NEURAL NETWORK BASICS FOR USE IN BUILDING MECHANICAL SYSTEMS

Darrell D. Massie, Ph.D., P.E.
Department of Civil and Mechanical Engineering
United States Military Academy, West Point, New York 10996 USA
845.938.4037 (phone), 845.938.5522 (fax), id4747@exmail.usma.edu

Peter S. Curtiss, Ph.D.
Peter Curtiss Consulting Engineers
Boulder, Colorado 80303 USA
303.440.6949 (phone), 845.938.6954 (fax), peterc@kassoc.com

ABSTRACT

Neural networks are massively parallel processors that have the ability to learn patterns through a training experience. Because of this feature, they are often well suited for modeling complex and non-linear processes such as those commonly found in the heating, ventilation and air conditioning (HVAC) industry. Since neural networks are a relatively new technology in HVAC, most of today's practicing engineers do not have a clear understanding of neural network basics and do not appreciate how neural networks can be integrated into today's building systems. This paper will lead the interested engineer through neural network terminology basics. It will also answer the important question of how neural networks are constructed and why they have the capacity to learn patterns. Additional discussions will cover how different types of common neural network models are constructed and common pitfalls associated with using neural network models.

Background

Traditionally, modeling of equipment and controller algorithms consists of computer programs that rely on complicated mathematics tailored for a specific application, and are generally not portable to other systems. Additionally, if an unusual perturbation or disturbance occurs, those models have difficulty making accurate predictions. An alternative to traditional models is through the use of neural networks (NN), from a branch of artificial intelligence. Neural networks are nonlinear computer algorithms that learn with feedback, and can model the behavior of complicated nonlinear processes.

The field of neural networks is vast and interdisciplinary. Therefore, this paper will focus on the two methods of most value to scientist and engineers – 1) cascaded feed forward networks, used to model equipment operating characteristics and 2) recurrent (sequential) networks that are useful for control modeling. The reader is directed to Rumelhart and McClelland (1), Cowan and Sharp (2), Wasserman (3) and Bishop (4) to learn more about the wide variety of NN architectures, structures and learning techniques.

Types of Learning

There are essentially two types of NN learning models – *supervised learning* and *unsupervised learning*. With supervised learning, for each input pattern the value of the desired output is specified and the goal of the network is to minimize an error function. To train a network, an input vector is applied to the network and the output of the network is calculated and compared to the corresponding target vector with the difference (error) being fed back through the network to change the weights so that the error is minimized.

Neural networks that do not rely on the use of target data are trained using unsupervised learning. Instead of trying to map the data input-output relationship, the goal is to find an underlying structure of the data, called clusters. Unsupervised learning has the advantages that:

- It does not need the collection and labeling of output values,
- If the characteristics or patterns of the data change over time, the network has the ability to change with the patterns, and
- In the early stages of an investigation, this method may provide valuable insight to a problem that can lead to a better end product.

While unsupervised learning has advantages and theoretically shows potential for solving difficult problems, is has remained largely unsuccessful in solving many engineering problems. As a result, it does not yet have widespread engineering applicability, so will not be further discussed in this paper.

Classes of Networks

There are many types of neural networks, each having specific functions and capabilities. Learning networks falls into two distinct classes – *regularity detection* and *associative learning*. Regularity detection networks learn to identify "interesting" trends within the data set. In this case, a teaching input is not explicitly provided, but is identified by the unit itself. This type of network falls into the category of unsupervised learning.

The associative learning case can further be broken down into two more subclasses – *auto-association* and *pattern association*. An auto-association network model is one where an input pattern is associated with itself. The goal of this type of network is pattern completion. Should only a portion of an input pattern be available, the network can fill in the remaining pattern. Pattern association makes a connection between an input set of data to an expected output value. The goal is

to find a set of connections so that when a particular set of values appear as input, the associated pattern will appear as the output value. These networks are usually trained by back-propagating errors based on the output training value. Since the vast majority of engineering problems that can benefit from neural network modeling fall into this category, the remainder of this paper will focus on pattern association learning.

Neural Network Configurations

Within pattern association networks there are many types of configurations, each with a specific architecture useful for solving different types of problems. In all cases, however, the goal is to map input to output while minimizing an error function.

The general arrangement for networks is to arrange them into layers (Figure 1 shows a generic layered architecture). Within layers there are nodes (depicted as circles) that are connected by weighting factors, called weights. There is no limit to the number of layers or nodes that are in a layer, although as the total number of nodes increases (and therefore weights) so do computational requirements. There are three types of layers, input, output and hidden. The input layer merely acts as a receptacle for input and is often referred to as layer zero. Nodes in each of the other layers collect information from the weighted upstream node outputs (usually by summation) and process the information with an *activation function*. A value computed from the activation function is then available to be sent to the next layer using more weights. This process will be discussed in greater depth later in the paper.

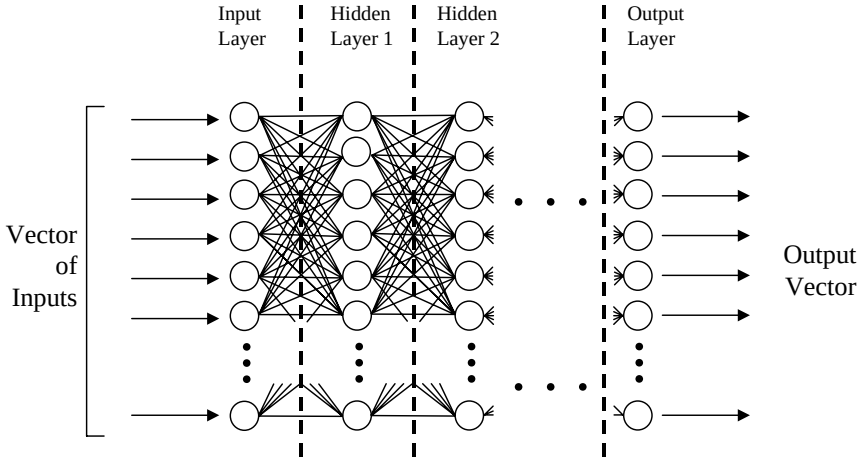

Figure 1. Typical architecture of layered neural network.

The *cascaded, feed-forward* network has a structure like that shown in Figure 1. An input vector is applied to layer zero and the goal of the network is to map the relationship between the input vector and the output vector (also referred to as target values). The number of elements in the input and output vector need not be the same. If the activation function is non-linear, then the mapping will also be non-linear. This network is particularly useful for modeling the non-linear processes that are often found in applications. As the architecture becomes more complex, then the greater the ability of the network to model non-linear relationships. However, there often becomes a point where increased network size does not result in increased performance and can actually decrease model accuracy.

The *recurrent* (*sequential)* network has a structure similar to Figure 2 and was proposed by Jordan (5). This network is particularly useful when modeling a continuous system in which the recent past history of a process is valuable in determining the process value at the next time step. In essence this type of network has a memory of current actions. The model consists of a sequence of actions, which are to be produced in the order of a "plan." Additional inputs come from the current "state" nodes that together with the plan nodes force the next state into a desirable state of action.

This type of network would be useful, for example, when modeling a car's speed as a function of the accelerator's position. The recent past history of the car's speed is important in determining the speed of the next time step. An example of the input values provided by the plan might include wind speed and direction or the slope of the road in which the car is traveling.

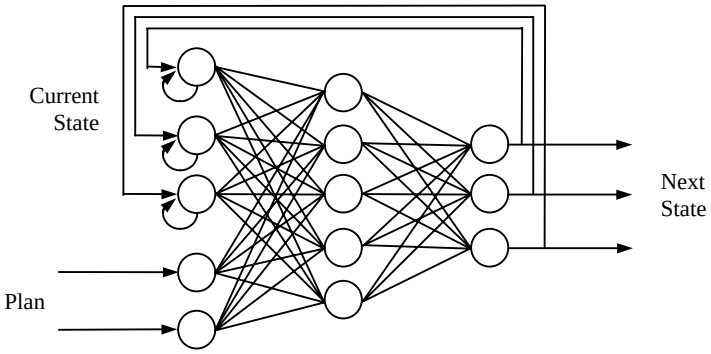

Figure 2. Jordan's sequential network configuration.

Activation Functions in Neural Networks

Each node in a network receives a set of inputs from either outside the network or from a previous layer. Figure 3 shows the basic structure of a receiving node accepting input from sending nodes.

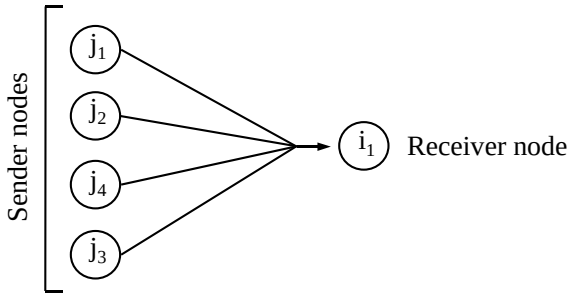

Figure 3. Sender and receiver nodes

Each input signal is multiplied by a weight and the product summed. The summation of products is termed *NET* and can be calculated for each node in the network using

$$NET_i = \sum_{j=f}^{l} A_j w_{ji} \qquad (1)$$

where

NET_i = net input to node i from the first sender f to the last sender l

A_j = output of node j and

w_{ji} = weight of connection between node j and i.

After *NET* is calculated, an activation function A_i, called the *activation*, is applied to modify it thereby producing the output signal. There are many functions that might be used as long as they are everywhere differentiable. If the activation function is a linear relationship, then the activation is equal to *NET*. However, for mapping nonlinear processes, a nonlinear function is required. A commonly used function, called a *sigmoid* function (sometimes called a *logistic* or *squashing function*) is used because it is self-limiting and has a simple derivative (hyperbolic tangent is also frequently use). The sigmoid function is given by

$$A_i = F(NET_i) = \frac{1}{1 + e^{-NET_i}} \qquad (2)$$

A graph of the sigmoid function is provided in Figure 4. An advantage of the sigmoid function is that it provides a form of automatic gain control (it is self-limiting) and output cannot grow infinitely large or small. For small values (*NET* near zero) the slope of the input/output curve is steep, producing high gain. As the magnitude of *NET* becomes larger, the gain decreases and small signals are allowed to pass through without excessive attenuation.

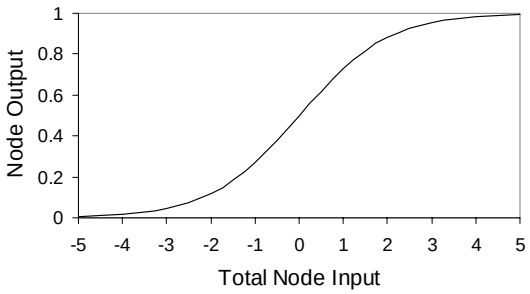

Figure 4. Sigmoid function.

Bias Nodes

It is often desirable to have a bias weight for each node. These weights are trainable just as other weights, however, their input is always set to one and they are not connected to other nodes. Their purpose is to offset the origin of the activation function, which can lead to more rapid convergence. Bias weights allow a node to have output even if the input is zero.

Training Neural Networks

The objective of training a neural network is to adjust the weights so that application of a set of inputs produces the desired set of outputs. Training assumes that each input vector is associated with an output vector. One input vector together with one output vector constitutes a training pair. Training a network involves the following four steps, Wasserman (1989):

1. Select a training pair from the training set and apply the input vector to the network input.
2. Calculate network output using a forward pass. Calculation of network output is accomplished by using a feed-forward process and application of an activation function for each layer (except layer zero) in the network.
3. Compute the difference between network output and the desired target value from the training pair output value.
4. Change the network weights in a way that minimizes the error.

A process known as *back-propagation* accomplishes the fourth step of the training process. Back-propagation is the name given to the computation of the error gradient by moving backwards from the output to the input. Given that there is a method to go from inputs to outputs and it is desired to compute the gradient of the output function with respect to an input, then the chain rule can be used to compute that derivative, provided that in between there are differentiable functions. This is indeed the case for neural networks where the nodes have differentiable functions.

One way to implement back-propagation is called the *delta rule* method. It is possibly the most common method used throughout neural networks and can be used whenever differentiable

functions are available. If the activation functions are not everywhere differentiable then the chain rule cannot be used and a finite different approach must be taken.

Delta Rule Method

The net input into a node is calculated using Equation (1) and the activation of a node can be calculated using many functions, the sigmoid function shown in Equation (2) being the most common. A more detailed discussion of this process can be found in Curtiss (1992) and is summarized here. Traditional training of neural networks seeks to minimize the error function

$$E = \sum_{p=1}^{N_p} \sum_{i=1}^{N_o} \left(t_{p,i} - A_{p,i} \right)^2 \tag{3}$$

where N_p is the number of input/output data pairs, N_o is the number of outputs of the network, $t_{p,i}$ is the desired output value for a set of inputs and $A_{p,i}$ is the activation function for a given set of input.

The partial derivative of the error function with respect to each weight is

$$\frac{\partial E}{\partial w_{j,i}} = 2 \sum_{p=1}^{N_p} \sum_{i=1}^{N_o} \left(t_{p,i} - A_{p,i} \right) \left(\frac{\partial t_{p,i}}{\partial w} - \frac{\partial A_{p,i}}{\partial w} \right) \tag{4}$$

If a positive change in a weight causes the error of the receiving node to increase, then a decrease in the weight will decrease the error. Weights are therefore adjusted proportionally to the negative of the error with respect to each weight

$$\Delta w_{j \to i} = -\varepsilon \frac{\partial E}{\partial w_{j \to i}} \tag{5}$$

where j is the sender node and i is the receiver node. ε is the *learning rate* of the network, a constant that controls the rate in which weights can be altered. Equation (5) can be rewritten as

$$\Delta w_{j,i} = \varepsilon \, \delta_i \, A_j \tag{6}$$

For output nodes located in the output layer, delta is given by

$$\delta_i = (t_i - A_i) f'(NET_i) \tag{7}$$

where $f'(NET_i)$ is the derivative of the activation function. For nodes located in the hidden layer

$$\delta_i = f'(NET_i) \sum_{j=f}^{l} \delta_j w_{j,i} \tag{8}$$

where f is the first and l is the last node sending data to

Finite Difference

When a network does not contain differentiable functions, the chain rule (as used in the delta rule back-propagation method) will not work. However, the change in the error with respect to each input parameter must still be computed to adjust weights. An alternative to using the chain rule is to use a finite difference approach.

Consider the network in Figure 5 where an input vector stimulates a non-differentiable activation function to produce an output. The activation function can be any function and need not be known a priori. An input layer, with linear activation, acts as a receptacle for the input vector and a set of weights connects the input layer with the activation function. The goal of the network is to modify the weights between the input layer and activation function so as to minimize the error.

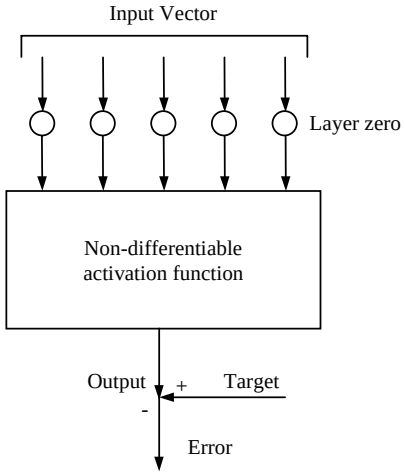

Figure 5. Network with a non-differentiable activation function.

The change in error with respect to each weight can be found by running the forward pass twice, once to determine the error and the second time to find the derivative with respect to each input parameter. Computationally this is accomplished by making a small change to the weight from a single input while all other weights remain constant. The associated change in error can be recorded and the process repeated with the next input.

Computing the derivative in this brute force pragmatic method certainly lacks elegance, however, it works. This approach is particularly useful for control problems where the weights can be made synonymous to the control settings by using the concept of virtual nodes.

Local Minima, Learning Rates and Weight Initialization

Back-propagation employs a type of gradient descent and therefore follows the slope of the error surface downward, continually adjusting the weights towards a minimum. The error surface of a complex network is full of local minima and maxima. Unfortunately if a network gets caught in a local minimum it is unable to discriminate between that local low point and the global low point. Furthermore, unless the weights are re-initialized and training initiated again, there is little chance of a network finding another solution.

The learning rate is related to the step size that weights can be changed in a training pass. Ideally the rate at which weights can be modified is infinitesimally small. This is clearly impractical as it implies a requirement for infinite training time. Likewise, if the learning rate is set at too large a value, then the network can become unstable and never settle to a solution. A technique that often improves training is to vary the learning rate throughout the training process by starting with a large initial learning rate and then decreasing it over time. This technique is called annealing.

All network weights must be set to initial values before training starts. If all weights between layers are set to the same value, then the delta rule learning algorithm dictates that those weights will remain equal regardless of the back-propagation error. If weights are too large then the network can become unstable and nodes saturated, and if weights are too small then weight changes will be painfully slow. The choice of initial weights is dependent upon the problem and normalization of variables. Values set to between negative one and plus one generally work well and should be chosen randomly.

Common Uses

Neural Networks are most extensively used for classification, modeling non-linear equipment characteristics and for local and global control of plants and processes. These uses will be discussed in an accompanying paper by Bailey and Curtiss [2001].

Conclusions

Neural networks are computer algorithms that have the ability to learn patterns by experience. Because of this feature, they are often well suited for modeling complex and non-linear processes such as those commonly found in the heating, ventilation and air conditioning (HVAC) industry. They are well founded in mathematics, however, a given weight does not carry a meaning such as would be found in a coefficient for many equations. Neural networks can have a variety of architectures that can be customized to solve a particular problem. Because of these features, they are a promising technology for the HVAC industry.

REFERENCES

REFERENCES

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

(2) Cowan, J.D.; Sharp, D.H. 1988. Neural nets and artificial intelligence. *Proceedings of the American Academy of Arts and Sciences*, Vol. 117, no. 1 pp.85-121.

(3) Wasserman, P.D. 1989. Neural Computing: Theory and Practice. New York: Van Nostrand Reinhold.

(4) Bishop, C.M. 1995. Neural Networks for Pattern Recognition. Oxford: Clarendon Press.

(5) Jordan, M.I. 1986. Attractor Dynamics and Parrallelism in a Connectionist Sequential Machine. *Proceedings of the 8th Annual Meeting of the Cognition Science Society*. Hillsdale, NJ. Erlbaum.

(6) Bailey, M.B., Curtiss, P.S. Neural Network Modeling and Control Applications in Building Mechanical Systems. CIBSE 2001.