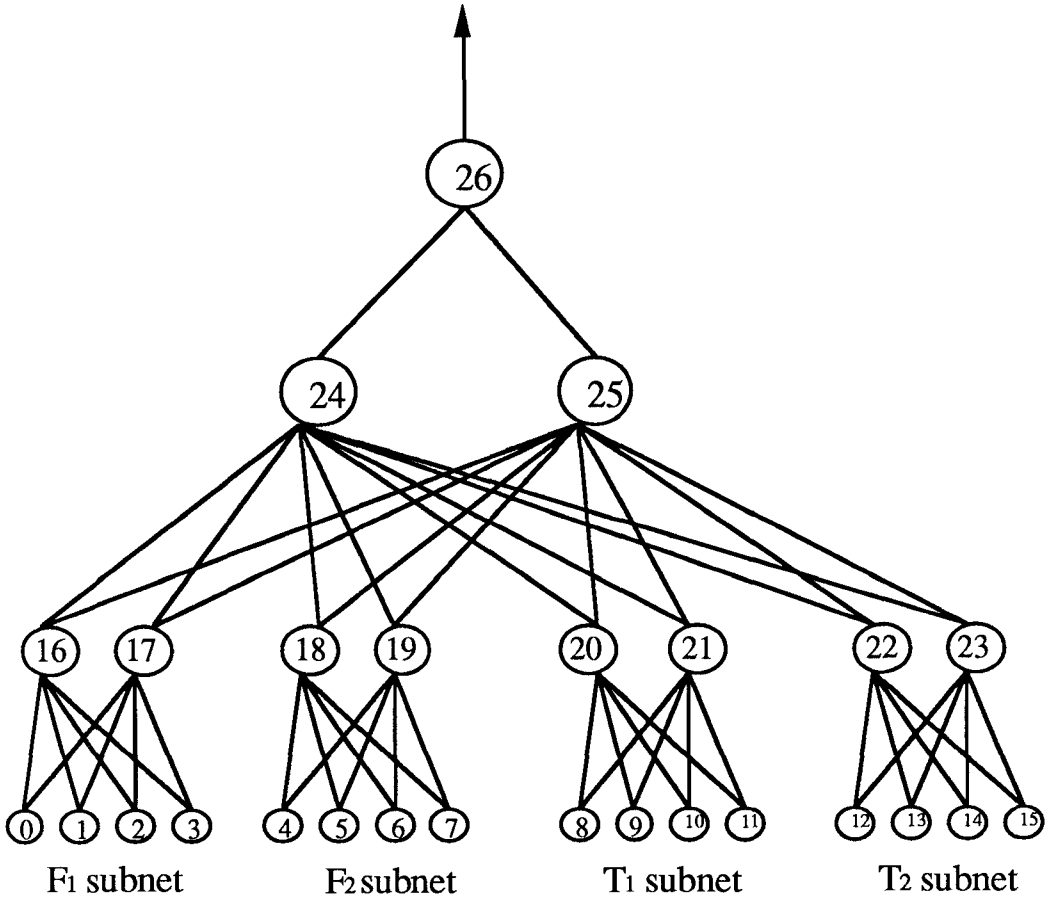


Figure 11. The network 8-2-HNN-p used in the sharp drop example. This network has two hidden layers.

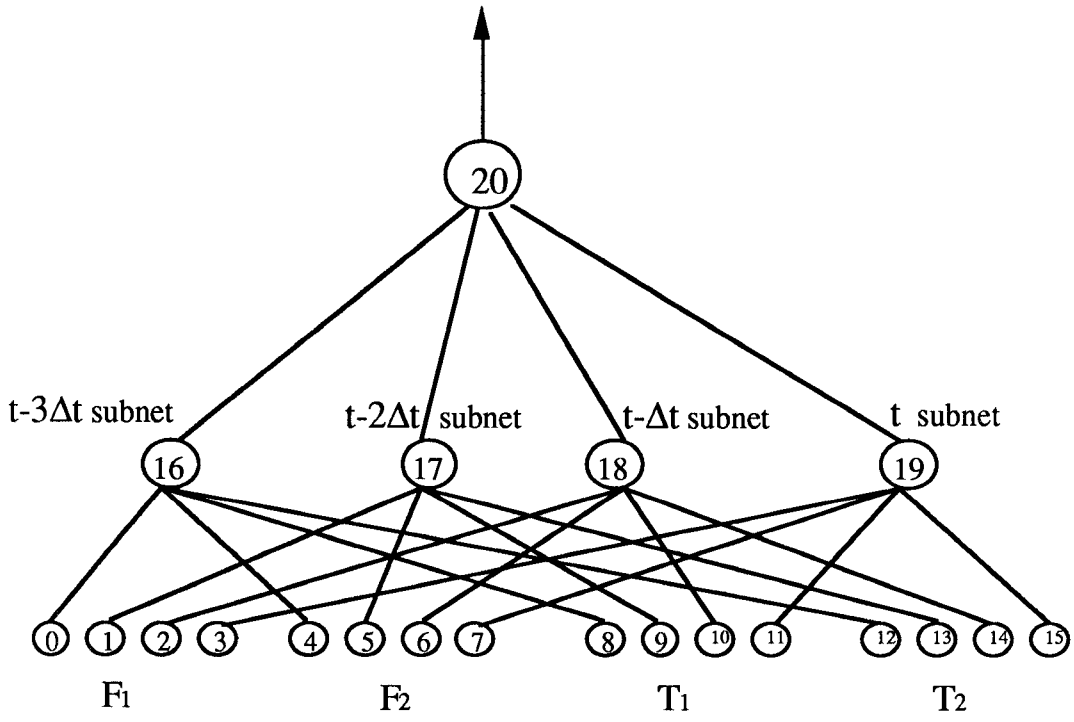


Figure 12. The network 4-HNN-t used in the sharp drop example. There are 4 subnets in this network structure, each subnet is responsible for examining one time-point. Each subnet is just one hidden node.

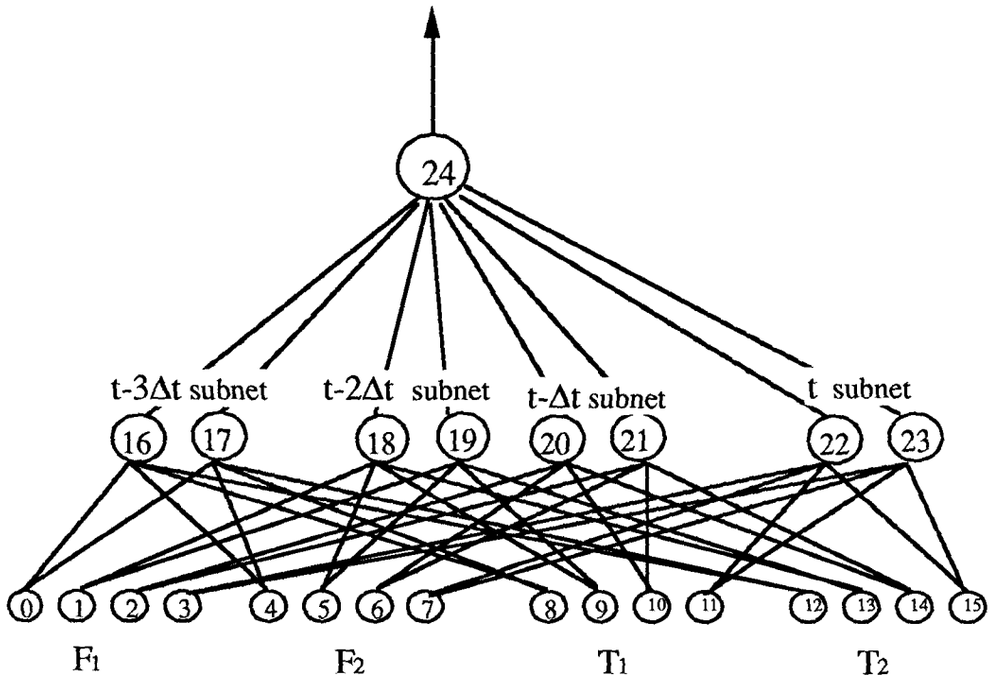


Figure 13. The network 8-HNN-t used in the sharp drop example. There are 4 subnets in this network structure, each consisting of two hidden nodes in a single layer.

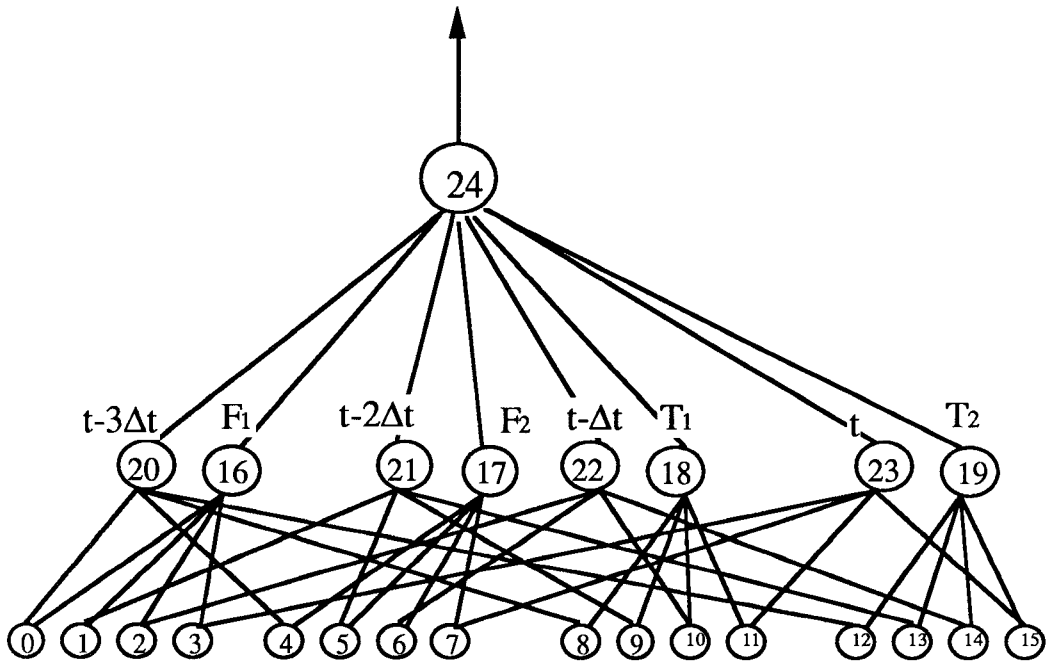


Figure 14. The network 8-HNN-pt used in the sharp drop example. There are four hidden nodes for the four time-point clusters and four hidden nodes for the four parameter clusters.

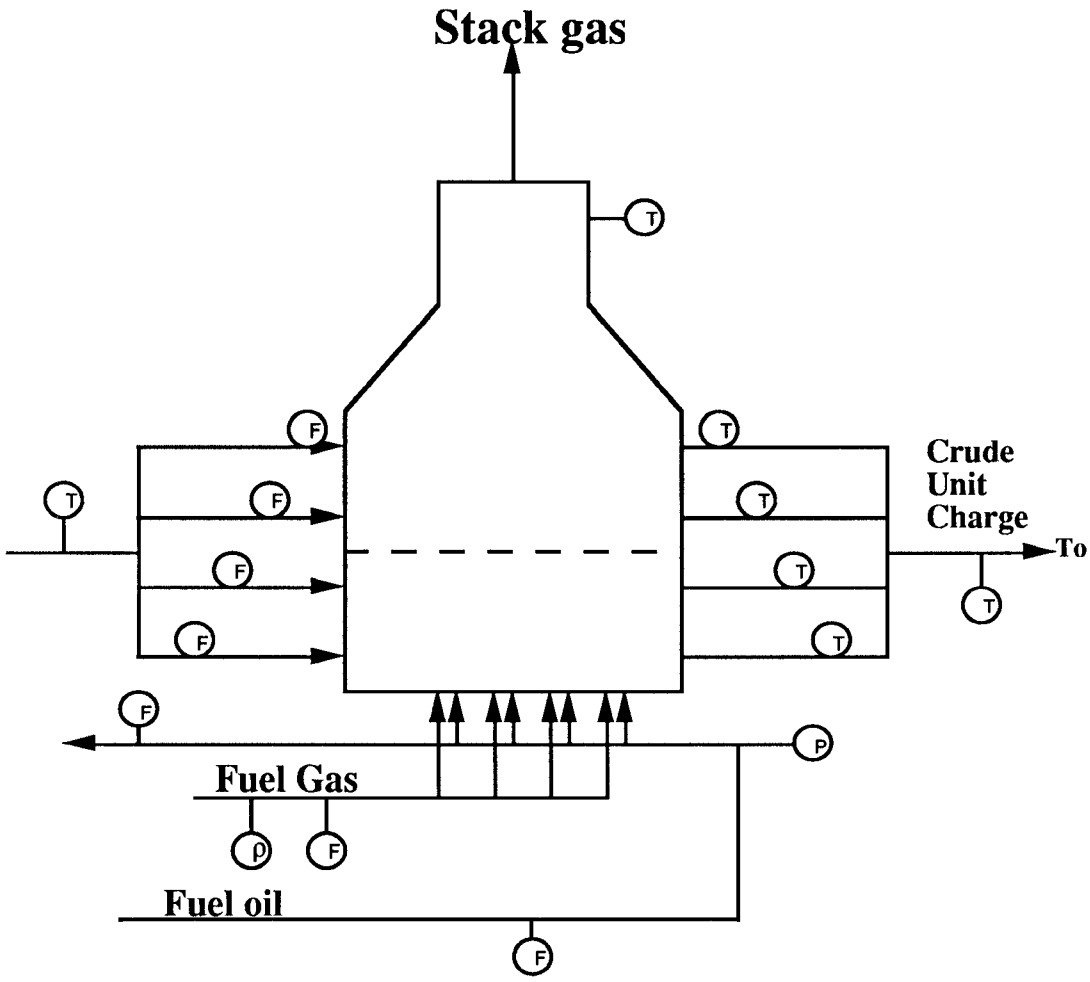


Figure 15. A crude oil heater. Important temperatures (T), flowrates (F), pressures (P), and a specific gravity (ρ) are shown.

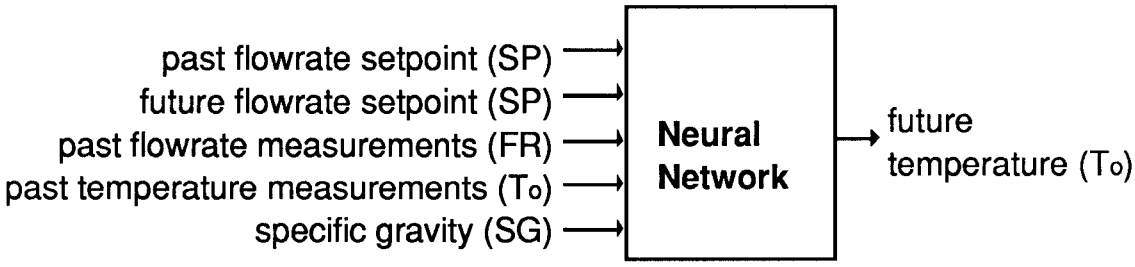


Figure 16. The dynamic-modelling task that neural networks will carry out for the crude oil heater of Figure 15.

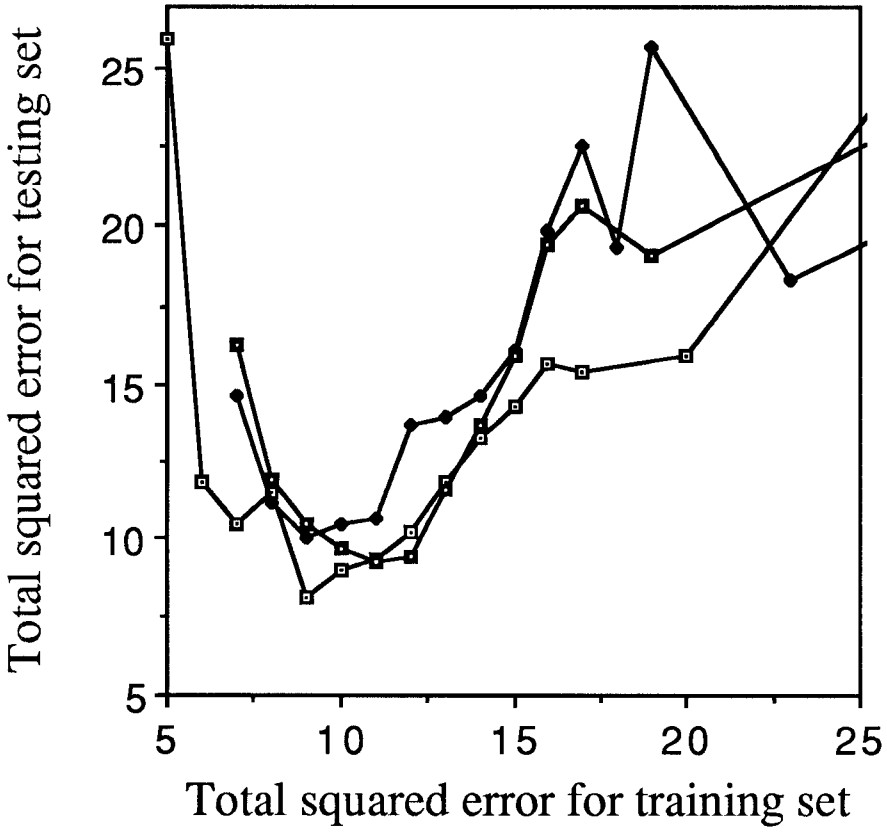


Figure 17. Total sum of squared errors on the testing patterns vs. total sum of squared errors on the training patterns for three runs of the 18-HNN (starting with different random weights) for the heater problem. The optimal testing error ranges from 8.1 to 9.9 and occurs when the training error is between 9 and 11.

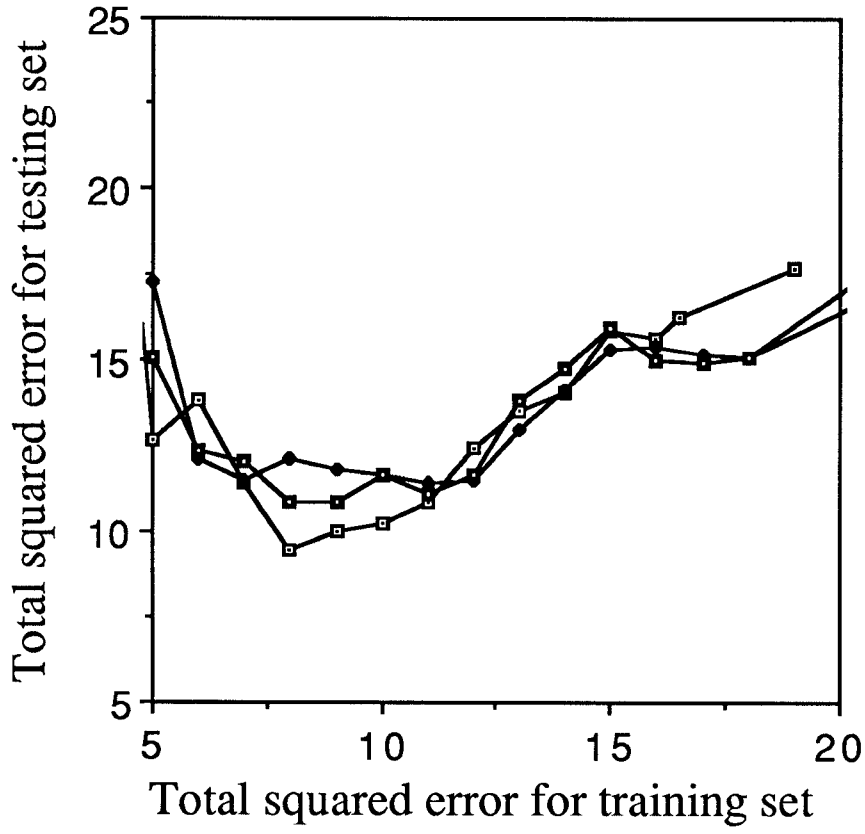


Figure 18. Total sum of squared errors on the testing patterns vs. total sum of squared errors on the training patterns for three runs of 5-TNN. The optimal testing error ranges from 9.5 to 11.4 and occurs when the training error is between 8 and 11.

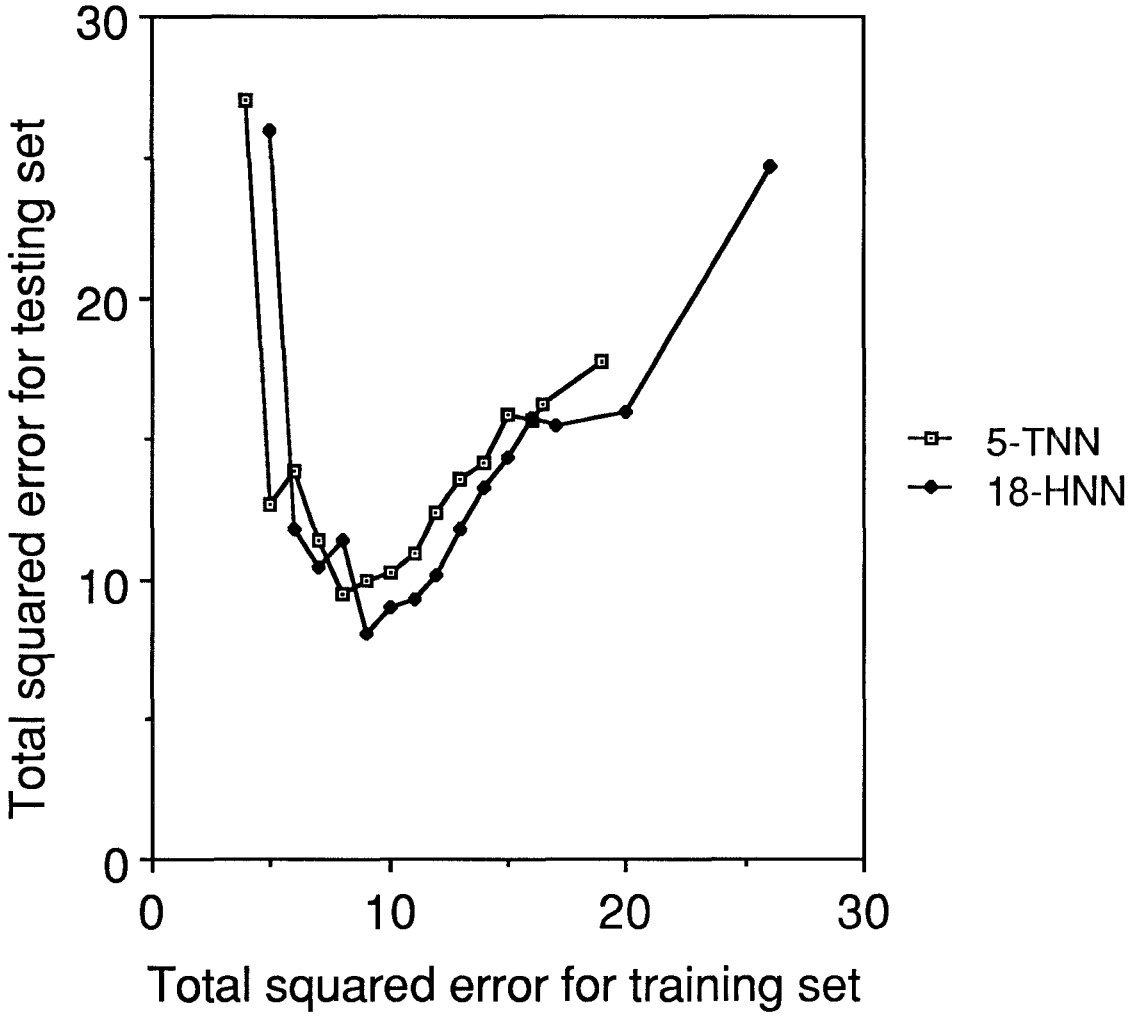


Figure 19. Total sum of squared errors on the testing patterns vs. total sum of squared errors on the training patterns for the best run of 5-TNN and the best run of 18-HNN. The testing error is 8.1 for 18-HNN and 9.5 for 5-TNN.