Extreme Learning Machine and its Variants for Time-Varying Neural Networks

Author: Yibin Ye; Tutor: Francesco Piazza

Department of Information Engineering
Università Politecnica delle Marche
Extended Summary

Abstract. System identification in nonstationary environment represents a challenging problem and an advaned neural architecture namely Time-Varying Neural Networks (TV-NN) has shown remarkable identification properties in nonlinear and nonstationary conditions. Time-varying weights, each being a linear combination of a certain set of basis functions, are used in such kind of networks instead of stable ones, which inevitalbly increases the number of free parameters. Therefore, an Extreme Learning Machine (ELM) approach is developed to accelerate the training procedure for TV-NN. What is more, in order to obtain a more compact structure, or determine several important parameters, or update the network more efficiently in online case, several variants of ELM-TV are proposed and discussed in the works. Related computer simulations have been carried out and show the effectiveness of the algorithms.

1 Introduction

Extreme Learning Machine (ELM) is a fast learning algorithm designed for single hidden layer feedforward neural networks (SLFNs) [1], first introduced by Huang et al. In ELM, the input weights of SLFNs do not need to be tuned and can be randomly generated, whereas the output weights are analytically determined using the least-square method, thus allowing a significant training time reduction. In recent years, ELM has been raising much attention and interest among scientific community, in both theoretic study and applications: A hybrid algorithm called Evolutionary Extreme Learning Machine (E-ELM) [2] uses the differential evolutionary algorithm to select the input weights. Incremental-based Extreme Learning Machine algorithms (I-ELM [3], EI-ELM [4] and EM-ELM [5]) can randomly add nodes to the hidden layer. A real-coded genetic algorithm approach called RCGA-ELM has been also proposed to select the optimal number of hidden nodes, input weights and bias values for better performance in multi-category sparse data classification problems [6]. B-ELM is able to optimize the weights of the output layer based on a Bayesian linear regression [7]. The Optimally Pruned Extreme Learning Machine (OP-ELM) extends the original ELM algorithm and wraps it within a methodology using a pruning of the neurons [8]. In CFWNN-ELM, composite functions are applied at the hidden nodes and learning is accomplished using ELM in a wavelet neural network [9]. Encoding

a priori information to obtain better generalization performance for function approximation is another possible way to improve ELM [10]. Radial-basis type neural networks [11, 12] also represent a fertile field where ELM paradigm has found application, as confirmed by some recent contributions[13, 1]. Among the various extensions, the online sequential learning algorithm based on ELM (OS-ELM) need to be cited, according to which learning is performed by using data one-by-one or chunk-by-chunk with fixed or varying chunk size [14]. The parameters of hidden nodes are randomly selected just as ELM, but the output weights are analytically updated based on the sequentially arriving data rather than the whole data set. Besides theoretical studies, some applications such as system identification and real-time data assessment have been considered to evaluate the effectiveness of ELM [15, 16].

Time-Varying Neural Network (TV-NN) represents a relevant example in neural architectures working properly in nonstationary environments. Such networks implement time-varying weights, each being a linear combination of a certain set of basis functions, whose independent variable is time. The candidate orthogonal basis function types employed in time-varying networks include Legendre polynomials, Chebyshev polynomials, Fourier sinusoidal functions, Prolate spheroidal functions, and more. An extended Back-Propagation algorithm has been first advanced to train these networks (BP-TV) [17]. Later on, an Extreme Learning Machine approach is also developed for TV-NN, accelerating the training procedure significantly (ELM-TV) [18].

2 Extreme Learning Machine

Let's assume that an SLFN with I input neurons, K hidden neurons, L output neurons and activation function $g(\cdot)$ is trained to learn N distinct samples $(\mathbf{X}, \mathbf{T})$, where $\mathbf{X} = \{x_i[n]\} \in \mathbb{R}^{N \times I}$ and $\mathbf{T} = \{t_l[n]\} \in \mathbb{R}^{N \times L}$ are the input matrix and target matrix respectively, $x_i[n]$ denotes the input data in i-th input neuron at n-th time instant, and $t_l[n]$ denotes the desired output in l-th output neuron at n-th time instant. In ELM, the input weights $\{w_{ik}\}$ and hidden biases $\{b_k\}$ are randomly generated, where w_{ik} is the weight connecting i input neuron to k-th hidden neuron, and b_k is the bias of k-th hidden neuron. Further let $w_{0k} = b_k$ and $x_0[n] = 1$. Hence, the hidden-layer output matrix $\mathbf{H} = \{h_k[n]\} \in \mathbb{R}^{N \times K}$ can be obtained by:

$$h_k[n] = g\left(\sum_{i=0}^{I} x_i[n] \cdot w_{ik}\right) \tag{1}$$

Let $\beta = \{\beta_{kl}\} \in \mathbb{R}^{K \times L}$ be the matrix of output weights, where β_{kl} denotes the weight connection between k-th hidden neuron and l-th output neuron; and $\mathbf{Y} = \{y_l[n]\} \in \mathbb{R}^{N \times L}$ be the matrix of network output data, with $y_l[n]$ the output data in l-th output neuron at n-th time instant. Therefore, this equation

can be obtained for the linear output neurons:

$$y_l[n] = \sum_{k=1}^{K} h_k[n] \cdot \beta_{kl} \tag{2}$$

$$\text{or} \qquad \mathbf{Y} = \mathbf{H} \cdot \beta \tag{3}$$

Thus, given the hidden-layer output matrix $\mathbf{H}$ and the targets matrix $\mathbf{T}$, to minimize $\|\mathbf{Y} - \mathbf{T}\|_2$, the output weights can be calculated by the minimum norm least-square(LS) solution of the linear system:

$$\hat{\beta} = \mathbf{H}^{\dagger} \cdot \mathbf{T}, \tag{4}$$

where $\mathbf{H}^{\dagger}$ is the MP generalized inverse of matrix $\mathbf{H}$. By computing output weights analytically, ELM achieve good generalization performance with speedy training phase.

3 ELM for Time-varying Neural Networks

The time-varying version of Extreme Learning Machine has been studied in [18]. In a time-varying neural network as shown in Fig. 1, the input weights, or output weights, or both are changing with time (both in training and testing phases). The generic weight $w[n]$ can be expressed as a linear combination of a certain set of basis function [19, 17]:

$$w[n] = \sum_{b=1}^{B} f_b[n] w_b \tag{5}$$

in which $f_b[n]$ is the known orthogonal function at n-th time instant of b-th order, w_b is the b-th order coefficient of the basis function to construct time-varying

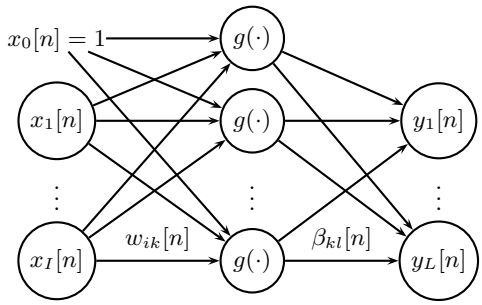

Fig. 1. Architecture of Time-Varying Neural Networks

weight w_n, while B (preset by user) is the total number of the bases.

If time-varying input weights are introduced in a SLFN, the hidden neuron output function can be written as:

$$h_k[n] = g\left(\sum_{i=0}^{I} x_i[n] \cdot w_{ik}[n]\right) \tag{6}$$

where $w_{ik}[n] = \sum_{b=1}^{B} f_b[n] w_{b,ik}$. Similarly, if time-varying output weights are introduced, the standard output equation can be written as:

$$y_l[n] = \sum_{k=1}^{K} h_k[n] \beta_{kl}[n] \tag{7}$$

where $\beta_{kl}[n] = \sum_{b=1}^{B} f_b[n] \beta_{b,kl}$.

To train a TV-NN, input weights $\{w_{b,ik}\}$ are randomly generated and hidden-layer output matrix $\mathbf{H}$ is computed according to equation (6). However, the values of a set of output weight parameters $\{\beta_{b,kl}\}$ can not be so straightforward calculated. Some transformations are needed. Expanding the time-varying output weights $\beta_{kl}[n]$ in (7) and assuming:

$$\begin{aligned}
\mathbf{f}[n] &= [f_1[n], f_2[n], \ldots, f_B[n]]^T \in \mathbb{R}^B, \\
\mathbf{h}[n] &= [h_1[n], h_2[n], \ldots, h_K[n]]^T \in \mathbb{R}^K, \\
\boldsymbol{\beta}_{kl} &= [\beta_{1,kl}, \beta_{2,kl}, \ldots, \beta_{B,kl}]^T \in \mathbb{R}^B, \\
\boldsymbol{\beta}_{(l)} &= [\boldsymbol{\beta}_{1l}, \boldsymbol{\beta}_{2l}, \ldots, \boldsymbol{\beta}_{Kl}] \in \mathbb{R}^{B \times K}, \\
\boldsymbol{\omega}_{(l)} &= [\boldsymbol{\beta}_{1l}^T, \boldsymbol{\beta}_{2l}^T, \ldots, \boldsymbol{\beta}_{Kl}^T] \in \mathbb{R}^{B \cdot K \times 1},
\end{aligned}$$

the following hold:

$$\begin{aligned}
y_l[n] &= \sum_{k=1}^{K} h_k[n] \cdot \left(\sum_{b=1}^{B} f_b[n] \cdot \beta_{b,kl}\right) \\
&= \sum_{k=1}^{K} \mathbf{f}[n]^T \cdot \boldsymbol{\beta}_{kl} \cdot h_k[n] \\
&= \mathbf{f}[n]^T \cdot \boldsymbol{\beta}_{(l)} \cdot \mathbf{h}[n] \\
&= \left(\mathbf{h}[n]^T \otimes \mathbf{f}[n]^T\right) \cdot \boldsymbol{\omega}_{(l)}
\end{aligned} \tag{8}$$

where $\otimes$ denotes the *Kronecker product* of $\mathbf{h}[n]^T$ and $\mathbf{f}[n]^T$. The last step consists in: $vec(\mathbf{AXB}) = (\mathbf{B}^T \otimes \mathbf{A}) vec(\mathbf{X})$ [20], (note that $vec(y_l[n]) = y_l[n]$) and $\boldsymbol{\omega}_{(l)} = vec(\boldsymbol{\beta}_{(l)})$ is the vectorization of the matrix $\boldsymbol{\beta}_{(l)}$ formed by stacking the columns of $\boldsymbol{\beta}_{(l)}$ into a single column vector. Moreover, let's define:

$$\mathbf{G} = \mathbf{H} * \mathbf{F} = \begin{bmatrix} \mathbf{h}[1]^T \otimes \mathbf{f}[1]^T \\ \cdots \\ \mathbf{h}[N]^T \otimes \mathbf{f}[N]^T \end{bmatrix}_{N \times B \cdot K} \tag{9}$$

where $\mathbf{H} = [\mathbf{h}[1], \mathbf{h}[2], \ldots, \mathbf{h}[N]]^T \in \mathbb{R}^{N \times K}$, $\mathbf{F} = [\mathbf{f}[1], \mathbf{f}[2], \ldots, \mathbf{f}[N]]^T \in \mathbb{R}^{N \times B}$, $*$ denotes the *Khatri-Rao product* of matrices $\mathbf{H}$ and $\mathbf{F}$, with $\mathbf{h}[n]^T$ and $\mathbf{f}[n]^T$ as their submatrices, respectively. Further assuming that $\mathbf{Y} = \{y_l[n]\} \in \mathbb{R}^{N \times L}$, $\mathbf{T} = \{t_l[n]\} \in \mathbb{R}^{N \times L}$, $\mathbf{\Omega} = [\boldsymbol{\omega}_{(1)}, \boldsymbol{\omega}_{(2)}, \ldots, \boldsymbol{\omega}_{(L)}] \in \mathbb{R}^{B \cdot K \times L}$, the following holds:

$$\mathbf{G} \cdot \mathbf{\Omega} = \mathbf{Y} \tag{10}$$

Since $\mathbf{F}$ is obtained by the type of the basis function predetermined by the user and $\mathbf{H}$ can be calculated by (6) once input weight parameters are randomly generated, hence $\mathbf{G}$ can be computed. Similarly to the ELM algorithm described in previous section, the time-variant output weight matrix $\mathbf{\Omega}$ can be computed by:

$$\hat{\mathbf{\Omega}} = \mathbf{G}^{\dagger} \cdot \mathbf{T} \tag{11}$$

where $\mathbf{G}^{\dagger}$ is the MP inverse of matrix $\mathbf{G}$, and consequently, $\hat{\mathbf{\Omega}}$ is a set of optimal output weight parameters minimizing the training error.

4 Group Selection Evolutionary ELM for TV-NNs

Similar to the standard ELM algorithm, the training of TV-NNs based on ELM requires relatively large amount of hidden neurons than that based on BP [18], due to the random generation of input weight parameters set. Moreover, when dealing with TV-NN we need to decide which type of the basis function has to be applied to the model. If the optimal input weight parameters set and the right type of basis function are selected, one may obtain good generalized performance with less hidden nodes (K) and less orders (B) of basis function, hence a compact network and faster response in the test phase. In this section, an algorithm named GSE-ELM-TV combined GS approach, DE algorithm and MP generalized inverse is proposed. We assume that TV-NN is trained by the FTV-ELM-NN: the proposed approach can be easily extended to the OTV-ELM-NN and ITV-ELM-NN case studies.

Firstly, we randomly generate the population, in which each individual consists of a set of input weight and bias parameters:

$$\begin{aligned}\boldsymbol{\theta} = &[w_{1,01}, w_{1,02}, \ldots, w_{1,0K}, w_{1,11}, w_{1,12}, \ldots, \\ &w_{1,1K}, \ldots, w_{1,IK}, \ldots, w_{2,IK}, \ldots, w_{B,IK}]\end{aligned}$$

Here, we consider the biases $\{b_{bk}\}$ as special weights $\{w_{b,0k}\}$. All $w_{b,ik}$ ($b = 1, 2, \ldots, B; i = 0, 1, \ldots, I; k = 1, 2, \ldots, K$) are randomly initialized within the range of $[-1, 1]$. We also denote Q as the dimension of the individual vector: $Q = B \cdot (I + 1) \cdot K$.

Secondly, we divide the population into several groups equally, and denote the population in each group as P_g. The amount of the groups M_g is set to be equivalent to the amount of available basis functions, so that each group represents one basis function type. This attribute addressed in each group can also be viewed as *group gene*, like culture in human society. Competition occurs

not only among individuals in each group, but also among groups along the whole evolution process. Only the fittest individuals and groups can survive from generation to the next.

Then, for each set of input weight parameters in each group, use its corresponding basis function type to compute hidden-layer output matrix $\mathbf{H}$ with (6), and then calculate the output weights $\hat{\mathbf{\Omega}}$ analytically by MP generalized inverse with (11) just as ELM-TV algorithm does, followed by computing network output matrix $\mathbf{Y}$ using (7) or (10). Finally evaluated the fitness of each individual in each group $f(\boldsymbol{\theta}_{p,m})$, $p = 1, 2, \ldots, P_g$;$m = 1, 2, \ldots, M_g$, e.g. root mean squared error(RMSE).

Now we introduce the concept of *Group Fitness*, and choose the best individual fitness in a specific group according to:

$$F(\boldsymbol{\Theta}_m) = f(\boldsymbol{\theta}_{best,m})$$

where $\boldsymbol{\Theta}_m = [\boldsymbol{\theta}_{1,m}, \boldsymbol{\theta}_{2,m}, \ldots, \boldsymbol{\theta}_{P_g,m}]$ represents the set of all the individuals in m-th group, and $\boldsymbol{\theta}_{best,m}$ is the individual with the best fitness of all in m-th group.

Once all the fitness of individuals in each group are computed, apply mutation, crossover and individual selection of DE, the same as in E-ELM algorithm, followed by the final step group selection to form a new generation.

Group Selection: Compare the computed $f(\boldsymbol{\gamma}_{p,m,G+1})$ among each group and obtain group fitness $F(\boldsymbol{\Gamma}_{m,G+1})$ of every group, which are then sorted out in the next step. The group with worst fitness is wiped out and replaced by that of best fitness, e.g. the groups for next generation are finally obtained by this rule:

$$\boldsymbol{\Theta}_{m,G+1} = \begin{cases} \boldsymbol{\Gamma}_{Best,G+1} & \text{,if } F(\boldsymbol{\Gamma}_{m,G+1}) = F(\boldsymbol{\Gamma}_{Worst,G+1}) \\ \boldsymbol{\Gamma}_{m,G+1} & \text{,otherwise} \end{cases}$$

Once new groups are generated, repeat the DE and GS process until the goal is met or a preset maximum learning epochs are completed. The summarized algorithm is shown in Algorithm 1.

Actually, it is not necessary to apply GS in every interaction of DE. For instance, we can introduce a frequency parameter d, and allow the algorithm to perform GS in every d interactions. So we have this algorithm named Group Selection Evolutionary ELM for Time-Varying neural networks (GSE-ELM-TV).

A simple time-varying system is constructed for simulations:

$$y[n] = g(w[n] \cdot x[n])$$

where $g(\cdot)$ is a sigmoid activation function $g(x) = \frac{1}{1+e^{-x}}$, and $w[n]$ is a single time-variant coefficient, which is combination of 3 prolate function bases, $w[n] = \sum_{b=1}^{3} f_b[n] \cdot w_b$, with setting $w_b = 0.5, b = 1, 2, 3$. The inputs are normalized (within the range $[-1, 1]$).

In BP-TV simulations, we compare different parameters(number of hidden nodes, type of basic function, and number of epochs), and list the best results

Algorithm 1 GSE-ELM-TV

Randomly generate individuals $\boldsymbol{\theta}_{p,m,1}$ of $P_g \cdot M_g$;
repeat
  Apply ELM-TV algorithm for $\boldsymbol{\Omega}_{p,m,G}$ and $f(\boldsymbol{\theta}_{p,m,G})$;
  Apply DE algorithm;
  if $G \equiv 0 \pmod{d}$ **then**
    Obtain $F(\boldsymbol{\Theta}_{m,G+1})$ and apply GS.
  end if
  $G \leftarrow G + 1$;
until $G = G_{max}$ or $F(\boldsymbol{\Theta}_{Best,G}) \leqslant F_{Goal}$
$\boldsymbol{\theta}_{best,Best,G}$; $\boldsymbol{\Omega}_{best,Best,G}$ is the solution.

with 150 epochs in Table 1. More hidden nodes are not helping for generalization performance in BP-TV. In ELM-TV simulations, hidden neurons are gradually increase with every 5 nodes and the best testing accuracies are selected as the final ones. We also test E-ELM-TV, which is Evolutionary ELM for Time-Varying NNs without GS, hence requiring selecting the basis function manually. Theoretically, only one hidden neuron is needed for the GSE-ELM-TV model to learn this TV perceptron system. The results of E-ELM, BP-TV, ELM-TV, E-ELM-TV and GSE-ELM-TV with different parameters are listed in Table 1, from which we can easily notice that though E-ELM-TV with Prolate basis function can achieve better testing accuracy with less training time than GSE-ELM-TV; but if improper basis type like Chebyshev or Legendre is selected, much worse performances would be obtained. Also note that varying the population size, the used generations, and/or the hidden nodes number in GSE-ELM-TV, we may reduce the training time with different testing performances. This allows us concluding that the proposed approach provides a flexible tradeoff between training time and number of hidden nodes (hence networks complexity).

5 Incremental-based ELM for TV-NNs

5.1 I-ELM-TV and EI-ELM-TV

It is proved in [3] and [4] that for one specific output neuron, when the k-th hidden neuron is added and $\beta_k = \frac{\tilde{\mathbf{h}}_k^T \cdot \tilde{\mathbf{e}}_{k-1}}{\tilde{\mathbf{h}}_k^T \cdot \tilde{\mathbf{h}}_k}$, $\|\tilde{\mathbf{e}}_k\| = \|\tilde{\mathbf{t}} - (\tilde{\mathbf{t}}_{k-1} + \beta_k \tilde{\mathbf{h}}_k)\|$ achieves its minimum and the sequence $\{\|\tilde{\mathbf{e}}_k\|\}$ decreases and converges. Note that $\beta_k = \frac{\tilde{\mathbf{h}}_k^T \cdot \tilde{\mathbf{e}}_{k-1}}{\tilde{\mathbf{h}}_k^T \cdot \tilde{\mathbf{h}}_k}$, is a special case of $\beta_k = \tilde{\mathbf{h}}_k^{\dagger} \tilde{\mathbf{e}}_{k-1}$ when $\tilde{\mathbf{e}}_{k-1}$ and $\tilde{\mathbf{h}}_k$ are vectors. Actually, the MP generalized inverse of $\tilde{\mathbf{h}}_k$ is just $\tilde{\mathbf{h}}_k^{\dagger} = (\tilde{\mathbf{h}}_k^T \cdot \tilde{\mathbf{h}}_k)^{-1} \tilde{\mathbf{h}}_k^T$ For our time-variant case, this can also be extended to matrix computations. Let $\delta\boldsymbol{\Omega}_k = \begin{bmatrix} \beta_{1,k1}, \cdots, \beta_{1,kL} \\ \cdots, \cdots, \cdots \\ \beta_{B,k1}, \cdots, \beta_{B,kL} \end{bmatrix}_{B \times L}$ and $\delta\mathbf{G}_k = \begin{bmatrix} h_k[1] \cdot \mathbf{f}[1]^T \\ \cdots \\ h_k[N] \cdot \mathbf{f}[N]^T \end{bmatrix}_{N \times B}$,

Algorithm	Training Time (s)	Training RMSE(dB)	Testing RMSE(dB)	Hidden neurons
E-ELM	6.66	-12.69	-12.77	1
BP-TV(Prolate;Epoch=150)	173.31	-21.92	-22.07	1
ELM-TV(Prolate)	0.17	-80.39	-77.86	50
ELM-TV(Legendre)	0.19	-72.65	-71.16	60
ELM-TV(Chebyshev)	0.20	-74.82	-71.30	65
E-ELM-TV(Prolate)	374.72	-124.37	-124.41	1
E-ELM-TV(Legendre)	375.94	-20.55	-20.57	1
E-ELM-TV(Chebyshev)	375.64	-20.62	-20.62	1
GSE-ELM-TV(Gen=20;P=600)	380.92	-105.41	-105.35	1
GSE-ELM-TV(Gen=5;P=300)	46.46	-28.61	-28.57	1
GSE-ELM-TV(Gen=5;P=100)	51.98	-75.50	-72.87	30

Table 1. Performance comparisons of E-ELM, BP-TV, ELM-TV, E-ELM-TV and GSE-ELM-TV in Time-Varying Perceptron System

(Note $\mathbf{\Omega} = \begin{bmatrix} \delta\mathbf{\Omega}_1 \\ \cdots \\ \delta\mathbf{\Omega}_K \end{bmatrix}$ and $\mathbf{G} = [\delta\mathbf{G}_1, \cdots, \delta\mathbf{G}_K]$.) Similarly, if $\delta\mathbf{\Omega}_k = \delta\mathbf{G}_k^{\dagger} \cdot \mathbf{E}_{k-1}$, $\|\mathbf{E}_k\| = \|\mathbf{T} - (\mathbf{T}_{k-1} + \delta\mathbf{G}_k\delta\mathbf{\Omega}_k)\|$ achieve its minimum and the sequence $\{\|\mathbf{E}_k\|\}$ would decrease and converges.

It is noted in [4] that some newly added hidden nodes may make residual error reduce less than others. In the Enhanced I-ELM method, at any step, among P trial of hidden nodes, the hidden nodes with greatest residual error reduction is chosen and added. This method is also applied in this work.

Hence, we have our time-variant version of I-ELM and EI-ELM. Assume we have a set of training data $\{(\mathbf{x}[n], \mathbf{t}[n])\}_{n=1}^{N}$, the target output matrix $\mathbf{T}$, the residual matrix $\mathbf{E}$, the maximum number of hidden nodes K_{max}, and the expected learning accuracy ϵ. We get Algorithm 2.

5.2 EM-ELM-TV

Similarly, with certain transformation, Error Minimized ELM approach can also be extended in time-variant case. Replace $\mathbf{H}_0$, $\delta\mathbf{H}_m$, β with $\mathbf{G}_1$,$\delta\mathbf{G}_k$, $\mathbf{\Omega}$, respectively, we get our time-variant version of EM-ELM. For the sake of presentation clarity, we only add hidden nodes one by one in our proposed EM-ELM-TV algorithm, which can be easily generalized to the group-by-group node addition means.

Assume we have a set of training data $\{(\mathbf{x}[n], \mathbf{t}[n])\}_{i=1}^{N}$, the target matrix $\mathbf{T}$, the residual matrix $\mathbf{E}_k = \mathbf{G}_k\mathbf{G}_k^{\dagger}\mathbf{T} - \mathbf{T}$, the maximum number of hidden nodes K_{max}, and the expected learning accuracy ϵ. We get Algorithm 3.

The performance comparison in terms of necessary hidden nodes between ELM-TV and EM-ELM-TV, has been conducted in these time-varying systems. The target training RMSE(dB) ϵ are set as: -40 for perceptron system, -35 for

Algorithm 2 I-ELM-TV(in the case of $P = 1$) and EI-ELM-TV

Let $k = 0$ and residual error $\mathbf{E} = \mathbf{T}$,
while $k < K_{max}$ and $\|\mathbf{E}\| > \epsilon$, **do**
  Increase by one the number of hidden nodes: $k = k + 1$;
  for $p = 1 : P$ **do**
    Assign random input weight and bias $\{w_{(p)b,ik}\}$ for new hidden node;
    Calculate the hidden layer output submatrix for new hidden node

$$\delta\mathbf{G}_{(p)} = \begin{bmatrix} h_{(p)k}[1] \cdot \mathbf{f}[1]^T \\ \cdots \\ h_{(p)k}[N] \cdot \mathbf{f}[N]^T \end{bmatrix}_{N \times B}$$

    Calculate the output weight $\delta\mathbf{\Omega}_{(p)}$ for the new hidden node

$$\delta\mathbf{\Omega}_{(p)} = \delta\mathbf{G}_{(p)}^{\dagger} \cdot \mathbf{E}$$

    Calculate the residual error after adding the new hidden node k:

$$\mathbf{E}_{(p)} = \mathbf{E} - \delta\mathbf{G}_{(p)} \cdot \delta\mathbf{\Omega}_{(p)}$$

  end for
  Let $p^* = \{p | min_{1 \leqslant p \leqslant P} \|\mathbf{E}_{(p)}\|\}$.
  Set $\mathbf{E} = \mathbf{E}_{(p^*)}$, $\{w_{b,ik} = w_{(p^*)b,ik}\}$, and $\delta\mathbf{\Omega}_k = \delta\mathbf{\Omega}_{(p^*)}$.
end while
The output weight matrix would be $\mathbf{\Omega} = \begin{bmatrix} \delta\mathbf{\Omega}_1 \\ \cdots \\ \delta\mathbf{\Omega}_K \end{bmatrix}_{B \cdot K \times L}$

MLP system and -18 for Narendra system. The optimal number of hidden nodes for ELM-TV is obtained by trial-and-error. Table 2 displays performance evaluation between ELM-TV and EM-ELM-TV, since I-ELM-TV and EM-ELM-TV are not able to reach the training goals of them within 500 nodes. Focusing on the MLP and Narendra Systems case studies for reasons explained above, results reported in Table 2 prove that EM-ELM-TV has similar generalization performance and optimal number of hidden nodes attainable with ELM-TV, but at reduced training time. Therefore, EM-ELM-TV is able to optimally select the number of hidden nodes more efficiently than the trial-and-error approach typically used in common ELM-TV, as well as concluded in the time-invariant case.

6 Online Sequential ELM for TV-NNs

All the training data (N samples) have to be available before using the batch algorithm ELM-TV stated in previous chapters. However, in some applications, the neural networks may receive the training data sequentially, or large amount of data to process in limited memory at the same time. Therefore, it is therefore convenient to develop an online algorithm for TV-NN.

Algorithm 3 EM-ELM-TV

1: Randomly generate one hidden node.
2: Calculate the time variant hidden layer output matrix $\mathbf{G}_1$ by (6) and (5):

$$\mathbf{G}_1 = \begin{bmatrix} h_1[1] \cdot \mathbf{f}[1]^T \\ \cdots \\ h_1[N] \cdot \mathbf{f}[N]^T \end{bmatrix}_{N \times B}$$

3: Calculate the output error $\|\mathbf{E}_1\| = \|\mathbf{G}_1\mathbf{G}_1^{\dagger}\mathbf{T} - \mathbf{T}\|$.
4: Let $k = 1$
5: **while** $k < K_{max}$ and $\|\mathbf{E}_k\| > \epsilon$, **do**
6: Randomly add another hidden node. The corresponding hidden layer output matrix becomes $\mathbf{G}_{k+1} = [\mathbf{G}_k, \delta\mathbf{G}_k]$, where

$$\delta\mathbf{G}_k = \begin{bmatrix} h_k[1] \cdot \mathbf{f}[1]^T \\ \cdots \\ h_k[N] \cdot \mathbf{f}[N]^T \end{bmatrix}_{N \times B}$$

7: Update the output weight $\mathbf{\Omega}$

$$\mathbf{D}_k = ((\mathbf{I} - \mathbf{G}_k\mathbf{G}_k^{\dagger})\delta\mathbf{G}_k)^{\dagger}$$
$$\mathbf{U}_k = \mathbf{G}_k^{\dagger} - \mathbf{G}_k^{\dagger}\delta\mathbf{G}_k\mathbf{D}_k$$
$$\mathbf{\Omega}^{(k+1)} = \mathbf{G}_{k+1}^{\dagger}\mathbf{T} = \begin{bmatrix} \mathbf{U}_k \\ \mathbf{D}_k \end{bmatrix}\mathbf{T}$$

8: $k = k + 1$
9: **end while**

It is here assumed that the number of samples N is large and the rank of time-varying hidden matrix $R(G) = K \cdot B$, i.e. the number of hidden nodes by the number of output bases; $\mathbf{G}^{\dagger}$ in (11) can be expressed as:

$$\mathbf{G}^{\dagger} = (\mathbf{G}^T\mathbf{G})^{-1}\mathbf{G}^T \tag{12}$$

Given the initial training set $\{(\mathbf{x}[n], \mathbf{t}[n])\}_{n=1}^{N_0}$ it is assumed that the number of samples is larger than the number of hidden nodes by the number of output bases, e.g. $N_0 > K \cdot B$. Using batch ELM-TV, according to (11) the optimal output weight matrix would be:

$$\mathbf{\Omega}^{(0)} = \mathbf{G}_0^{\dagger} \cdot \mathbf{T}_0 = (\mathbf{G}_0^T\mathbf{G}_0)^{-1}\mathbf{G}_0^T\mathbf{T}_0 = \mathbf{C}_0^{-1}\mathbf{A}_0 \tag{13}$$

where $\mathbf{C}_0 = \mathbf{G}_0^T\mathbf{G}_0$;$\mathbf{A}_0 = \mathbf{G}_0^T\mathbf{T}_0$;

$$\mathbf{G}_0 = \begin{bmatrix} \mathbf{h}[1]^T \otimes \mathbf{f}[1]^T \\ \vdots \\ \mathbf{h}[N_0]^T \otimes \mathbf{f}[N_0]^T \end{bmatrix}_{N_0 \times K \cdot B} ; \text{and } \mathbf{T}_0 = \begin{bmatrix} \mathbf{t}[1]^T \\ \vdots \\ \mathbf{t}[N_0]^T \end{bmatrix}_{N_0 \times L} \tag{14}$$

Systems [stop RMSE(dB)]	Algorithms	Training Time (s)	Testing RMSE(dB)	Hidden Nodes
Perceptron [-40]	ELM-TV	0.0775	-41.80	13.7
	EM-ELM-TV	0.1166	-41.75	13.2
MLP [-35]	ELM-TV	3.0707	-30.69	197.7
	EM-ELM-TV	0.3463	-30.07	203.8
Narendra [-18]	ELM-TV	0.2039	-16.13	48.1
	EM-ELM-TV	0.1604	-16.75	49.5

Table 2. Performance Comparisons of ELM-TV and EM-ELM-TV for the Number(average of 10 trials) of Hidden Nodes When the Expected Accuracy is Reached

Let's suppose that another chunk of data $\{(\mathbf{x}[n], \mathbf{t}[n])\}_{n=N_0+1}^{N_0+N_1}$ arrives, the optimal output weight matrix has to be modified as:

$$\mathbf{\Omega}^{(1)} = \mathbf{C}_1^{-1}\mathbf{A}_1 \tag{15}$$

where

$$\mathbf{C}_1 = \begin{bmatrix}\mathbf{G}_0\\ \mathbf{G}_1\end{bmatrix}^T \begin{bmatrix}\mathbf{G}_0\\ \mathbf{G}_1\end{bmatrix} = \begin{bmatrix}\mathbf{G}_0^T & \mathbf{G}_1^T\end{bmatrix} \begin{bmatrix}\mathbf{G}_0\\ \mathbf{G}_1\end{bmatrix} = \mathbf{C}_0 + \mathbf{G}_1^T\mathbf{G}_1 \tag{16}$$

$$\mathbf{A}_1 = \begin{bmatrix}\mathbf{G}_0\\ \mathbf{G}_1\end{bmatrix}^T \begin{bmatrix}\mathbf{T}_0\\ \mathbf{T}_1\end{bmatrix} = \begin{bmatrix}\mathbf{G}_0^T & \mathbf{G}_1^T\end{bmatrix} \begin{bmatrix}\mathbf{T}_0\\ \mathbf{T}_1\end{bmatrix} = \mathbf{A}_0 + \mathbf{G}_1^T\mathbf{T}_1 \tag{17}$$

$$\mathbf{G}_1 = \begin{bmatrix}\mathbf{h}[N_0+1]^T \otimes \mathbf{f}[N_0+1]^T\\ \vdots\\ \mathbf{h}[N_0+N_1]^T \otimes \mathbf{f}[N_0+N_1]^T\end{bmatrix}_{N_1 \times K\cdot B}; \mathbf{T}_1 = \begin{bmatrix}\mathbf{t}[N_0+1]^T\\ \vdots\\ \mathbf{t}[N_0+N_1]^T\end{bmatrix}_{N_1 \times L} \tag{18}$$

The aim here is to express $\mathbf{\Omega}^{(1)}$ as a function of $\mathbf{\Omega}^{(0)}$,$\mathbf{C}_1$,$\mathbf{G}_1$ and $\mathbf{T}_1$. According to (13) and (16), $\mathbf{A}_0$ can be written as:

$$\mathbf{A}_0 = \mathbf{C}_0\mathbf{C}_0^{-1}\mathbf{A}_0 = (\mathbf{C}_1 - \mathbf{G}_1^T\mathbf{G}_1)\mathbf{\Omega}^{(0)} = \mathbf{C}_1\mathbf{\Omega}^{(0)} - \mathbf{G}_1^T\mathbf{G}_1\mathbf{\Omega}^{(0)} \tag{19}$$

Combining (15),(16),(17), and (19), the following can be obtained:

$$\mathbf{\Omega}^{(1)} = \mathbf{C}_1^{-1}(\mathbf{C}_1\mathbf{\Omega}^{(0)} - \mathbf{G}_1^T\mathbf{G}_1\mathbf{\Omega}^{(0)} + \mathbf{G}_1^T\mathbf{T}_1) = \mathbf{\Omega}^{(0)} + \mathbf{C}_1^{-1}\mathbf{G}_1^T(\mathbf{T}_1 - \mathbf{G}_1\mathbf{\Omega}^{(0)}) \tag{20}$$

Since $\mathbf{C}_1^{-1}$ is used for computing $\mathbf{\Omega}^{(1)}$, then by setting $\mathbf{P}_0 = \mathbf{C}_0^{-1}$ and $\mathbf{P}_1 = \mathbf{C}_1^{-1}$, and using the Woodbury formula [21] the following can be derived:

$$\mathbf{P}_1 = (\mathbf{C}_0 + \mathbf{G}_1^T\mathbf{G}_1)^{-1} = \mathbf{P}_0 - \mathbf{P}_0\mathbf{G}_1^T(\mathbf{I} + \mathbf{G}_1\mathbf{P}_0\mathbf{G}_1^T)^{-1}\mathbf{G}_1\mathbf{P}_0 \tag{21}$$

Generalizing the previous arguments, when m-th chunk of data set arrives:

$$\mathbf{P}_m = \mathbf{P}_{m-1} - \mathbf{P}_{m-1}\mathbf{G}_m^T(\mathbf{I} + \mathbf{G}_m\mathbf{P}_{m-1}\mathbf{G}_m^T)^{-1}\mathbf{G}_m\mathbf{P}_{m-1} \tag{22}$$

$$\mathbf{\Omega}^{(m)} = \mathbf{\Omega}^{(m-1)} + \mathbf{P}_m\mathbf{G}_m^T(\mathbf{T}_m - \mathbf{G}_m\mathbf{\Omega}^{(m-1)}) \tag{23}$$

When the training data is received one-by-one instead of chunk-by-chunk, e.g. $N_m = 1$, the above formula have the following simple format:

$$\mathbf{P}_m = \mathbf{P}_{m-1} - \frac{\mathbf{P}_{m-1}\mathbf{g}_m\mathbf{g}_m^T\mathbf{P}_{m-1}}{1 + \mathbf{g}_m^T\mathbf{P}_{m-1}\mathbf{g}_m} \tag{24}$$

$$\mathbf{\Omega}^{(m)} = \mathbf{\Omega}^{(m-1)} + \mathbf{P}_m\mathbf{g}_m(\mathbf{t}_m^T - \mathbf{g}_m^T\mathbf{\Omega}^{(m-1)}) \tag{25}$$

where $\mathbf{g}_m = (\mathbf{h}_m^T \otimes \mathbf{f}_m^T)^T$

Now, our proposed online algorithm, namely OS-ELM-TV can be summarized as follows. Assume that a single hidden layer time-varying neural network is to be trained, with K hidden nodes and B output basis functions, and receiving the training data set $\{(\mathbf{x}[n], \mathbf{t}[n])\}$ sequentially. Algorithm 4 is thus attained. Note that the time invariant OS-ELM algorithm is just a special case of the proposed OS-ELM-TV (when $B = 1$).

Algorithm 4 OS-ELM-TV

1: Randomly generate the input weights set $\{\omega_{b,ik}\}$ and choose a set of basis funtions.
2: Accumulate N_0 samples of training data (make sure $N_0 > K \cdot B$).
3: Calculate the time-varying hidden layer output matrix $\mathbf{G}_0$ by (14)
4: Calculate $\mathbf{P}_0 = (\mathbf{G}_0^T\mathbf{G}_0)^{-1}$
5: Calculate the initial output weight matrix $\mathbf{\Omega}^{(0)}$ by (13)
6: **for** $m = 1$ to M **do**
7: When the m-th chunk of data arrives, calculate the time-varying partial hidden layer output matrix $\mathbf{G}_m$.
8: Update the output weight matrix $\mathbf{\Omega}^{(m)}$ by,

$$\mathbf{P}_m = \mathbf{P}_{m-1} - \mathbf{P}_{m-1}\mathbf{G}_m^T(\mathbf{I} + \mathbf{G}_m\mathbf{P}_{m-1}\mathbf{G}_m^T)^{-1}\mathbf{G}_m\mathbf{P}_{m-1} \tag{26}$$

$$\mathbf{\Omega}^{(m)} = \mathbf{\Omega}^{(m-1)} + \mathbf{P}_m\mathbf{G}_m^T(\mathbf{T}_m - \mathbf{G}_m\mathbf{\Omega}^{(m-1)}) \tag{27}$$

9: $m \leftarrow m + 1$
10: **end for**

In practice, in order to apply OS-ELM-TV, $N_0 > 1.2KB$ is usually chosen to make sure $R(G) = KB$ and $\mathbf{G}^T\mathbf{G}$ a full rank matrix so that $\mathbf{P}_0$ exists and $\mathbf{P}_m$ can be computed recursively. In some applications, the selected parameters K and B can not be too large, otherwise there would occur that $R(G) < KB$ and the online algorithm might likely diverge.

The results of Electricity Load Estimation [22] are depicted in Table 3. The numbers in parentheses in first column of the tables represent the numbers of basis functions in input and output layers. Such results show that OS-ELM-TV has comparable generalization performance to ELM-TV. On the other hand, OS-ELM-TV consumes much less training time when updating the output weights in the applications in which the training data set arrived sequentially. Especially in the case of large number of hidden nodes and large number of output bases, as shown in Table 3 for electricity load estimation task, once the TV-NN receives a

Basis Function (#input, #output)	Algorithms	Training RMSE(dB)	Validation RMSE(dB)	Training Time (s)
None(1,1)	ELM	-14.84	-14.83	0.0123
	OS-ELM	-14.66	-14.64	0.0002
Legendre(3,1)	ELM-TV	-12.90	-12.90	0.0173
	OS-ELM-TV	-12.90	-12.91	0.0003
Legendre(1,3)	ELM-TV	-15.74	-15.69	0.0616
	OS-ELM-TV	-15.45	-15.41	0.0004
Legendre(3,3)	ELM-TV	-13.98	-13.92	0.0781
	OS-ELM-TV	-14.07	-14.03	0.0004
Chebyshev(1,3)	ELM-TV	-15.62	-15.60	0.0656
	OS-ELM-TV	-15.60	-15.59	0.0004
Prolate(1,3)	ELM-TV	-15.74	-15.71	0.0679
	OS-ELM-TV	-15.69	-15.67	0.0004
Fourier(1,3)	ELM-TV	-15.41	-15.40	0.0607
	OS-ELM-TV	-15.32	-15.30	0.0004

Table 3. Performance Comparisons of ELM-TV and OS-ELM-TV with different basis functions and number of input/output bases when updating the 2000th sample in electricity load estimation task, with 30 hidden nodes.

new training sample, OS-ELM-TV takes only about 0.4 milliseconds to update its output weights, while ELM-TV takes more than 60 milliseconds to retrain its network.

7 Conclusions

As a learning technique for nonstationary environments, ELM-TV and its variants have shown good potentials to resolving regression problems. However, there are some open issues on ELM-TV worth investigating in the future.

- As the most important parameters in ELM-TV, the number of hidden nodes or the number of basis functions can be determined by EM-ELM-TV or EM-OB respectively. How to develop a joint incremental approach to set the optimal numbers of both hidden nodes and basis functions remains open.
- ELM-TV and its variants aim at nonstationary environment. More applications in this categaory are to be tested to further verify the effectiveness of these algorithms.

References

1. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: Theory and applications. Neurocomputing **70**(1-3) (2006) 489 – 501 Neural Networks - Selected Papers from the 7th Brazilian Symposium on Neural Networks (SBRN '04).

2. Zhu, Q.Y., Qin, A., Suganthan, P., Huang, G.B.: Evolutionary extreme learning machine. Pattern Recognition **38**(10) (2005) 1759 – 1763
3. Huang, G.B., Chen, L., Siew, C.K.: Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Transactions on Neural Networks **17**(4) (July 2006) 879–892
4. Huang, G., Chen, L.: Enhanced random search based incremental extreme learning machine. Neurocomputing **71**(16-18) (2008) 3460–3468
5. Feng, G., Huang, G.B., Lin, Q., Gay, R.: Error Minimized Extreme Learning Machine With Growth of Hidden Nodes and Incremental Learning. IEEE Transactions on Neural Networks **20**(8) (aug. 2009) 1352 –1357
6. Suresh, S., Saraswathi, S., Sundararajan, N.: Performance enhancement of extreme learning machine for multi-category sparse data classification problems. Engineering Applications of Artificial Intelligence **23**(7) (2010) 1149–1157
7. Soria-Olivas, E., Gomez-Sanchis, J., Martin, J., Vila-Frances, J., Martinez, M., Magdalena, J., Serrano, A.: BELM: Bayesian Extreme Learning Machine. IEEE Transactions on Neural Networks **22**(3) (march 2011) 505 –509
8. Miche, Y., Sorjamaa, A., Bas, P., Simula, O., Jutten, C., Lendasse, A.: OP-ELM: Optimally pruned extreme learning machine. IEEE Transactions on Neural Networks **21**(1) (2010) 158–162
9. Cao, J., Lin, Z., Huang, G.: Composite function wavelet neural networks with extreme learning machine. Neurocomputing **73**(7-9) (2010) 1405–1416
10. Han, F., Huang, D.S.: Improved extreme learning machine for function approximation by encoding a priori information. Neurocomputing **69**(16-18) (2006) 2369 – 2373
11. Huang, D.: Radial basis probabilistic neural networks: Model and application. International Journal of Pattern Recognition and Artificial Intelligence **13** (1999) 1083–1101
12. Huang, D.S., Du, J.X.: A Constructive Hybrid Structure Optimization Methodology for Radial Basis Probabilistic Neural Networks. IEEE Transactions on Neural Networks **19**(12) (dec. 2008) 2099 –2115
13. Huang, G., Siew, C.: Extreme learning machine: RBF network case. In: Control, Automation, Robotics and Vision Conference, 2004. ICARCV 2004 8th. Volume 2., IEEE (2004) 1029–1036
14. Liang, N.Y., Huang, G.B., Saratchandran, P., Sundararajan, N.: A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks. IEEE Transactions on Neural Networks **17**(6) (November 2006) 1411–1423
15. Li, M.B., Er, M.J.: Nonlinear System Identification Using Extreme Learning Machine. In: Proc. 9th International Conference on Control, Automation, Robotics and Vision ICARCV '06. (December 5–8, 2006) 1–4
16. Xu, Y., Dong, Z., Meng, K., Zhang, R., Wong, K.: Real-time transient stability assessment model using extreme learning machine. Generation, Transmission & Distribution, IET **5**(3) (2011) 314–322
17. Titti, A., Squartini, S., Piazza, F.: A new time-variant neural based approach for nonstationary and non-linear system identification. In: Proc. IEEE International Symposium on Circuits and Systems ISCAS 2005. (May 23–26, 2005) 5134–5137
18. Cingolani, C., Squartini, S., Piazza, F.: An extreme learning machine approach for training Time Variant Neural Networks. In: Proc. IEEE Asia Pacific Conference on Circuits and Systems APCCAS 2008. (November 2008) 384–387
19. Grenier, Y.: Time-dependent ARMA modeling of nonstationary signals. IEEE Transactions on Acoustics, Speech and Signal Processing **31**(4) (1983) 899–911

20. Horn, R., Johnson, C.: Topics in matrix analysis. Cambridge University Press (1994)
21. Golub, G., Loan, C.: Matrix computations, 3rd ed. Johns Hopkins studies in the mathematical sciences. Johns Hopkins University Press (1996)
22. Deoras, A.: Electricity Load and Price Forecasting Webinar Case Study (2010)