Control of Air-Cooled Chiller Condenser Fans using Clustering Neural Networks

Gregor P. Henze, Ph.D. **Richard E. Hindman, Ph.D.**
Associate Member ASHRAE

Gregor P. Henze is an assistant professor at the University of Nebraska – Lincoln and Richard E. Hindman is a research associate at the University of Colorado at Boulder.

ABSTRACT

This study investigates task-blind and task-specific training methods to determine appropriate radial basis function based neural network architectures. These neural nets identify system behavior of air-cooled chiller condensers by grouping dominant features (clustering) of measured chiller performance data. Task-specific clustering proved superior but more computationally demanding than task-blind methods in learning a difficult task of fan operation for an air-cooled chiller. Seven measured variables were selected as relevant for the operation of two variable-speed and multiple fixed-speed condenser fans. All neural network architectures investigated successfully learned the functional 7-input/4-output mapping. Process control logic post-processes the noisy neural network control signal and evaluates operational and temporal constraints. The neurocontroller trained on the measured data set exhibits roughly similar performance compared to manufacturer-provided control, while offering reduced development time and efforts.

INTRODUCTION

Basics of Refrigeration Cycles

Air-cooled chillers are devices used for comfort control in commercial buildings. In contrast to the typical air-conditioner used in residential environments that make use of the refrigerant's cooling capacity directly, chillers generate chilled water in an intermediate step. The chilled water is used to cool down the air supplied to the conditioned spaces in order to compensate for the gains generated by people, equipment, or solar radiation gains through windows.

The refrigerant as the primary working fluid undergoes a cyclic thermodynamic process known as the vapor-compression cycle. In the initial step the fluid enters a heat exchanger – the evaporator – at low pressure and in two-phase state of low quality and by absorbing heat from the inflowing water continuously evaporates until it becomes a slightly superheated gas. The subsequent compression of the fluid increases its pressure and temperature. In order to remove the heat absorbed in the evaporator (and compressor), a second heat exchanger – the condenser – is necessary.

Since the fluid entering the condenser has a temperature above ambient, either water or air can be used to extract the heat from the fluid, thereby condensing and slightly subcooling the refrigerant. If ambient air is used for that purpose, as in the given investigation, the chiller is said to be *air-cooled*, in the other case *water-cooled*. To close the cycle, an expansion valve is provided to reduce the pressure (and temperature) in the refrigerant, thereby transferring it from the slightly subcooled liquid state to the two-phase, low-quality state, a point at which it enters the evaporator and the cycle is closed. Please refer to Figure 1 b) for illustration.

Problem Statement

Part of the problem of controlling an entire chiller, is the issue of controlling the condenser fans that draw ambient air across the condenser coils. Air-side control is typically accomplished by one of three methods or a combination of two of them: (1) fan cycling, (2) modulating dampers, and (3) fan speed

control (ASHRAE, 2000). The systems investigated in this study use fan cycling and fan speed control. The heat transfer rate is – among other parameters – a function of the speed of air flowing around the coils. For a given refrigerant flow rate and condenser entering conditions, the air flow determines at which temperature and pressure the refrigerant condenses. In the given chiller configuration there is one evaporator and two helical-rotary compressors each serving an individual condenser. Refrigerant charge is 148 lbs (67 kg) R-22. Each of the two condensers has one bank of 26-inch (66 cm) diameter propeller fans. Two types of fans are used: one inverter-driven variable-speed fan (VAV) and a set of fixed air-flow fans (CV), the number of which depends on the chiller size. For the given case of 120 ton (420 kW) chiller, each bank of fans has five fixed-speed fans and one variable-speed fan. For reasons of clarity, Figure 1 b) shows only one of the two compressors and refrigerant circuits.

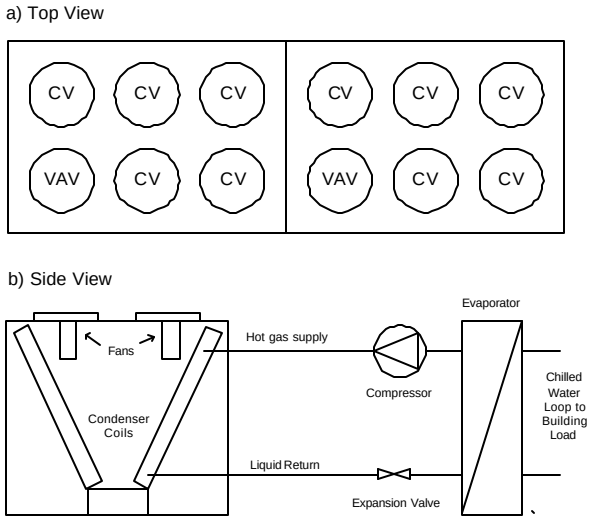


Figure 1: Schematic of air-cooled chiller condenser and fans to be controlled.

A list of requirements has been formulated for the fan control to perform adequately:

1. On startup, a minimum pressure differential between the saturated condensing and suction pressure of about 350 kPa (50 psi) has to be achieved quickly.
2. An operating pressure differential within a specified range such as 415–480 kPa (60–70 psi) should be maintained.
3. A maximum saturated condensing pressure (e.g., 2,500 kPa or 375 psi) may not be exceeded.
4. Fan cycling is to be minimized.
5. The control signal to the variable-speed fan should be stable.

The motivation for this investigation is to determine the usefulness of incorporating artificial neural networks to achieve similar or even superior control performance when compared to conventional PID control. Future work will evaluate the potential of using the trained nets for chillers whose PID control parameters would have to be determined by empirical loop tuning methods such as the response curve or ultimate frequency methods. Neural networks hold the promise of adequate generalization from one specific system to another and a reduction in development time. However, implicit to connectionist modeling is the drawback that no physical model will be created to validate one's thinking. For that purpose, data from dynamic tests were made available by the manufacturer of the investigated chiller model. The data were used to train networks off-line, with the purpose of appropriately generalizing the behavior of the chiller and the corresponding control signal. The underlying assumption is that the system, which the data were acquired from, controls the condenser fans in a desired fashion.

The last three items in the above list of control objectives include a time-dependency that restricts the change in control signal from one time step to the next. Since the network does not know the sampling rate of the data acquisition system and to remain flexible to changes with respect to the latter, it appeared appropriate to divide the control sequence into two parts: In the first part the sensor readings are fed into the input layer of the neural network which in turn produces a continuously-valued output signal with four

control variables in the interval [0,1]. This output signal represents the fractional use of the fixed speed and variable-speed fans for the two refrigerant circuits. If this signal were to be used without any post-processing, it is possible that the network forces the fans to undergo unacceptably high cycling in a single fan or drastic changes in the total number of fans running. To avoid the above mentioned complications, the second part of the control sequence will enforce the constraints necessary to keep the fans under stable control. A process control logic (PLC) will keep track of all time-related issues such as the minimal ramp time for the variable-speed fan, the minimal hold period before a fan can be turned on again after it has been turned off (or vice versa) and others.

DESCRIPTION OF DATA

A total of approximately 150 data points were monitored by the manufacturer in the dynamic testing of the investigated chiller system. The sampling rate was 25 seconds; 341 time stamps were taken resulting in an overall testing time of close to 2 hours. The overall trend of the test was that a slowly increasing chilled water return temperature at constant flow rate imposed an increasing demand for cooling from the chiller. The chiller nominal capacity is 120 tons of refrigeration (420 kW_{th}) and the chiller load was varied between 0% and 100% of this value. Chiller capacity is controlled by varying the refrigerant mass flow rate. To condense an increasing flow of refrigerant to desired conditions implies an increase in the available air flow rate. The test represented a continuous ramping of the water-side cooling capacity over the entire period, except for a short period of 4 minutes about 15 minutes into the test during which no cooling is required and the condenser fans shut off. This particular period was expectedly the one hardest to train for.

Each condenser fan was individually monitored. To translate these 12 signals into a smaller number of output variables and remain flexible with respect to the total number of installed fans, it was decided to "lump" the five constant-speed fans together into one signal for each of the two circuits. The third and fourth output signals are the control signals for the variable-speed fans of both circuits. For each of the fan circuits, seven measurements points were chosen to be closely associated with condenser fans and their control as shown in Table 1 Future work will investigate the effect of adding and subtracting input variables on the networks ability to appropriately generalize the control behavior. All variables were normalized to the range [0, 1] before network training.

Table 1: Selected input and output variables

Input Variable	Output Variable
Outdoor air temperature [°F]/ [°C]	Number of fixed-speed fans on, circuit 1
Evaporator cooling capacity [tons]/[kWth]	Number of fixed-speed fans on, circuit 2
Total refrigerant flow rate [lbm/min]/[kg/s]	Variable fan speed, circuit 1
Entering condenser refrigerant temperature, circuit 1 [°F]/ [°C]	Variable fan speed, circuit 2
Entering condenser refrigerant temperature circuit 2 [°F]/ [°C]	
Leaving subcooler refrigerant temperature, circuit 1 [°F]/ [°C]	
Leaving subcooler refrigerant temperature, circuit 2 [°F]/ [°C]	

BASICS OF NEURAL NETWORKS

Connectionist modeling has been a research area of rapid growth over the last decade, though many of the fundamental concepts had been developed in the 1960's and earlier. Neural networks set themselves apart from sequential computation by distributing the computational tasks of a problem onto many identical simple units ("neurons") that are highly interconnected and can work in parallel. Hence, the commonly used term *parallel distributed processing* (Rumelhart and McClelland, 1988). The name "neural network" was coined due to the fact that these networks show some – but not significant – resemblance to the brain. The largest networks in use today are perhaps comparable to a bee's brain. Originally, connectionist models were developed as models of groups of neurons and less as an entire brain (Hertz et al., 1991).

The concept of artificial neural networks as applied to adaptive control is considered attractive for three dominant reasons. The first reason is that they have proven to deal surprisingly well with problems where numerous simultaneous constraints have to be considered. The trade-off between control objectives such as maintaining setpoints and minimizing energy use can therefore be incorporated. The second reason is that they are capable of performing any arbitrary *nonlinear* mapping of input-output patterns, i.e., they can approximate any continuous function to any degree of accuracy. This use for regression alone, which

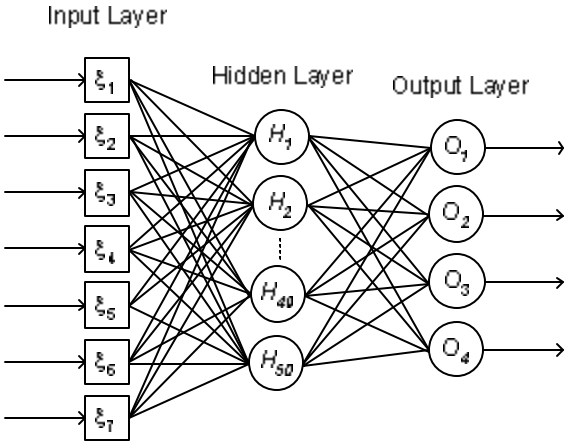
e.g. is shared with polynomials and splines, does not explain their popularity. Rather, as the third reason, by virtue of *backpropagation* (Anderson and Rosenfeld, 1988) – a means of automatically adjusting the network's parameters as to minimize a specified error function involving the observed (target) and predicted output value – they exhibit the ability to "learn" changes that occur in the environment, a fact that makes them ideal for uncertain and non-stationary surroundings. The topic of continuous learning and how to implement when and how much to learn, is an involved topic not further discussed here.

In recent years, artificial neural networks have been applied increasingly to complex nonlinear tasks relevant to the operation of buildings such as building control (Jeanette et al. 1998), building fault diagnostics (Breekweg et al., 2000), and building energy prediction (Dodier and Henze 1996).

Moreover, neural networks have proven to be superior for state-space reconstruction and are therefore well suited for time series prediction, such as load and weather forecasting (Weigend and Gershenfeld 1994). In this paper, particular emphasis is placed on radial basis networks, which are applied for tasks in which features of large sets of patterns are to be grouped into smaller sets of feature clusters, using a process called *clustering*. Neural networks are well-suited for modeling problems which lie in the *data-rich* and *theory-poor* domain. In summary, neural networks appear to be well-matched for the identification of systems which lie in the data-rich and theory-poor domain as well as for control of problems that involve non-stationary, nonlinear, and dynamic behavior. A glossary of neural network terms is provided at the end of this article.

INVESTIGATED NETWORK ARCHITECTURES

Artificial neural networks are, in their broadest interpretation, nonlinear regression models, which develop a functional relationship (mapping) between input variables and output variables. The network architecture describes how the input variables are processed on the way to the output variables. All networks discussed here are feedforward networks that, unlike their autoregressive peers, do not use past outputs as inputs. Sets of values of input variables are called pattern vectors (ξ) as each value is associated with a particular input variable. These pattern vectors are presented to the input layer of the feedforward neural network. Each input variable unit is connected with each neuron (same as unit) in the subsequent layer, known as the first hidden layer, and scaled by a weight factor. These neurons H_i are simple numerical elements that carry out any of the linear and nonlinear computations listed in the section on activation functions. The input variables to each of these units are given weights w and a bias term b accommodates an additional scaling effect. The results of these calculations are the outputs of the hidden neurons of the first layer. In the simplest case, these outputs are connected to the subsequent output layer and processed using another set of weights and biases to form the output vector. The output units O_i can employ any activation function, typically however, a linear activation is chosen for the output. This would be called a two-layer network since commonly the input layer is not counted. The number of input, hidden, and output units as well hidden layers, the type of activation function in each layer (except for the input) characterize the network architecture. Figure 2 shows a feedforward network architecture typical for this study: the input layer consisting of seven units (one for each of the selected relevant parameters) is followed by a hidden layer with 50 units, which is followed by the four-unit output layer representing the control signal.



Activation Functions

Below are descriptions of the activation functions used in the investigated architectures. The activation function describes how each input of the processed to form the neuron's output. The first and most important is based on radial basis functions. These are the clustering elements used in each of the networks. Short descriptions of sigmoid and linear units are also included.

Radial Basis Function

All the architectures explored in this project investigate the effects of clustering. This clustering is achieved with a hidden layer of radial basis function (`rbf`) neurons (Hertz et al. 1991). The activation functions for this `rbf` layer are a modified version of the normalized Gaussian form

$$g_j(x) = \frac{\exp\left[-\frac{1}{2}\left(x - w_j\right)^2 b_j^2\right]}{\sum_k \exp\left[-\frac{1}{2}\left(x - w_k\right)^2 b_k^2\right]}. \quad (1)$$

In the above equation, the activation value for a neuron increases as the Euclidean distance of the weight vector w_j and the pattern vector ξ decreases. The activation, i.e., the output, for a neuron approaches a maximum when the weight vector is very near the pattern vector (distance is small). Thus, input patterns that excite the same `rbf` neuron in the hidden layer are classified in the same cluster. The bias b_j is defined as the inverse of the standard deviation of the Gaussian. As the neuron bias increases, the standard deviation decreases, and the peak of the `rbf` function becomes narrower. Of course, the opposite is also true. This allows the cluster to become more or less selective as needed. The activations for the `rbf` layer are then normalized such that the sum of the `rbf` activations is unity. This ensures that winning clusters for different patterns can be properly compared. In addition, varying numbers of hidden units can be used without the total activation to the next layer changing. However, activations are not normalized for all network architectures, specifically when backpropagation is used through the `rbf` layer. Figure 3 illustrates an example where two different pattern vectors ξ_1 = {8, 23, 26} and ξ_2 = {4, 28, 34} are presented to the network below featuring a bias of unity in both `rbf` neurons. According to Eq. (1), when the distance between the pattern and weight vector of an investigated `rbf` neuron j is small $\left(\left(x - w_j\right)^2 \to 0\right)$, the level of activation will be high, i.e., close to unity when normalized.

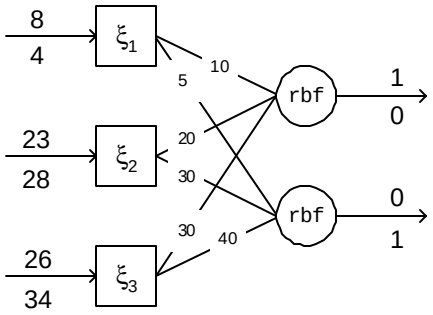


Figure 3: Simple radial basis function activation example.

Sigmoid Function

The log-sigmoid activation function is used as a second hidden layer in one of the explored architectures. It has the form of

$$g_j(x) = \frac{1}{1 + \exp(-h)} \text{ where } h = x \cdot w_j + b_j. \quad (2)$$

This function approaches zero for large negative values of h, and unity for large positive values. The smooth rise from zero to one occurs around $h = b_j$; b_j can therefore be used to shift the activation function as desired.

Linear Function

Linear activation functions are used in all architectures for the output units. Linear unit have the simplest form

$$g_j(x) = x \cdot w_j + b_j \tag{3}$$

This equation states that the output is a weighted sum of the previous layer activations plus bias.

Selected Architectures

The structures of the tested networks are very similar. They mostly consist of an input layer, a hidden layer of clustering `rbf` neurons, and a linear output layer as shown in Figure 2. There are slight structural differences from network to network, but the main difference is in how their hidden clustering layers are trained. Their training can either be *task-blind* or *task-specific*. When the clustering layer is trained without any information of the desired output, the training is called task-blind. However, if the weights entering the clustering layer are determined in the process of learning the particular input-output, it is using task-specific training. Figure 4 illustrates the network architectures investigated in this study. Training is accomplished using standard backpropagation (Hertz et al., 1991), i.e., gradient descent in error space using 80-90% of the available test data as randomly selected training data and the remaining 10-20% of the data set for validation (testing) of learning progress. Withholding 10-20% of the data set allows for the assessment of the network performance on previously unseen events, which is indicative of how well the network generalizes. Each pattern consists of an input pattern vector and the output vector.

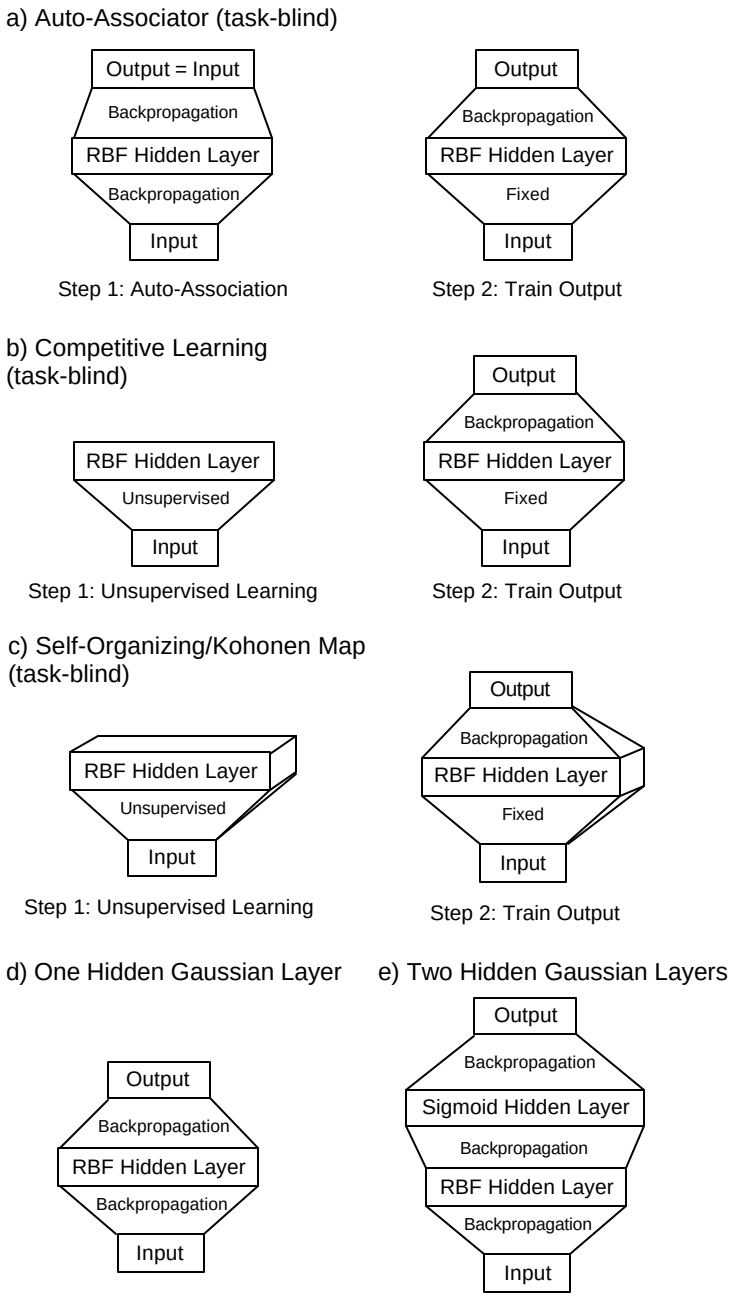


Figure 4: Investigated clustering neural network architectures

Task-Blind Clustering

There are three task-blind clustering approaches investigated in this work. Referring to Figure 4 a) through c), task-blind describes the process of finding patterns in the input space and grouping these features in clusters independent of the control task at hand. Please refer to (Hertz et al., 1991) for a description of the network architectures described as well as excellent treatment of the underlying theory. The only major network architecture not investigated is that of recurrent models such as Boltzmann, autoregressive, and reinforcement learning machines, in which connections are allowed both ways between a pair of units, and even from a unit to itself.

Auto-Associator: The auto-associator network architecture derives its name from the fact that the clustering layer is trained as the hidden layer of an auto-associator configuration: During the first step of

the two-step training process, the network features an equal number of input and output units, and a hidden layer consisting of the previously mentioned `rbf` neurons. This network is trained using backpropagation through both layers. The two-step procedure is illustrated in Figure 4 a). It is called an auto-associator because the network is trained to associatively memorize identical output responses triggered by the corresponding inputs. After the auto-associator is properly trained, the output layer is discarded, the hidden layer weights are fixed, and a new output is added. The weights of the connection to this new output layer are also trained using backpropagation, but the weights from the inputs to the hidden layer remain unchanged. This is an example of task-blind clustering. During the first stage of training, the hidden layer learns to cluster the input patterns in a way that reduces the error of the auto-associator. It has no prior information of what our final outputs will be. The training of this hidden layer is done through *supervised* learning, i.e., comparing the network-predicted output with the known target output and employing gradient descent to adjust each weight in the network to minimize the error contribution of the weight in question. The next architecture is an example of unsupervised, task-blind training.

Competitive Learning: This network uses an *unsupervised* method of training for the hidden clustering layer, i.e., without exploitation of an error signal that compares predicted with target output: The network presents one input pattern and computes each hidden unit's activation. The unit that shows the largest activation is the unit whose weight vector most closely approximates the input pattern. This is the only unit whose weights are updated. The weight vector is moved a small fraction of the distance toward the input pattern. The procedure as sketched in Figure 4 b) is repeated many times with randomly selected input patterns. This moves one weight vector into the center of each cluster of the input patterns. Since no comparison between input and desired output is carried out, there is no error signal available, which may be used to update the weights as is done in supervised learning. This clustering layer is then fixed, and the linear output layer is added. The output layer weights are then trained using backpropagation and the four-unit output target values. In this example, the hidden layer is trained to reflect the structure of the inputs. These assigned clusters, however, are independent of the target set, and the network is therefore task-blind.

Self-Organizing Map (Kohonen Network): The self-organizing map is the last architecture to use task-blind training. It is similar to the competitive network above, but the clustering layer is trained in a slightly different manner. The hidden `rbf` layer is arranged in a rectangular fashion as sketched in Figure 4 c). The size of this so-called Kohonen map was selected to be seven-by-seven, 49 units in total; seven represents the length of the input vector and seven rows chosen arbitrarily. In this arrangement, neurons that are close together are said to be in a neighborhood. During training – again unsupervised – the activations of the clustering layer are recorded for the currently investigated input pattern. The neuron with the highest activation is moved closer to the input pattern, just as in the case of competitive learning. But instead of only training that weight vector, all the weight vectors in its neighborhood are also moved closer to the input pattern. This type of training produces a topological map of the input space, with similar inputs mapping to neighboring neurons. A sample topological map is provided in Figure 5, which is produced after training has been completed and an arbitrary pattern vector is presented. Although the topological information has not been made use of, training multiple weight vectors for a given pattern results in fewer weight vectors being ignored and never trained. Again, after training the clustering layer, its weights are fixed, and a linear output neuron is added. This is trained using backpropagation, as before.

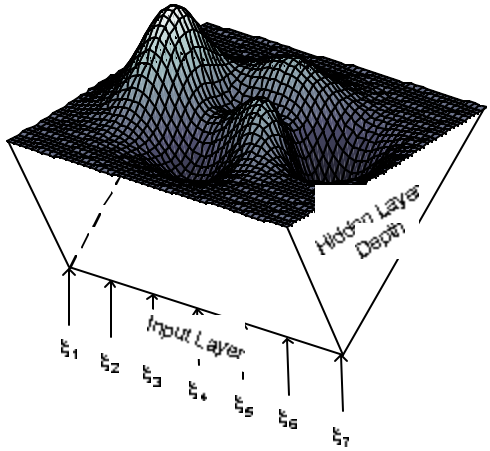


Figure 5: Sample activation of a self-organizing (Kohonen) map with 7 input variables and 20 rows in the hidden layer when presented with a pattern vector

Task-Specific Clustering

The next two architectures are very similar both in form and in training. They again have an input layer, an `rbf` clustering layer, and a single output neuron. In the second network, an additional second hidden layer is incorporated. This layer consists of sigmoidal neurons and increases the network's computational capacity. Both networks train all layers simultaneously using backpropagation.

One Hidden Gaussian Layer: This network consists of an input layer, a hidden `rbf` layer, and one linear output. Please refer to Figure 4 d). Because backpropagation is used to teach the entire network simultaneously, the clustering layer is trained to minimize the output error. This task-specific training produced a lower overall error, but training times are high due to the large amount of computations required.

Two Hidden Layers: The above network is slightly altered in this example by adding a second non-linear hidden layer between the clustering layer and the output as illustrated in Figure 4 e). The non-linear nature of the sigmoidal activation function allows more complex output functions to be approximated by the network. Errors were found to be reduced, but the additional hidden layer adds further computation to an already slow training process.

Training Techniques

While the neural network architecture is central to the performance of developing the desired mapping between the input and output space, training the networks effectively proved nontrivial. For this reason, this section describes the training techniques that were adopted in order to successfully train the investigated network architectures. Momentum and dynamic learning rate are standard tools and described only for completeness sake, while input normalization, random pattern weighting, and local minimum avoidance are less commonly employed approaches, which proved beneficial in this study.

Momentum is a common enhancement to backpropagation used to help avoid the problems associated with local minima of the error function. Momentum allows the network to pass through most local minima without getting stuck. It is simply used by making the direction of the next change in weights largely dependent on the previous direction of change, and only slightly dependent on gradient descent.

Dynamic Learning Rate: A trade-off encountered in the use of backpropagation is the learning rate size. Small learning rates are more stable but take a long time to train, and large learning rates, although fast to train, have difficulties in narrow basins of the error function. One solution is the use of a dynamic learning rate: If the network is doing well as judged by a monotonic decrease in training error, the learning rate will increase and the network will train more quickly. If, however, the learning rate is too large and producing an increase in error, the learning is decreased.

Input Normalization: Both of the unsupervised training examples (competitive and self-organizing maps) compute the distance between an input vector and a neuron's weight vector during training. To ensure similarity of the input and the weight vectors, they should first be normalized to unit length. This places them on the surface of a multi-dimensional hypersphere. However a problem can arise if some inputs are approximately scaled versions of other input patterns. After standard normalization, these patterns would be mapped to the same space on the unit hypersphere. A way to avoid this is to add an additional input variable that brings the un-normalized vector up to unit length. This extra variable adds an additional input dimension to the input space. Short input patterns are mapped to higher areas in this extra dimension, allowing smaller inputs to be separated from larger but similar inputs.

Bonus Learning Rate Multiplier: One problem not solved by a dynamic learning rate as described is a gradual descent of the error function. If the weights enter a flat region of the error surface with a small learning rate, it can take a long time to pass through the region. This occurs because the learning rate will not significantly increase due to the gentle slope. Thus, a new concept was introduced, the *bonus learning rate multiplier*. This factor can only increase the learning rate. It increases the learning rate by a factor proportional to the number of consecutive steps that produced a decrease in error. A single increase in error resets the bonus multiplier to one, resulting in no scaling for the next learning rate. This technique rewards slow but steady descent on the error surface with larger learning rates. This method was found to decrease backpropagation training times.

Random Pattern Weighting: With all these techniques pushing the network in various directions, it became necessary to prevent large, sudden increases in error. Thus, weight changes that resulted in an error increase of more than five percent were ignored. But how was the network to decide on a new direction to explore? One technique used altered the way gradient descent works while batching input patterns. Instead of weighting each pattern in the batch equally and producing the same gradient decent vector each time, a random weighting of input patterns was used. It was expected that this randomness would add significant noise to the system to allow it to escape local minima. This technique did prove beneficial to backpropagation training.

Discard – Local Minimum Avoidance: With all the above techniques, the network would still get trapped in a position where it would not allow a large increase in error. In some situations, the network could not get away from these points on the error surface. The following technique was explored: If no changes are allowed for more than four cycles in a row due to possible large increases in error, the net decided to allow the increase and accept the new weights and biases. Most often after a permitted jump in error, the network would quickly decrease the error on the next step, dropping below the error of the initial sticking point. This technique has been the most valuable technique used to improve network error reduction.

COMPARISON OF TRAINING RESULTS

The following discussion is based on the same set of data measured by the manufacturer, which has been described above. Of the 341 measured samples (or patterns) of seven input and four output variables, 20% were randomly withheld for validation (or testing) of network performance on previously unseen data. The results presented here are only valid for the particular data set used in this study and cannot serve to make generalized conclusions on the effectiveness of each neural network architecture. Every data set is the unique outcome of the physical phenomena involved as well as any noise present in the observation of the process. For this reason, a network choice that performs well on one task may perform poorly on another. While it is true that a network with an *infinite* number of nonlinear neurons (e.g., sigmoidal) in the hidden layer(s) can approximate any arbitrary nonlinear functional relationship, this theoretical insight has little bearing on the performance of neural networks with *finite* computational resources.

Table 2 summarizes the results of the experiments conducted in the context of this study. In the three investigated cases of task-blind clustering, there are two steps involved as illustrated in Figure 4 a) through c). When there are two entries in Table 2, the first entry refers to the first step and the second entry to the second training step. An 'epoch' has elapsed when all training patterns have been used in the training process; a 'cycle' is the presentation of only one training pattern in the training process. Again, see the glossary of terms at the end of the article. Table 2 reveals that task-blind clustering is faster but less accurate when compared to the task-specific counterpart.

Table 2: Comparison of the training results

Network Architecture	Training Epochs	Training CPU Time	Training Error	Validation Error
Auto-Associator	5,000/20,000 epochs	1236 sec	0.0766/ 0.3347	0.0645/ 0.2418
Competitive Learning	100,000 cycles/ 100,000 epochs	422 sec	0.2003	0.1538
Self-Organizing Map	100,000 cycles/ 100,000 epochs	515 sec	0.2115	0.1780
One Hidden RBF Layer	10,000 epochs	1663 sec	0.0904	0.1442
Two Hidden Layers	30,000 epochs	6598 sec	0.0906	0.0768

Task-Blind Clustering

Auto-Associator: During the training of the different networks, the advantages and disadvantages soon become apparent. The auto-associator network is one of the worst performers. The first task of training the clustering layer as the hidden layer of an auto-associator, can be accomplished accurately. The normalized error for this part of the network can be pushed down as far as 0.08 (an error value of 1.00 is the equivalent of always predicting the mean of the data set). However, the number of computations becomes very large for 50 hidden neurons, resulting in long training times.

These long training times occur for two reasons. In the first step, training using backpropagation through an `rbf` layer is a computationally expensive procedure, plus the number of outputs – and thus the number of second layer weights trained in this section – is large compared to the other architectures. The

second step is to discard the multiple outputs, and replace them with one linear output neuron. This new layer must be retrained to produce the correct target values when the input is fed into the network. This layer is also trained with backpropagation, but it is much quicker because the hidden layer is now frozen. The error performance of this final configuration is only mediocre, however. Training errors leveling off around 0.33 or higher occurred quite often. Its poor behavior can be traced to the task-blind clustering layer. When the activations of the 50 hidden units for the 64 validation input patterns (~20% of the total data set) are plotted as shown in Figure 6, two features can be noticed. First, several of the clustering units are fully active, i.e., $g_j(\xi) = 1$, for all input patterns. These units are of little benefit in predicting the target output. Also, some units are never activated, and the same argument holds. The rest of the units' behavior is difficult to classify.

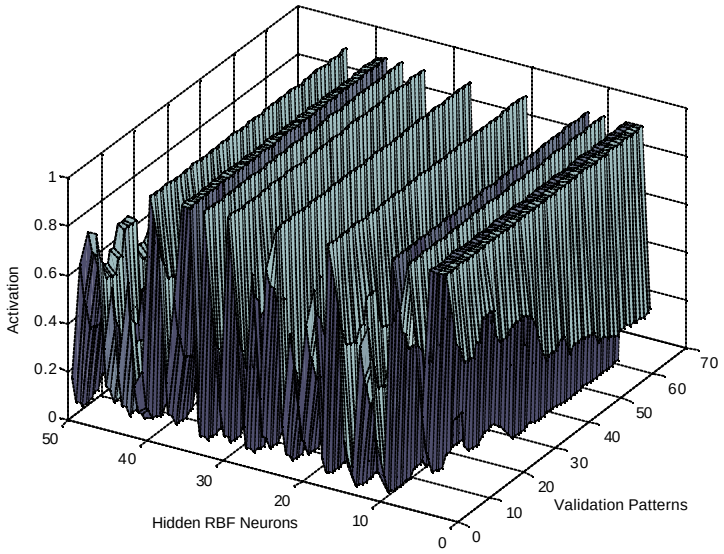


Figure 6: Auto-associator cluster activations

Competitive Learning: The next architecture is a hybrid of unsupervised and supervised learning as first presented by Moody and Darken (1989). It utilizes unsupervised learning to train its clustering layer, and supervised to teach the output layer. This competitive learning example trains very quickly, for it only presents one input pattern at a time instead of using batching. This network's training speed allowed the hidden layer of 50 units to be trained for 100,000 cycles or more within a few minutes. The plot of the activations for each of the input patterns in Figure 7 shows the network learned to cluster some of the early patterns quite well: The earlier validation patterns, i.e., those from the beginning of the measurement phase during which the chiller was operating at low to medium part loads, lead to wider spectrum of activations and more unique patterns. The absolute value of the activations is of no relevance. The later patterns that exhibit a high degree of similarity produce a more even distribution of activation values. Again a single output neuron is trained with backpropagation. The lowest error achieved is much improved, around 0.2, which is 40% less of that attained by the auto-associator.

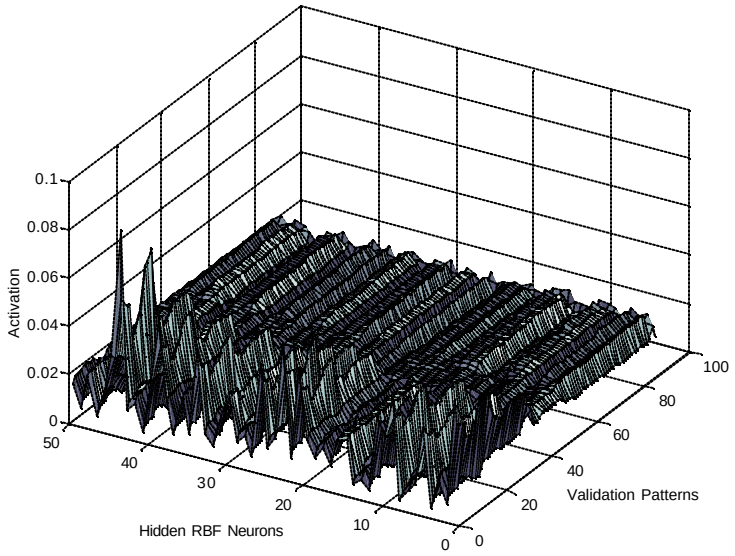


Figure 7: Competitive network cluster activations

Self-Organizing Map (Kohonen Network): The self-organizing map is trained almost identically to the competitive learning example above. The training is now completed for the winning neuron and the neurons in its neighborhood. This topological constraint produces an interesting activation plot as seen in Figure 8. This plot shows the network learned to classify early training patterns distinctly separate from the later ones. This effect is expected due to the similarities in the data set. But again the network was only unable to produce a low output error, approaching 0.2 as in the competitive learning case with similarly short training times.

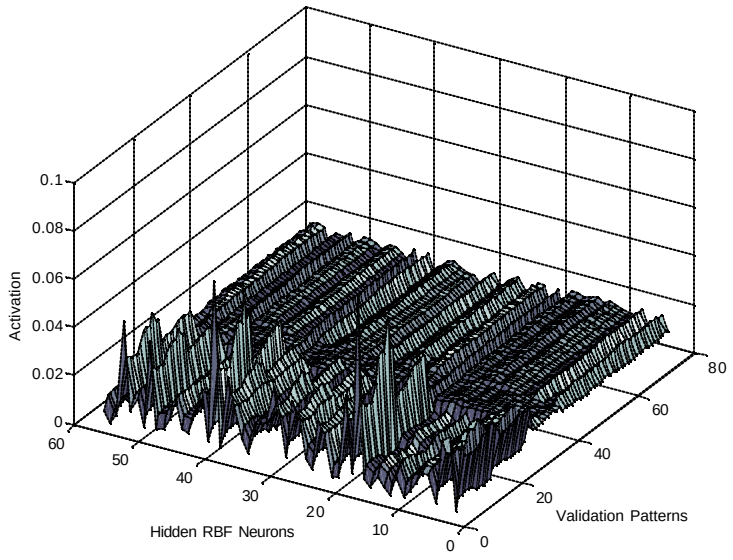


Figure 8: Self-organizing map cluster activations

Task-Specific Clustering

The first three models were examples of task-blind training of the hidden layer. In the next two models, the effectiveness of task-specific training will be examined.

One Hidden Gaussian Layer: The first architecture using task-specific training is simply an `rbf` hidden layer and a linear output neuron. The entire network is trained simultaneously with backpropagation. The activation of the clustering layer as shown in Figure 9 appears to be similar to that of the auto-associator. After 10,000 training epochs an error around 0.09 can be achieved, while the error measured on the validation set is about 0.14. Training is slow for 50 hidden units, even on an 850 MHz Pentium III based computer. This is due to backpropagation through the `rbf` layer. Obviously, task-specific training yields training errors substantially (<50%) below the best task-blind architecture.

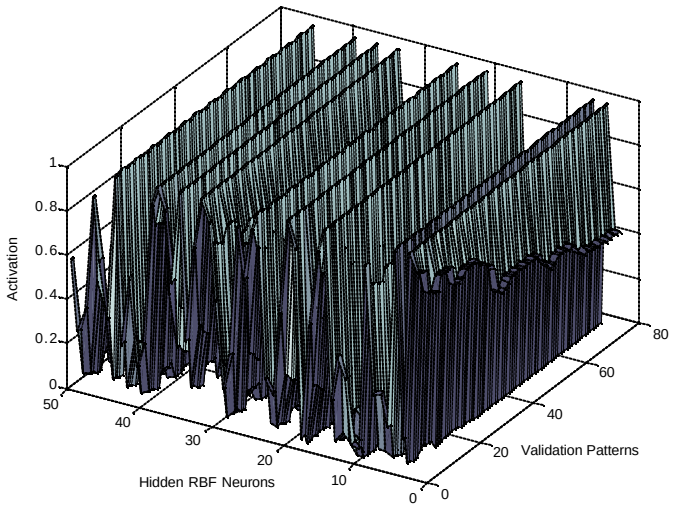


Figure 9: One hidden RBF cluster activations

Two Hidden Layers: This architecture did not perform significantly better than the previous task-specific case. It consists of the hidden `rbf` layer, a non-linear sigmoid layer, and the output. One would expect the sigmoid layer to add processing power to the network, but better results were not achieved, i.e., again an error of 0.09. The validation error was decreased by a factor of two compared to the architecture with one hidden layer. Training speed becomes even slower due to backpropagation through three layers. Figure 10 reveals that after 30,000 epochs, the three-layer architecture uses most of the neurons of the first hidden `rbf` layer effectively, only a few neurons are fully active for the entire set of validation patterns.

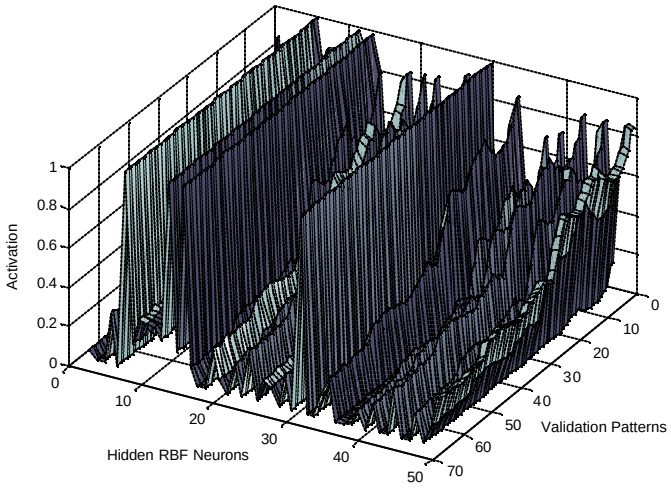
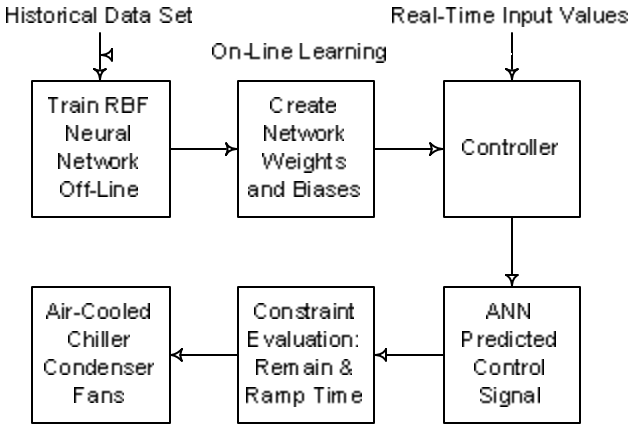


Figure 10: Two hidden RBF cluster activations

DESCRIPTION OF THE CONTROLLER IMPLEMENTATION PROCEDURE

Figure 11 illustrates the controller implementation procedure; the dashed line shows the on-line learning, the implementation of which is reserved for future study. The feedforward clustering neural networks with one hidden radial basis function layer and a single linear output described above are applied to identify the system behavior of the air-cooled condenser control problem, while process control logic is responsible for the temporal constraint evaluation. Because the measured data set is available and the controller is to be trained off-line, lower prediction errors were desired accepting the corresponding longer training time as an expected disadvantage. The network is trained separately on the selected training data subset. When training is completed, the weight matrices and bias vectors to the hidden and output layers are saved to a separate file, therefore making the network available for later use.



contains some high-frequency noise, which has to be attenuated using temporal constraints described below.

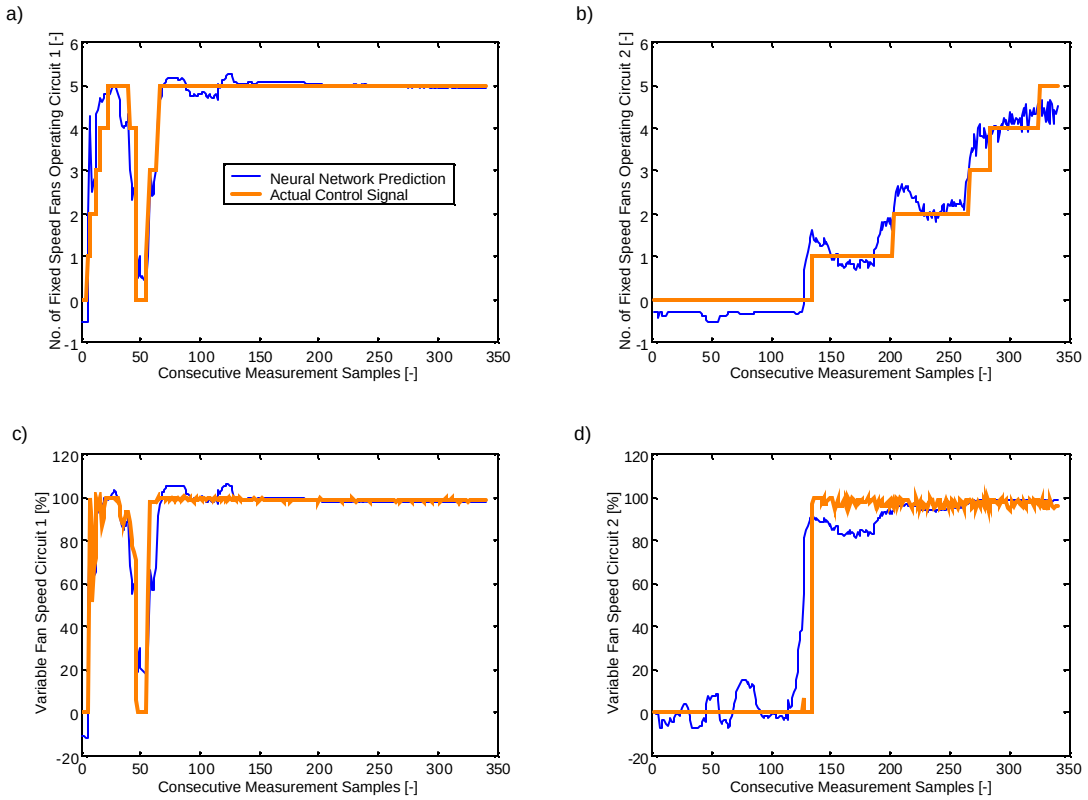
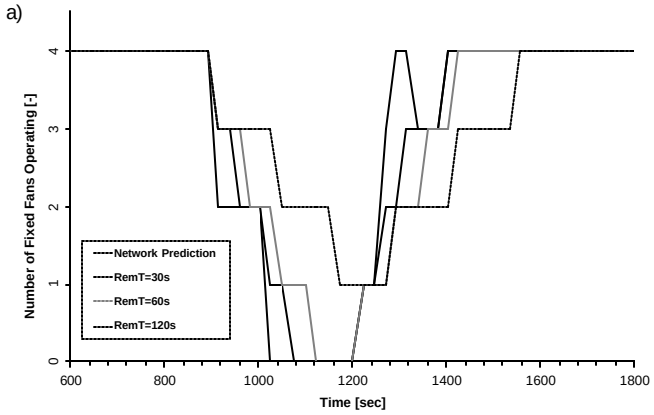


Figure 12: Target vs. recommended control for one hidden RBF network

One way of enforcing temporal constraints is through the use of minimal remain time and minimal ramp time. Figure 13 a) illustrates how the actual control signal tracks the network recommendation as a function of minimal remain time. It shows the effect of minimal remain time compared to a network without remain time constraints. As expected, the longer the minimal remain time, the more the actual signal lags behind the network prediction and the lower the "willingness" to follow high-frequency spikes in fan cycling. The opposite holds true for shorter minimal remain times. In a similar fashion, the minimal ramp time, as seen in Figure 13 b) defines how closely the variable speed fan follows the network prediction. The longer the minimal ramp time, the smaller the allowed changes of the inverter control signal.



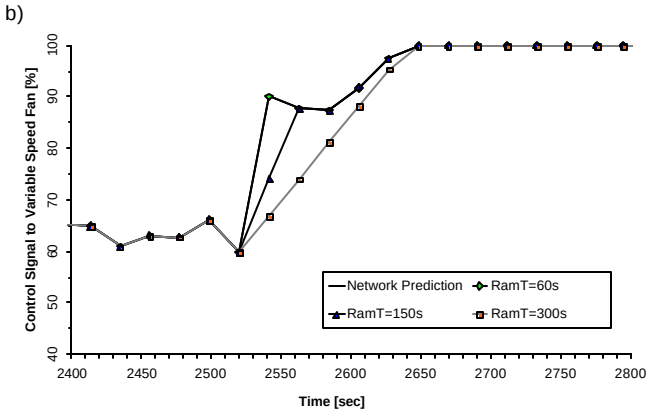


Figure 13: Effect of a) minimal remain time on cycling of fixed-speed fans and b) minimal ramp time on stability of variable-speed fan

CONCLUSIONS

Clustering Neural Network Architecture

It became evident that task-blind training produces larger errors than task-specific training. One advantage of task-blind training is that it can be trained unsupervised, arriving at relatively high training speeds. If the maximum error constraint can be loosened, the competitive learning or the self-organizing map examples can produce adequate results in very short time periods. If training accuracy is of dominating importance, task-specific training is required, but one has to be prepared to encounter extensive training times.

One possible direction of future study would be to initially use task-blind unsupervised training on the clustering layer as in the competitive learning example, but instead of freezing the hidden layer weights, allow the backpropagation algorithm to also train and optimize the clustering units in a task-specific sense. This may potentially reduce the number of epochs required to reach the error goal, thus reducing training time. Other areas to be investigated are the effect of adding past outputs as inputs to the network. This will allow the *autoregressive* network to recognize trends and more accurately predict the next output. This method is reminiscent of time series prediction, but the network would also utilize information on the current state of the system. As mentioned in the section describing the data set, additional inputs might also prove beneficial to training. The available data set consists of approximately 150 variables, of which seven were selected as the most relevant. If more of the available inputs were used, improved performance might result.

Control

With the objective of a neural network controller that is trained on-line to incorporate changes in its environment allowing for adaptive control, an architecture that allows for short training times is desirable. This allows for adequate adaptation to changes while reducing the impact on the current control performance. For this purpose, the competitive or self-organizing map architectures are recommended. However, to allow a newly installed controller to perform adequately, the feedforward architectures with their low errors should be preferred since training time is not a crucial issue. The design of the off-line controller was chosen accordingly and satisfactory preliminary results were achieved. The next step will be to refine the controller design and apply it to the control of an actual air-cooled condenser and compare its performance with the one of the conventional controller. Subsequent work can then determine the usefulness of the neurocontroller in reducing development time when applied to different chillers for which the condenser control has not yet been developed. The true benefit of the work presented, however, will be realized when the neural network is initially trained by manufacturer's measured data, while subsequent on-line training will allow for continuous improvement of energy consumption and/or other desired features through predictive optimal control or reinforcement learning control (Henze and Dodier, 2002).

REFERENCES

Anderson, J.A. and E. Rosenfeld, eds. 1988. *Neurocomputing: Foundations of Research*. Cambridge, MA: MIT Press.

ASHRAE. 2000. *Handbook of Systems and Equipment*. p. 35.12. American Society of Heating, Refrigerating, and Air-Conditioning Engineers, Atlanta, Georgia.

Breekweg, M.R.B., P. Gruber, and O. Ahmed. 2000. Development of a Generalized Neural Network Model to Detect Faults in Building Energy Performance – Part I. ASHRAE Transactions, Vol. 106, Part 2, American Society of Heating, Refrigerating, and Air-Conditioning Engineers, Atlanta, Georgia.

Dodier, R.H. and G.P. Henze. 1996. Statistical Analysis of Neural Networks as Applied to Building Energy Prediction. Proceedings of the ASME International Solar Energy Conference, New York, NY: American Society of Mechanical Engineers.

Henze, G.P. and R.H. Dodier. 2002. "Adaptive Optimal Control of a Grid-Independent Photovoltaic System." Proceedings of SOLAR 2002, American Society of Mechanical Engineers, New York, New York.

Hertz, J., Krogh, A., Palmer, R.G. 1991. Introduction to the Theory of Neural Computation. Redwood City, CA: Addison-Wesley.

Jeannette, E., K. Assawamartbunlue, and P.S. Curtiss. 1998. Experimental results of a predictive neural network HVAC controller. ASHRAE Transactions, Vol. 104, Part 2, American Society of Heating, Refrigerating, and Air-Conditioning Engineers, Atlanta, Georgia.

Moody, J., Darken, C.J. 1989. Fast Learning in Networks of Locally-Tuned Processing Units. Neural Computation, 1, 281–294. Cambridge, MA: Massachusetts Institute of Technology.

Rumelhart, D.E. and McClelland, J.L. 1988. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations. Cambridge, MA: MIT Press.

Weigend, A.S. and Gershenfeld, N.A., eds. 1994. Time Series Prediction: Forecasting the Future and Understanding the Past. Reading, MA: Addison-Wesley.

GLOSSARY

architecture - A description of the number of layers in a neural network, each layer's transfer function, the number of neurons per layer, and the connections between layers.

backpropagation learning rule - A learning rule in which weights and biases are adjusted by error-derivative (delta) vectors backpropagated through the network. Backpropagation is commonly applied to feedforward multilayer networks. Sometimes this rule is called the generalized delta rule.

bias - A neuron parameter that is summed with the neuron's weighted inputs and passed through the neuron's transfer function to generate the neuron's output.

bias vector - A column vector of bias values for a layer of neurons.

competitive learning - The unsupervised training of a competitive layer with the instar rule or Kohonen rule. Individual neurons learn to become feature detectors. After training, the layer categorizes input vectors among its neurons.

cycle - A single presentation of an input vector, calculation of output, and new weights and biases.

distance function - A particular way of calculating distance, such as the Euclidean distance between two vectors.

dynamic learning rate - A learning rate that is adjusted according to an algorithm during training to minimize training time.

epoch - The presentation of the set of training (input and/or target) vectors to a network and the calculation of new weights and biases. Note that training vectors can be presented one at a time or all together in a batch.

error vector - The difference between a network's output vector in response to an input vector and an associated target output vector.

feedforward network - A layered network in which each layer only receives inputs from previous layers.

function approximation - The task performed by a network trained to respond to inputs with an approximation of a desired function.

global minimum - The lowest value of a function over the entire range of its input parameters. Gradient descent methods adjust weights and biases in order to find the global minimum of error for a network.

gradient descent - The process of making changes to weights and biases, where the changes are proportional to the derivatives of network error with respect to those weights and biases. This is done to minimize network error.

hidden layer - A layer of a network that is not connected to the network output. (For instance, the first layer of a two-layer feedforward network.)

input layer - A layer of neurons receiving inputs directly from outside the network.

input vector - A vector presented to the network.

Kohonen learning rule - A learning rule that trains selected neuron's weight vectors to take on the values of the current input vector.

layer - A group of neurons having connections to the same inputs and sending outputs to the same destinations.

learning - The process by which weights and biases are adjusted to achieve some desired network behavior.

learning rate - A training parameter that controls the size of weight and bias changes during learning.

learning rules - Methods of deriving the next changes that might be made in a network OR a procedure for modifying the weights and biases of a network.

local minimum - The minimum of a function over a limited range of input values. A local minimum may not be the global minimum.

mean square error function - The performance function that calculates the average squared error between the network outputs a and the target outputs t.

momentum - A technique often used to make it less likely for a backpropagation networks to get caught in a shallow minima.

neighborhood - A group of neurons within a specified distance of a particular neuron.

neuron - The basic processing element of a neural network. Includes weights and bias, a summing junction and an output transfer function. Artificial neurons, such as those simulated and trained with this toolbox, are abstractions of biological neurons.

output layer - A layer whose output is passed to the world outside the network.

output vector - The output of a neural network. Each element of the output vector is the output of a neuron.

output weight vector - The column vector of weights coming from a neuron or input. (See outstar learning rule.)

pattern - A vector.

radial basis networks - A neural network that can be designed directly by fitting special response elements where they will do the most good.

supervised learning - A learning process in which changes in a network's weights and biases are due to the intervention of any external teacher. The teacher typically provides output targets.

test vectors - A set of input vectors (not used directly in training) that is used to test the trained network.

training - A procedure whereby a network is adjusted to do a particular job. Commonly viewed as an "offline" job, as opposed to an adjustment made during each time interval as is done in adaptive training.

training vector - An input and/or target vector used to train a network.

unsupervised learning - A learning process in which changes in a network's weights and biases are not due to the intervention of any external teacher. Commonly changes are a function of the current network input vectors, output vectors, and previous weights and biases.

validation vectors - A set of input vectors (not used directly in training) that is used to monitor training progress so as to keep the network from overfitting.

weighted input vector - The result of applying a weight to a layer's input, whether it is a network input or the output of another layer.

weight function - Weight functions apply weights to an input to get weighted inputs as specified by a particular function.

weight matrix - A matrix containing connection strengths from a layer's inputs to its neurons.