Available online at www.sciencedirect.com
ScienceDirect
Neurocomputing 70 (2006) 409–419
NEUROCOMPUTING
www.elsevier.com/locate/neucom

Short-term ANN load forecasting from limited data using generalization learning strategies

Zeke S.H. Chan[a,*], H.W. Ngan[b], A.B. Rad[b], A.K. David[b], N. Kasabov[a]

[a]*Knowledge Engineering and Discovery Research Institute, Auckland University of Technology, New Zealand*
[b]*Department of Electrical Engineering, Hong Kong Polytechnic University, Kowloon, Hong Kong*

Received 2 March 2005; received in revised form 6 October 2005; accepted 27 December 2005
Communicated by A. Zobaa
Available online 18 August 2006

Abstract

The emergence of the new competitive electricity market environment has made short-term load forecasting a more complex task, owing to the effect of marketers' behavior on the load pattern and the reduction of available information due to commercial reasons. In recent years, many ANN-based forecasters are proposed for learning the highly nonlinear load pattern, yet their effectiveness are limited by the reduction of training data, which causes these ANN models to be susceptible to "over-fitting". "Over-fitting" is a common ANN problem that describes the situation that the model memorizes the training data but fails to generalize well to new data.

This paper discusses the problem of "over-fitting" and some common generalization learning techniques in the ANN literature, as well as introducing a new Genetic Algorithm-based regularization method called "GARNET" for short-term load forecasting. As an illustration, four generalization learning techniques, including Early-Stopping, Bayesian Regularization, Adaptive-Regularization and GARNET are applied to train Multi-Layer Perceptrons networks (MLP) for day-ahead load forecasting on limited amount of hourly data from a US utility. Results show that forecasters trained by these four methods consistently produce lower prediction error than those trained by the standard error minimization method.

Keywords: Regularization; Day-ahead forecasting; Open electricity market; Genetic algorithm

1. Introduction

As many competitive markets emerge in recent years to encourage greater customer participation and higher efficiency in electricity generation, day-ahead load forecasting has become an important commercial function for the power utilities. For optimal bidding in these markets that are usually based on day-ahead auctions, the utilities require accurate day-ahead load forecasts [6,7]. However, the new role of forecasting has also raised the difficulties of the task, due to (a) the changes in the load pattern caused by the marketers' bidding behavior and the inclusion of new input variables such as spot prices, and (b) the reduction of the amount of training data available to the forecasting models. The later is due to the relatively short history of market existence, the frequent changes in market structure, e.g. the addition of new rules or competitors, and the non-disclosure of much information that is considered commercially sensitive and private.

To model the increasingly complex load pattern, many Artificial Neural Network (ANN)-based forecasters have been proposed in recent years. See Ref. [15] for a comprehensive review. ANN models are capable of learning nonlinear functional mapping and are suitable for adapting to increasingly complex load functions. In fact, ANN load forecasters have been adopted by at least 70 utilities in the US and NEMMCO[1] [18]. However, with limited amount of training data, these ANN forecasters are susceptible to "overfitting", which describes rigid memorization of the training data but failure to predict unseen

*Corresponding author. Tel.: +649 622 0705.
E-mail address: zekechan@vodafone.net.nz (Z.S.H. Chan).

[1]NEMMCO Operating Procedure: Load Forecasting. Document Number SO_OP3710. Date saved:21/3/00.

0925-2312/$ - see front matter © 2006 Published by Elsevier B.V.
doi:10.1016/j.neucom.2005.12.131

data. In such cases, improving the generalization performance of the forecasters becomes a critical issue. Ref. [9,26] report that generalization learning strategies significantly improve the performance of ANN load forecasters.

Meanwhile, generalization learning is not widely discussed in the load forecasting literature. Most of the existing literature on ANN short-term load forecasting does not mention employing generalization learning. A few, [11,15,26,32,33], use Early-Stopping or Bayesian Regularization. This work introduces some common, as well as more advanced generalization learning methods from main classes of methods: cross-validation, regularization and pruning [5]. Methods include Early-Stopping (ES), Bayesian Regularization (BR) by MacKay [23,24,26], Adaptive Regularization (AR) by Larsen et al. [21], Genetic Algorithm (GA) for Radial Basis Functions networks (RBF) by Chen [10], Monte-Carlo [30], Monte-Carlo Markov Chain [27–29] and Genetic Programming by Yao [34]. We will also describe a new method called GARNET (Genetic Algorithm for Regularization of Neural Network) that is designed for large-scale regularization of Multi-Layer Perceptrons networks (MLP) [8]. In the experiments, we apply ES, BR, ES and GARNET to train MLP day-ahead load forecasters on 1 yr of hourly-data obtained from a US utility to demonstrate the problem of day-ahead load forecasting on limited data. The results show that forecasters that employ generalization learning record significantly lower prediction error than forecasters that employ the standard Maximum Likelihood (ML) learning method.

The rest of this paper is organized as follows. Section 2 describes the problem of ANN over-fitting, the definition of generalization performance, and then reviews some generalization learning methods taken from the classes of cross validation, regularization and pruning. Section 3 introduces GARNET and its implementation. Section 4 shows the experimental results on training day-ahead load forecasters to verify the benefits of ANN generalization. The paper is concluded in Section 5.

2. Review of ANN over-fitting and generalization learning strategies

In this paper, we limit our study to three-layer MLP, which is the most widely-used class of ANNs for both regression and classification problems [5]. A typical MLP is shown in Fig. 1. Hyperbolic tangent and linear activation functions are assumed for the hidden and output layers, respectively. The network parameter vector **w** (the weight vector) is the concatenation of all vectorized weights and biases.

2.1. The problem of ANN over-fitting

The problem of over-fitting under limited training data can be illustrated with the following example. Two

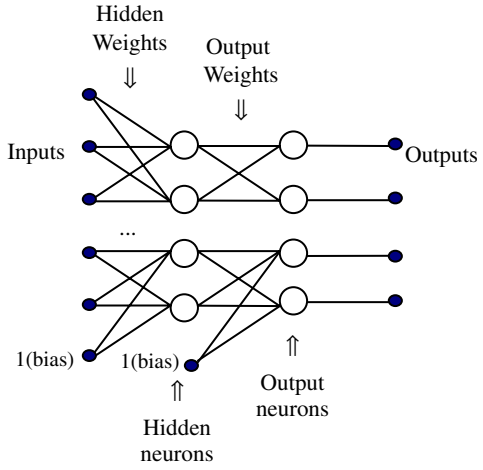

Fig. 1. A typical MLP.

identical three-layer MLPs with 1-5-1 configuration are trained on samples generated from a noisy sine function $f(x) = \sin(x) + \varepsilon$, $\varepsilon \sim N(0,0.5)$ where $N(\mu,\sigma)$ is a Gaussian distributed noise source with mean μ and standard deviation σ, and x is uniform sampled from the interval $[0,2\pi]$. The first MLP trains on 200 samples while the second MLP trains on 50 samples only. The training objective is the conventional error minimization. The task is to learn the true generating function $\sin(x)$. Fig. 2 shows the training data used by these MLPs and their prediction over the interval $[0,2\pi]$.

Results show that the model trained with 200 samples predicts closely to $\sin(x)$, whereas the one trained with 50 samples follows the exact representation of the corrupted training data, suffering the typical symptom of "over-fitting". Models with insufficient training data are susceptible to over-fitting either because there is a lack of inadequate statistical information for reconstructing the true generating function, or because the model is excessively complex, which causes it to memorize the exact representation of the training data. The latter can be remedied through finding the least complex model that adequately explains the data, so that rigid memorization is prevented. This principle of parsimony in explaining or modeling a phenomenon, known as the *Occam's Razor*, is central to the three classes of generalization learning methods of Cross-Validation (CV), Regularization and Pruning. CV provides a measure of model optimality over the training process that is often used for preventing model over-training; regularization adds a penalty function to the training objective so that model complexity and prediction error are minimized simultaneously; and pruning physically removes excessive neurons to produce the least size network. Note that to apply these three classes of methods, networks should be oversized rather than undersized to allow control over the learning process of model complexity reduction.

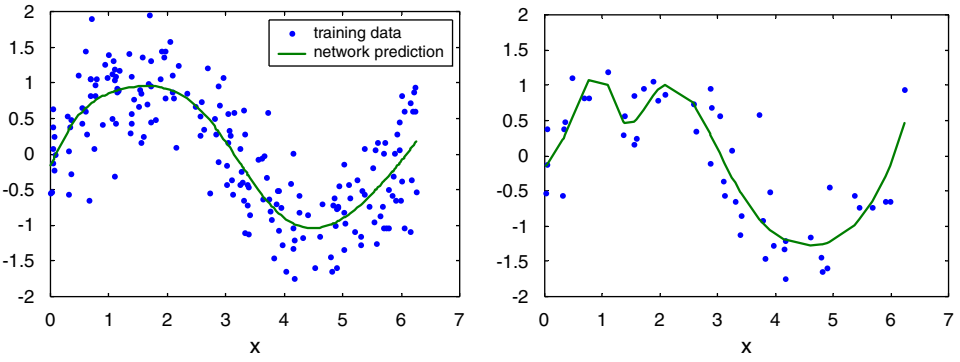

Fig. 2. Predictions and training data of ANNs that are trained with 200 samples (left) and 50 samples (right).

2.2. Cross-validation and early-stopping

CV provides continuous monitoring of the generalization performance of the model over the learning process. CV requires splitting the data into 2–3 non-overlapping subsets: the training set which is used for parameter learning, the validation set on which the prediction error is used to approximate the generalization error, and the optional test set on which the prediction error is used to approximate the unbiased generalization error after the learning is completed (note that the test set is not seen by the model over the learning process). The prediction error on the validation set, the so-called Cross-Validation error (CV error), provides an empirical indication on the optimality of the model. A commonly-used extension is the k-fold CV, which requires first splitting the data evenly into k non-overlapping sets, and then performing CV through all k sets using one set as the training set and the rest as the validation set. The average CV error is used to approximate the generalization error. Due to the ease of its implementation, CV is used in conjunction with many learning methods for monitoring the progress of model training. One such example is ES.

The goal of ES is to prevent the model from over-learning the data by terminating the training process at which the model complexity is optimal, which is indicated by the emergence of the lowest CV error. The following pseudo-code summarizes the implementation.

Step 1: Set $gen = 0$

Step 2: Optimize $\mathbf{w}^{gen} \rightarrow \mathbf{w}^{gen+1}$ for *n_inner* iterations on the training set.
$gen = gen + 1$

Step 3: Determine termination with CV:
IF the validation error of the new solution is smaller than that of the best solution found
THEN the new solution becomes the best solution
ELSE if no better solution has been found in the last n_{stop} trials, terminate
Go to *step 2* if $gen < gen_{\max}$, else terminate

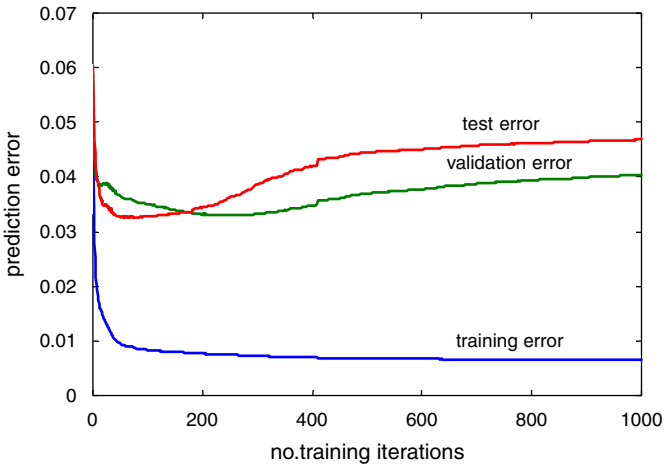

Fig. 3. A plot of validation error and training error versus the amount of training, taken from the ML training conducted in Section 4.

The problem-dependent control parameters n_{inner} and $gen_{\max}$ define the CV interval and the maximum training period, respectively. n_{stop} defines the length of the trial training period, which is used for robust training against noisy validation error. If the validation error is smooth and convex, the training algorithms can stop once validation error hits the minimum and starts rising; however, the validation error is usually noisy and multi-modal in practice and requires a trial training period n_{stop} for escaping local minima. A plot of the validation, test and training error versus the amount of training is taken from the experiments conducted in Section 4 and is shown in Fig. 3. Note that the validation error is noisy and is not strictly convex.

2.3. Regularization

Regularization training requires minimizing the network weight sizes as well as the prediction errors to preserve model parsimony over the training process. Given the network weights $\mathbf{w}$, actual input $\mathbf{x}$ and output $\mathbf{t}$, network prediction $y(.)$, N the no. outputs, P the no. samples, the

common error minimization function is given by

$$E_{\mathrm{D}}(\mathbf{w}) = \frac{1}{2}\sum_{p=1}^{P}\sum_{i=1}^{N}\left(t_i^{(p)} - y_i(\mathbf{x}^{(p)};\mathbf{w})\right)^2. \quad (1)$$

The minimization of (1) produces the Maximum Likelihood Estimates (MLE) of the model at the given training data t. As illustrated with the over-fitting example in Section 2.1, MLE is not always optimal because the model tends to memorize rather than generalize. Regularization requires inclusion of a weight penalty function $E_{\mathrm{W}}(\mathbf{w})$ in the training objective. Since weight sizes and model complexity are positively related [5,24], the penalty function helps suppressing model complexity. The penalty function can take many forms, but the most common "weight-decay" function is given by

$$E_{\mathrm{W}}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{W}\mathbf{w}_i^2. \quad (2)$$

The regularization objective is a weighted sum of $E_{\mathrm{D}}(\mathbf{w})$ and $E_{\mathrm{W}}(\mathbf{w})$

$$E = \beta E_{\mathrm{D}}(\mathbf{w}) + \alpha E_{\mathrm{W}}(\mathbf{w}) \quad (3)$$

The weighing coefficients $\{\beta,\alpha\}$, known as the *hyperparameters*, control the balance in the minimization of prediction error and weight magnitude. The generalization learning problem has now become one of finding the set of hyperparameters that leads to the optimal weights. Relative to the error minimization function, the regularization function is a better-defined training objective because the inclusion of the penalty function constrains the feasible solution space, therefore yielding more unique and better quality solutions.

2.4. Multiple hyperparameters

Since different classes of weights (e.g. weights functioning at different layers) and input variables have different relevance to the outputs, it is more accurate to assign them with separate hyperparameters. All experiments conducted in Section 4 assigns a separate hyperparameter to the class of hidden weights, hidden biases, output weights and output biases and to each input variable, which totals $H = (\text{no. inputs} + 4)$ hyperparameters. The use of separate hyperparameters for each input variable is equivalent to the Automatic Relevance Determination (ARD) technique used in Bayesian Regularization (discussed later) for suppressing irrelevant input variables. Given the hyperparameters $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots \alpha_H]$ and $\mathbf{w}_c$ the weights that belong to class c, the penalty function now becomes

$$\sum_{c=1}^{H}\alpha_c E_{\mathrm{W}}(\mathbf{w}_c) = \sum_{c=1}^{H}\left(\frac{\alpha_c}{2}\sum_{j\in c} w_j^2\right). \quad (4)$$

Putting together (3) and (4), the final regularization equation becomes

$$E = \beta E_{\mathrm{D}}(\mathbf{w}) + \sum_{c=1}^{H}\alpha_c E_{\mathrm{W}}(\mathbf{w}_c). \quad (5)$$

To find the hyperparameters $\{\beta,\boldsymbol{\alpha}\}$ and the corresponding weight set that minimize the cross-validation error, a robust search scheme is required because the hyperparameters are typical multi-modal and the cross-validation error is noisy. In the following, some common regularization methods that use (1) different hyperparameter optimization methods and (2) different optimization criteria are discussed. Their performance is problem-dependent. For notational simplicity, we combine all hyperparameters into the vector $\boldsymbol{\alpha} = [\beta, \alpha_1, \alpha_2, \ldots \alpha_H]$.

2.4.1. The Bayesian regularization method (BR)

The BR method, introduced by MacKay [5,23,24,26], optimizes the hyperparameters through maximizing the model "evidence", which describes the probability of a model given the data set under Bayesian theory. Since the evidence embodies the "*Occam's razor*" or favoritism towards the simplest model that adequately describes the data set, such maximization increases model parsimony and therefore generalization performance. The evidence maximization method uses the inverse weight Hessian for computing the local quadratic optimizer, therefore requiring a computational complexity of $O(W^3)$ where W is the number of network weights.

BR is the only generalization learning method listed in this paper that can be implemented without a validation set, because the maximization of evidence does not require evaluation of the CV error. However, in practice, CV is still often used to guarantee model optimality [26]. The procedure of BR is as follows.

Step 1: Set $gen = 0$.
Initialize $\mathbf{s}^0 = [\mathbf{w}{:}\boldsymbol{\alpha}]^0$
SCG-optimize $\mathbf{w}^0$ wrt $\boldsymbol{\alpha}^0$ for n_{init} iterations.

Step 2: Perform Evidence Maximization to update $\boldsymbol{\alpha}^{gen} \rightarrow \boldsymbol{\alpha}^{gen+1}$ (see Appendix A.1 for details)
SCG-optimize $\mathbf{w}^{gen} \rightarrow \mathbf{w}^{gen+1}$ wrt $\boldsymbol{\alpha}^{gen+1}$ for n_{inner} iterations
$gen = gen + 1$

Step 3: Determine termination with CV

Because evidence maximization works more effectively on already-converged models, MacKay suggested using large starting value of β [25] so that the initial models over-fit rather than under-fit. The control parameter n_{init} defines the amount of training given to the initial model.

2.4.2. The adaptive regularization method (AR)

The AR method, introduced by Larsen et al. [21], optimizes $\boldsymbol{\alpha}$ through iteratively minimizing the mean n_{w}-fold CV error. As n_{w}-fold CV error is used for approximating the generalization error, higher value of n_{w} is expected to increase the accuracy of the approximation and thus AR generalization performance. However, the computational complexity also increases proportionally as n_{w} inverse Hessians are required for computing the local quadratic minimizer. AR requires computational

complexity of $n_w \cdot O(W^3)$. The value of β is set to 1 to simplify calculations. The follow pseudo-code describes the procedures of AR.

Step 1: Set $gen = 0$
Initialize n_w sets of weights $\mathbf{W}_0 = \{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \ldots \mathbf{w}^{(n_w)}\}^0$ and $\boldsymbol{\alpha}^0$
SCG-optimize each set in $\mathbf{W}^0$ wrt $\boldsymbol{\alpha}^0$ for n_{init} iterations.

Step 2: Compute the gradient of n_w-fold CV errors with respect to the hyperparameters $\boldsymbol{\alpha}^{gen}$ (see Appendix A.2 or details).
Use gradient descent method to update $\boldsymbol{\alpha}^{gen} \rightarrow \boldsymbol{\alpha}^{gen+1}$ in the direction of the error gradient
SCG-optimize each set in $\mathbf{W}^{gen} \rightarrow \mathbf{W}^{gen+1}$ wrt $\boldsymbol{\alpha}^{gen+1}$ for n_{inner} iterations
$gen = gen + 1$

Step 3: Choose the best set in $\mathbf{W}^{gen}$ as $\mathbf{w}^{best}$
Determine termination with CV

2.4.3. The Monte-Carlo method

The Monte-Carlo method [30] is a computationally intensive heuristic search. n_s samples of $\boldsymbol{\alpha}$, typically spread out in logarithmic scale, are evaluated for n_w-fold cross-validation error. The optimal $\boldsymbol{\alpha}$ is then determined subjectively through graphing and studying the plot of validation error against $\boldsymbol{\alpha}$, or more objectively through computing the weighted average of $\boldsymbol{\alpha}$ wrt their frequency of scoring the minimum validation error. The Monte-Carlo approach is robust in approximating $\boldsymbol{\alpha}$, but when applied to optimize large number of hyperparameters, the required sample size n_s and computational intensity for building sufficient statistics can become forbiddingly large. Assuming the Scaled Conjugate Gradient method (SCG), a fast algorithm requiring complexity of only $O(W)$, spends an average of *niter* iterations for training the network, the Monte-Carlo method requires $(n_s \times n_w \times n_{iter})$ SCG-iterations.

2.4.4. The Markov chain Monte-Carlo method

The Markov Chain Monte-Carlo method (MCMC) is a simple but powerful stochastic simulation technique that has enabled the application of Bayesian inferences to many complex real-life problems in recent years [12,13]. It is first applied to ANN regularization by Neal [27–29]. MCMC requires first interpreting the ANN as a probabilistic model, and then sampling the posterior distribution of E. It is a Markov chain that uses a specifically designed transition kernel for determining the next move and the well-known Metropolis algorithm for determining the acceptance of the move. The hybrid Monte-Carlo algorithm by Neal [27] is one of the most efficient MCMC methods, which uses the stochastic dynamic method as the transition kernel. The major drawbacks are the high computational intensity required to simulate the Markov chain, and the lack of convergence diagnosis methods.

2.4.5. The genetic algorithm method

Since the 1990's, EC has emerged as a generic and robust global optimization tool for solving many complex engineering problems, of which GA [16] and GP [20] have been widely applied for optimizing ANN structure and weights, see [1–4,20] for a comprehensive review. The application of EC to regularization is however rare, and here we mention only Chen's GA for the RBF networks [10]. Chen introduces a two-level GA for regularizing RBFs. The GA population includes μ randomly initialized pairs of $\{\mathbf{w},\boldsymbol{\alpha}\}$. At the bottom level, individual $\mathbf{w}$'s are separately optimized using local optimizer and at the top level, the whole population of $\boldsymbol{\alpha}$'s are genetically operated and evolved simultaneously. Assuming GA evolves for gen_{max} generations, its computational requirement is $(\lambda \times n_{iter} \times gen_{max})$ SCG-iterations which could still be forbiddingly large for optimizing large number of hyperparameters. Chen's GA optimizes only two hyperparameters in his work and does not provide comparison with other regularization methods.

2.5. Pruning and structural evolution with evolutionary programming

Pruning physically removes redundant weights to reduce model complexity [5]. The choice of weight deletion is usually based on trial-and-error, minimization of the cross-validation error, or the information on the number of effective weights computed using BR [24]. In [34], the task is automated through EP [19], which is a branch of EC. Genetic operations include mutation that deletes or inserts connections and crossover that swaps tree-structures between ANNs. Yao's mutation operator [34] encourages model parsimony by preferring deletion to addition. The major difficulty with applying GP is the lack of behavioral link between parents and offspring (or the lack of continuity in the search space).

In general, we believe that a GA method for regularization would perform better than a GP method for pruning, mainly because the search of GA in the continuous space of the hyperparameters would be more effective and efficient than the search of GP in the discrete space of model architecture. This phenomenon can be illustrated by observing how each method eliminates the effect of redundant weights: regularization eliminates the effects of irrelevant weights or input variables by evolving small corresponding hyperparameters that suppress their effects, while EP eliminates them by physically deleting them from the architecture. The former implements a small and continuously change to the hyperparameters so that corresponding network weights only need to be partially updated, whereas the later changes the architecture and requires complete re-training of the network weights, which causes high computational intensity and noisy fitness function that could mislead the evolution. Similar comments on the advantage of 'soft pruning' with adjusting hyperparameter values over physical pruning are found in [9].

2.6. Discussions on the reviewed methods

Each of the reviewed methods has its own strengths and weaknesses. BR and AR are efficient, local optimization methods. They are however, easily trapped in local optima and misled by noise, since hyperparameter search space is highly multi-modal and noisy. The Monte-Carlo method requires manual interpretation and it is applicable to problems of low dimensionality only. The hybrid MCMC is a powerful stochastic simulation method. It incorporates the leap-frog algorithm that facilitates effective exploration into the search space and escape from local optima. It is however, computationally intensive because each leap-frog step requires evaluating both the objective function and its derivative, and each MCMC run typically requires a large number of leap-frog steps. In [29], Neal spent 1–6 million evaluations of leap-frog steps on sampling the posterior of a 3-layer MLP to predict MacKay's Robot arm data. The last method, Chen's GA, is robust to multi-modality and noises, but it is not computationally efficient and its application is limited to the regularization of small RBF network using only two hyperparameters. Its performance has not been compared with other standard neural network training methods. In comparison, the proposed method GARNET (detailed in the next section) will be tested with the regularization of large networks that contains 509 weights and 76 hyperparameters, and its performance will be compared against several ANN generalization learning methods.

3. GARNET: implementation and features

GARNET [8] combines the strengths of GA and a gradient-based method (in this case, the SCG method) in a two-level structure. At the top (wrapper) level, GARNET adopts the standard structure of GA. The search is performed with a population of individuals, each being a pair of solutions consisting of a hyperparameter set and the corresponding weight set. For each individual, we obtain the initial weight solutions using ES, and then sample the initial hyperparameter solutions from a probability distribution. This is for reducing training time in the initial phase. In each subsequent generation, GARNET searches the hyperparameter space by evaluating new hyperparameter solutions that are created using standard genetic operators like uniform crossover, mutation and selection for overcoming noises and multimodality. The corresponding new weight solutions are then passed to the second level, where they are re-trained using the SCG method on a small, fixed number of iterations from the last estimate, and *not* from a randomized network. Note that this is an important procedure of GARNET for (1) economizing training time and (2) reducing the noises that stem from the one-to-many mapping in the hyperparameter-to-weight space. As the majority of the genetically operated hyperparameters changes only by a small, continuous amount from the old estimate, and the old weight solution has already converged to a local optimum, re-training from the old weight solution generally requires much less time (we use 20 SCG iterations only) than re-training from a randomized network (economizing time), and tends to place the new weight solution at a nearby optimum (reducing one-to-many mapping). Finally, the fitness of each individual is measured by its performance in minimizing the CV error. The algorithm finishes when the maximum number of generations is reached. A schematic diagram of GARNET is shown in Fig. 4.

3.1. Procedures of GARNET

This section presents detail description of the GARNET flowchart shown in Fig. 4. Each individual is a pair of solutions consisting of a hyperparameter set and the corresponding weight set. The ith individual in the population is thus denoted by $\mathbf{s}(i) = \{\boldsymbol{\alpha}(i), \boldsymbol{w}(i)\}$. The parent and offspring population contain μ and λ individuals, respectively. The genetic operators consist of the standard uniform crossover and mutation. The selection method is (μ,λ). The guidelines for designing GA, choosing the genetic operators and setting their parameters roughly follow [1–4]. The following pseudo-code describes the purpose and implementation of each procedure.

(1) *Initialization:* Generate an initial population of μ parents: $\mathbf{s}(m)$, $m = [1, \ldots, \mu]$. For each parent, the weights are obtained by performing Early-Stopping from random weights. The hyperparameters are the absolute values of random samples taken from $N(0,\sigma^2_{\text{init}})$ (Normal distribution with mean zero and standard deviation σ_{init}).

(2) *Reproduction:* Reproduce λ offspring from the μ parents, $\mathbf{s}(k)$, $k = [1, \ldots, \lambda]$. For each offspring, two parents, $\mathbf{s}(i)$ and $\mathbf{s}(j)$ are randomly selected
   a. Encode the hyperparameters of the parents, $\boldsymbol{\alpha}(i)$ and $\boldsymbol{\alpha}(j)$, as binary strings $\mathbf{b}(i)$ and $\mathbf{b}(j)$ respectively, using either the standard base-two coding or Gray's coding.
   b. Genetically operate on the parent binary strings with uniform crossover and mutation to create one offspring binary string. The implementations are described as follows:
      Uniform crossover: For each bit position, swap the binary values of the parents strings $\mathbf{b}(i)$ and $\mathbf{b}(j)$ with crossover probability p_c. Uniform crossover creates two offspring, $\mathbf{b}_c(i)$ and $\mathbf{b}_c(j)$. To reduce statistical dependency within the offspring population, only one of the two offspring is used and the other one is discarded.
      Mutation: For each bit position, invert the binary value of the remaining offspring, say $\mathbf{b}_c(i)$, with mutation probability p_{m} to create $\mathbf{b}_{\text{m}}(i)$.

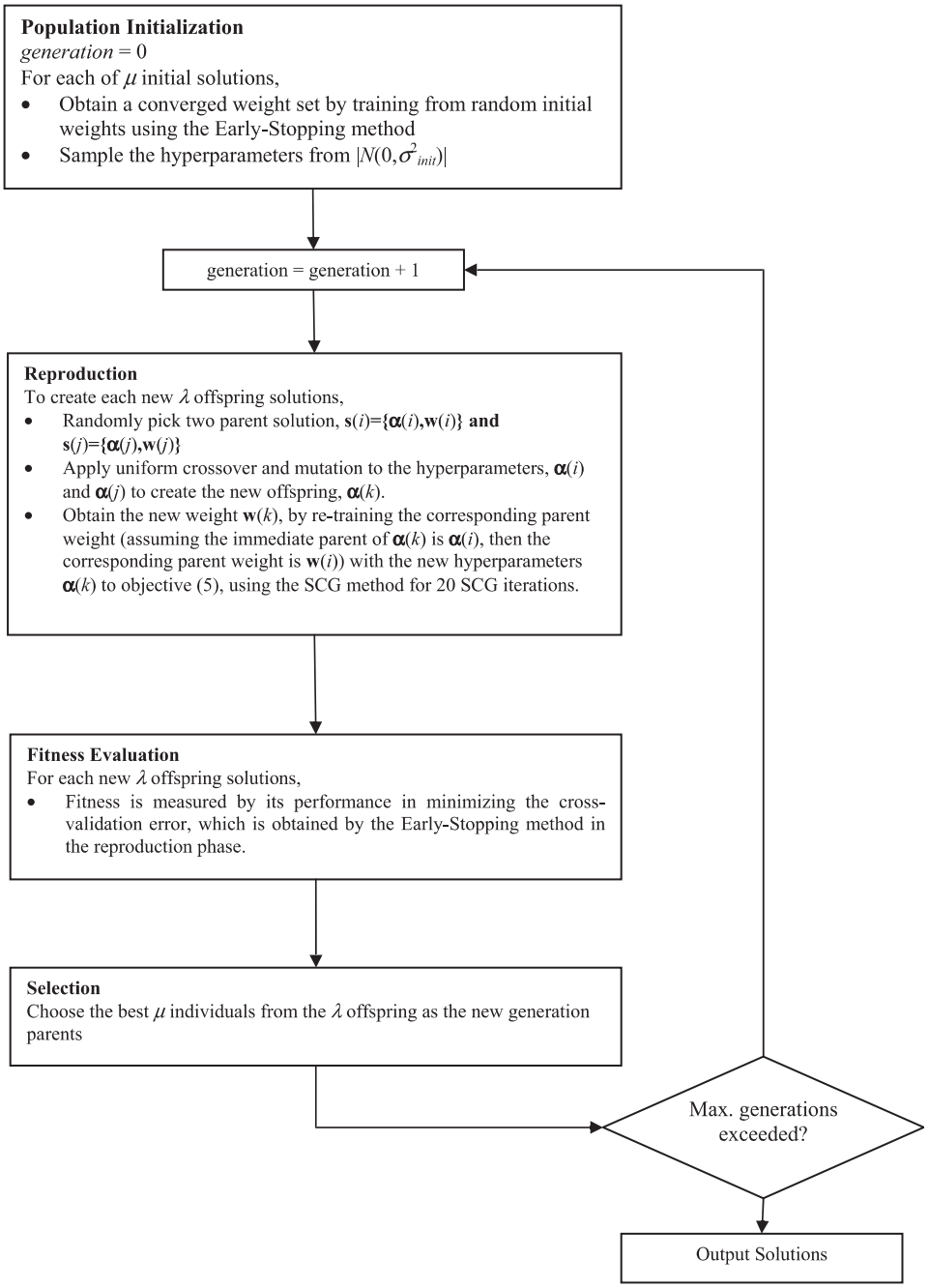

Fig. 4. GARNET flowchart.

The choice of the parameters p_c and p_m will be explained later. An illustration of uniform crossover and mutation is shown in Fig. 5.

c. Convert the resultant offspring binary string $\mathbf{b}_m(i)$ to real vector, which becomes the new hyperparameter solution, $\boldsymbol{\alpha}(k)$. It is important to identify the "immediate parent" of the surviving offspring, so that we can use the corresponding weight set as part of the new solution in the next stage (see the illustration in Fig. 5. Here we assume that the immediate parent of $\mathbf{b}_m(i)$ is $\boldsymbol{\alpha}(i)$ and so we will use the corresponding weight set $\mathbf{w}(i)$ in the next stage.

d. Using the new hyperparameter solution $\boldsymbol{\alpha}(k)$ in the regularization training objective **(5)**, optimize the corresponding parent weight set $\mathbf{w}(i)$ with the SCG method for 20 epochs to obtain the new weight set, $\mathbf{w}(k)$. Thus now, we have obtained a new pair of solutions, $\mathbf{s}(k) = \{\boldsymbol{\alpha}(k), \boldsymbol{w}(k)\}$.

(3) Measure the fitness of each of the λ offspring by its performance in minimizing the CV error.

(4) (μ,λ) *Selection.* Select μ individuals from the λ offspring to be the next generation parents.

(5) Go back to step (2) unless the maximum number of generation has been exceeded.

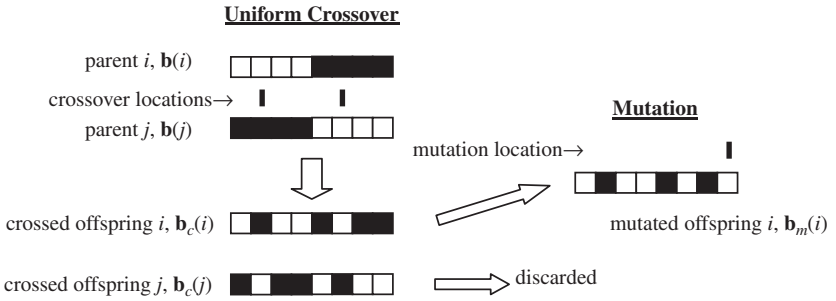

Fig. 5. Illustration of uniform crossover and mutation. Dark boxes represent one's and white boxes represent zero's. In this example, parent i, b_i, is "0000 1111" and parent j, b_j, is "1111 0000". The immediate parent of the mutated offspring $b_m(i)$ is parent i.

3.2. Features of GARNET

GARNET is designed with emphasis on (i) enhancing efficiency in performing the twofold optimization of the hyperparameters and weights, and on (ii) increasing robustness to the noises and multi-modality that are commonly found with the validation errors and hyper-parameter space, respectively. Efficiency is particularly important because GAs are intrinsically computation-intensive optimization algorithms. GARNET has the following features:

- For each candidate hyperparameter set, GARNET optimizes just one weight set in order to reduce computational requirement. This is unlike the Monte-Carlo method and AR, which optimize multiple weight sets per hyperparameter set.
- The initial weights are pre-trained with ES for reducing the amount of search required by GARNET in the initial phase.
- For re-training of weights with respect to the updated hyperparameters (procedure 2(d) above), GARNET re-trains from the current estimate of weights, like that of Adaptive Regularization. This is unlike Chen's GA and the Monte-Carlo method, which re-train the weights from random initial values. This design greatly reduces the training time. Empirically, we find that GARNET requires only 20 SCG-iterations to obtain required accuracy. Thus, the total computational requirement is $(\lambda \times n_{\text{iter}} \times gen_{\max})$ SCG-iterations for $n_{\text{iter}} = 20$. This is much lesser than that required by Chen's GA, which is $(\lambda \times n_{\text{iter}} \times gen_{\max})$ iterations where n_{iter} is typically in the range of $[10^2,10^4]$.
- For selection operator, GARNET uses the non-elitist method (μ,λ) instead of the fast, elitist method $(\mu+\lambda)$ for increasing the robustness to noise and multimodality. The reason is as follows. In noisy and multi-modal environment, there are many high-fitness spikes. As the $(\mu+\lambda)$ selection keeps the best strings observed in the population, these spikes may remain in the population and mislead the search for many generations. On the other hand, non-elitist selection selects only from the current offspring population and the effect of spikes will last only one generation. Therefore, the search is predominantly attracted towards the more global and the larger-volume basins. Fig. 6 shows the difference between the (μ,λ) and $(\mu+\lambda)$ selection method in search: the former leads the solutions towards the larger-volume and the more stable equilibrium, while the later leads the solutions towards the narrow high-fitness spike. Fig. 7 plots the fitness of the best solution in each generation of a typical GARNET run. Note that the fitness value does not follow a rapidly and monotonically decreasing trajectory that is typical of $(\mu+\lambda)$, but rather a gradual and rugged trajectory that is typical of (μ,λ).

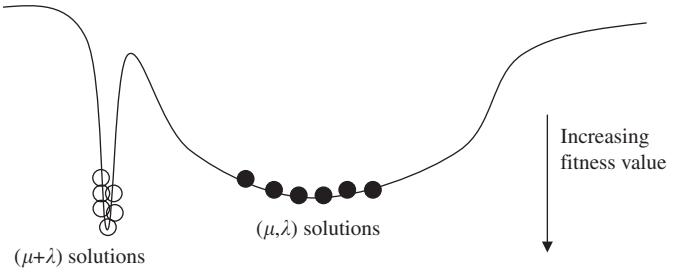

Fig. 6. Difference between the (μ,λ) and $(\mu+\lambda)$ selection method in the placement of solutions in a one-dimensional fitness landscape.

4. Load data and experimental results

To illustrate the advantages of generalization learning on ANN load forecasters under limited data, we compare the performance of MLP load forecasters that are trained using five different strategies on day-ahead, 24-h load forecasting using data from a US utility. The training data contains 1 yr of hourly load and climatic information. For illustration, we limit the experiment to weekday data only. Separate models are usually required for weekend and holiday data as they exhibit different load pattern. 150 input–output pairs of weekday data are extracted from February to September, which include the summer peak that occurs in August (winter peak occurs in December). The amount of training data is insufficient for some commercial load forecasters [14,17,18,22], but is

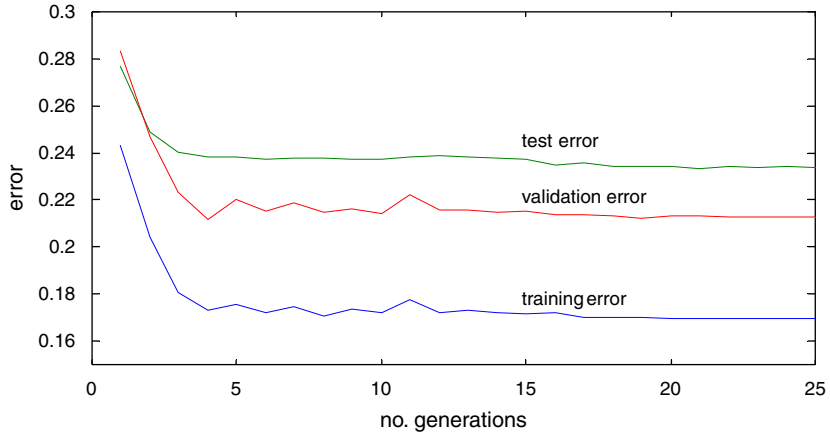

Fig. 7. Evolution of the prediction errors of the best solutions in a typical GARNET run.

appropriate here to illustrate training with limited data in a new market environment.

Inputs to the MLP comprise 24-h load and 24-h effective-temperature of previous day and 24-h weather forecast for the next day. These input vectors are denoted by $load(t)$, $temp_{\text{eff}}(t)$ and $temp_{\text{eff}}(t+1)$, respectively. Outputs of the model are the 24-h load forecasts for the next day, denoted by $load(t+1)$.

The number of hidden neurons is chosen as the minimum that provides sufficient model complexity, and it is empirical determined as the least number of hidden neurons that causes the model to overfit on the training data. In our case, five hidden neurons are found to be sufficient. In total, the model contains 509 weights and 76 hyperparameters. Such network topology is similar to that in [18].

Hourly weather data is readily available from the US Weather Bureau. It includes temperature, relative humidity and wind speed. These three components are combined into a single "effective-temperature" ($temp_{\text{eff}}$) for input. There are many methods for computing the effective-temperature, for example [18,31]. We use the former approach ([18], Eq. (6)), as it provides satisfactory results for our data set. The scalar quantity of ψ and ζ are empirically determined.

$$temp_{\text{eff}} = \begin{cases} temp + \psi \cdot Humidity, & temp > 75^{\circ}, \\ temp, & 65^{\circ} \leqslant temp \leqslant 75^{\circ}, \\ temp - \zeta \cdot Wind \cdot (65^{\circ} - temp)/100, & T < 65^{\circ}. \end{cases} \tag{6}$$

Prediction error is expressed in Mean Absolute Prediction Error (MAPE), a standard for load forecasting error.

The five training strategies applied are: ML , ES, AR, BR and GARNET. ML is the standard error-minimization training method and it is used here mainly to illustrate the symptoms of over-fitting. It applies SCG to train the model until the prediction error on the training set falls below a threshold value, or until the maximum iteration is exceeded. In this case, the threshold value and the maximum iteration are set to 0.5% MAPE and 20,000, respectively (empirically we find that over-fitting would have occurred in either case). The experimental settings for ES, AR, BR and GARNET are listed in the appendix. GARNET uses $gen_{\max} = 200$, $\mu = 10$, $\lambda = 20$, $p_c = 0.7$, $p_{\text{m}} = 1/n_{\text{bits}}$, $\sigma_{\text{init}} = 0.01$. The settings for μ, λ, p_c and p_{m} are quite common in EC literature; the rest are specific to GARNET operation and are empirically verified. The experimental settings of ES, BR and AR are listed in Table 1.

Table 1
Settings of ES, BR and AR

	ES	BR	AR
$gen_{\max}$	400	200	400
n_{init}	/	500	500
n_{inner}	20	20	20
n_{stop}	20	10	10

The entire training data is randomly split into three non-overlapping sets of training set, validation set and test set for cross validation. Each of the five training methods is applied to train the MLP forecasters from the same 10 sets of randomly initialized weight. Forecasting performance is measured by the average prediction error on the validation set and the test set, which are the seen and unseen approximation of generalization error. Results and the training time are averaged over the 10 sets and are tabulated in Table 2. The training methods are listed in the order of descending validation error.

Table 2 shows that although ML scores the lowest in the training error (0.58%), it scores the highest in the validation error (5.68%) and test error (5.79%), which indicates that the model over-fits the training set but generalizes badly to new data (this result also confirms that a five-hidden-neurons MLP is sufficiently complex for the task). A typical ML training was shown earlier in Fig. 3, demonstrating symptoms of over-fitting that the validation

Table 2
Mean training error, validation error and test error of ML, AR, ES, BR and GARNET

	Time (s)	Train.err (%)	Valid.err (%)	Test.err (%)
ML	957	0.58	5.68	5.79
AR	380	0.88	4.80	4.83
ES	46	1.95	3.95	4.30
BR	708	1.94	3.80	4.10
GARNET	2045	1.32	3.19	3.48

and test error increases while the training error decreases over the learning process. On the other hand, the four generalization learning methods scores relatively high in the training error but low in the generalization error. GARNET, the best performer, scores 1.32% in the training error, but only 3.19% and 3.48% in the validation error and test error respectively. Generalization learning clearly improves the performance of MLP forecasters in predicting unseen data. In terms of training time, ML spends 957 s, more than all except GARNET. ES spends 46 s only, the least of all. When cost-effectiveness is critical, we recommend ES, which is a fast and effective generalization learning method that requires little additional programming effort to ML.

5. Conclusions

Short-term load forecasting is given new significance as well as complexity due to the emergence of the new competitive electricity market environment. ANN-based forecasters that are proposed to cope with the additional non-linearities in the load pattern are also susceptible to the problem of "over-fitting" caused by the shortage of training data. This work illustrates the problem of "over-fitting", and introduces some common generalization learning methods for remedy, which include the classes of CV, Regularization and Pruning. Experiments include applying ES, AR, BR and a new GA-based regularization method called GARNET for training MLPs for short-term load forecasting using 1 yr of data from a US utility. Results show that forecasters trained by these four methods consistently produce lower prediction error than that trained by the standard error minimization method, verifying the importance of generalization learning when training data is limited.

Acknowledgement

The first author acknowledges the financial support (account S728) given to this project by the Research Committee of the Hong Kong Polytechnic University and the KEDRI postdoctoral fellow research grant.

Appendix A

A.1. Evidence maximization

Evidence maximization requires intrepreting the MLP as a probabilistic model and taking Gaussian approximation on the posterior distribution. In which case, the evidence at $\mathbf{w}_{\mathrm{MP}}$ is maximized by setting $\alpha = \alpha_{\mathrm{MP}}$ and $\beta = \beta_{\mathrm{MP}}$ where

$$\gamma = k - \alpha \mathrm{Trace}(\mathbf{A}^{-1}), \tag{7}$$

$$1/\alpha_{\mathrm{MP}} = \sum_i \mathbf{w}_i^{\mathrm{MP}^2}/\gamma, \tag{8}$$

$$1/\beta_{\mathrm{MP}} = 2E_{\mathrm{D}}/(N_{\mathrm{f}} - \gamma), \tag{9}$$

where $\mathbf{A}$ is the Hessian of the regularization objective E (3) at $\mathbf{w}_{\mathrm{MP}}$ (see [5] for a discussion of methods for Hessian computation), k is the total number of parameters and N_{f} is the degree of freedom given by (no. outputs $N \times$ no. training samples P). Note that in probabilistic terms, β and α actually represent the inverse variance of the weights and prediction error, respectively. Often different weight classes have different variances, so it is inaccurate to assign them with equal values of α. In these cases, multiple α_c can be applied so that (8) becomes

$$1/\alpha_c^{\mathrm{MP}} = \sum_{i \in c} w_i^{\mathrm{MP}^2}/\gamma_c, \tag{10}$$

where

$$\gamma_c = k_c - \alpha_c \mathrm{Trace}_c(\mathbf{A}^{-1}), \tag{11}$$

where k_c denotes the number of weights in class c.

A.2. The gradient of the cross-validation error with respect to the hyperparameters

Let $\boldsymbol{\nu}^{(n)}$ denote the prediction error on the nth validation set in an n_{w}-fold CV. Its gradient with respect to the hyperparameter $\boldsymbol{\alpha}_c$ is given by

$$\frac{\partial v^{(n)}(\mathbf{w}_c)}{\partial \alpha_c} = -\frac{\partial \alpha_c E_{\mathrm{W}}(\mathbf{w}_c)}{\partial \mathbf{w}_c^{\mathrm{T}}} \mathbf{A}^{-1}(\mathbf{w}_c) \frac{\partial v^{(n)}(\mathbf{w}_c)}{\partial \mathbf{w}_c}, \tag{12}$$

where $A(\mathbf{w}_c)$ is the Hessian of the cth weight class $\mathbf{w}_c$ of the regularization training objective. The gradient of the overall cross-validation error, denoted by $\delta \boldsymbol{\nu}^{(\mathrm{cv})}/\delta \boldsymbol{\alpha}_c$, is simply the average of the n_{w}-fold gradients

$$\frac{\partial v^{(cv)}}{\partial \alpha_c} = \frac{1}{nw} \sum_{n=1}^{n_{\mathrm{w}}} \frac{\partial v^{(n)}}{\partial \alpha_c}. \tag{13}$$

References

[1] T. Baeck, Evolutionary Algorithm in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford University Press, New York, 1995.
[2] T. Baeck, U. Hammel, H.-P. Schwefel, Evolutionary computation: comments on the history and current state, IEEE Trans. Evol. Comput. 1 (1997) 3–17.

[3] T. Baeck, D.B. Fogel, Z. Michalewicz, Evolutionary Computation I. Basic Algorithm and Operators, vol. 1, Institute of Physics Publishing, Bristol, 2000.
[4] T. Baeck, D.B. Fogel, Z. Michalewicz, Evolutionary Computataion II. Advanced Algorithm and Operators, 2, Institute of Physics Pub., Bristol, 2000.
[5] C. Bishop, Neural Networks for Pattern Recognition, Oxford University Press Inc., 1995.
[6] D.W. Bunn, Forecasting loads and prices in competitive power markets, Proc. IEEE 88 (2000) 163–169.
[7] D.W. Bunn, E.D. Farmer, Economic and operational context of electric load prediction, in: D.W. Bunn, E.D. Farmer (Eds.), Comparative Models for Electrical Load Forecasting, Wiley, Newyork, 1985, pp. 3–12.
[8] Z.S.H. Chan, H.W. Ngan, A.B. Rad, T.K. Ho, Alleviating "overfitting" via genetically-regularised neural network, Electronics Letter, 2002.
[9] W. Charytoniuk, M.-S. Chen, Very short-term load forecasting using artificial neural networks, IEEE Trans. Power Syst. 15 (2000) 263–268.
[10] S. Chen, Combined genetic algorithm optimization and regularized orthogonal least squares learning for radial basis function networks, IEEE Trans. Neural Networks 10 (1999) 1239–1243.
[11] E. Doveh, P. Feigin, D. Greig, L. Hyams, Experience with FNN models for medium term power demand predictions, IEEE Trans. Power Syst. 14 (1999) 538–546.
[12] D. Gamerman, Markov Chain Monte-Carlo:Stochastic simulation for Bayesian inference, Chapman & Hall, London, 1997.
[13] W.R. Gilks, S. Richardson, D.J. Spiegelhalter, Markov Chain Monte-Carlo in Practice, Chapman & Hall, London, 1996.
[14] W.M. Grady, L.A. Groce, T.M. Huebner, Q.C. Lu, M.M. Crawford, Enhancement, implementation and performance of an adaptive short-term load forecasting algorithm, IEEE Trans. Power Systems 6 (1991) 1404–1410.
[15] H.S. Hippert, C.E. Pedreira, R.C. Souza, Neural networks for short-term load forecasting: a review and evaluation, IEEE Trans. Power Systems 16 (2001) 44–55.
[16] J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, MI, 1975.
[17] A. Khotanzad, R. Afkhami-Rohani, L. Tsun-Liang, A. Abaye, M. Davis, D.J. Maratukulam, ANNSTLF-a neural-network-based electric load forecasting system, IEEE Trans. Neural Networks 84 (1997) 835–846.
[18] A. Khotanzad, R. Afkhami-Rohani, D.J. Maratukulam, ANNSTLF-artificial neural network short-term load forecaster-generation three, IEEE Trans. Power Systems 134 (1998) 1413–1422.
[19] S. Kotz, N. Balakrishnan, N.L. Johnson, Continuous Multivariate Distributions, second ed, Wiley, Newyork, 2000.
[20] J.R. Koza, Genetic Programming, MIT Press, Cambridge, MA, 1992.
[21] J. Larsen, C. Svarer, L.N. Andersen, L.K. Hansen, Adaptive regularization in neural network modeling, in: G.B. Orr, K.-R. Muller (Eds.), Neural Networks: Tricks of the Trade, Springer, Berlin, 1998, pp. 113–132.
[22] Q.C. Lu, W.M. Grady, M.M. Crawford, An adaptive algorithm for short-term multinode load forecasting in power systems, IEEE Trans. Circuits Syst. 35 (1988) 1004–1010.
[23] D.J.C. MacKay, A practical Bayesian framework for backprop networks, in: J.E. Moody, S.J. Hanson, R.P. Lippmann, (Eds.), Adv. Neural Inform. Process. Syst., 4 (1992) 839–846.
[24] D.J.C. MacKay, Probable networks and plausible predictions - a review of practical Bayesian methods for supervised neural networks, Comput Neural Syst. 6 (1995) 469–505.
[25] D.J.C. MacKay, Bayesian methods for neural networks - FAQ, vol. 2001, 2000.
[26] D.J.C. MacKay, Bayesian Methods for Neural Networks: Theory and Applications, Cavendish Laboratory, Cambridge Neural Networks Summer School lecture notes, 2001.
[27] R.M. Neal, Bayesian Training of Backpropagation Networks by the Hybrid Monte-Carlo Method, Connectionist Research Group, Department of Computer Science, University of Toronto CRG-TR-92-1, 10 April 1992.
[28] R.M. Neal, Probabilistic Inference Using Markov Chain Monte-Carlo Methods, Department of Computer Science, University of Toronto CRG-TR-93-1, 25 September 1993.
[29] R.M. Neal, Bayesian Learning for Neural Networks; Springer, New York, 1996.
[30] T. Rognvaldsson, A Simple Trick for Estimating the Weight Decay Parameter, in: G.B. Orr, K.-R. Muller (Eds.), Neural Networks: Tricks of the Trade, Springer, Berlin, 1998, pp. 71–92.
[31] A.M. Schneider, T. Takenawa, D.A. Schiffman, 24-hour electric utility load forecasting, in: D.W. Bunn, E.D. Farmer (Eds.), Comparative Models for Electrical Load Forecasting, Wiley, New York, 1985, pp. 87–108.
[32] K. Siwek, S. Osowski, Regularization of neural networks for improved load forecasting in power system, presented at ICECS 2001, 2001.
[33] E.H. Tito, G. Zaverucha, M. Vellasco, M. Pacheco, Bayesian neural networks for electric load forecasting, presented at International Conference on Neural Information Processing, 1999.
[34] X. Yao, Y. Liu, A new evolutionary system for evolving artificial neural networks, IEEE Trans. Neural Networks 8 (1997) 694–713.

Dr. Zeke S.H. Chan's research interest lies in the area of Bayesian inference and computation intelligence; in particular, Markov chain Monte Carlo, evolutionary computation, Bayesian parameter estimation and prediction for neural networks in forecasting, control and bioinformatics applications.