Neural Comput & Applic (1997)6:42–56
© 1997 Springer-Verlag London Limited

Neural Computing & Applications

Short Term Electric Load Forecasting Using a Neural Network with Fuzzy Hidden Neurons

S. Kuusisto[1], M. Lehtokangas[1], J. Saarinen[1] and K. Kaski[2]

[1]Tampere University of Technology, Signal Processing Laboratory, Tampere, Finland; [2]Helsinki University of Technology, Laboratory of Computational Engineering, Finland

Short term electric load forecasting with a neural network based on fuzzy rules is presented. In this network, fuzzy membership functions are represented using combinations of two sigmoid functions. A new scheme for augmenting the rule base is proposed. The network employs outdoor temperature forecast as one of the input quantities. The influence of imprecision in this quantity is investigated. The model is shown to be capable of also making reasonable forecasts in exceptional weekdays. Forecasting simulations were made with three different time series of electric load. In addition, the neuro-fuzzy method was tested at two electricity works, where it was used to produce forecasts with 1–24 hour lead times. The results of these one month real world tests are represented. Comparative forecasts were also made with the conventional Holt–Winters exponential smoothing method. The main result of the study is that the neuro-fuzzy method requires stationarity from the time series with respect to training data in order to give clearly better forecasts than the Holt–Winters method.

Keywords: Forecasting; Membership function; Neuro-fuzzy system; Neural network; Rule-base fine-tuning, Rule-based system

1. Introduction

A well known fact is that electric energy cannot be stored efficiently using current technology. Therefore, knowledge of how the electric load will behave in the future is precious information for electricity works and power plants. In power supply management in particular, load forecasts are used to ensure economical and reliable operation. These forecasts are made for time-spans of different lengths, which vary from a few minutes to over 20 years. Very short term forecasts for a few minutes ahead are used in the minute-to-minute allocation of loads for the benefit of the generating units. Short term forecasts from one hour to a day or so are used in managing the unit commitment, which means planning and controlling the start-up and shut-down scheduling of the generating units. Unit committing is carried out to ensure that there is a sufficient amount of generating capacity to meet the varying load. These forecasts are also used in planning the buying and selling of energy in interconnected systems. Medium and long term forecasts are made from several days to several years ahead. These forecasts are used in system planning, which concerns building new generators and transmission lines. In this paper, we concentrate on hourly Short Term Load Forecasting (STLF) within the window from 1 to 24 hours.

Short term load forecasting is a subject that has been studied extensively, and several different methods have been developed during the last few decades. These different approaches have been discussed and compared with each other [1,2]. Conventional statistical forecasting methods can be divided into time series and regression methods. An excellent discussion of conventional STLF models is [3]. In the time series models, the prediction is based on the past values and prediction errors of a variable [4–7]. In methods based on simple or multiple regression, the relationships between some inde-

Correspondence and offprint requests to: Mr S. Kuusisto, Tampere University of Technology, Signal Processing Laboratory, P.O. Box 553, FIN-33101 Tampere, Finland. Email: seppoq @cs.tut.fi.

pendent variables and the dependent variable are modelled [8,9]. These relationships are estimated from the past values of these variables, and they are used to forecast the dependent variable by using the future evaluations of the independent ones. However, load demand has multiple determining factors which have complex inter-relations. This is a significant drawback for regression models, since they assume linear relationships between the explanatory variables and the dependent variable, but in practise these relationships are more or less nonlinear. As an alternative solution to conventional forecasting methods, expert systems have also been successfully applied [10–13]. The advantage of expert systems is that one does not have to make any assumptions about the linearity or nonlinearity of relationships prevailing in the system of interest. Instead, the rules in expert systems are collected from human experts who themselves are capable of making intelligent forecasts. This requires that the expert is able to represent his or her knowledge so that it can be programmed into a computer. However, programming and rule production of an expert system can take a great deal of time and co-ordination.

Another interesting information processing paradigm is offered by artificial neural networks. These self-learning algorithms were applied to electric load forecasting for the first time by Dillon *et al.* [14]. Recently, numerous applications in load forecasting that apply neural networks have been proposed [15–36]. Neural networks can be used in modelling the complex nonlinear relationships between the electric load and its determining factors like the load history and weather variables. Prior knowledge or assumptions about the characteristics of the relationships are not needed when neural networks are applied. This is because they can learn and capture relationships between input data and observed outputs by themselves from a set of examples. This set is called 'training data', which consists of input–output pairs that are typical for the system of interest. During training, the input–output pairs are represented to the network and its parameters are adjusted iteratively so that the total sum of errors (i.e. the differences between the network output and the desired output) for the training data will be minimised. When the error is small enough, the parameters are fixed and the training is finished. Generalising the dependencies in the set of examples, the network can also produce correct outputs for input patterns that are not included in the examples. As the concepts of fuzzy sets and inference are incorporated into artificial neural networks, the result is a fuzzy neural network or neuro-fuzzy system. These systems form an input–output mapping that is based on fuzzy rules which can be expressed with linguistic and imprecise statements. The system produces its output according to an inference mechanism, which is based on the rules extracted from the set of examples. In the last few years, neuro-fuzzy systems have been applied, for example, to nonlinear time-series prediction and electric load forecasting [37–44].

In Sect. 2 we give a detailed description of a conventional time-series forecasting method, Holt–Winters exponential smoothing. In Sect. 3 we describe a modern neuro-fuzzy method. The neuro-fuzzy model is a learning system that generates the rules automatically using example data as the initialisation data. Moreover, the approximation of triangular membership functions with sigmoid-triangular membership functions is described. In Sect. 4, we propose an algorithm which determines whether or not the rule base should be augmented. The results of the forecasting simulations and real world test-runs are represented in Sect. 5, and they are discussed in Sect. 6. Finally, in Sect. 7 we give concluding remarks and suggestions for further research. The OLS-algorithm and formulas for optimising the parameters of the rule-base are presented in Appendices A and B.

2. Holt–Winters Method

The Holt–Winters method belongs to a class of autoprojective forecasting techniques. This means that in forecasting the future values of the time series, previous values of only the same time series are used. The Holt–Winters method is a modification of the exponential smoothing technique, in which a forecast is computed as a weighted sum of previous time series values. Exponential smoothing can be applied merely to stationary time series. However, the Holt–Winters method is developed to cope with *nonstationarity, seasonality* and *trend* by estimating these properties and employing these estimates in the forecast. In addition, all the estimates are updated as the time evolves and as new time series values become available.

The estimates of mean, trend and seasonal component are scalars. The seasonal component is the deviation from the mean which is typical for each moment of the season. In the following, we use the notation $x(t)$ for the time series value, $m(t)$ for the estimate of the mean, $r(t)$ for the estimate of the trend and $s(t)$ for the estimate of the seasonal deviation at time t. First we have to find the starting values for these quantities, i.e. the values for $m(0)$, $r(0)$ and $s(-d+1)$, $s(-d+2)$, …, $s(0)$, where d is the length of the season. If the first forecast is to be made for time $t = 1$, the time series values from k preceding seasons are needed to initialise the estimates. Thus, the time series values which are used in initialising the mean, seasonal components and trend are denoted $X_{\text{init}} = \{x(-kd+1), x(-kd+2), \ldots, x(0)\}$. The estimate of the mean is initialised simply as the mean of X_{init}.

$$m(0) = \frac{1}{kd} \sum_{i=-kd+1}^{0} x(i) \tag{1}$$

An initial estimate for the trend can be calculated, for example, as the difference between the mean values of the last two seasons divided by the length of the season, which can be written

$$r(0) = \frac{1}{d}\left[\left(\frac{1}{d}\sum_{i=-d+1}^{0} x(i)\right) - \left(\frac{1}{d}\sum_{i=-2d+1}^{-d} x(i)\right)\right] \tag{2}$$

The initial values for the seasonal component $s(-d+j)$ can be calculated as the average deviation from the mean $m(0)$ at corresponding instants $t = -kd+j, -(k-1)d+j, -(k-2)d+j, \ldots, -d+j$, which is

$$s(-d+j) = \frac{1}{k}\sum_{l=1}^{k}(x(-ld+j) - m(0)) \tag{3}$$

where $j = 1, \ldots, d$.

When all the necessary variables and parameters are initialised, a forecast $\hat{x}(t+h)$ can be made h steps ahead, according to the formula

$$\hat{x}(t+h) = m(t) + hr(t) + s(t-d+h) \tag{4}$$

As can be seen from the above equation, the forecast is the sum of the estimates of the mean, trend and seasonal components. For some time series it would be more convenient to assume that seasonal effects are *multiplicative* instead of *additive*, but in the case of electric load data, the assumption of additivity is preferable.

As more measurements $x(1), x(2), \ldots$ become available, the estimates for $m(t)$, $r(t)$ and $s(t)$ are updated according to the formulas

$$m(t) = \alpha(x(t) - s(t-d)) + (1-\alpha)(m(t-1) + r(t-1)) \tag{5}$$

$$s(t) = \beta(x(t) - m(t)) + (1-\beta)s(t-d) \quad \text{and} \tag{6}$$

$$r(t) = \gamma(m(t) - m(t-1)) + (1-\gamma)r(t-1) \tag{7}$$

where d is the length of the season.

Before the Holt–Winters method can be used, the three parameters in Eqs (5)–(7) that control the adaptation speed have to be determined. These parameters are denoted as α, β and γ $(0 < \alpha, \beta, \gamma < 1)$, and they correspond to the adaptation speed of the estimates of mean, seasonal components and trend, respectively. Because there is no analytical technique to solve the optimal values of α, β and γ for a given time series, they must be determined by trying different combinations. Nevertheless, a systematic search is quite fast due to the simple nature of the Holt–Winters method. In addition, the accuracy of the forecast is not very sensitive to small fluctuations in parameter values α, β and γ. The nearly optimal values for these parameters can be found by discretising their continuous range of variation and searching through the whole discrete parameter space. A suitable discretisation step was found to be 0.1 for all three parameters. If we use 10 values 0.05, 0.15, 0.25, ..., 0.95, we end up with 10^3 different combinations of parameter values. For each combination, one step forecasts are made for a time series segment $t = 1, \ldots, N$, which follows immediately after the initialisation segment. The average forecasting error E is the average of the squared errors of one step forecasts

$$E(\theta(i)) = \frac{1}{N}\sum_{t=1}^{N}(\hat{x}(t;\theta(i)) - x(t))^2 \tag{8}$$

where $\theta(i)$ denotes the parameter combination $\{\alpha_i, \beta_i, \gamma_i\}$. In this method, the parameter combination that yields the smallest E is chosen.

3. Neural Network with Fuzzy Hidden Neurons

In this method we construct a network of fuzzy rules which is capable of learning the relationships between explanatory variables and target variables from the measured data. Because this method belongs to a class of connectionist methods, we call the vectors of explanatory variables the input patterns and the target variable the output. The network has a predetermined feed-forward structure with a large number of free parameters. The input patterns are fed into the network, a series of computational operations occurs in the network, and it produces the output. This output depends only on the input pattern and on the parameters of the network. The network is trained with a set of input–output pairs, which constitutes the training data.

The training is an iterative procedure in which the parameters of the network are adjusted to make it produce the desired output for each input pattern.

The training must be done iteratively, because the functions in the network are highly nonlinear, which is why there is no method to solve the optimal parameters directly. At each iteration, adjustments made to the parameters are relatively small. Because of this, the training data has to be presented to the network several times. One presentation of the whole training data is called an *epoch*. Once the network has learned to produce the desired output with a satisfactory accuracy for all the input patterns, or when the learning has stopped, the parameters of the network are fixed. With these parameters the network should be capable of producing reasonable outputs not only for those input patterns that were in the training data, but also for such unseen patterns that were not included in that data set.

The parameters of the network determine the rules by which the network produces its output. A rule consists of a partial condition for each input variable and a consequence. The fulfilments of the partial conditions are combined to measure the fulfilment of the whole condition. The structure of the network of fuzzy rules is presented in Fig. 1. The objects depicted as circles are called *neurons*. The inputs are connected directly to the *fuzzy neurons*, and the *output neuron* produces the output of the network. The output of a fuzzy neuron corresponds to the fulfilment of the condition being unity if the whole condition is 100% true, and zero if it is not true at all. Depending on the degree of fulfilment of the condition, the output of a fuzzy neuron can get values in the continuous range [0, 1].

The function of a fuzzy neuron is depicted in Fig. 2. It contains a triangular *membership function* for every input variable shown. The membership function can have values between 0 and 1. The condition implemented by a fuzzy neuron in Fig. 2 can also be expressed in the following linguistic form,

If x_1 is *about* 2 and x_2 is *about* 3
and x_3 is *about* 1, then …

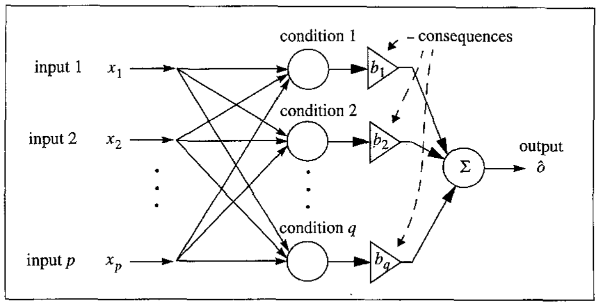

Fig. 1. Structure of the network of fuzzy rules.

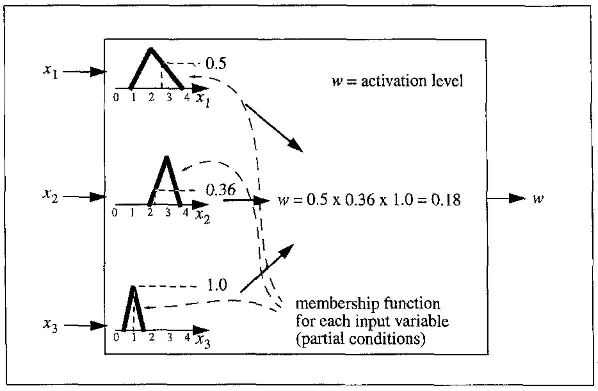

Fig. 2. Structure and principle of the function of a fuzzy neuron with three inputs.

The condition of each rule consists of several partial conditions which are specified using membership functions. The fulfilment of the whole condition is the product of the fulfilments of the partial conditions. Instead of multiplication, different rule combination methods can also be used. Some of those are described in [45,46]. The network contains several rules, and every rule has one consequence. The output of the whole network is the weighted mean of the consequences b_i,

$$\hat{o} = \sum_{i=1}^{q} w_i b_i \bigg/ \sum_{i=1}^{q} w_i \tag{9}$$

A triangular membership function is specified by three parameters p_1, p_2 and p_3, as depicted in Fig. 3. This membership function is approximated with a combination of two sigmoidal functions of the form

$$f(x) = \frac{1}{1 + \exp((x - \tau)/\sigma)} \tag{10}$$

where τ is a value at which $f(\tau) = 0.5$, and σ is a parameter which determines the steepness of the function. Parameters τ and σ are solved to approximate the slopes of the triangular membership function $\hat{R}(x)$. The sigmoid-triangular membership function is denoted by $\tilde{R}(x)$, and it is determined with sigmoid functions f_l (left) and f_r (right) of the form in Eq. (10). We use parameters p_1 and p_2 to find such parameters τ_l and σ_l that satisfy equations $f_l(p_1) = 0.1$ and $f_l(p_2) = 0.9$. Similarly, we find the

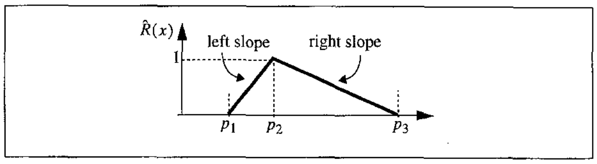

Fig. 3. Representation of a triangular membership function.

parameters τ_r and σ_r for a function that satisfies $f_r(p_2) = 0.9$ and $f_r(p_3) = 0.1$. Consequently, the following relations can be written:

$$\tau_l = \frac{p_1 + p_2}{2}, \quad \tau_r = \frac{p_2 + p_3}{2},$$

$$\sigma_l = \frac{p_1 - p_2}{2 \ln 9} \quad \text{and} \quad \sigma_r = \frac{p_3 - p_2}{2 \ln 9} \tag{11}$$

Finally, we scale the sigmoid functions by multiplying them by 1/0.9 so that $\tilde{R}(p_2) = 1$. Now we can write the sigmoid-triangular membership function in the form

$$\tilde{R}(x) = \frac{1}{0.9} \min \{f_l(x), f_r(x)\} \tag{12}$$

The resulting approximation of a triangular membership function is illustrated in Fig. 4. The activation of the jth fuzzy neuron with an input pattern $\boldsymbol{x}$ is

$$R_j(\boldsymbol{x}) = \prod_{i=1}^{p} \tilde{R}_i(x_i) \tag{13}$$

which is the product of the approximated membership functions.

The reason why purely triangular membership functions are not used is that they are not as tolerant as sigmoid-triangular membership functions. Intolerance means that if the input variable $x < p_1$ or $x > p_3$, the purely triangular membership function evaluates to zero. If even one membership function (i.e. partial condition) in a fuzzy neuron evaluates to zero, the condition is not true at all, which means that the output of that neuron is identically zero. This can easily lead to a situation in which there are input patterns for which the outputs of all the neurons are zero. This is an undefined situation, because the denominator in Eq. (9) becomes zero. On the other hand, this kind of situation cannot emerge with sigmoid-triangular membership functions, because those functions always evaluate to greater than zero with finite arguments.

Each of the fuzzy neurons is locally active, which means that it evaluates to unity for only one specific input pattern, called the centre of the condition, denoted by $\boldsymbol{c}$. For all other input patterns than $\boldsymbol{c}$, the activation of the neuron is less than unity. The width of the membership function determines the size of the region in which the neuron gives significant activation. The distribution of the centres is very significant with respect to the performance of the network. When an input pattern hits close to one condition centre, this particular neuron becomes activated distinctly more than the other neurons. Therefore, the consequence of that certain rule becomes weighted more than others. As a result, the output of the network is close to the consequence of this rule. So, the centres of the conditions are fixed points which produce the outputs that are close to the corresponding consequences. To be exact, when sigmoid-triangular membership functions are used, every rule always has at least a small contribution to the output, even if the input pattern is far from the centre of that rule. However, when the pattern hits close to one centre, the output is dictated practically by that single rule. With sigmoid-triangular membership functions, all input patterns cause an interpolation of the consequences to take place. Because of this, it is very important that the centres of the conditions are distributed effectively to the input space.

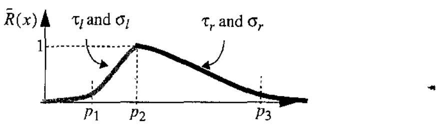

Fig. 4. Approximation of a triangular membership function.

There is no analytic method to find the optimal centres for given training data. Yet satisfactory results can be obtained by first creating a representative set of centres and then selecting only the best of those. Once the set of conditions is formed, the widths and centres can be further optimised, for example with some gradient-based optimising method. One exhaustive way to form the set of candidates for the centres is to make a candidate centre for each input pattern. The main disadvantage of this approach is that, in many cases, there is too much data for this. However, all the data is not needed to create a representative set of condition centres. It is essential that the centres cover those parts of the input space where the input data exists. One method to create the candidate centres is to use K-means clustering. It has been found to be a fast and efficient clustering algorithm. By clustering it is possible to determine just a few clusters, which are distributed into those parts of the input space where the data is. After clustering, we are not interested in what cluster each of the input patterns belongs to, but just where the centres of the clusters are. The vectors referring to the cluster centres are good representatives of the whole data. These centres can then be regarded as the centres of the candidate conditions. In the following, the K-means algorithm is presented [47].

The data to be clustered is $\boldsymbol{x}_i$, $i = 1, \ldots, N$, and $\boldsymbol{x}_i = [x_{i1}, x_{i2}, \ldots, x_{ip}]^T$. The number of clusters is k.

Step 1. Select k datavectors to be the initial cluster centres $\boldsymbol{c}_s$, $s = 1, \ldots, k$. The selection method could be, for example: $\boldsymbol{c}_s = \boldsymbol{x}_s$, $s = 1, g+1, 2g+1, 3g+1, \ldots, (k-1)g+1$, where $g = \lfloor N/k \rfloor$.

Step 2. For every datavector $\boldsymbol{x}_i$, compute the euclidean distance $d_{si} = \|\boldsymbol{x}_i - \boldsymbol{c}_s\|$ for $s = 1, \ldots, k$. Assign datavector $\boldsymbol{x}_i$ to the cluster

$$\xi(i) = \underset{s}{\operatorname{argmin}} \{d_{si} | s = 1, \ldots, k\} \qquad (14)$$

If all datavectors became assigned to the same clusters as in the previous iteration, stop the algorithm. Otherwise, go to step 3.

Step 3. Compute the new cluster centres $\boldsymbol{c}_s$ for each cluster $s = 1, \ldots, k$. The new cluster centre is equal to the mean of those datavectors which are assigned to that particular cluster. Go back to step 2.

The parameters p_2 which represent the centres of the membership functions in candidate conditions have now been determined, but the widths are not. The membership functions should cover the input space, and they are allowed to overlap with each other. These properties are preferable but not necessary. Because the input variables are not normalised, they may have different ranges of variation. This means that the widths of the membership functions for each input variable has to be scaled by the range of variation of that input quantity. Another point that affects the width of the membership function is the number of rules in the network we are aiming to design. If we are designing a network with a lot of rules, the membership functions should be narrower than if the network is going to have only a few rules. The third point that should be considered in choosing the widths is the dimensionality of the input space. If there are q rules in a p-dimensional hypercube and the rules are assumed to be uniformly distributed, there are about $\sqrt[p]{q}$ membership functions projected into one input variable axis. Even though the membership function does not have to be equally wide for both directions (left and right from the centre), which means that the one slope can be steeper than the other, at this stage both slopes are set equally steep. Thus, the conditions in the rules are identical in widths, but the centres are in different locations. The width of the membership function for the jth input variable is denoted by d_j and calculated as

$$\mathrm{d}_j = \frac{(\max_j - \min_j)}{\sqrt[p]{q}} \qquad (15)$$

where $\max_j = \max \{x_{ij} | i = 1, \ldots, N\}$ and $\min_j = \min \{x_{ij} | i = 1, \ldots, N\}$, p is the number of inputs and q is the number of rules. The parameters p_1 and p_3 for each membership function are now computed according to $p_1 = p_2 - d_j$ and $p_3 = p_2 + d_j$.

At this stage, parameters have been defined for k candidate conditions. However, our goal is to design a network of q rules ($q \ll k$). Next we use the Orthogonal Least Squares (OLS) algorithm to select the best conditions denoted by $\boldsymbol{C}_j$, $j = 1, \ldots, q$. In the field of neural computation, the OLS-algorithm was originally developed for training Radial Basis Function networks, but it can also be applied to fuzzy neural networks [48,49]. This algorithm is described in Appendix A. The candidate conditions are actually candidate fuzzy neurons which all respond to the input patterns differently. We apply the input patterns to the candidate neurons and record their normalised outputs for each input pattern. The normalised outputs of the jth candidate neuron are denoted by vector $\boldsymbol{r}_j = [r_{j1}, r_{j2}, \ldots, r_{jN}]^{\mathrm{T}}$, where

$$r_{ji} = R_j(\boldsymbol{x}_i) \Bigg/ \left(\sum_{l=1}^{k} R_l(\boldsymbol{x}_i) \right) \qquad (16)$$

The consequences for the conditions $\boldsymbol{C}_j$ can now be solved in the least squares sense. As mentioned earlier, the output of the network is given in Eq. (9). If we make use of matrix notation, the same equation yields

$$\begin{bmatrix} \mathrm{w}_{11} & \cdots & \mathrm{w}_{1q} \\ \cdots & \cdots & \cdots \\ \mathrm{w}_{N1} & \cdots & \mathrm{w}_{Nq} \end{bmatrix} \begin{bmatrix} b_1 \\ \cdots \\ b_q \end{bmatrix} = \begin{bmatrix} \hat{o}_1 \sum_{l=1}^{q} w_{1l} \\ \cdots \\ \hat{o}_N \sum_{l=1}^{q} w_{Nl} \end{bmatrix}$$

$$\text{or} \quad \boldsymbol{Wb} = \hat{\boldsymbol{o}} \operatorname{diag}(\boldsymbol{W1}) \qquad (17)$$

where N is the number of training pairs $\{(\boldsymbol{x}_i, o_i) | i = 1, \ldots, N\}$, q is the number of rules and w_{ij} is the output of the jth fuzzy neuron when $\boldsymbol{x}_i$ is applied to the input. The notations diag($\boldsymbol{a}$) and $\mathbf{1}$ are

$$\operatorname{diag}(\boldsymbol{a}) = \begin{bmatrix} a_1 & 0 & 0 \\ 0 & \ldots & 0 \\ 0 & 0 & a_N \end{bmatrix} \quad \text{and} \quad \mathbf{1} = \begin{bmatrix} 1_1 \\ \ldots \\ 1_N \end{bmatrix} \qquad (18)$$

Now, the vector of consequences is to be solved when the vector of desired outputs is known. Let us rearrange the terms in Eq. (17), replace the output of the network $\hat{\boldsymbol{o}}$ with the desired output of the network $\boldsymbol{o}$ and solve with respect to $\boldsymbol{b}$:

$$\boldsymbol{b} = (\boldsymbol{W}^{\mathrm{T}}\boldsymbol{W})^{-1} \boldsymbol{W}^{\mathrm{T}}\boldsymbol{o} \operatorname{diag}(\boldsymbol{W1}) \qquad (19)$$

The parameters of the rule base are now determined, but they can still be fine-tuned. In the optimisation, a gradient-based method is used to minimise the sum of squared errors between the output of the model and the desired output [50–52]. The derivation of the formulas used in training is described in Appendix B.

4. Adding New Rules to the Network

When the network of fuzzy rules has been built and optimised for a given set of training data, it can be fed with unseen input patterns that resemble the input patterns in the training data. We call these input patterns *insiders*, because they lie more or less in the input space over which the condition centres are scattered. However, if the unseen input pattern is dissimilar to all the training data, the network cannot produce any reasonable output. This is because the network has not learned any input–output relationships for that input pattern. These input patterns we call *outsiders*.

There are real world systems whose properties gradually change, and the training data thus becomes out-dated. To model such systems the network needs to be trained again when its performance has deteriorated too much with respect to the most recent data. There are two ways to keep the network of fuzzy rules up-to-date: optimising the network continuously with the newest data available; or building up the whole rule base for the set of newest data. The optimisation process can be used when there are only small fluctuations in the properties of the system. In the optimisation, mainly the widths of the membership functions and the consequences are adapted, but the centres of the conditions remain practically almost fixed. This is why the whole rule base has to be rebuilt when the properties of the system have developed considerably.

Optimisation of the network with the latest data is computationally less expensive than rebuilding the whole model. Because of this, it is not worth rebuilding the whole model every time there is a new measurement to teach to the network. Instead, the newest measurement can be included into the training data for which the network is optimised. If the real world system that is being modelled is continuously changing in the long run, and mainly to the same direction, the network will eventually have to be rebuilt. The exact moment when the optimisation is not enough and the rebuilding is needed depends on many things, like the size and complexity of the network, computational power and time available, precision needed from the network, rate of change in the modelled system, and so on.

We have developed a method which increases the lifetime of the rule base in changing conditions. This method is based on augmenting the rule base with a new rule when an outsider is detected. Adding rules to the rule base does not compensate fully for rebuilding of the network, but it allows the old rule base to be used for longer. The rule base augmenting algorithm is much more heuristic than the OLS-algorithm which is used in selecting the condition centres in the network building stage. Therefore, the rules that are added on-line to the network are not optimal, whereas the original rules are.

The locations of the condition centres $\boldsymbol{C}_i$, $i = 1, \ldots, q$, with respect to the new input pattern $\boldsymbol{x}$, are used to determine whether or not an input pattern is an outsider. The procedure of classification of an input pattern into an insider or outsider is divided into two steps, which are described below and illustrated in Fig. 5:

Step 1:

Calculate the distances $d_{1i} = \|\boldsymbol{x} - \boldsymbol{C}_i\|$ for $i = 1, \ldots, q$.

Denote the nearest condition centre $\boldsymbol{C}_{\text{near}}$.

Calculate the difference vector $\boldsymbol{v} = \boldsymbol{x} - \boldsymbol{C}_{\text{near}}$.

Generate a test pattern $\boldsymbol{x}_{\text{test}} = \boldsymbol{C}_{\text{near}} + 1.01 \cdot \boldsymbol{v}$.

Step 2:

Calculate the distances $d_{2i} = \|\boldsymbol{x}_{\text{test}} - \boldsymbol{C}_i\|$ for $i = 1, \ldots, q$.

If $\exists i$, for which $d_{2i} < d_{1i}$, the pattern $\boldsymbol{x}$ is an insider. => No need for a new rule, stop.

Otherwise the pattern is an outsider and a new rule has to be generated.

Generation of the new rule

New condition centre is $\boldsymbol{C}_{q+1} = \boldsymbol{C}_{\text{near}} + 2 \cdot \boldsymbol{v}$. Membership function width ω_i for the ith variable is the mean of the widths of all other rules for that variable. The meaning of the condition centre c_i and the width ω_i is illustrated in Fig. 6.

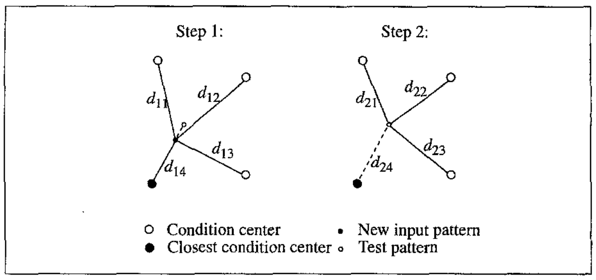

Fig. 5. An example of determining whether the new input pattern is an insider or an outsider. The pattern turns out to be an insider because $d_{21} < d_{11}$ (also $d_{22} < d_{12}$).

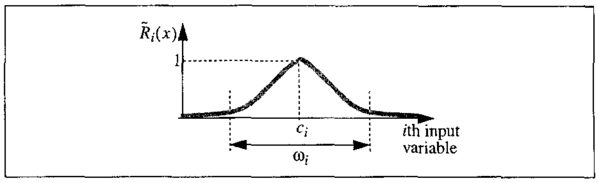

Fig. 6. Shape and parameters of the membership function for *i*th variable in a new rule.

The consequence of the new rule is the observed output for that input pattern.
The number of rules is incremented by 1: Set $q = q + 1$.

5. The Results of the Forecasts

The forecasting simulations were made for three different time series of hourly electric load. The simulations were made with both the Holt–Winters (HW) method and the neuro-fuzzy (NF) method. In the Holt–Winters method, only the previous values of the electric load time series were used in forecasting. In the neuro-fuzzy model, the input variables shown in Table 1 were used.

The two-dimensional coding of the time-of-day and day-of-week information can be interpreted as the *xy*-coordinates of the circumference of a unit circle. The hourly forecasts were made for data which are illustrated in Fig. 7 with dotted line. We have chosen three time series periods, 76 days (series 1), 38 days (series 2) and 11 days (series 3) in length.

The models were initialised with data that was from four weeks preceding the forecasting period. This initialisation part of the time series is plotted with a solid line in Fig. 7. During the forecasting simulations, models were updated hour by hour with the newest data available. The HW model was updated according to Eqs (5)–(7). The NF model was trained hourly with the data from the latest 24

Table 1. The input variables used in the neuro-fuzzy model

Number	*Name of the input variable*
1	Electric load of the previous hour
2	Average outdoor temperature of the previous 6 hours
3	Time-of-day (*x*-coordinate)
4	Time-of-day (*y*-coordinate)
5	Day-of-week (*x*-coordinate)
6	Day-of-week (*y*-coordinate)

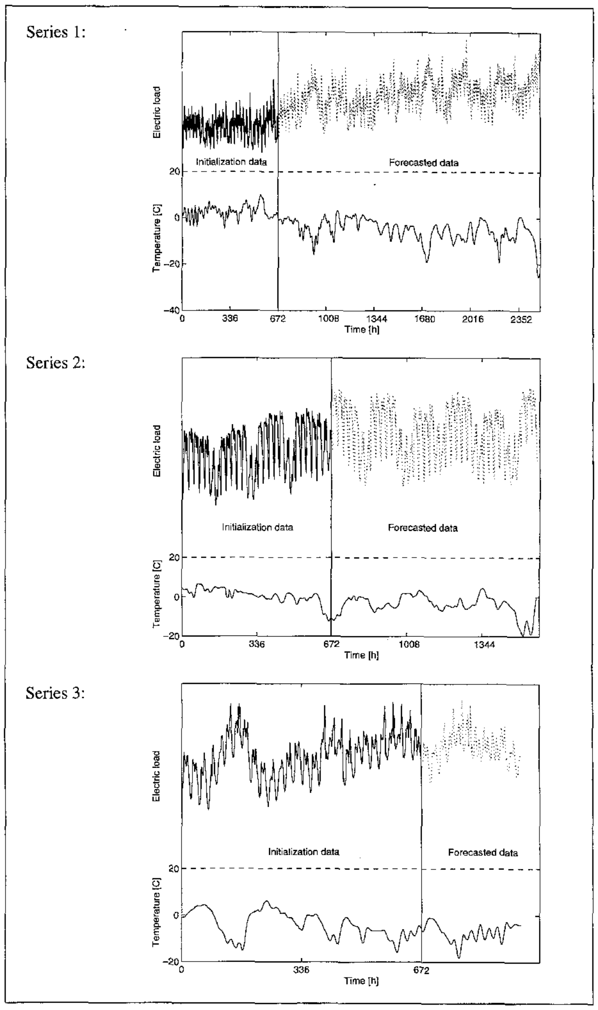

Fig. 7. The electric load time series used in forecasting simulations.

hours for a fixed number of epochs. Also, new rules were added to the rulebase when it was found to be necessary according to the algorithm. The forecasts were made hourly for 1–12 hours ahead recursively with 1-hour forecasts, so that the forecasted load was regarded as an actual load in forecasting the load of the following hour.

Three different measures were used to examine the accuracy of the forecasting methods: Mean Squared Error (MSE), Mean Absolute Error (MAE) and Mean Absolute Peak Error (MAPE). MSE and MAE are self-evident, and the MAPE is the mean of the absolute forecasting errors in daily peaks of electric load. MAPE is included because it shows the accuracy of the forecasts for those hours for which it is most important to know the electric load. The results for all the series are depicted in Fig. 8.

The simulations were made with the assumption that accurate temperature forecasts were available. In real application, however, more or less inaccurate

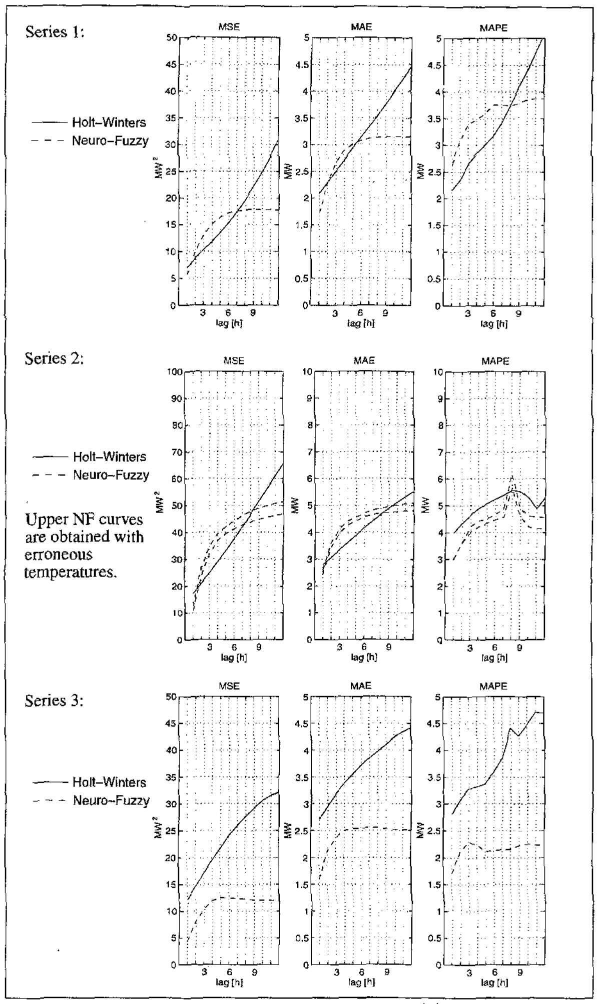

Fig. 8. The forecasting errors in simulations for series 1–3.

temperature forecasts would have to be used. The effect of error in temperature forecast was briefly tested with an additional simulation with series 2. The error was added to the measured temperature time series. Because the actual temperature forecasts were not available for this particular period, the typical error in temperature forecasts was obtained from another period as a difference between the forecasted and measured temperature. So the error in temperature resembles the actual forecasting error for 24 hours ahead. In Fig. 8 those errors that increase faster as a function of the time lag are made with erroneous temperatures. The MSE of the temperature error was 3.2 (°C)^2 and the MAE of it was 1.5°C.

The neuro-fuzzy method gives the user the possibility to select which kind of daytype is used in forecasting the electric load for a given day. This can be utilised to cope with so-called 'special' days. In most special days, the electric load profile resembles the profile during weekends. Therefore, the special day can be regarded as either a Saturday or Sunday. This information can be given to the neuro-fuzzy model by replacing the day-of-week code of the weekday by the code of the weekend day. A simulation of a period of one week was done with Monday being the Independence Day in Finland, a national holiday. The simulation was made with both normal weekday coding (1, 2, ..., 7) and with Monday's code replaced by Saturday's code (6, 2, 3, ..., 7). The results of this simulation are shown in Fig. 9. Those forecasts were made with a 12 hour forecasting horizon.

Both forecasting methods were also tested for one month's period in two electricity works, denoted by EW1 and EW2. The test period for EW1 was January 25th–February 28th, and for EW2 it was February 10th–March 13th in 1995. The results from the test-runs are represented in Fig. 10. In EW1 there was a reasonable temperature forecast available, but in EW2 the temperature forecast was extremely poor. The errors of the temperature forecasts are illustrated in Fig. 10 in the lowest two plots. Both the MSE and MAE of the temperature forecasts are plotted as a function of the forecasting lag. The effect of the poor temperature forecast in EW2 can be seen in Fig. 10 by comparing the electric load forecasting errors in EW1 and in EW2.

6. Discussion

For simulations with precise temperature forecasts the results from series 1 and 2 are quite similar. The MSE of forecasting with the Holt–Winters method increases almost linearly as a function of the forecasting lag. In one hour forecasts the error is almost the same with the HW and neuro-fuzzy methods. In forecasts from 2 to 7 hours, the error of the NF method is greater than with the HW method, but with over 7 hour forecasts the error is less. However, with series 3, the neuro-fuzzy method clearly gives

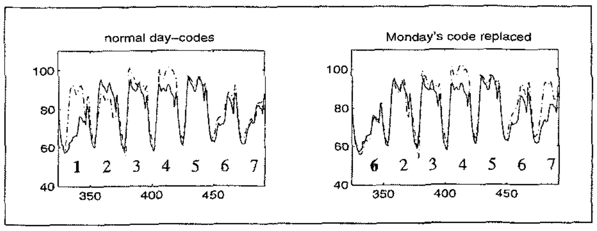

Fig. 9. The results of forecasting the electric load on Monday which is a special day (Independence Day in Finland 6.12.1993) with the code of Monday (left plots) and with the code of Saturday (right plots). The forecast is plotted for a whole week and the Independence Day is the first day in plots. Solid line = actual load, dashed line = forecasted load.

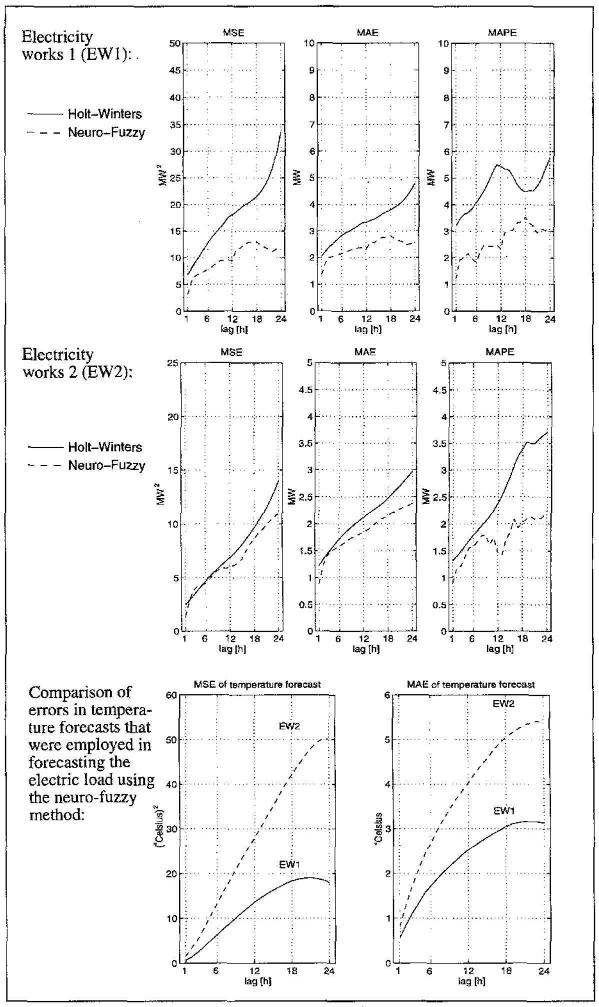

Fig. 10. The forecasting errors in two test runs in real electricity works.

better forecasts than the Holt–Winters method for all forecasting lags from 1 to 12 hours. The reason why the neuro-fuzzy model performed better with series 3 than with series 1 and 2 is hidden in the relationship between the initialisation data and the forecasted data. If we look at Fig. 7, we can see that the only series where the electric load stays relatively stationary is in series 3. Because the rule base of the neuro-fuzzy model is optimised for the initialisation data, and the test data resembles the initialisation data, the performance is quite good. But as the test data drifts away from the initialization data, the network cannot adapt efficiently to the changing conditions. This adaptation would be needed in series 1 and 2, but not in series 3. In addition, it must be noted that in the training of the neuro-fuzzy model, the negative errors were weighted more than the positive errors. This is because the forecasts were preferred to be too large instead of being too low. This weighting weakens the performance measured with MSE and MAE, but improves it when the measure is MAPE. This effect can be seen most clearly in Fig. 8, series 2 and in Fig. 10, electricity works 2. In these figures the MAPE measure indicates better performance than the other measures.

The forecasting with erroneous temperatures for series 2 shows the sensitivity of the forecasting accuracy to the outdoor temperature with the neuro-fuzzy model. The results show that with relatively small errors in temperature forecasts, the performance is not drastically deteriorated. The MSE error increased about 10% in 12 hour forecasts. The simulation concerning the forecasting for special days shows that the normal weekday rhythm can easily be altered to also cope with special days which otherwise would be difficult to forecast and would deteriorate forecasts for other days. In Fig. 9 two effects can be seen when Monday's code has been replaced with Saturday's code. First, the forecast for Monday is clearly improved due to the replacement of the day-of-week code. Second, the forecast for Tuesday is also improved. The explanation for this effect is that the neuro-fuzzy model assumes Monday and Tuesday to be quite similar, and therefore the special Monday profile is regarded as a new model for the weekday load. However, the replacement prevents the model from wrongly learning that Mondays are gradually starting to resemble Saturdays. Virtually, the coding of the day-of-week was done with *xy*-coordinates, and not with integer numbers as depicted in Fig. 9 for simplicity.

The results from the real world tests illustrated in Fig. 10 are similar to those that were obtained with simulations. One remarkable difference between the simulations and the real world test-run is, however, that the simulation forecasts were made for 1–12 hours ahead, but in the real world tests the forecasts were made for 1–24 hours ahead. Therefore, the real world tests give additional information on how the methods behave when the forecasting horizon is pushed further away. The forecasts that are obtained with the Holt–Winters method start deteriorating fast for near 24 hour forecasts, whereas the neuro-fuzzy method gives quite reliable forecasts even for 24 hours ahead, at least for EW1. In the case of EW2, as mentioned earlier, all the forecasts get poorer very quickly as a function of the forecasting lag. This is probably because the temperature forecasting error also increases very quickly. The temperature forecasting errors for over 6 hour forecasts are greater than the error added in simulation of forecasting series 2. This implies that if the temperature forecasts had been more accurate, the forecasting accuracy might also have been better for both EW1 and EW2.

7. Conclusion

In this study a modern short term electric load forecasting method was developed and its performance quantitatively compared with a more conventional forecasting method. The modern method was a feedforward neural network that uses fuzzy membership functions in the hidden layer. The membership functions were approximations of triangular membership functions. The approximation was done as a combination of two sigmoidal functions. Nonstationarity in the electric load time series raises the need for augmenting the rule base. For this purpose a new scheme was proposed. The variables used as inputs to the model were the electric load of the previous hour, the outdoor temperature averaged over the last 6 hours, the time-of-day and the day-of-week. Time and day information were coded as points in the xy-coordinates on the circumference of a unit circle. The forecasting performance was tested with simulations and with one month test-runs in two real world electricity works. A traditional forecasting method used for comparison was the Holt–Winters exponential smoothing method.

The superiority of the modern method was not so evident in simulations, but in real world test-runs it outperformed the conventional Holt–Winters method. In particular, it was found that nonstationary time series are difficult to forecast with the proposed network of fuzzy rules. Also, the importance of the accuracy of the temperature forecast was discovered. However, even though the temperature forecasts available were extremely poor, the electric load forecasting error with the modern method did not exceed the error of the Holt–Winters method. In addition, the modern method was also shown to be capable of forecasting the electric load on special days with relatively good accuracy.

The proposed method was shown to be a promising forecasting method and a subject for further research. The performance of the network of fuzzy rules in forecasting nonstationary time series could probably be improved by developing a more efficient rule adding algorithm.

Acknowledgements. This work has been financially supported by Enermet Oy and Foundation for the Advancement of Technology in Finland.

References

1. Bunn DW, Farmer ED. Comparative Models for Electrical Load Forecasting. Wiley, Chichester, 1985
2. Moghram I, Rahman S. Analysis and evaluation of five short-term load forecasting techniques. IEEE Trans Power Syst 1989; 4(4): 1484–1491
3. Gross G, Galiana F. Short term load forecasting. Proc IEEE 1987; 75(12): 1558–1573
4. Box GEP, Jenkins GM. Time Series Analysis, Forecasting and Control. Holden Day, San Francisco, 1970
5. Chatfield C. The Analysis of Time Series: An Introduction (third edition). Chapman & Hall, London, 1984
6. Hagan MT, Behr SM. The time series approach for short term load forecasting. IEEE Trans Power Syst 1987; 2(3): 785–791
7. Montgomery DC, Johnson LA. Forecasting and Time Series Analysis. McGraw Hill, USA, 1976
8. Papalexopoulos AD, Hesterberg TC. A regression-based approach to short-term system load forecasting. IEEE Trans Power Syst 1990; 5(4): 1535–1544
9. Satoh R, Tanaka E, Hasegawa J. Daily load forecasting using a neural network combined with regression analysis. In: Proc Int Conf Intelligent System Application to Power Systems, vol. 2, Montpellier, France, 5–9 September 1994; 345–352
10. Ho KL, Hsu Y-Y, Chen C-F, Lee T-E, Liang CC, Lai T-S, Chen KK. Short term load forecasting of Taiwan power system using a knowledge based expert system. IEEE Trans Power Syst 1990; 5(4): 1214–1221
11. Hsu Y-Y, Ho KL. Fuzzy expert systems: an application to short term load forecasting. IEE Proc C 1992; 139(6): 471–477
12. Jabbour K, Riveros JFV, Landsbergen D, Meyer W. ALFA: automated load forecasting assistant. IEEE Trans Power Syst 1988; 3(3): 908–914
13. Ranman S, Bhatnagar R. An expert system based algorithm for short term load forecast. IEEE Trans Power Syst 1988; 3(2): 392–399
14. Dillon TS, Morsztyn K, Phua K. Short term load forecasting using adaptive pattern recognition and self organising techniques. In: Fifth Power Systems Computation Conference, PSCC Proceedings, Cambridge, September 1–5 1975
15. Bacha H, Meyer W. Automated load forecasting using neural networks. In: Proc American Power Conference. 54(2), Illinois Institute of Technology, Chicago, IL, 1992; 1144–1149
16. Chaudhary SD, Kalra PK, Srivastava SC, Vinod Kumar DM. Short term electric load forecasting using artificial neural network. In: Proc Expert System Application to Power Systems IV, La Trobe, Melbourne, Australia, 4–8 January 1993; 159–163
17. Chen S-T, Yu DC, Moghaddamjo AR. Weather sensitive short-term load forecasting using nonfully connected artificial neural network. IEEE Trans Power Syst 1992; 7(3): 1098–1105
18. Connor JT, Atlas LE, Martin D. Recurrent neural networks and load forecasting. In: Proc 1st Int Forum on Applications of Neural Networks to Power Systems, Seattle, WA, 23–26 July 1991; 22–25
19. Dash PK, Dash S, Rahman S, Chandrasekharaigh HS. Short term load forecasting using artificial neural network with a fast learning algorithm. In: Proc Expert System Application to Power Systems IV, La Trobe, Melbourne, Australia, 4–8 January 1993; 169–174
20. Dillon TS, Sestito S, Leung S. Short term load fore-

casting using an adaptive neural network. Electrical Power & Energy Syst 1991; 13(4): 186–192
21. Djukanovic M, Babic B, Sobajic DJ, Pao Y-H. Unsupervised/supervised learning concept for 24-hour load forecasting. IEE Proc C 1993; 140(4): 311–318
22. El-Sharkawi MA, Oh S, Marks RJ, Damborg MJ. Short-term electric load forecasting using an adaptively trained layered perceptron. Proc 1st Int Forum on Applications of Neural Networks to Power Systems, Seattle, WA, 23–26 July 1991; 41–45
23. Ho K-L, Hsu Y-Y, Yang C-C. Short term load forecasting using a multilayer neural network with an adaptive learning algorithm. IEEE Trans Power Systems 1992; 7(1): 141–149
24. Hsu Y-Y, Yang C-C. Design of artificial neural networks for short-term load forecasting. Part I: Self-organising feature maps for day type identification. IEE Proc C 1992; 138(5): 407–413
25. Hsu Y-Y, Yang C-C. Design of artificial neural networks for short-term load forecasting. Part II: Multilayer feedforward networks for peak load and valley load forecasting. IEE Proc C 1992; 138(5): 414–418
26. Hwang J-N, Moon S. Temporal difference method for multi-step prediction application to power load forecasting. In: Proc 1st Int Forum on Applications of Neural Networks to Power Systems, Seattle, WA, 23–26 July 1991; 41–45
27. Lee KY, Cha YT, Park JH. Short-term load forecast using an artificial neural network. IEEE Trans Power Systems 1992; 7(1): 124–132
28. Lu CN, Wu HT, Vemuri S. Neural network based short term load forecasting. IEEE Trans Power Syst 1993; 8(1): 336–342
29. Park DC, El-Sharkawi MA, Marks II RJ. An adaptively trained neural network. IEEE Trans Neural Networks 1991; 2(3): 334–345
30. Park DC, El-Sharkawi MA, Marks II RJ, Atlas LE, Damborg MJ. Electric load forecasting using an artificial neural network. IEEE Trans Power Syst 1991; 6(2): 442–449
31. Park DC, Mohammed O, El-Sharkawi MA, Marks II RJ. An adaptively trainable neural network and its application to electric load forecasting. In: Proc 1st Int Forum on Applications of Neural Networks to Power Systems, Seattle, WA, 23–26 July 1991; 7–11
32. Peng TM, Hubele NF, Karady GG. Conceptual approach to the application of neural network for short term load forecasting. In: IEEE Int Symposium on Circuits and Systems, New Orleans, LA, May 1990; 2342–2345
33. Peng TM, Hubele NF, Karady GG. Advancement in the application of neural networks for short-term load forecasting. IEEE Trans Power Syst 1992; 7(1): 250–257
34. Peng TM, Hubele NF, Karady GG. An adaptive neural network approach to one-week ahead load forecasting. IEEE Trans Power Syst 1993; 8(3): 1195–1202
35. Srinivasan D, Liew AC, Chen JSP. Short term forecasting using neural network approach. In: Proc 1st Int Forum on Applications of Neural Networks to Power Systems, Seattle, WA, 23–26 July 1991; 12–16
36. Wu H-T, Lu C-N. Using artificial neural network for providing hourly load update and next day load profile. In: Proc Int Conf Advances in Power System Control, Operation and Management, Hong-Kong, November 1991; 895–901
37. Dash PK, Dash S, Rahman S. A hybrid artificial neural network-fuzzy expert system for short term load forecasting. In: Proc Expert System Application to Power Systems IV, La Trobe, Melbourne, Australia, 4–8 January 1993; 175–180
38. Dash PK, Liew AC. A comparative study of load forecasting models using fuzzy neural networks. Proc Int Conf Intelligent System Application to Power Systems, vol. 2, Montpellier, France, 5–9 September 1994; 865–872
39. Jang J-SR, Sun C-T. Predicting chaotic time series with fuzzy if-then rules. In: Proc 2nd IEEE Int Conf Fuzzy Systems, vol. 2, San Francisco, CA, 1993; 1079–1084
40. Jang J-SR. ANFIS: Adaptive-network-based fuzzy inference system. IEEE Trans Neural Networks 1993; 23(3): 665–685
41. Katayama R, Kajitani Y, Kuwata K, Nishida Y. Self-generating radial basis function as neuro-fuzzy model and its application to nonlinear prediction of chaotic time series. Proc 2nd IEEE Int Conf Fuzzy Systems, vol 1, San Francisco, CA, 28 March–1 April 1993; 407–414
42. Kim KH, Park D-Y, Park J-K. A hybrid model of artificial neural network and fuzzy expert system for short term load forecast. Proc Expert System Application to Power Systems IV, La Trobe, Melbourne, Australia, 4–8 January 1993; 164–168
43. Makino K, Shimada T, Ichikawa R, Ono M, Endo T. Short-term load forecasting using an artificial neural network of locally active units. In: Proc Int Conf Intelligent System Application to Power Systems, vol 2, Montpellier, France, 5–9 September 1994; 849–856
44. Mori H, Kobayshi H. A fuzzy neural net for short term load forecasting. Proc Int Conf Intelligent System Application to Power Systems, vol 1, Montpellier, France, 5–9 September 1994; 775–782
45. Nakanishi H, Turksen IB, Sugeno M. A review and comparison of six reasoning methods. Fuzzy Sets and Systems 1993; 57(1): 257–294
46. Turksen IB, Tian Y. Combination of rules or their consequences in fuzzy expert systems. Fuzzy Sets and Systems 1993; 58(1): 3–40
47. Jain AK, Dubes RC. Algorithms for Clustering Data. Prentice Hall, New Jersey, 1988
48. Jang J-SR, Sun C-T. Functional equivalence between radial basis function networks and fuzzy inference systems. IEEE Trans Neural Networks 1993; 4(1): 153–159
49. Wang L-X, Mendel JM. Fuzzy basis functions, universal approximation and orthogonal least squares learning. IEEE Trans Neural Networks 1992; 3(5): 807–814
50. Guély F, Siarry P. Gradient descent method for optimising various fuzzy rule bases. Proc 2nd IEEE Int Conf Fuzzy Systems, vol 2, San Francisco, CA, 1993; 1241–1246
51. Horikawa S, Furuhashi T, Uchikawa Y. On identification of structures in premises of a fuzzy model using a fuzzy neural network. In: Proc 2nd IEEE Int Conf Fuzzy Systems, vol 1, San Francisco, CA, 28 March–1 April 1993; 661–666
52. Rumelhart DE, McClelland JL, and the PDP Research Group. Parallel Distributed Processing, vol 1, MIT Press, Cambridge, MA, 1988

Appendix A: OLS-algorithm

The OLS-algorithm starts by selecting the neuron that has the biggest contribution to the vector of the desired outputs of the network. This is done in step 1:

```
Step 1. For $i = 1$ to $k$ do $\left\{ g_i = \frac{\boldsymbol{r}_i^{\mathrm{T}} \boldsymbol{o}}{\boldsymbol{r}_i^{\mathrm{T}} \boldsymbol{r}_i} \quad \mathrm{err}_i = g_i^2 \frac{\boldsymbol{r}_i^{\mathrm{T}} \boldsymbol{r}_i}{\boldsymbol{o}^{\mathrm{T}} \boldsymbol{o}} \right\}$
    Set $\boldsymbol{h}_1 = \boldsymbol{r}_\xi$, $\xi$ satisfying $\xi = \underset{i}{\mathrm{argmax}} \{\mathrm{err}_i | i = 1, \ldots, k\}$.
    Set $\boldsymbol{C}_1 = \boldsymbol{c}_\xi$ to be the best centre.
    Drop column vector $\boldsymbol{r}_\xi$ out of $\{\boldsymbol{r}_i | i = 1, \ldots, k\}$.
```

In step 2 the rest of the condition centres are selected one-by-one. The measure according to which the column vectors are selected is the *error reduction ratio*, err. The bigger the additional contribution of the condition centre j is to the variance of the output vector $\boldsymbol{o}$, the bigger the err_j. Step 2 of the OLS-algorithm is as follows:

```
Step 2. For $j = 2$ to $q$ do {
    Compute for all remaining column vectors $\{\boldsymbol{r}_x\}$:
    {
    $\alpha_{ij} = \frac{\boldsymbol{h}_i^{\mathrm{T}} \boldsymbol{r}_x}{\boldsymbol{h}_i^{\mathrm{T}} \boldsymbol{h}_i}$, for $i = 1, \ldots j - 1$.
    $\psi_x = \boldsymbol{r}_x - \sum_{l=1}^{j-1} \alpha_{lj} \boldsymbol{h}_l$
    $g_x = \frac{\psi_x^{\mathrm{T}} \boldsymbol{o}}{\psi_x^{\mathrm{T}} \psi_x} \quad \mathrm{err}_x = g_x^2 \frac{\psi_x^{\mathrm{T}} \psi_x}{\boldsymbol{o}^{\mathrm{T}} \boldsymbol{o}}$
    }
    Set $\boldsymbol{h}_j = \psi_\xi$, $\xi$ satisfying
    $\xi = \underset{i}{\mathrm{argmax}} \{\mathrm{err}_i | i = 1, \ldots, k$, except the selected indexes$\}$
    Set $\boldsymbol{C}_j = \boldsymbol{c}_\xi$ to be the $j$th best centre.
    Drop column vector $\boldsymbol{r}_\xi$ out of $\{\boldsymbol{r}_x\}$.
    }
```

Appendix B: Derivation of Formulas used in Network Training

The error to be minimised can be written as

$$E = \sum_{i=1}^{N} (\hat{o}_i - o_i)^2 \tag{20}$$

where $\hat{o}$ is the output of the model and o is the desired output. In the following, we consider only one single output value $\hat{o}$ and corresponding desired output value o instead of the whole set of N values in deriving the formulas for updating. Parameters p_1, p_3 and the consequence vector $\boldsymbol{b}$ were updated directly with the gradient method. The network is trained with the so-called batch-training method. In batch training, all training data is presented to the network while the parameters of the network remain constant. Each training pair reveals an adjustment for every parameter. These adjustments are accumulated separately for each of the parameters. When all training patterns are represented, the accumulated adjustments are applied to the parameters. Next the process is repeated with the same data, but with these newly adjusted parameters. Parameters p_1 and p_3 were updated with the amount of Δp_1 and Δp_3, respectively. Δp_i is a resulting cumulative sum of change to parameter p_i, when the whole training data is presented to the network. Parameter p_2 was updated by $(\Delta p_1 + \Delta p_3)/2$ if both Δp_1 and Δp_3 were of the same sign. In the following derivation, we denote p_1 as l (left parameter), p_3 as r (right parameter) and p_2 as m (parameter of the tip of the triangle). To know how to update the parameters, we need to calculate the derivative of E with respect to each parameter. They are calculated by using the chain rule. In the following, the equations for computing the updates for $\boldsymbol{b}$, l_{ij} and r_{ij} are derived:

$$\frac{\partial E}{\partial b_i} = \frac{\partial E \partial \hat{o}}{\partial \hat{o} \partial b_i} \tag{21}$$

$$\frac{\partial E}{\partial l_{ij}} = \frac{\partial E \partial \hat{o} \partial w_i}{\partial \hat{o} \partial w_i \partial f_{ij}} \left[\frac{\partial f_{ij} \partial \sigma_{ij}}{\partial \sigma_{ij} \partial l_{ij}} + \frac{\partial f_{ij} \partial \tau_{ij}}{\partial \tau_{ij} \partial l_{ij}} \right] \tag{22}$$

$$\frac{\partial E}{\partial r_{ij}} = \frac{\partial E \partial \hat{o} \partial w_i}{\partial \hat{o} \partial w_i \partial f_{ij}} \left[\frac{\partial f_{ij} \partial \sigma_{ij}}{\partial \sigma_{ij} \partial r_{ij}} + \frac{\partial f_{ij} \partial \tau_{ij}}{\partial \tau_{ij} \partial r_{ij}} \right] \tag{23}$$

In Eqs (21)–(23), w_i is the output of the ith hidden fuzzy neuron, f_{ij} is a sigmoid function of the jth input variable in the ith rule, σ_{ij} is the steepness coefficient of the sigmoid function f_{ij} and τ_{ij} is the offset of the sigmoid function f_{ij}. Every f_{ij} has two values for σ_{ij} and τ_{ij} (one for the left slope and another for the right), and the left one is selected when the jth input variable $x_j < p_2$ and the right one is selected when $x_j > p_2$. In Eqs (22) and (23), there are two terms in brackets because parameters l_{ij} and r_{ij} affect both the steepness of the sigmoid and the offset of the sigmoid. The terms in Eqs. (21)–(23) can be written as

$$\frac{\partial E}{\partial \hat{o}} = 2(\hat{o} - o), \quad \frac{\partial \hat{o}}{\partial b_i} = \frac{w_i}{\sum_{k=1}^{q} w_k},$$

$$\frac{\partial \hat{o}}{\partial w_i} = \frac{b_i - \hat{o}}{\sum_{k=1}^{q} w_k}, \quad \frac{\partial w_i}{\partial f_{ij}} = \frac{w_i}{f_{ij}} \tag{24}$$

$$\begin{aligned} \frac{\partial f_{ij}}{\partial \sigma_{ij}} &= \frac{x_j - \tau_{ij}}{\sigma_{ij}^2} \left[1 + \exp\left(\frac{x_j - \tau_{ij}}{\sigma_{ij}} \right) \right]^{-2} \\ &= \frac{x_j - \tau_{ij}}{\sigma_{ij}^2} (f_{ij}(x))^2 \end{aligned} \tag{25}$$

$$\begin{aligned} \frac{\partial f_{ij}}{\partial \tau_{ij}} &= \frac{1}{\sigma_{ij}} \left[1 + \exp\left(\frac{x_j - \tau_{ij}}{\sigma_{ij}} \right) \right]^{-2} \\ &= \frac{1}{\sigma_{ij}} (f_{ij}(x))^2 \end{aligned} \tag{26}$$

$$\frac{\partial \sigma_{ij}}{\partial l_{ij}} = \frac{1}{2 \ln 9}, \quad \frac{\partial \sigma_{ij}}{\partial r_{ij}} = \frac{1}{2 \ln 9} \tag{27}$$

$$\frac{\partial \tau_{ij}}{\partial l_{ij}} = \frac{1}{2}, \quad \frac{\partial \tau_{ij}}{\partial r_{ij}} = \frac{1}{2} \tag{28}$$

The parameters b_i are updated after epoch number e according to the equation

$${}^{e+1}b_i = {}^{e}b_i - \Delta^{e}b_i, \quad \text{where}$$

$$\Delta^{e}b_i = {}^{e}\gamma \sum_{s=1}^{n} \left(\left.\frac{\partial E}{\partial b_i}\right|_{pattern = \{x_s|o_s\}} \right) \tag{29}$$

The expression *pattern* = $\{x_s|o_s\}$ means that the derivative is evaluated when $\boldsymbol{x}_s$ is applied to the input and o_s is the desired output. The coefficient ${}^{e}\gamma$ is called a *step-length*, and by changing its value the amount of the update in each epoch can be adjusted. The number of updates to be made to parameters r_{ij} and l_{ij} is calculated similarly as in Eq. (29), but the actual adjustments are put into effect only under the following conditions:

$${}^{e+1}l_{ij} = \begin{cases} {}^{e}l_{ij} - \Delta^{e}l_{ij}, & \text{if } \Delta^{e}l_{ij} > (-0.5({}^{e}m_{ij} - {}^{e}l_{ij})) \\ {}^{e}l_{ij}, & \text{otherwise} \end{cases} \tag{30}$$

$${}^{e+1}r_{ij} = \begin{cases} {}^{e}r_{ij} - \Delta^{e}r_{ij}, & \text{if } \Delta^{e}r_{ij} < (0.5({}^{e}r_{ij} - {}^{e}m_{ij})) \\ {}^{e}r_{ij}, & \text{otherwise} \end{cases} \tag{31}$$

These restricting conditions arise from the fact that the parameters must satisfy the inequality ${}^{e}l_{ij} < {}^{e}m_{ij} < {}^{e}r_{ij}$. The updating of parameter m_{ij} is done according to

$${}^{e+1}m_{ij} = \begin{cases} {}^{e}m_{ij} - \dfrac{\Delta^{e}l_{ij} + \Delta^{e}r_{ij}}{2}, & \text{if } \Delta^{e}l_{ij}\Delta^{e}r_{ij} > 0 \\ {}^{e}m_{ij}, & \text{otherwise} \end{cases} \tag{32}$$

However, the same restriction applies, so if

$${}^{e+1}m_{ij} \leq {}^{e+1}l_{ij} \quad \text{or} \quad {}^{e+1}m_{ij} \geq {}^{e+1}r_{ij} \tag{33}$$

the update cannot be made, which means that ${}^{e+1}m_{ij} = {}^{e}m_{ij}$.

In the optimisation of the parameters, an adaptive step-length was used. Adaptiveness was put into effect by lengthening the step slightly by a factor of 1.05 when the total error had become smaller, and by drastically shortening the step by a factor of 0.5 when the total error had increased.

Symbols

$x(t)$	time series value at time t
$m(t)$, $r(t)$, $s(t)$	estimate of the mean, trend and seasonal deviation at time t
X_{init}	set of time series values used in initialization
$\hat{x}(t+h)$	prediction for h steps ahead at time t
α, β, γ	adaptation speed for mean, trend and seasonal deviation
$E(\theta(i))$	average prediction error with parameter set
$\theta(i)$	set of parameters $\{\alpha_i,\beta_i,\gamma_i\}$
x_i	value of ith input quantity
$\hat{o}$	output of the network
b_j	consequence of the jth rule
w_j	activation of the jth hidden neuron
τ_l, τ_r, σ_l, σ_r	parameters of a sigmoid-triangular membership function
$\hat{R}(x)$	triangular membership function
p_1, p_2, p_3	parameters of a triangular membership function
$\tilde{R}(x)$	sigmoid-triangular membership function
$f_l(x)$, $f_r(x)$	sigmoid function for left and right slopes, respectively
$R_j(\boldsymbol{x})$	activation of the jth hidden neuron when input vector is $\boldsymbol{x}$
$\boldsymbol{x}$	input vector
$\boldsymbol{c}_s$	sth cluster center
d_{si}	Euclidean distance between sth cluster centre and ith input vector
$\xi(i)$	index of the cluster centre to which ith input vector has been assigned
$\sqrt[p]{q}$	pth root of q
d_j	width of the membership function for jth input quantity
$\boldsymbol{C}_j$	jth best cluster centre which has been selected amongst the candidates
$\boldsymbol{r}_j$	vector of normalized hidden unit activations when input vector is $\boldsymbol{x}_j$
$\boldsymbol{W}$	matrix of activations of all hidden neurons for all input vectors
$\boldsymbol{b}$	vector of consequences
$\hat{\boldsymbol{o}}$	vector of outputs of the network
diag($\boldsymbol{a}$)	diagonal matrix with a_i:s in the diagonal
$\boldsymbol{a}$	column vector whose elements are a_i, $i = 1, \ldots, N$
$\mathbf{1}$	column vector whose each element is unity
$\boldsymbol{C}_{\text{near}}$	condition centre nearest to the input vector
$\boldsymbol{v}$	difference vector between $\boldsymbol{x}_{\text{test}}$ and $\boldsymbol{C}_{\text{near}}$
$\boldsymbol{x}_{\text{test}}$	input vector used in testing whether a new rule has to be generated

Special Symbols used in Appendix A

g_i	scalar valued coefficient
$\boldsymbol{r}_i^{\mathrm{T}}$	transpose of a vector $\boldsymbol{r}_i$
$\boldsymbol{h}_i$	column vector
α_{ij}	scalar coefficient
ψ_i	column vector

Special Symbols used in Appendix B

Δp_i	update of parameter p_i
l_{ij}	left parameter of a triangular membership function
r_{ij}	right parameter of a triangular membership function

m_{ij}	parameter of the tip of the triangular membership function
τ_{ij}, τ_{ij}, σ_{ij}, σ_{ij}	parameters of a sigmoid-triangular membership function
f_{ij}	sigmoid function
${}^{e}b_i$	parameter b value at epoch e (parameter can be also l, r or m)
$\Delta^{e}b_i$	parameter b update at epoch e (parameter can be also l or r)
${}^{e}\gamma$	step-length at epoch e