Dynamic Adjustment of Hidden Layer Structure for Convex Incremental Extreme Learning Machine

Yongjiao Sun, Yuangen Chen, Ye Yuan and Guoren Wang

Abstract Extreme Learning Machine (ELM) is a learning algorithm based on generalized single-hidden-layer feed-forward neural network. Since ELM has an excellent performance on regression and classification problems, it has been paid more and more attention recently. The determination of structure of ELM plays a vital role in ELM applications. Essentially, determination of the structure of ELM is equivalent to the determination of the hidden layer structure. Utilizing a smaller scale of the hidden layer structure can promote faster running speed. In this paper, we propose algorithm PCI-ELM (Pruned-Convex Incremental Extreme Learning Machine) based on CI-ELM (Convex Incremental Extreme Learning Machine). Furthermore, we also present an improved PCI-ELM algorithm, EPCI-ELM (Enhanced Pruned-Convex Incremental Extreme Learning Machine), which introduces a filtering strategy for PCI-ELM during the neurons adding process. In order to adjust the single-hidden-layer feed-forward neural network more flexibly and achieve the most compact form of the hidden layer structure, in this paper, we propose a algorithm which can dynamically determine hidden layer structure, DCI-ELM (Dynamic Convex Incremental Extreme Learning Machine). At the end of this paper, we verify the performance of PCI-ELM, EPCI-ELM and DCI-ELM. The results show that PCI-ELM, EPCI-ELM and DCI-ELM control hidden layer structure very well and construct the more compact single-hidden-layer feed-forward neural network.

Keywords Extreme learning machine · Dynamic adjustment · Feed-forward neural network · Convex optimal increment

1 Introduction

ELM [1], which is now an important branch of neural networks, gains high testing accuracy, extremely fast learning speed and good generalization performance. Increasing attention has been drawn to ELM in both industrial and academic fields.

Y. Sun (✉) · Y. Chen · Y. Yuan · G. Wang
Northeastern University, Shenyang 110819, Liaoning, China
e-mail: sunyongjiao@ise.neu.edu.cn

J. Cao et al. (eds.), *Proceedings of ELM-2015 Volume 2*,
Proceedings in Adaptation, Learning and Optimization 7,
DOI 10.1007/978-3-319-28373-9_30

In the computing theory of ELM, the input weights and bias are generated randomly. The output weights are calculated using input data matrix, input weights and bias. Then the whole structure of neural networks are generated. From this point of view, we draw a conclusion that the calculation cost are directly related to the structure of the neural network. Researchers have done some work trying to simplify the structure of ELM on the premise of keeping the advantages of extremely fast learning speed and generalization performance, so that the overfitting problem can be avoided.

Many researchers focused on improvement of the testing accuracy, generalization performance and training speed. Hai-Jun Rong et al. proposed P-ELM (Pruned Extreme Learning Machine) [2, 7–9], which analyzes the relativity of neurons with mathematical statistics methods and eliminates the neurons insensitive of class labels. P-ELM reduces the calculation cost and is capable of real-time structure adjustment. Yoan Miche et al. focused on the influence of the incorrect training data and proposed OP-ELM (Optimally Pruned Extreme Learning Machine) [3]. Different from P-ELM, OP-ELM takes the influence of irrelative training data into consideration and increases the generalization performance and robustness using pruning strategies. Guorui Feng et al. proposed E-ELM (Error Minimized Extreme Learning Machine) [4] to realize increment of single or multiple neurons by minimizing error. Rui Zhang et al. analyzed the weights of the hidden layer nodes and proposed D-ELM (Dynamic Extreme Learning Machine) [5] to further evaluate the changes after the increment of neurons. Guorui Feng et al. proposed DA-ELM [6] (Dynamic Adjustment Extreme Learning Machine) based on theory of application circle expectation minimization to reduce errors.

Given a specific application, how to determine the structure of the neural network remains an open question. In this paper, we try to solve the problem in the following aspects.

- Generate appropriate neural network structures automatically for various applications. Increasing strategies of hidden layer neurons have to be proposed to minimize training errors.
- Evaluate neurons to organize hidden layer with higher performance. Neurons with less contribution have to be eliminated to generate a reduced but efficient neural network.
- Alter the complexity of the hidden layer structure. Adjustment strategies of hidden layer make the alteration more reasonable.

2 Improved Convex Incremental ELM

During the adjustment of the ELM structure, the increment of hidden layer nodes without consideration of the effectiveness will cause the redundant of the ELM structure. Furthermore, the additive node will also affect the effectiveness of the neurons

already added into the hidden layer. Therefore, when eliminating the neurons with low effectiveness, the measure of neurons faces great challenges.

2.1 Pruned Convex Incremental Extreme Learning Machine

The pruning ELMs usually measure and sort the neurons by some criteria to eliminate the neurons with low effectiveness. Based on the same idea, we propose Pruned Convex Incremental Extreme Learning Machine (PCI-ELM), which measures the neurons by the output weights. In the case of multiple output nodes, PCI-ELM measures the neurons by the norm of the output weights:

$$\|\beta_i\| = \sqrt{\beta_{i1}^2 + \cdots + \beta_{im}^2} \tag{1}$$

where $\beta_i = [\beta_{i1}, \ldots, \beta_{im}]^T$ is the output weights of the ith neurons; m is the number of output nodes. However, even if the output weight is not small, but the output of the activation function, taking sigmoid as example, is small, thus, in this case, the output weight cannot represent the importance of the neuron. In CI-ELM [10], the weights are updated as

$$\beta_i = (1 - \beta_L)\beta_i \tag{2}$$

If we view β_i as the function of additive weights β_L, β_i is a monotonical decreasing function. $o = [o_1, \ldots, o_m]^T$ is the output of ELM, $h_i(x_j)$ is the output of the jth sample on the ith neuron:

$$o_i = h_1(\mathbf{x}_i)\beta_1 + \cdots + h_L(\mathbf{x}_i)\beta_L \tag{3}$$

Since the parameters of the existing neurons do not change, the outputs of hidden layer neurons $h(\mathbf{x})$ do not change either. Given any training sample $\mathbf{x}$, the influence of the existing neurons $h_i(\mathbf{x})\beta_i$, $i \in (1, L-1)$ will change due to the Lth additive node. Therefore, in CI-ELM, the output weights indicate the effectiveness of hidden layer nodes very well. According to the output weights, we sort the hidden layer nodes and eliminate the ones with little effectiveness. A threshold of the weights γ is set, which is initiated as

$$\gamma' = \gamma = \frac{1}{L}\sum_{i=1}^{L} |\beta_i| \tag{4}$$

The elimination is determined by the training accuracy: if the training accuracy $\varepsilon' < \varepsilon$ after the elimination, the neurons have to be kept in the network; if $\varepsilon' > \varepsilon$ after the elimination, the network has to be pruned.

In order to provide better adjustment, PCI-ELM eliminates the neurons with low effectiveness to construct a neural network with simplest structure.

2.2 Enhanced Pruned Convex Incremental Extreme Learning Machine

The pruning after the construction of the ELM simplifies the network structure. However, the calculation of the effectiveness of the neurons and the performance variation lead to much more calculation cost. Therefore, a better way is to verify the effectiveness of the neurons before they are added into the network.

Since the smaller the norm of the output weights is, the better generalization performance the network gains [11], we also use the norm of the output weights $\|\boldsymbol{\beta}\|$ as a criterion to choose additive neurons. After the output weights are updated as $\beta_i = (1-\beta_L)\beta_i$, the neuron with smaller $\|\boldsymbol{\beta}\|$ will be added into the network. The selection is summarized as $\Psi_{L+1} = \begin{cases} \Psi_{L+1}^{(1)} \\ \Psi_{L+1}^{(2)} \end{cases} if \parallel E\left(\Psi_{L+1}^{(2)}\right) \parallel < \parallel E\left(\Psi_{L+1}^{(1)}\right) \parallel and \parallel \beta^{\Psi_{L+1}^{(2)}} \parallel < \parallel \beta^{\Psi_{L+1}^{(1)}} \parallel$ where $\Psi_{(L+1)}$ is the network with L+1 hidden layer neurons. The output weight of the Lth neuron is calculated as:

$$\left\|\beta^{\mathrm{L}-1}\right\| = \sqrt{\beta_1^2 + \cdots + \beta_{\mathrm{L}-1}^2} \tag{5}$$

When the Lth neuron is added, the output weights is calculated as:

$$\begin{aligned} \left\|\beta^{\mathrm{L}}\right\| &= \sqrt{\left(1-\beta_L\right)^2 \cdot \left(\beta_1^2 + \cdots + \beta_{L-1}^2\right) + \beta_L^2} \\ &= \sqrt{\left(1-\beta_L\right)^2 \left\|\beta^{\mathrm{L}-1}\right\|^2 + \beta_L^2} \\ &\geq \sqrt{\frac{\left\|\beta^{\mathrm{L}-1}\right\|^2}{\left\|\beta^{\mathrm{L}-1}\right\|^2 + 1}} = \frac{\left\|\beta^{\mathrm{L}-1}\right\|}{\sqrt{\left\|\beta^{\mathrm{L}-1}\right\|^2 + 1}} \end{aligned} \tag{6}$$

As to $\left\|\beta^{\mathrm{L}}\right\|$, $\left\|\beta^{\mathrm{L}-1}\right\|$ is a constant, thus, $\left\|\beta^{\mathrm{L}}\right\|$ can be viewed as a dependent variable, β_L as an independent variable. Equation 6 can be viewed as function $f(\mathbf{x}) = \sqrt{(1-\mathbf{x})^2 a^2 + \mathbf{x}^2}$, where a is a constant, $\mathbf{x} \in R$, thus, $f(\mathbf{x})$ is monotonic decreasing in the interval $\left[-\infty, \frac{a^2}{1+a^2}\right]$, monotonic increasing in the interval $\left[\frac{a^2}{1+a^2}, +\infty\right]$.

Note that the method mentioned above only tries to guarantee that the existing neurons have certain effectiveness. Although in each step we try to make $\|\boldsymbol{\beta}\|$ as small as possible, the output weight $\|\boldsymbol{\beta}\|$ is not smaller than $\|\boldsymbol{\beta}\|$ in CI-ELM for sure.

The redundancy of the network cannot be avoided, which is because the remaining training error is calculated as:

$$\begin{aligned}\Delta &= \left\|e_{L-1}\right\|^2 - \left\|e_L\right\|^2 \\ &= \left\|e_{L-1}\right\|^2 - \left\|e_{L-1} - \beta_L\left(\mathbf{H} - \mathbf{F}_{L-1}\right)\right\|^2 \\ &= 2\beta_L\left\langle e_{L-1}, \mathbf{H}_L - \mathbf{F}_{L-1}\right\rangle - \beta_L^2\left\|\mathbf{H}_L - \mathbf{F}_{L-1}\right\|^2 \\ &= \left\|\mathbf{H}_L - \mathbf{F}_{L-1}\right\|^2\left(\frac{\left\langle e_{L-1}, \mathbf{H}_L - \mathbf{F}_{L-1}\right\rangle^2}{\left\|\mathbf{H}_L - \mathbf{F}_{L-1}\right\|^4} - \left(\beta_L - \frac{\left\langle e_{L-1}, \mathbf{H}_L - \mathbf{F}_{L-1}\right\rangle}{\left\|\mathbf{H}_L - \mathbf{F}_{L-1}\right\|^2}\right)^2\right)\end{aligned} \tag{7}$$

When $\beta_L = \frac{\left\langle e_{L-1}, \mathbf{H}_L - \mathbf{F}_{L-1}\right\rangle}{\left\|\mathbf{H}_L - \mathbf{F}_{L-1}\right\|^2}$, Δ is maximum, $\Delta_{max} = \frac{\left\langle e_{L-1}, \mathbf{H}_L - \mathbf{F}_{L-1}\right\rangle^2}{\left\|\mathbf{H}_L - \mathbf{F}_{L-1}\right\|^2}$. The greater β_L is, the greater Δ_{max} is.

From Eq. 6 we can see that, if the output weight of the additive node is $\beta^{\mathrm{L}} = \frac{\left\|\beta^{\mathrm{L}-1}\right\|^2}{\left\|\beta^{\mathrm{L}-1}\right\|^2+1}$, the norm $\|\boldsymbol{\beta}\|$ is minimum. Therefore, we try to choose larger $\beta_L = \frac{\left\|\beta^{\mathrm{L}-1}\right\|^2}{\left\|\beta^{\mathrm{L}-1}\right\|^2+1}$, which means larger $\left\|\beta^{\mathrm{L}-1}\right\|$.

If a large enough $\left\|\beta^1\right\|$ is set in the neural network, the output of the additive node can also be large. In the initiate phase, K neurons are generated randomly as $h_1, \ldots, h_K$, $\beta_i = \frac{\mathbf{E}\cdot[\mathbf{E}-(\mathbf{F}-\mathbf{H}_i)]^{\mathrm{T}}}{[\mathbf{E}-(\mathbf{F}-\mathbf{H}_i)]\cdot[\mathbf{E}-(\mathbf{F}-\mathbf{H})_i]^{\mathrm{T}}}$, $i = 1,\ldots, K$, larger β will be chosen as the initiate neuron.

Based on the random search strategy, the probability of finding a neuron with output weight $\frac{\left\|\beta^{\mathrm{L}-1}\right\|^2}{\left\|\beta^{\mathrm{L}-1}\right\|^2+1}$ is nearly zero. Thus, it is not necessary to pursue the minimum $\left\|\beta^L\right\|$. Like the random search in EI-ELM, a maximum search time k is set, the neurons with more training error decrement and the least norm is added into the network. In other words, these neurons must exists in the circle with the center located as $\beta^{\mathrm{L}} = \frac{\left\|\beta^{\mathrm{L}-1}\right\|^2}{\left\|\beta^{\mathrm{L}-1}\right\|^2+1}$, γ as the radius. In most cases, the output weights near center $\frac{\left\|\beta^{\mathrm{L}-1}\right\|^2}{\left\|\beta^{\mathrm{L}-1}\right\|^2+1}$ are more likely to be chosen.

3 Dynamic Convex Incremental Extreme Learning Machine

In this section, we introduce the Convex Incremental Extreme Learning Machine (DCI-ELM). We merge the pruning of useless neurons into the process of adding new neurons. So we can delete the useless or inefficient neurons from the ELM network

earlier, simplify the neuron network structure as early as possible, and make the most effort to get the most compact and efficient ELM hidden structure.

The less error the new added neurons using CI-ELM bring to the ELM network training process, the more efficient the network will be. But this may lead to the decrease of the effectiveness of other neurons in the hidden layer. Meanwhile, the method that CI-ELM construct the front-feedback single hidden layer network can be treated as ordering a sequence of neurons in the hidden layer. So trying some kinds of ordering method for each new added neuron can maximally mining its effectiveness. Thus, for the ELM network with i neurons in the hidden layer, there would exist a set Φ_i with size V_{max} recording the network hidden layers with currently smallest training error for the i neurons in the hidden layer.

Before the processing of DCI-ELM, all the sets can be treated as not existing. At this time, no ELM network with any size of hidden layer is constructed. The first step of DCI-ELM is to construct an ELM network $\Psi_1^{(1)}$ with only one neuron in its hidden layer. Using the same training method of the first neuron with CI-ELM, we get its corresponding weight in the output layer. Then, $\Psi_1^{(1)}$ is treated as an element and add to the set Φ_1. We update the set Φ_1 with the maximal training error $\left\|\mathbf{E}(\Phi_1)\right\|_{\max}$ and size $V_1 = 1$.

If the constructed single-hidden-layer feed-forward neural network $\Psi_1^{(1)}$ cannot satisfy the given training target, then DCI-ELM randomly generate a neuron $\mathbf{H}_2$, $\mathbf{H}_2 = [G(\mathbf{a}_2, b_2, \mathbf{x}_1), G(\mathbf{a}_2, b_2, \mathbf{x}_2), \ldots, G(\mathbf{a}_2, b_2, \mathbf{x}_N)]^T$. At this time, for the neuron $\mathbf{H}_2$, there exist two choices: (1) construct a new ELM network $\Psi_1^{(2)}$ containing only $\mathbf{H}_2$; (2) add $\mathbf{H}_2$ into ELM $\Psi_1^{(1)}$ and construct a new network $\Psi_2^{(1)}$ containing two neurons in the hidden layer. To mostly utilize the generated neurons to construct the optimal ELM network, and mine the effectiveness of each neuron in the hidden layer, $\mathbf{H}_2$ will conduct these two processes. At the same time, to make each set full faster and drive the compete among them to select preferable middle networks, the update strategy of the sets is bottom up, i.e., the adding order of the new generated neurons is $\Phi_1, \Phi_2, \ldots$ Then, when there is an ELM network reaches the training accuracy, the algorithm halts. This ensures that the simpler and efficient ELM networks are chosen in prior. Thus, h_2 firstly constructs the ELM network, and then is added to the set Φ_1:

$$\Psi_1^{(2)} \in \Phi_1, V_1 = V_1 + 1 \tag{8}$$

where, $\Phi_1 = \{\Psi_1^{(1)}, \Psi_1^{(2)}\}$. If $\exists \Psi_1 \in \Phi_1$, makes $\| \mathbf{E}(\boldsymbol{\Psi}_\mathbf{1}) \| < \varepsilon$, then it means we have constructed an appropriate ELM network, and the algorithm halts. Otherwise, based on the elements in Φ_1, we add neuron $\mathbf{H}_2$ to these ELM networks. We use the following equations to calculate the corresponding output weight β_2 of $\mathbf{H}_2$, and the training error E of each new generated network:

$$\beta_2 = \frac{\mathbf{E} \cdot [\mathbf{E} - (\mathbf{F} - \mathbf{H}_2)]^{\mathrm{T}}}{[\mathbf{E} - (\mathbf{F} - \mathbf{H}_2)] \cdot [\mathbf{E} - (\mathbf{F} - \mathbf{H})_2]^{\mathrm{T}}} \tag{9}$$

$$\mathbf{E} = (1 - \beta_2)\mathbf{E} + \beta_2(\mathbf{F} - \mathbf{H}_2) \tag{10}$$

Thus, we can obtain two more complex networks $\Psi_2^{(1)} = \Psi_1^{(1)} + \beta_2^{(1)}\mathbf{H}_2$ and $\Psi_2^{(2)} = \Psi_1^{(2)} + \beta_2^{(2)}\mathbf{H}_2$, and add them to the set Φ_2, $\Phi_2 = \{\Psi_2^{(1)}, \Psi_2^{(2)}\}$.

When the Lth neurons are generated if the volume of each set $V_{max} < L$, all the sets are full, and the set containing the most complex ELM network is Φ_{L-1}, in which the hidden layer of each network has L-1 neurons. For the sets $\Phi_1, \Phi_2, \ldots, \Phi_{L-1}$, the neuron $\mathbf{H}_L$ are added to the middle network of these sets in order. Thus, compared to the original network, in the new generated middle network, the more accurate network remains in the set and continue to be increased, and the ones with less accuracy are pruned. Thus, for each time we add neurons and generate a new single-hidden-layer feed-forward neural network, it has the following comparison process with the existing sets that contain the same size of neurons in the hidden layer:

If $V_i = V_{max}, i < L$, and let k=1, when $k \leq V_{max}$,

$$\Psi_i^{\max} = \begin{cases} \tilde{\Psi}_i^{(k)}, if \left\|\mathbf{E}\left(\Psi_i^{\max}\right)\right\| > \left\|\mathbf{E}\left(\tilde{\Psi}_i^{(k)}\right)\right\| \\ \Psi_i^{\max}, if \left\|\mathbf{E}\left(\Psi_i^{\max}\right)\right\| \leq \left\|\mathbf{E}\left(\tilde{\Psi}_i^{(k)}\right)\right\| \end{cases} \tag{11}$$

Here, Ψ_i^{max} presents the element whose training error is the largest in the set Φ_i; $\tilde{\Psi}_i^{(k)}$ presents the kth result in the single-hidden-layer feed-forward neural network. Thus, this kind of comparison or competition makes the accuracy of any neuron in each set keeps increasing, so that we can obtain the network meeting the training target as soon as possible. When the Lth neuron is added to the set Φ_{L-1}, the set Φ_L is empty. So after adding a neuron to the set Φ_{L-1}, all the generated V_{L-1} single-hidden-layer feed-forward neural networks will be added to the set Φ_L.

Therefore, when any single-hidden-layer feed-forward neural network added into any set will active the comparison with the training target ε. If the ones whose training error is smaller than the training target ε have existed in the set, the algorithm halts. To consolidate the generalization ability of ELM, for the situation in which more than one single-hidden-layer feed-forward neural networks reach the target, we will choose the one in which $\|\beta\|$ is the smallest. In order words, the final chosen structure of ELM is as follows:

$$\Psi = \Psi_i, if \left\|\mathbf{E}(\Psi_i)\right\| \leq \varepsilon \ and \ \forall\Psi \in \Phi_i, min\|\boldsymbol{\beta}\| \tag{12}$$

4 Performance Evaluation

In this Section, we test the algorithms proposed in the former using extensive experiments. The tests focus on three targets: proceeding time, generalization ability, and the number of neurons in the hidden layer. Meanwhile, we compare the algorithms, PCI-ELMEPCI-ELM, and DCI-ELM, introduced in the former, with the existing typical ELM algorithms, such as I-ELMEI-ELM and CI-ELM. All the evaluations were carried out in MATLAB R2009a in a Intel Core *i*3 processor with 3.3 GHz and 4GB RAM. A group of real datasets [12] about the regression problem are used to test the performance the PCI-ELM, EPCI-ELM, and DCI-ELM algorithms.

To ensure the effectiveness of the training results, each dataset is separated in to training data and testing data using a proportion of 2:1, so that the training data and testing data are independent and not repeated. At the same time, before using these data to train the networks, the input data in all dimensions are normalized into a range of [–1,1] according to the following equation.

$$Input(:,i) = \frac{Input(:,i) - \min(Input(:,i))}{\max(Input(:,i)) - \min(Input(:,i))} \times 2 - 1 \tag{13}$$

Here, Input means the matrix of the input data, and Input$(:,i)$ means a line of the input matrix, i.e., all the input data in the same attribute or dimension.

We will analyze the experiment results of I-ELM, CI-ELM, PCI-ELM, and DCI-ELM algorithms. We compare the training time of different algorithms over the same dataset, predicting time, and the number of neurons in the hidden layer w.r.t the learning problems.

To avoid the occasionality of a single experiment result, in this paper, all our experiments over each dataset are conducted 30 times and calculate the average value as the final result. Moreover, for each algorithm, the number of neurons is counted from 0 to 1000. Meanwhile, when EI-ELM and EPCI-ELM searching for prior neurons, we set the maximal searching times and the size of sets both as 5. Thus, for the given training target, if the training result cannot reach , the result is also recorded when the number of neurons in the hidden layer reaches 1000.

Figures 1 and 2 show the results of using sigmoid function and RBF function in different algorithms respectively. Even though the effect would be somehow different when using different excitation function, in general, the training time has obvious similarity between the two experiments. As we delete useless or inefficient neurons during the process of training, the training time of PCI-ELM is longer than that of CI-ELM conducted over the same dataset. As EPCI-ELM as to PCI-ELM is just like EI-ELM as to I-ELM, there exists a more strategy to select the neurons, and thus the training time of EPCI-ELM is longer than that of PCI-ELM. However, this is not absolutely. Because the searching of neurons can speed up the convergence of algorithm, there exists such situation that EPCI-ELM is faster than PCI-ELM or even CI-ELM. For the same reason, similar situation also happen in the process of EI-ELM and I-ELM.

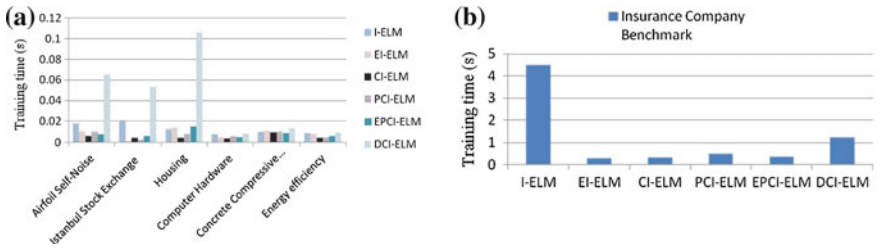

Fig. 1 Comparison chart of training time by using sigmoid function. **a** Without insurance company bench-mark. **b** Insurance company benchmark

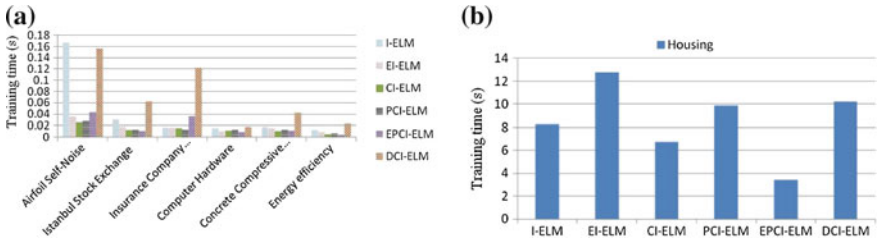

Fig. 2 Comparison chart of training time by using RBF function. **a** Without housing. **b** Housing

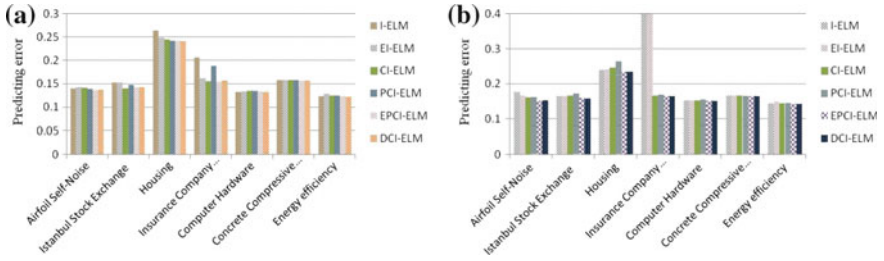

Fig. 3 Comparison chart of predicted results. **a** Sigmoid function. **b** RBF function

From the experiment results, we can see that the training time has similarity when using sigmoid or RBF function. Usually, EI-ELM and EPCI-ELM enhance the selection of neurons in the hidden layer, so it will spend more training time. But this is not absolute. As EI-ELM and EPCI-ELM speed up the convergence of algorithms, they may be faster sometimes. So as to DCI-ELM, even though training more than one networks spends more training time, it also speed up the convergence when constructing the hidden layer structures.

Figure 3 shows the predicting results of the algorithms conducted over different datasets. Firstly, we can see that the pruning of hidden layer structures in CI-ELM influences little in the approximate ability. Secondly, as optimization methods for generalization ability exist in EPCI-ELM and DCI-ELM, the predicting results of these two algorithms is not worse or even better than those of CI-ELM.

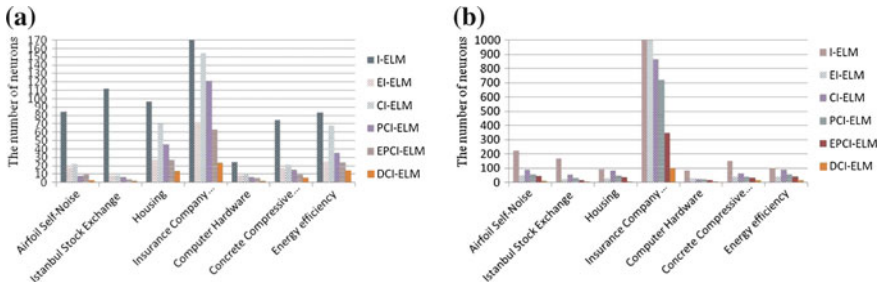

Fig. 4 Comparison chart of hidden layer scale. **a** Sigmoid function. **b** Sigmoid function

From Fig. 4, we can see that the pruning that PCI-ELM providing to CI-ELM largely decrease the redundancy when constructing the networks suing CI-ELM, so that the hidden layer is more compact. EPCI-ELM improves the searching strategy of the neurons in the hidden layer, so that it further avoiding the attendance of redundant neurons. In other words, it optimize the constructed single-hidden-layer feed-forward neural networks from the very beginning. The DCI-ELM makes an adjustment to the network construction in a largest degree. In other words, it mines the most compact structure constructed by the generated neurons in a largest degree, so that the needed neurons is fewer and make the ELM predicting efficient.

5 Conclusion

In this paper, we firstly introduce the ELM theory and its typical dynamic construction algorithms, and analyze the corresponding idea of structure selection. Then, we propose two hidden layer structure optimization algorithms: PCI-ELM and EPCI-ELM. We further adjust the hidden layer structure constructed by CI-ELM to make it more compact and efficient. After this, we propose a DCI-ELM algorithm, and provide a more dynamic hidden layer structure adjustment algorithm, so that it can allocate the pruning to the process of adding neurons. Finally, we take extensive experiments over real datasets and analyze the effectiveness of our proposed algorithms.

References

1. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: theory and applications. Neurocomputing **70**(1), 489–501 (2006)
2. Rong, H.J., Ong, Y.S., et al.: A fast pruned-extreme learning machine for classification problem. Neurocomputing **72**(1), 359–366 (2008)

3. Miche, Y., Sorjamaa, A., et al.: OP-ELM: optimally pruned extreme learning machine. Neural Netw. **21**(1), 158–162 (2010)
4. Feng, G., Huang, G.B., et al.: Error minimized extreme learning machine with growth of hidden nodes and incremental learning. Neural Netw. **20**(8), 1352–1357 (2009)
5. Zhang, R., Lan, Y., Huang, G.B., et al.: Dynamic extreme learning machine and its approximation capability. Cybernetics **43**(6), 2054–2065 (2013)
6. Feng, G., Lan, Y., et al.: Dynamic adjustment of hidden node parameters for extreme learning machine. Cybernetics **45**(2), 279–288 (2015)
7. Huang, G.B., Zhou, H., et al.: Extreme learning machine for regression and multiclass classification. Cybernetics **42**(2), 513–529 (2012)
8. Liang, N.Y., Huang, G.B., et al.: A fast and accurate online sequential learning algorithm for feedforward networks. Neural Netw. **17**(6), 1411–1423 (2006)
9. Huang, G.B., Chen, L.: Enhanced random search based incremental extreme learning machine. Neurocomputing **71**(16), 3460–3468 (2008)
10. Huang, G.B., Chen, L.: Convex incremental extreme learning machine. Neurocomputing **70**(16), 3056–3062 (2007)
11. Bartlett, P.L.: The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. Inform. Theory **44**(2), 525–536 (1998)
12. Blake, C., Merz, C.: UCI Repository of Machine Learning Databases, Department of Information and Computer Sciences, University of California, Irvine, USA, 1998. http://archive.ics.uci.edu/ml/datasets.html