Comparing Error Minimized Extreme Learning Machines and Support Vector Sequential Feedforward Neural Networks for Classification Problems

Enrique Romero and René Alquézar

Abstract— **Recently, error minimized extreme learning machines (EM-ELMs) have been proposed as a simple and efficient approach to build single-hidden-layer feedforward networks (SLFNs) sequentially. They add random hidden nodes one by one (or group by group) and update the output weights incrementally to minimize the sum-of-squares error in the training set. Other very similar methods that also construct SLFNs sequentially had been reported earlier with the main difference that their hidden-layer weights are a subset of the data instead of being random. By analogy with the concept of support vectors original of support vector machines (SVMs), these approaches can be referred to as support vector sequential feedforward neural networks (SV-SFNNs). An experimental study on ten benchmark classification data sets, comparing EM-ELMs and SV-SFNNs, was carried out under the same conditions for the two models. Although both models have the same (efficient) computational cost, a statistically significant improvement in generalization performance of SV-SFNNs vs. EM-ELMs was found in six out of the ten benchmark problems.**

I. Introduction

Feed-forward Neural Networks (FNNs) are a popular machine learning approach for classification and regression problems with very interesting properties (see, for example, [2]). As a specific type of FNNs, the single-hidden-layer feedforward networks (SLFNs) play an important role in practical applications. Since the optimal number of hidden nodes is problem dependent and unknown in advance, users usually choose the number of hidden nodes by trial-and-error. Once the architecture is fixed, an iterative learning algorithm such as back-propagation gradient descent is usually applied to adjust the weights in the output and hidden layers simultaneously.

There exist, however, FNN models that construct the network sequentially, so that the number of hidden units is a result of the learning process rather than being fixed a priori. For a review of constructive FNNs see, for example, [8]. Recently, error minimized extreme learning machines (EM-ELMs) have been proposed as a simple and efficient approach to build SLFNs sequentially [4]. EM-ELMs are an incremental extension of the previously presented extreme learning machines (ELMs) [6]. Both methods use random hidden nodes and find the output weights to minimize the sum-of-squares error in the training set by solving a linear system of equations. The specific features of EM-ELMs with respect to ELMs are that they add random hidden nodes one by one (or group by group) and update the output weights incrementally in an efficient way by taking advantage of the incremental construction of the hidden-layer output matrix involved in the linear system.

Other very similar methods that also construct SLFNs sequentially had been reported earlier ([3], [14], [11]). They all find the optimal linear weights of the output layer by solving the same linear system. In fact, the idea of adding random hidden units was already stated in the Sequential Approximation with Optimal Coefficients and Interacting Frequencies (SAOCIF) algorithm ([10], [11]) as a possible strategy to be used and, as shown in [11], the solution of the linear system can be computed efficiently thanks to the incremental construction of the hidden-layer output matrix. The EM-ELMs and SAOCIF with *random* selection strategy can easily be shown to be equivalent (see Section II).

Another strategy proposed in [10], [11] to be used within SAOCIF was to take hidden-layer weights always as a subset of the data (*input* strategy). In this case, the resulting method is equivalent to the Orthogonal Least Squares Learning algorithm [3] and to Kernel Matching Pursuit with prefitting [14]. All of them select the hidden-layer weights among the input vectors. By analogy with the concept of support vectors original of support vector machines (SVMs) [13], these approaches can be referred to as support vector sequential feedforward neural networks (SV-SFNNs) [12].

SV-SFNNs and SVMs were compared experimentally in [12]. Very similar accuracies were found, although computational times were lower for SVMs. Regarding the number of support vectors, SV-SFNNs constructed models with less hidden units than standard SVMs and in the same range as "sparse" SVMs [7]. On the other hand, EM-ELMs were compared in [4] with other sequential algorithms, namely resource allocation network (RAN) [9] and minimum resource allocation network (MRAN) [15], as well as with the original ELMs [6]. EM-ELMs obtained better performance and less training time than RAN and MRAN, and a similar performance but less training time than ELMs.

This work focuses on the comparison of EM-ELMs (i.e. SAOCIF with *random* strategy) and SV-SFNNs (i.e. SAOCIF with *input* strategy). An experimental study on ten benchmark data sets for classification problems is presented in which the two methods are compared in the same conditions

This work was supported in part by the Ministerio de Ciencia e Innovación (MICINN), under project TIN2009-13895-C02-01.

Enrique Romero is with the Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Spain.

René Alquézar is with the Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Barcelona, 08028 Spain (e-mail: ralquezar@iri.upc.edu).

and using the same software. Since both approaches can be adjusted to have the same computational cost (each candidate weight vector for a hidden unit is either generated randomly or selected randomly among the input vectors), the goal is finding out whether there is any difference in generalization performance between EM-ELMs and SV-SFNNs. In other words, does the use of inputs (support vectors) as hidden unit weights provide any advantage over pure random selection?

Although it may be argued that the *input* strategy is using some sort of information present in the training data (to set the hidden-layer weights) whereas the *random* strategy is not, this does not necessarily imply that the generalization performance of the former has to be better than that of the latter. Therefore, an empirical comparative study on a wide range of problems is interesting to assess this point. Moreover, it should be noted that, on one hand, EM-ELMs have been proposed to build SLFNs as an alternative superior (due to their simplicity, efficiency and effectiveness) to common neural net approaches like backpropagation gradient descent [4] and, on the other hand, input vectors are rarely used as hidden-layer weights in SLFNs, especially in the case of additive units (i.e. two-layer perceptrons).

II. Background

The output function of an SLFN (i.e. a fully connected FNN with a single hidden layer of N_h units and m linear output units) can be expressed as a linear combination of simple (basis) functions:

$$f_{N_h}(\mathbf{x}) = \lambda_0 + \sum_{i=1}^{N_h} \lambda_i \, \varphi\left(\boldsymbol{\omega}_i, b_i, \mathbf{x}\right) \tag{1}$$

where $\boldsymbol{\omega}_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$ are the learning parameters of the hidden units, $\lambda_i \in \mathbb{R}^m$ are the output-layer weights connecting the i-th hidden unit to the m output units, φ is the activation function of the hidden units, $\varphi(\boldsymbol{\omega}_i, b_i, \mathbf{x})$ is the output of the i-th hidden unit with respect to the input $\mathbf{x}$, and $\lambda_0 \in \mathbb{R}^m$ denotes the bias terms (if any) of the linear output units.

Although a lot of activation functions φ (even not neuron alike) can be used that allow universal approximation capability, the more usual choices are the Gaussian RBF applied to a distance between an input vector and a centre

$$\varphi\left(\boldsymbol{\omega}_i, b_i, \mathbf{x}\right) = \mathbf{gau}\left(b_i \, ||\mathbf{x} - \boldsymbol{\omega}_i||\right) \tag{2}$$

and the sigmoid (e.g. hyperbolic tangent) applied to a scalar product of the input and weight vectors (this will be referred to as a sigmoid additive unit [4])

$$\varphi\left(\boldsymbol{\omega}_i, b_i, \mathbf{x}\right) = \mathbf{tnh}\left(\boldsymbol{\omega}_i \cdot \mathbf{x} + b_i\right). \tag{3}$$

For Gaussian RBF units, $\boldsymbol{\omega}_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}^+$ are the centre and the impact factor of the i-th RBF unit. For sigmoid additive units, $\boldsymbol{\omega}_i \in \mathbb{R}^n$ is the weight vector connecting the input layer to the i-th hidden unit and $b_i \in \mathbb{R}$ is the bias of the i-th hidden unit. In our experiments presented in Section III, a third activation function has also been tested, the sin applied to the scalar product (i.e. a sin additive unit)

$$\varphi\left(\boldsymbol{\omega}_i, b_i, \mathbf{x}\right) = \mathbf{sin}\left(\boldsymbol{\omega}_i \cdot \mathbf{x} + b_i\right). \tag{4}$$

For a given set of training examples $\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{L} \subset \mathbb{R}^n \times \mathbb{R}^m$, if the outputs of the network are equal to the targets, we have

$$f_{N_h}(\mathbf{x}_j) = \sum_{i=1}^{N_h} \lambda_i \, \varphi\left(\boldsymbol{\omega}_i, b_i, \mathbf{x}_j\right) = \mathbf{t}_j, \qquad j = 1, \ldots, L. \tag{5}$$

Equation (5) can be written compactly as

$$\mathbf{H}\boldsymbol{\lambda} = \mathbf{T} \tag{6}$$

where $\mathbf{H}$ is an $L \times N_h$ matrix called the hidden-layer output matrix of the network ($\mathbf{H}_{ji} = \varphi\left(\boldsymbol{\omega}_i, b_i, \mathbf{x}_j\right)$), $\boldsymbol{\lambda}$ is an $N_h \times m$ matrix containing the output-layer weights, and $\mathbf{T}$ is an $L \times m$ matrix containing the target values in the training set. The output layer biases can be added by including in H a first column with a fixed value of 1 (and increasing N_h by 1).

Normally, the number of training examples L will be much greater than the number of hidden units N_h and an exact solution of (6) cannot be expected. Then, the usual cost function in SLFNs (and in general in FNNs) is the sum-of-squares error

$$E = \frac{1}{2} \sum_{j=1}^{L} ||f_{N_h}(\mathbf{x}_j) - \mathbf{t}_j||^2. \tag{7}$$

It is well known (e.g. [2]) that, to minimize E, the optimal output-layer weights can be computed as

$$\hat{\boldsymbol{\lambda}} = \mathbf{H}^{\dagger}\mathbf{T} \quad \text{where} \quad \mathbf{H}^{\dagger} = \left(\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T \tag{8}$$

is the pseudo-inverse (or Moore-Penrose generalized inverse) of the hidden-layer output matrix $\mathbf{H}$. The sum-of-squares error can be expressed as

$$E(\mathbf{H}) = \frac{1}{2} \, ||\mathbf{H}\boldsymbol{\lambda} - \mathbf{T}||^2 = \frac{1}{2} \, ||\mathbf{H}\mathbf{H}^{\dagger}\mathbf{T} - \mathbf{T}||^2. \tag{9}$$

A. Error Minimized Extreme Learning Machines (EM-ELMs)

Huang et al. [5] have shown that SLFNs with random weights in the hidden layer have universal approximation capability for many different choices of the activation function, including the ones stated in eqs. 2 to 4. Based on this result, they propose the ELMs learning algorithm [6], which can be summarized as follows:

Algorithm for ELMs: Given a set of training examples $\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{L} \subset \mathbb{R}^n \times \mathbb{R}^m$, the hidden-layer activation function $\varphi\left(\boldsymbol{\omega}, b, \mathbf{x}\right)$, and an *a-priori* fixed number N_h of hidden units:

1) randomly assign hidden-unit parameters $(\boldsymbol{\omega}_i, b_i)$, $i = 1, \ldots, N_h$
2) calculate the hidden-layer output matrix $\mathbf{H}$
3) calculate the output-layer weight matrix $\boldsymbol{\lambda}$ using (8)

In order to avoid the need of setting in advance the number N_h of hidden units and to reduce the training computational time, a fast sequential extension of the ELMs algorithm called EM-ELMs has been recently reported by Feng et al. [4].

Algorithm for EM-ELMs: Given a set of training examples $\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{L} \subset \mathbb{R}^n \times \mathbb{R}^m$, the maximum number of hidden units $N_{\max}$, and the expected learning accuracy ϵ:

1) *Initialization phase*:
- initialize the SLFN with a small group of N_0 randomly generated hidden units $(\boldsymbol{\omega}_i, b_i)$, $i = 1, \ldots, N_0$
- calculate the hidden-layer output matrix $\mathbf{H}_0$
- calculate the corresponding output error

2) *Recursively growing phase*:
- let $k := 0$
- **while** $N_k < N_{\max}$ and $E(\mathbf{H}_k) > \epsilon$ **do**
  - let $k := k + 1$
  - randomly add δN_k hidden units to the existing SLFN; the total number of hidden units becomes $N_k = N_{k-1} + \delta N_k$ and the corresponding hidden-layer output matrix $\mathbf{H}_k = [\mathbf{H}_{k-1}, \boldsymbol{\delta}\mathbf{H}_k]$, where $\boldsymbol{\delta}\mathbf{H}_k$ contains the new δN_k columns computed
  - the output-layer weights $\boldsymbol{\lambda}$ are updated as

$$\boldsymbol{\lambda}_k = \mathbf{H}_k^{\dagger}\mathbf{T} = \begin{bmatrix} \mathbf{U}_k \\ \mathbf{D}_k \end{bmatrix} \mathbf{T}, \text{ where}$$

$$\mathbf{D}_k = \left(\left(\mathbf{I} - \mathbf{H}_{k-1}\mathbf{H}_{k-1}^{\dagger} \right) \boldsymbol{\delta}\mathbf{H}_k \right)^{\dagger} \text{ and}$$

$$\mathbf{U}_k = \mathbf{H}_{k-1}^{\dagger} \left(\mathbf{I} - \boldsymbol{\delta}\mathbf{H}_k \mathbf{D}_k \right)$$

- **end while**

B. Sequential Approximation with Optimal Coefficients and Interacting Frequencies (SAOCIF)

A rather general constructive method for SLFNs, called SAOCIF, was proposed in [10], [11] by Romero and Alquézar. The specific features of SAOCIF are: *i*) the optimal (in a least squares sense) output-layer weights are recalculated each time a hidden unit is added by solving a linear equations system, and *ii*) the added hidden unit is selected among a set of candidates taking into account its interaction with the previously added hidden units (i.e. to minimize together the training error). The SAOCIF algorithm can be described as follows:

Algorithm for SAOCIF: Given a set of training examples $\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{L} \subset \mathbb{R}^n \times \mathbb{R}^m$, the maximum number of hidden units $N_{\max}$, a *strategy* to generate the candidates, the maximum number of candidates for any hidden unit $C_{\max}$, and the expected learning accuracy ϵ:

- let $N := 0$ // in this case, N is equivalent to both k and N_k in EM-ELMs, because $\delta N_k = 1$ for SAOCIF
- let $\mathbf{H}_0 = [\,]$
- **repeat**
  - let $N := N + 1$
  - let $c := 0$ // c is the number of valid candidates tested for the N^{th} hidden unit
  - **while** $c < C_{\max}$ **do**
    - generate a candidate $(\boldsymbol{\omega}, b)$ for the N^{th} hidden unit with the given strategy, and store in a temporary matrix $\mathbf{H}$ the corresponding hidden-layer output matrix $\mathbf{H} = [\mathbf{H}_{N-1}, \boldsymbol{\delta}\mathbf{H}]$, where $\boldsymbol{\delta}\mathbf{H}$ contains the new column computed
    - find the optimal output-layer weights $\boldsymbol{\lambda} = \mathbf{H}^{\dagger}\mathbf{T}$ for the current candidate $(\boldsymbol{\omega}, b)$ using the incremental method described in [11] (and equivalent to the incremental method described in the algorithm for EM-ELMs)
    - **if** the current candidate is considered valid (linear system without numerical problems, etc) **then**
      - let $c := c + 1$
      - calculate the corresponding output error $E(\mathbf{H}) = \frac{1}{2} ||\mathbf{H}\mathbf{H}^{\dagger}\mathbf{T} - \mathbf{T}||^2$
      - **if** $E(\mathbf{H})$ is the minimum error found for the tested candidates in the current loop **then**
        - let $(\boldsymbol{\omega}_N, b_N) = (\boldsymbol{\omega}, b)$; $\boldsymbol{\lambda}_N = \boldsymbol{\lambda}$; $\mathbf{H}_N = \mathbf{H}$
      - **end if**
    - **end if**
  - **end while**
- **until** $N = N_{\max}$ or $E(\mathbf{H}_N) \leqslant \epsilon$

Note that if we set $C_{\max} = 1$ and a *random* strategy to generate the candidates in the SAOCIF algorithm and we set $\delta N_k = 1$ (for all k) in the algorithm for EM-ELMs, then both algorithms are equivalent.

C. Support Vector Sequential Feed-forward Neural Networks (SV-SFNNs)

Apart from the *random* strategy, other possibilities are allowed in the SAOCIF approach to generate the candidates. In particular, let us define the *input* strategy as the one in which the candidates are only selected among the input examples in the training set; more precisely, $\boldsymbol{\omega} = \mathbf{x}_j$, for some j not already used, and b is a constant depending on the activation function (e.g. $b = 1$ for RBF units and $b = 0$ for additive units). Then, if we set $C_{\max} = L - N + 1$ and generate the candidates using the *input* strategy, then the resulting method, which has been called SV-SFNNs [12], is equivalent to the Orthogonal Least Squares Learning (OLSL) algorithm [3] and to Kernel Matching Pursuit with *pre-fitting* (KMP-*prefit*) [14]. Actually, OLSL was only proposed for RBF units and KMP-*prefit* for kernel-based activation functions, while SAOCIF with *input* strategy permits as well any other activation function with universal approximation capabilities (e.g. sinusoidal additive units).

III. Comparing EM-ELMs and SV-SFNNs

This section compares EM-ELMs with SV-SFNNs and explains the methodology followed in the experiments.

A. Compared Methods

At first, the computational cost of the EM-ELMs algorithm is lower than that of the original SV-SFNNs defined in the

TABLE I
FEATURES OF THE CLASSIFICATION BENCHMARK DATA SETS

Data Set	#Inputs	#Classes	#Examples
Australian	43	2	690
Gene	120	3	3175
German	56	2	1000
Ionosphere	34	2	351
Iris	4	3	150
Satimage	36	6	6435
Segment.	16	7	2310
Sonar	60	2	208
Vehicle	18	4	846
Wine	13	3	178

previous section, just because in EM-ELMs the inner while loop of SAOCIF is not carried out. However, in order to make a comparison as fair as possible, we can easily adjust both methods to work in the same conditions and taking the same computation time, with the only difference residing on whether the candidates are randomly generated or taken randomly from the input patterns (i.e. *random* versus *input* strategy).

To this end, we have defined two settings. In the former, $C_{\max} = 1$, so the original EM-ELMs are confronted with a very limited version of SV-SFNNs in which a randomly selected input (not the best) yields the single candidate. In the latter, $C_{\max} = 59$, so an extended version of EM-ELMs (with the upgrade of selecting the best random candidate among $C_{\max}$ at each step) is confronted with a not so limited version of SV-SFNNs in which not the best of the remaining candidates but the best of a randomly selected subset (of size $C_{\max}$) of the remaining candidates is added. The choice of $C_{\max} = 59$ is justified because, in order to obtain a candidate that is with probability 0.95 among the best 5% of all candidates, a random subset of size $\lceil \log 0.05 / \log 0.95 \rceil = 59$ suffices.

B. Software

We have used our own implementation in C setting the algorithm parameters as explained in the preceding paragraph.

C. Data Sets

Ten benchmark data sets of classification problems from the UCI repository [1] were used for the comparison, namely *Australian Credit*, *Gene*, *German Credit*, *Ionosphere*, *Iris*, *Satimage*, *(Image) Segmentation*, *Sonar*, *Vehicle* and *Wine*. The features of these data sets are summarized in Table I.

D. Methodology

- **Preprocessing** Categorical attributes were converted to dummy variables. The rest of the attributes were scaled to mean zero and variance one.
- **Activation functions** Three types were used: Gaussian RBF (2), sigmoid additive (3) and sin additive (4) units, but with a further multiplicative positive parameter γ introduced for a wider search. Specifically, γ multiplies the distance $||\mathbf{x} - \boldsymbol{\omega}_i||$ in the RBF units and the scalar product $\boldsymbol{\omega}_i \cdot \mathbf{x}$ in the additive units.
- **Parameters and model selection** A hidden-unit candidate weight vector was not considered valid if the associated linear equations system could not be solved or if the 1-norm of the solution (the output-layer weights) was greater than a certain value M. This can be seen as a form of regularization. M was set to 1024. We fixed $N_{\max} = 99$ and $\epsilon = 0$, so that $N_{\max}$ hidden units are always added. In order to get an adequate value for the γ parameter, a search was performed ranging γ from 2^{-10} to 2^5. The same search was performed for all the models, and repeated for every activation function.
- **Model training and testing** The methods were trained and tested over 30 training-validation-test different random partitions (80% training, 10% validation, 10% test) of the whole data set. For every configuration (defined by a given strategy, $C_{\max}$, activation function and γ), the networks with the lowest errors in the validation subsets were selected as the final models. The accuracies of the final models were given by the average accuracies measured in the test subsets. The sizes of the final models are defined by their average number of hidden units.

E. Experimental Results

Tables II and III show the accuracies of the best final models (among all γ) for the two strategies (*input* and *random*) and the three activation functions tried (Gaussian RBF, sin additive, sigmoid additive) using the methodology previously described for the ten UCI data sets studied. Table II displays the results of the methods for $C_{\max} = 1$, where the *input* strategy is fully comparable to EM-ELMs, and Table III displays the results for $C_{\max} = 59$. It can be observed that *Iris* and *Wine* data sets correspond to easy problems that have been learnt perfectly using both strategies. For the other eight data sets, test accuracies look similar between the two strategies in some cases and a superior performance of the *input* strategy can be appreciated in the rest. The best values for each strategy and data set are marked in bold; not surprisingly, all of them are included in Table III (i.e. they have been obtained using $C_{\max} = 59$). In order to obtain an objective statistical measure, a Student's t-test was applied to each data set to check if the difference between the best mean results of the two strategies was statistically significant (p-value = 0.05, i.e. confidence of 95%). In six of the data sets (*Australian Credit*, *Gene*, *Ionosphere*, *Satimage*, *Segmentation*, *Sonar*) the t-test gave a significant difference with a superior mean accuracy of the *input* strategy, whereas no significant difference was found in the other ones (*German Credit* and *Vehicle*).

Tables IV and V show, for $C_{\max} = 1$ and $C_{\max} = 59$ respectively, the average number of hidden units in the final models selected for each combination of strategy and activation function. Although no clear trend is observed about the number of hidden units selected by both strategies (it depends quite a lot on the specific activation function), the *input* strategy seems to need more units than the *random* strategy in the case of Gaussian RBF hidden units (this can be

TABLE II

COMPARISON OF AVERAGE TEST ACCURACY - ONE CANDIDATE

	Gaussian RBF		Sin MLP		Sigmoid MLP	
Data Set	Input	Rand.	Input	Rand.	Input	Rand.
Australian	83.86	83.00	83.67	83.14	84.25	83.48
Gene	84.71	83.38	84.75	83.13	84.82	82.95
German	77.37	78.23	77.73	77.20	77.83	77.10
Ionosphere	93.87	90.19	90.10	88.67	90.19	88.67
Iris	100	99.11	100	99.78	100	99.56
Satimage	82.98	80.57	79.17	77.73	79.40	77.62
Segment.	86.62	81.65	86.42	86.08	86.70	86.39
Sonar	89.83	80.83	77.17	76.17	77.50	75.83
Vehicle	85.48	85.52	86.11	84.80	85.87	85.20
Wine	99.61	100	99.80	100	100	100

TABLE III

COMPARISON OF AVERAGE TEST ACCURACY - BEST OF 59 CANDIDATES

	Gaussian RBF		Sin MLP		Sigmoid MLP	
Data Set	Input	Rand.	Input	Rand.	Input	Rand.
Australian	84.49	83.48	83.77	83.43	**85.02**	**83.82**
Gene	**86.60**	**86.05**	86.34	85.86	86.36	85.92
German	77.13	**78.33**	77.60	77.30	**78.03**	77.37
Ionosphere	**93.90**	**90.67**	89.52	88.67	90.00	88.83
Iris	100	100	100	100	100	100
Satimage	**86.35**	**83.31**	82.83	77.56	81.73	77.50
Segment.	**88.56**	83.39	86.61	86.64	87.30	**86.83**
Sonar	**96.50**	**81.83**	87.67	76.67	75.00	74.17
Vehicle	86.67	85.56	**86.87**	86.11	86.75	**86.63**
Wine	100	100	100	100	100	100

TABLE IV

AVERAGE NUMBER OF HIDDEN UNITS - ONE CANDIDATE

	Gaussian RBF		Sin MLP		Sigmoid MLP	
Data Set	Input	Rand.	Input	Rand.	Input	Rand.
Australian	43.53	30.33	25.17	37.40	61.33	44.90
Gene	97.07	98.87	97.40	98.60	97.20	98.00
German	40.47	43.23	44.53	33.87	36.77	36.80
Ionosphere	73.10	40.63	19.40	24.30	14.97	24.60
Iris	55.97	31.63	3.87	3.60	3.87	3.67
Satimage	95.30	96.80	97.93	97.13	95.47	97.73
Segment.	41.53	25.90	63.10	74.30	53.40	82.07
Sonar	66.53	64.00	47.67	43.97	38.13	42.10
Vehicle	74.77	38.00	67.30	61.73	64.03	63.97
Wine	11.57	11.77	7.73	10.00	7.90	10.90

TABLE V

AVERAGE NUMBER OF HIDDEN UNITS - BEST OF 59 CANDIDATES

	Gaussian RBF		Sin MLP		Sigmoid MLP	
Data Set	Input	Rand.	Input	Rand.	Input	Rand.
Australian	50.63	23.33	15.53	32.67	24.93	21.77
Gene	95.23	89.87	90.33	87.07	88.33	84.80
German	12.20	12.20	12.97	14.23	13.83	16.90
Ionosphere	65.73	29.03	10.40	12.10	12.53	12.67
Iris	19.13	4.50	3.87	2.97	3.97	2.77
Satimage	97.67	89.73	96.23	96.83	95.40	97.43
Segment.	95.83	32.00	75.33	63.57	91.87	68.27
Sonar	88.27	55.43	31.17	10.57	25.13	21.13
Vehicle	81.73	37.13	44.80	50.77	58.37	54.27
Wine	4.90	7.13	5.00	6.13	5.13	6.23

seen easily in Table V). Regarding the number of candidates, final models obtained with $C_{\max} = 1$ usually have more hidden units than those obtained with $C_{\max} = 59$.

IV. CONCLUSIONS AND FUTURE WORK

The experimental comparison between EM-ELMs and SV-SFNNs presented in the paper draws two interesting conclusions that can be further investigated in future research. The first one is that selecting the hidden-layer weights as a subset of the input data, even if this selection is done randomly, yields better generalization results than selecting them in a purely random manner from scratch (like EM-ELMs do). As discussed at the end of Section I, this is not an obvious result. Indeed, the average accuracies obtained by the two strategies were very similar in four of the benchmark problems, but SV-SFNNs showed a statistically significant improvement in generalization performance in the other six. One might ask whether there is any noticeable difference between these two groups of classification problems. With the exceptions of the *German credit* and the *Segmentation* problems, data sets with a higher number of variables (see Table I and imagine for instance an arbitrary threshold of 20 inputs) were the ones in which SV-SFNNs outperformed EM-ELMs. Although this can be considered as a reasonable result, which may be justified by the difficulty in finding adequate decision boundaries in high-dimensional input spaces from randomly distributed hidden-layer weights, the underlying hypothesis needs further validation in future studies.

The second conclusion of the study is that, independently of the strategy used (*input* or *random*), the number of candidates for the hidden-layer weights is a parameter that controls the trade-off between the generalization performance, the computational cost and the number of hidden units of the final models. In general terms, by increasing the number of candidates at each step of the sequential algorithm (recall that in the originally proposed EM-ELMs this number is 1), the generalization is improved and the final number of hidden units is reduced at the expense of a higher training time.

REFERENCES

[1] A. Asuncion and D. J. Newman. UCI machine learning repository, 2007. University of California, Irvine, School of Information and Computer Science. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[2] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press Inc., New York, 1995.

[3] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, 1991.

[4] Feng G., G. B. Huang, Q. Lin, and R. Gay. Error Minimized Extreme Learning Machine with Growth of Hidden Nodes and Incremental Learning. *IEEE Transactions on Neural Networks*, 20(8):1352–1357, 2009.

[5] G. B. Huang, L. Chen, and C. K. Siew. Universal Approximation using Incremental Constructive Feedforward Networks with Random Hidden Nodes. *IEEE Transactions on Neural Networks*, 17(4):879–892, 2006.

[6] G. B. Huang, Q. Y. Zhu, and C. K. Siew. Extreme Learning Machine: Theory and Applications. *Neurocomputing*, 70(1-3):489–501, 2006.

[7] S. S. Keerthi, O. Chapelle, and D. DeCoste. Building Support Vector Machines with Reduced Classifier Complexity. *Journal of Machine Learning Research*, 7:1493–1515, 2006.

[8] T. Y. Kwok and D. Y. Yeung. Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems. *IEEE Transactions on Neural Networks*, 8(3):630–645, 1997.

[9] J. Platt. A Resource-Allocating Network for Function Interpolation. *Neural Computation*, 3(2):213–225, 1991.

[10] E. Romero and R. Alquézar. A New Incremental Method for Function Approximation using Feed-forward Neural Networks. In *International Joint Conference on Neural Networks*, volume 2, pages 1968–1973, 2002.

[11] E. Romero and R. Alquézar. A Sequential Algorithm for Feed-forward Neural Networks with Optimal Coefficients and Interacting Fr equencies. *Neurocomputing*, 69(13-15):1540–1552, 2006.

[12] E. Romero and D. Toppo. Comparing Support Vector Machines and Feed-forward Neural Networks with Similar Hidden-layer Weights . *IEEE Transactions on Neural Networks*, 18(3):959–963, 2007.

[13] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, NY, 1995.

[14] P. Vincent and Y. Bengio. Kernel Matching Pursuit. *Machine Learning*, 48(1-3):165–187, 2002. Special Issue on New Methods for Model Combination and Model Selection.

[15] L. Yingwei, N. Sundararajan, and P. Saratchandran. A Sequential Learning Scheme for Function Approximation using Minimal Radial Basis Function Neural N etworks. *Neural Computation*, 9(2):461–478, 1997.