IEICE TRANS. INF. & SYST., VOL.E90–D, NO.4 APRIL 2007

PAPER

Incremental Leaning and Model Selection for Radial Basis Function Network through Sleep

Koichiro YAMAUCHI[†a)], ***Member and*** **Jiro HAYAMI**[†], ***Nonmember***

SUMMARY The model selection for neural networks is an essential procedure to get not only high levels of generalization but also a compact data model. Especially in terms of getting the compact model, neural networks usually outperform other kinds of machine learning methods. Generally, models are selected by trial and error testing using whole learning samples given in advance. In many cases, however, it is difficult and time consuming to prepare whole learning samples in advance. To overcome these inconveniences, we propose a hybrid on-line learning system for a radial basis function (RBF) network that repeats quick learning of novel instances by rote during on-line periods (awake phases) and repeats pseudo rehearsal for model selection during out-of-service periods (sleep phases). We call this system Incremental Learning with Sleep (ILS). During sleep phases, the system basically stops the learning of novel instances, and during awake phases, the system responds quickly. We also extended the system so as to shorten the periodic sleep periods. Experimental results showed the system selects more compact data models than those selected by other machine learning systems.

key words: *incremental learning, sleep learning, model selection, RBF*

1. Introduction

In 1974, Akaike showed that the suitability of a statistical model for a target dataset depends on the number of parameters in the the model and its error, and he proposed an information criteria "AIC" to measure the model's suitability. Later, his method was improved so it could be used to design learning machines [1]–[4]. According to these methods, a compact learning machine, which has no redundant parameters, confers a high level of generalization. Moreover, a compact learning machine also saves resources. However, because the appropriate number of parameters is initially unknown, it has to be found through trial and error learning using whole samples. This trial and error learning task is usually called "model selection."

Actual applications using machine learning systems usually need to achieve incremental learning in which they learn instances in an on-line manner. In this learning scheme, the machines are asked to memorize new instances immediately after their presentation without forgetting old memories. Moreover, there are no guarantees that the instances presented in an on-line manner are independently and identically distributed instances (iid), which is an essential condition to approximate off-line learning effects in an on-line manner. In such environments, the learning with "model-selection" to get the simplest data model is even harder since the past samples are lost.

To overcome this problem, we present a hybrid learning system capable of both incremental learning and model selection.

The appropriate number of parameters, however, is highly dependent on the learning machine architecture. For example, the usual way to achieve incremental learning is by using instance-based learning methods, such as k-nearest neighbors and the generalized regression neural network (GRNN) [5], [6], but they often waste huge amounts of resources to learn instances by rote. Although nearest neighbor editing (NN-editing) [7] prunes redundant records after learning, it still needs a large amount of resources to achieve accurate recognition. On the other hand, the traditional radial basis function (RBF) [8] and multi-layered perceptrons can construct more compact data models if they employ an effective pruning strategy [9], [10]. However, these pruning algorithms do not support incremental learning. Moreover, while the evolving fuzzy neural network (EFuNN) [11] and receptive field weighted regression (RFWR) [12] do achieve incremental learning together with pruning, their resources saving effect is relatively small compared with the RBF with a pruning algorithm that is we report on in this paper.

In this paper, we propose a new learning system that achieves both incremental learning and model selection for the conventional RBF. The RBF is more plausible than multi-layered perceptrons in terms of the growing and pruning strategies of the network, so we employ it as the main architecture of our system. The new system has following properties:

- Incremental learning of non-independent instances, which are correlated with time, without forgetting old memories.
- Dual parallel learning architecture for simultaneous model selection and learning of new instances.

First, we propose the base model of the new system, which is called 'Incremental Learning with Sleep 1 (ILS1).' ILS1 repeats new instance learning and model selection learning alternately. During the model selection learning, ILS1 does not learn new instances.

Note that ILS1 is an incremental learning method inspired by biological learning behavior rather than a model of biological sleep learning. So, the noun 'sleep' used in the name of this system refers to an analogy of biological sleep

Manuscript received December 21, 2005.
Manuscript revised August 22, 2006.

[†]The authors are with the Graduate School of Information Science and Technology, Hokkaido University, Sapporo-shi, 060–0814 Japan.
a) E-mail: yamauchi@complex.eng.hokudai.ac.jp

DOI: 10.1093/ietisy/e90–d.4.722

behavior.

Next, we extend this model; ILS2 simultaneously achieves new instance learning and model selection learning because it uses a dual parallel learning architecture.

Section 2 discusses related work. Section 3 outlines ILS1, and Sects. 4 and 5 explain the two phases in the ILS1 learning procedure. Section 6 explains ILS2, which enables the learning of novel instances during the sleep phase. Section 7 shows an example of ILS's behavior, and Sect. 8 discusses the benchmark test results.

2. Related Work

Hoya and Chambers proposed a variant of the generalized regression neural network (GRNN) for clustering [13]. It has two learning stages: growing and shrinking. In the growing stage, the system adds hidden units for misclassified samples, and during the shrinking stage, it prunes redundant units. They showed that the system performance improved after the redundant units were removed. The main difference between their work and ours is that theirs is not for continuous function approximation whereas ours is. The design principle of a learning system for continuous function approximation is quite different from that of a clustering system.

Kasabov proposed a network similar to GRNN for a fuzzy system [11], which they called "EFuNN." The EFuNN is a fuzzy neural network, and its architecture is very similar to the GRNN, in which the output values are calculated by normalizing consequent results of fuzzy rules of which preconditions are matched to current inputs. Thus, it can support regression problems. It learns instances incrementally by adding new hidden units, and it prunes out redundant units by aggregating similar units.

We consider that the network architectures of Hoya's system and EFuNN are variants of the "table look up" method that reduces redundant records. This method sometimes need too many resources to approximate a target function due to the look up table's structural restriction, as is shown in this paper's experimental section.

Hayashi et al. proposed an improved version of adaptive resonance theory (ART), called the TAM-network [14]. Their system performs incremental learning and pruning of redundant connections in the network. First, it learns new inputs based on ART's learning method. After the learning, it starts the pruning process. It is a fuzzy neural network, so we can extract the reasoning rule from the resultant network, as can be done in the EFuNN. It is also for clustering and is not for regression.

RFWR [12], which is similar to the normalized Gaussian network [15], convenes localized linear models to approximate the target function. Although it is more adaptable than GRNN, it needs too many hidden units.

French [16] and Ans and Roussert [17] have proposed architectures that are similar to our system, i.e., dual network architectures to achieve incremental learning. Their models consist of two multi-layer perceptrons. If a new instance is assigned, one side of the network learns the new instance interleaved with pseudo instances generated by the other side of the network. After the learning, the learned new instance is copied or exchanged with the other side of the network. These models, however, do not support model selection learning. Moreover, their pseudo samples are not easily generated from the perceptrons, so one of the models needs an additional associative memory to recall old instances.

Yamauchi et al. [18]–[21] proposed models for quick learning of new instances, model selection, and recognition. These models basically consist of two neural networks: a fast learning network (F-Net) and a slow learning network (S-Net). The F-Net learns new instances quickly to compensate for errors of the S-Net. The final output is the sum of both the S- and F-Net outputs and is always near the desired output. Although the systems quickly respond to new instances during the awake phase, they sometimes fail to learn non-independently distributed inputs during the awake phase for certain parameter settings. This is because the F-Net employs normal gradient based learning. In contrast, the system proposed in this paper overcomes this problem by using a MGRNN [6] as the F-Net. Although preliminarily results of ILS1 in this paper are reported in [22], the new system architecture ILS2 is an improvement on the previous models in which dual parallel learning of new instances is enabled.

The incremental learning method for support vector regression (SVR) is a support vector machine (SVM) for regression, namely accurate online support vector regression (AOSVR) [23]. Using this method, the learning system detects the optimal number of hidden units to achieve the ability of generalization. Although AOSVR performs incremental learning in a support vector regression manner, its generalization ability is very dependent on the choice of kernel, which is the same limitation as that of the original SVM. The kernel has to be manually designed using prior knowledge [24] or by trial and error testing. For example, the variance of each kernel is fixed a certain width so that the SVR sometimes needs a large number of support vectors to achieve accurate learning. Moreover, AOSVR needs to store all instances even if they are not the "support vectors," because these will be used as support vectors in later learning steps. On the other hand, our system does not need to store past instances because the model selection learning uses pseudo instances generated by the internal parameters.

Umano et al. also proposed a Fuzzy system that aims to realize sequential construction of rules [25]. Their system, however, is not for regression and the structure and learning methods are quite different from those of ours.

3. ILS1

This section describes the basic incremental learning method with sleep 1 (ILS1). ILS1 is designed to build a compact data model through incremental learning. To build the compact data model, ILS1 employees a normal radial

basis function network (RBF) as the core learning machine. Note that the local internal representation of RBF is suitable for the incremental learning [26]. Moreover, RBF allow a nested internal representation using several radial function so that RBF is also suitable for building the compact data model. Although other models such as GRNN [5], EFuNN [11] and RFWR [12] are also suitable for incremental learning, their internal representation make it hard to build a nested one so that the amount of resources they require is usually larger than that of RBF.

Actually, the model selection of RBF is a time-consuming trial and error process. Thus, by itself, it is unsuitable for achieving quick responses in incremental learning. To avoid this inconvenience, the system has two learning phases: awake and sleep. During the awake phase, the system quickly learns new instances by rote, while it achieves model-selection to get a compact RBF during the sleep phase. We believe that the sleep phase can be made to coincide with the out-of-service period so this time-consuming process should not cause inconvenience to the user.

To perform the above tasks, the system consists of three networks: a main network (M-Net), a fast-learning network (F-Net), and a slow-learning network (S-Net) (Fig. 1). The M-Net and S-Net are consist of radial basis function network. The M-Net is a fixed network that keeps the network structure acquired during the latest sleep phase. The M-Net is also used to get precise output values for the recognition during the preceding awake phase and for getting pseudo instances for the next sleep phase.

During the awake phase, novel instances are presented to the system. If the M-Net error for the current instance exceeds a threshold, the F-Net learns the current input-output pair. The F-Net consists of an MGRNN [6]. This procedure achieves one-pass learning of new inputs like that of nearest neighbors and smoothly interpolates the outputs between instances[†].

The system output is the weighted sum of the F- and M-Net outputs. The weight is determined according to the outputs of the F-Net's hidden units. If the outputs are large, the F-Net's weight is large while M-Net's weight is small. As a result, the system output becomes close to the desired one immediately after encountering the novel instance.

During the sleep phase, the S-Net should learn the instances stored in the F- and M-Net with reduction its number of hidden units. In terms of the quality of the learning, S-Net learning method should be an offline learning method with pruning strategy. Because, the quality of resulting S-Net using the offline learning is better than that of online learning method. The offline learning method, however, wastes huge amount of resources to store all instances in the storage space. So, during the sleep phase, the S-Net learns the pseudo instances generated by the F- and M-Nets in online manner. The desired output for the S-Net is the weighted sum of the M- and F-Net outputs. The S-Net learns the pseudo instances so as to minimize not only its error but also its parameters. Reducing the parameters in this way

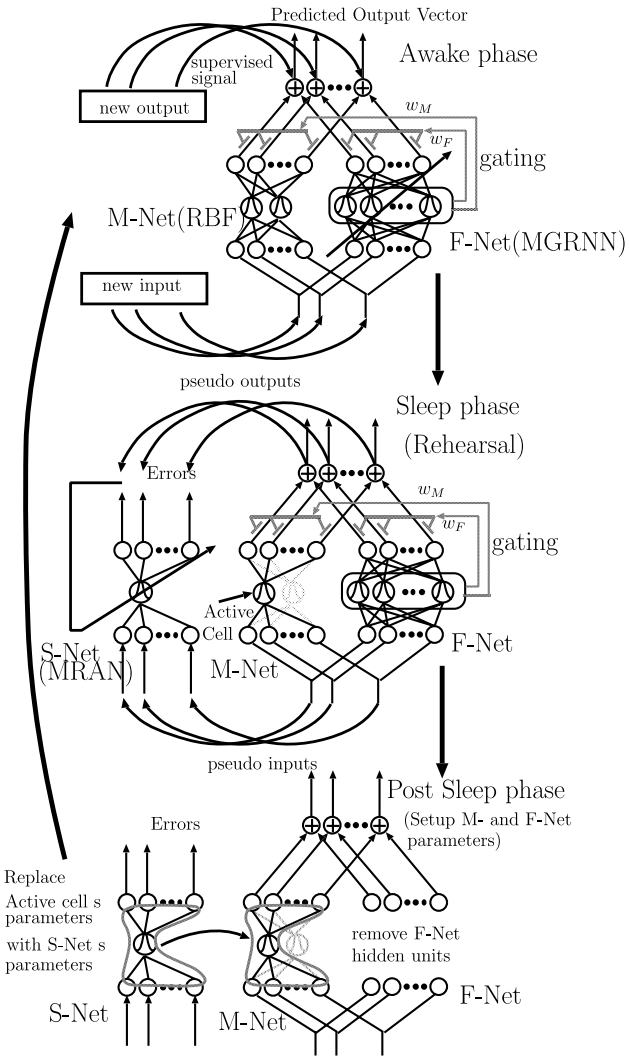

Fig. 1 Structure and behavior of system.

should improve the S-Net's generalization ability. The S-Net's learning method is a modified version of MRAN [9], which uses a growing strategy that prunes redundant cells, as will be described later. The modified MRAN performes stable learning, even if the distribution of inputs is unstable. The S-Net continues learning until its outputs are sufficiently close to those of the pseudo outputs. Note that the pseudo instances are generated as i.i.d samples.

To avoid wasting computational power for unnecessary learning, only some of the total instances are used as the pseudo instances for the S-Net learning process. Therefore, the pseudo instances are generated from the hidden units in the F-Net and some of the M-Net's hidden units, which are activated during the preceding awake phase. This means the pseudo instances are the novel instances interleaved with old instances, which are to be interfered from the incremen-

[†]Our previous version of the system has the Fnet which consists of Resource allocating network (RBF). However, RAN is interfered with by the learning of new instances. To avoid this drawback, the new system employs MGRNN.

tal learning. After the S-Net learning(post sleep phase in Fig. 1), the M-Net's parameters are basically replaced by the S-Net's parameters. As a result, the M-Net becomes a more suitable network for the recognition. After that, all hidden F- and S-Net units are removed.

The F-Net then begins to learn new instances again.

However, the S-Net needs a long learning period, which is much longer than that of the F-Net, to complete learning, meaning that the pruning process of redundant hidden units is much slower than that of growing process of the F-Net's hidden units. Therefore, if the system learns the S- and F-Nets simultaneously, the number of hidden units would increase until the system approximates the total volume of the input space. To solve this problem, F-Net learning is suspended during S-Net learning. Consequently, the system stops reading new novel instances during the learning period, and this results in alternating learning processes for the S- and F-Nets. The learning process is repeated until no more new instances are recorded by the F-Net. Although the total learning period required for this system is not reduced in comparison with that of other learning systems, the system does show very quick learning behavior during the awake phase. Therefore, in practice, we can use it as a quick learner during the awake phase, if its S-Net learning is accomplished during out-of-service periods.

However, if the average interval of new instances is sufficiently large, it is also possible that the system will complete the S-Net learning during the awake phase. In this case, system needs one more F-Net (see Sect. 6).

In the following, $\boldsymbol{f}^*_M(\boldsymbol{x}, \boldsymbol{\theta}_M)$, $\boldsymbol{f}^*_f(\boldsymbol{x}, \boldsymbol{\theta}_f)$ and $\boldsymbol{f}^*_s(\boldsymbol{x}, \boldsymbol{\theta}_s)$ denote the respective output vectors of the M-Net, F-Net and S-Net. Here, $\boldsymbol{\theta}_*$ and $\boldsymbol{x}$ denote the parameter vector of the networks and the input vector, respectively. The M-Net and S-Net outputs are the same as that of RBF. For example, the i-th output of the M-Net is

$$f^*_{Mi}(\boldsymbol{x}, \boldsymbol{\theta}_M) = \sum_{M\alpha} c^M_{i\alpha} \phi_{M_\alpha}(\boldsymbol{x}), \tag{1}$$

where

$$\phi_{M_\alpha}(\boldsymbol{x}) \equiv \exp\left(-\frac{\left\| \boldsymbol{x}_j - \boldsymbol{u}_{M\alpha} \right\|^2}{2\sigma^2_{M\alpha}} \right). \tag{2}$$

Here, $c^M_{i\alpha}$ is the connection strength between the $M\alpha$-th hidden unit and the i-th output cell and $\phi_{M_\alpha}(\boldsymbol{x})$ denotes the output value from the $M\alpha$-th hidden unit, $\boldsymbol{t}_{M\alpha}$ and $\sigma_{M\alpha}$ are the reference vector and kernel width.

On the other hand, the F-Net output is the MGRNN output:

$$f^*_{Fi}(\boldsymbol{x}, \boldsymbol{\theta}_F) = \frac{\sum_{F\alpha} c^F_{i\alpha} \phi_{F_\alpha}(\boldsymbol{x}) / \sigma_{F_\alpha}}{\sum_{F_j} \phi_{F_j}(\boldsymbol{x}) / \sigma_{F_j}} \tag{3}$$

where $\phi_{F_j}(\boldsymbol{x})$ is the output of each hidden unit, which has the same form as Eq. (2).

4. Awake Phase

During the awake phase, the system recognizes familiar inputs by calculating the sum of the outputs from the F-Net and M-Net while the F-Net learns novel instances quickly. Let $\boldsymbol{f}^*(\boldsymbol{x})$ and $\boldsymbol{D}(\boldsymbol{x})$ be the final output vector of the system and the desired output vector to $\boldsymbol{x}$, respectively, where $\boldsymbol{f}^*(\boldsymbol{x})$ is the weighted sum of the F-Net and M-Net outputs:

$$\boldsymbol{f}^*(\boldsymbol{x}) = w_F(\boldsymbol{x}) \boldsymbol{f}^*_F(\boldsymbol{x}, \boldsymbol{\theta}_F) + w_M(\boldsymbol{x}) \boldsymbol{f}^*_M(\boldsymbol{x}, \boldsymbol{\theta}_M) \tag{4}$$

The weight $w_F(\boldsymbol{x})$ and $w_M(\boldsymbol{x})$ are the weights for the F- and M-Net outputs and they vary depending on the activity of hidden F-Net units. Therefore,

$$\begin{aligned} w_F(\boldsymbol{x}) &= \frac{1}{1 + \exp(-A\{\sum_{F_j} \phi_{F_j}(\boldsymbol{x}) - \theta_d\})} \\ w_M(\boldsymbol{x}) &= 1 - w_F(\boldsymbol{x}) \end{aligned} \tag{5}$$

where A denotes a gain that determines the smoothness of the weighting function and θ_d is the threshold for determining the trusted input region for the F-Net's output. We set A and θ_d to 100 and 0.95, which are determined through numerical tests.

The F-Net learns a new unknown instance $(\boldsymbol{x}_n, \boldsymbol{D}(\boldsymbol{x}_n))$ by adding new $N+1$-th hidden unit if $\|\boldsymbol{D}(\boldsymbol{x}_n) - \boldsymbol{f}^*(\boldsymbol{x}_n)\| > \epsilon_E$, where ϵ_E is a threshold.

$$\begin{aligned} c^F_{iN+1} &= D_i(\boldsymbol{x}_n) \\ \sigma_{F_{N+1}} &= \lambda_F \min\left\{ \min_{\alpha \neq N+1} \|\boldsymbol{x}_n - \boldsymbol{u}_{F_\alpha}\|, \min_{\beta} \|\boldsymbol{x}_n - \boldsymbol{u}_{M_\beta}\| \right\} \\ \boldsymbol{u}_{F_{N+1}} &= \boldsymbol{x}_n \end{aligned} \tag{6}$$

where λ_F denotes overlap ratio for the F-Net. Note that the overlap ratio is for determining the kernel width: σ^F_{N+1}. A larger λ_F makes σ^F_{N+1} wider. According to the MGRNN heuristic, λ_F is set to 0.5, with which the slope of the regression curve in the middle between the nearest two samples equals the slope of the connecting line. To do this, the kernel width of the old F-Net unit, which is the nearest to the new allocated hidden unit, needs to be the same width as that of the new kernel. Therefore, if the F_j-th F-Net unit is the nearest unit,

$$\sigma_{F_j} := \sigma_{F_{N+1}} \tag{7}$$

else if the nearest unit is a M-Net unit, we do not touch the M-Net kernel width to keep its memory.

According to the MGRNN algorithm, after this initialization, the kernel width σ_{F_i} should be optimized to minimize the generalization error. However, in this paper, we do not discuss such an optimization process since it is wasteful to optimize the F-Net parameters for temporal use of the F-Net. The above equation is repeated whenever a novel instance appears.

5. Sleep Phase

The learning during the sleep phase aims to modify the M-Net's parameters using the S-Net so as to fit the new instances stored in the F-Net.

Fundamentally, the M-Net should learn the F-Net's instances. However, this learning style causes catastrophic

forgetting of the M-Net. To avoid this, the M-Net must accomplish both the learning of the F-Net's new instances and the re-learning of the pseudo instances recalled from the M-Net itself, simultaneously. During such learning, however, the M-Net's memory is also modified due to the parameter changes, so the M-Net fails to learn correct outputs. To overcome the problem, the M-Net has to recall and store the instances in a learning pattern buffer prior to the re-learning process. However, this strategy has the drawback that the buffer usually wastes a huge amount of storage space.

To solve these problems, in this system, the M-Net itself plays the role of the buffer and the S-Net learns the pseudo instances instead of the M-Net. Therefore, the F-Net and M-Net generate pseudo instance $(\hat{\boldsymbol{x}}, \boldsymbol{D}^*(\hat{\boldsymbol{x}}))$, where $\boldsymbol{D}^*(\hat{\boldsymbol{x}})$ denotes the pseudo desired output vector. Then the S-Net learn the pseudo instances in online learning manner. This is for achieving buffer less learning. Therefore, if the S-Net learning is an offline learning, the system need a buffer to store all the pseudo instances. This is not suitable for our system in terms of saving storage space. So, S-Net learns the pseudo instances in online learning manner.

Prior to the S-Net learning, this system copies all or some of the M-Net's parameters to the S-Net as the initial parameters. To avoid wasting computational power, the M-Net copies the parameters $\boldsymbol{c}_j^M$, $\boldsymbol{u}_j^M$, σ_j^M associated with only the hidden units which are activated during the adjacent awake phase to the S-Net. During the awake phase, each hidden unit in the M-Net checks its "activated-flag" when the unit is activated. If $\phi_{M_i}(\boldsymbol{x})$ exceeds the threshold ϵ_o, then $ActivatedFlag_i = true$. If $ActivatedFlag_i = true$, then the i-th M-Net hidden unit is regarded as "active unit." Below, we call the hidden unit an 'active unit.'

Note that the active units in the M-Net represent the memory that will be interfered with by the learning of the new instances in the F-Net. Therefore, the system realizes effective learning using the pseudo instances generated from the active units (see Fig. 2).

5.1 Pseudo Instances

The $(\hat{\boldsymbol{x}}, \boldsymbol{D}^*(\hat{\boldsymbol{x}}))$ are generated based on the modified version of the method proposed in Ref. [26] as follows. The system randomly chooses one hidden unit from all the hidden units of the F-Net and active units in the M-Net every time one pseudo input is generated. Using this strategy, the system generates independent and identically distributed (iid) pseudo inputs for the S-Net online learning. As a result, the S-Net can learn the F-Net outputs correctly.

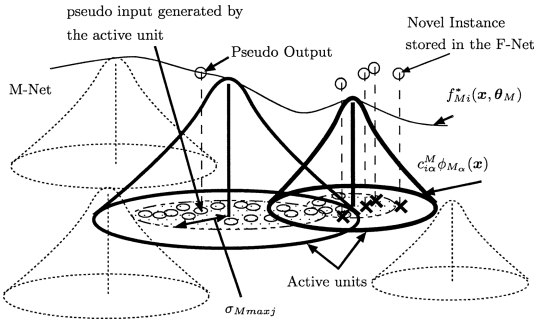

Fig. 2 Pseudo instances generation. The active units are activated by the new instances stored in the F-Net. The system learns the new instances interleaved with the pseudo instances generated by these active units.

The pseudo input vector $\hat{\boldsymbol{x}}$ generation algorithm is varied depending on which network, F- or M-Net the hidden unit belongs to:

$$\hat{\boldsymbol{x}} = \begin{cases} \boldsymbol{u}_{F_j} & \text{if F-Net's hidden unit} \\ \boldsymbol{u}_{M_j} + l\,\sigma_{M\max_j}\boldsymbol{\kappa} & \text{if M-Net's active unit} \end{cases} \tag{8}$$

where $\boldsymbol{\kappa}$ is a random vector with unit length, and l is a random value in the interval $[0, 1]$. Here, $\sigma_{M\max\ j}$ denotes maximum Euclid distance between the j-th hidden unit center and instances. If the j-th M-Net hidden unit center is the closest to the center of the k-th F-Net's hidden unit and $\sigma_{M\max\ j} < \|\boldsymbol{u}_{Mj} - \boldsymbol{u}_{Fk}\|$, then $\sigma_{M\max\ j}$ is updated to be the current distance: $\sigma_{M\max\ j} := \|\boldsymbol{u}_{Mj} - \boldsymbol{u}_{Fk}\|$. This parameter setting is achieved just before a removing process for all F-Net hidden units at the end of sleep phase.

The output vector $\boldsymbol{D}^*(\hat{\boldsymbol{x}})$ is basically the weighted sum of the F- and M-Net outputs without non-active hidden units. Precisely, $\boldsymbol{D}^*(\hat{\boldsymbol{x}})$ should be the outputs minus the M-Net output without active hidden units:

$$\begin{aligned} \boldsymbol{D}^*(\hat{\boldsymbol{x}}) \equiv\ & w_F(\hat{\boldsymbol{x}})\boldsymbol{f}_F^*(\hat{\boldsymbol{x}}) + w_M(\hat{\boldsymbol{x}})\boldsymbol{f}_M^*(\hat{\boldsymbol{x}}, \boldsymbol{\theta}_M) \\ & - \boldsymbol{f}_M^*(\hat{\boldsymbol{x}}, \boldsymbol{\theta}_{Mnon-active}). \end{aligned} \tag{9}$$

where $\boldsymbol{\theta}_{Mnon-active}$ is the parameters of the M-Net associated with the non-active hidden units. This is because, the parameters of the active M-Net's hidden units are to be replaced by the parameters of the S-Net (see 5.3), so the S-Net output must fit to the subtracted value.

5.2 S-Net Learning (sleep phase)

The S-Net learns the pseudo instances $(\hat{\boldsymbol{x}}, \boldsymbol{D}^*(\hat{\boldsymbol{x}}))$. Actually, we can employ various learning methods for the S-Net. The conditions for the S-Net learning method are

- It is an online learning method, and
- The method reduces redundant parameters by a pruning or merging strategy.

The first condition is needed for bufferless learning; and the second is the essential condition for our system.

In this paper, we use a modified version of the MRAN learning algorithm [9], which is an online learning with a pruning strategy. MRAN is superior to other methods in terms of its generalization ability and the smaller quantity of resources consumed during learning. In the pruning strategy, the S-Net prunes any hidden units whose contribution ratio is smaller than a threshold. The contribution ratio of the S_j-th hidden unit is the relative magnitude of outputs, as per the equation below.

$$r_j(\hat{\boldsymbol{x}}) = \|\boldsymbol{c}_j\phi_{S_j}(\hat{\boldsymbol{x}})\| / \max_i \|\boldsymbol{c}_i\phi_{S_i}(\hat{\boldsymbol{x}})\|, \tag{10}$$

for each pseudo instance $(\hat{\boldsymbol{x}}, \boldsymbol{f}^*(\hat{\boldsymbol{x}}))$
  for each S_j-th hidden unit
  if $\|\hat{\boldsymbol{x}} - \boldsymbol{u}_{S\alpha}\| < \sigma_{S\alpha}$ then
    Calculate the contributing ratio $r_{S_j}(\hat{\boldsymbol{x}})$ with Eq. 10
    if $r_{S_j}(\hat{\boldsymbol{x}}) < \delta$ for M consecutive $\hat{\boldsymbol{x}}$ then
      prune the S_j-th hidden unit;

Fig. 3 Pruning algorithm.

where $\boldsymbol{c}_j$ is the connection strength vector between the S_j-th hidden unit and the output units. Note that this equation estimates the contribution ratio for only one input vector $\hat{\boldsymbol{x}}$. To estimate the contribution ratio for all input space roughly, the Minimum RAN algorithm estimates Eq. (10) using several different input vectors. Therefore, if $r_j(\hat{\boldsymbol{x}})$ is less than a threshold for M consecutive pseudo input $\hat{\boldsymbol{x}}$, the hidden unit is pruned. This estimation method is the same as that of the original MRAN [9].

The distribution of pseudo inputs varies depending on the change in the number of hidden units. The original MRAN pruning method, however, usually discards hidden units that are needed when the distribution of $\hat{\boldsymbol{x}}$ inputs varies. To overcome this shortcoming, we slightly modified it to measure the confidence of each hidden unit. In the modified method, the confidence ratio of each hidden unit is only measured when the unit is active. Therefore, the contribution ratio $r_j(\hat{\boldsymbol{x}})$ is only estimated when $\|\hat{\boldsymbol{x}} - \boldsymbol{u}_{S\alpha}\| < \sigma_{S\alpha}$, where $\sigma_{S\alpha}$ is the kernel width of the hidden unit. This restriction means that, the resulting ratio is unaffected by variations in the distribution of pseudo input $\hat{\boldsymbol{x}}$. The summarized pruning algorithm is given in Fig. 3.

In the box, δ denotes a performance-threshold and this is also determined through preliminary experiments to minimize Akaike's Information Criterion (AIC) [27], which evaluates the statistical suitability of linear models for a dataset. According to AIC, the model of minimal number of parameters which are essential for the learning of the dataset is good. Although the AIC is for evaluating linear models, we can use it for rough evaluation of the parameter δ.

In fact, however, δ can be set roughly. As shown in 8, in many cases, ILS1 is superior to other systems even if δ is set to a certain value. In the experiment, we set $\delta = 0.4$, to which the averaged ILS1 performance is almost the best over several datasets.

The S-Net's learning method is that of M-RAN proposed by Yingwei et al. [9] with the modified pruning algorithm, but the extended Kalman filter (EKF) used in the original M-RAN is replaced by the standard gradient descent method, for which Fig. 4 shows the whole S-Net learning procedure. In this figure, e_{rmsn} denotes the root mean square error of S-Net.

The leaning process is repeated $N_{optimize}$ times, where $N_{optimize}$ is set N_o times the number of hidden F-Net units and M-Net's active units. In the experiment described later, N_o was set to 200.

for $n = 1$ **to** $N_{optimize}$ **times**
  Get pseudo pattern $\hat{\boldsymbol{x}}$ from the F- and M-Net.
  $e_{rmsn} := \sqrt{\frac{\sum_{i=t-M}^{T} \|\boldsymbol{f}^*(\hat{\boldsymbol{x}}_i) - \boldsymbol{f}_S^*(\hat{\boldsymbol{x}}_i)\|^2}{M}}$
  $e_n = \max\{\epsilon_{max}\gamma^n, \epsilon_{min}\}$, where $\gamma < 1$ is a decay constant
  if $\min_{S\alpha} \|\hat{\boldsymbol{x}} - \boldsymbol{u}_{S\alpha}\| > e_n$ and
  $\|\boldsymbol{f}^*(\hat{\boldsymbol{x}}) - \boldsymbol{f}_S^*(\hat{\boldsymbol{x}})\| > e_E$ and
  $e_{rmsn} > e'_{rmsn}$ and
  Number of allocated cells is less than that of F-Net
  then
    allocate new cell $S\alpha$
    $\boldsymbol{u}_{S\alpha} = \hat{\boldsymbol{x}}$
    $\boldsymbol{c}_{S\alpha} = \boldsymbol{f}^*(\hat{\boldsymbol{x}}) - \boldsymbol{f}_S^*(\hat{\boldsymbol{x}})$
    $\sigma_{S\alpha} = \min\left(\lambda_S \min_{S\beta} \|\hat{\boldsymbol{x}} - \boldsymbol{u}_{S\beta}\|, \sigma_{\max}\right)$
  else
    $\boldsymbol{u}_{S\alpha} := \boldsymbol{u}_{S\alpha} - \varepsilon\nabla_{\boldsymbol{u}_{S\alpha}} \|\boldsymbol{f}(\hat{\boldsymbol{x}}) - \boldsymbol{f}^*(\hat{\boldsymbol{x}})\|^2$
    $\boldsymbol{c}_{S\alpha} := \boldsymbol{c}_{S\alpha} - \varepsilon\nabla_{\boldsymbol{c}_{S\alpha}} \|\boldsymbol{f}(\hat{\boldsymbol{x}}) - \boldsymbol{f}^*(\hat{\boldsymbol{x}})\|^2$
    $\sigma_{S\alpha} := \sigma_{S\alpha} - \varepsilon\nabla_{\sigma_{S\alpha}} \|\boldsymbol{f}(\hat{\boldsymbol{x}}) - \boldsymbol{f}^*(\hat{\boldsymbol{x}})\|^2$
  endif
  Execute the pruning algorithm(see Figure 3)
endfor

Fig. 4 S-Net learning procedure.

5.3 Post Sleep Phase

After learning with the S-Net, the system removes both active M-Net hidden units and all F-Net hidden units and their associated parameters, after which all parameters of the S-Net are moved to the M-Net. Then, the system restarts the awake phase.

6. ILS2

During the sleep phase, the system normally does not support any other new novel instances. This restriction is intended to fix the output function of the F- and M-Net so that they generate consistent pseudo-instances for the S-Net learning. We can overcome this restriction by making a minor extension to this system.

The extended system has an additional F'-Net, which also consists of an MGRNN-like F-Net (see Fig. 5). Even if the system is in the sleep phase, the M-Net and F-Net can predict the output vector to the current input vector by suspending the sleep learning process for a moment. If the predicted output vector is not correct, the F'-Net learns the new novel instance. F'-Net learning rule is almost the same as that of F-Net, but each hidden unit's kernel width is adjusted according to both the M- and F-Net hidden units. Therefore,

$$\sigma_{F'_{N+1}} = \lambda_{F'} \min\left\{\min_{\alpha} \|\boldsymbol{x}_n - \boldsymbol{u}_{F_\alpha}\|, \min_{\beta} \|\boldsymbol{x}_n - \boldsymbol{u}_{M_\beta}\|, \min_{\gamma \neq N+1} \|\boldsymbol{x}_n - \boldsymbol{u}_{F'_\gamma}\|\right\} \quad (11)$$

where $\sigma_{F'_{N+1}}$ denotes the kernel width of $N+1$-th F'-Net hidden unit and $\boldsymbol{u}_{F'_\gamma}$ denotes γ-th F'-Net center.

The output vector of the system is the weighted sum of the F'-Net and the original system's output vectors. The

weights are calculated in the same manner as Eq. (5). In the extended system, all the F'-Net's hidden units are moved to F-Net when the number of the F'-Net's hidden units reaches the upper limit and the sleep phase has finished. However, the kernel witdh's of the moved hidden units should be re-adjusted according to the M-Net hidden unit's centers, which are determined by the last sleep phase. To achieve this, ILS2 moves F'-Net's hidden units one by one, and adjusts moved hidden unit's kernel-width using Eq. (6), where $\boldsymbol{x}_n$ is replaced by the moved kernel center $\boldsymbol{u}_{F'\alpha}$.

The system performs sleep learning during the intervals between the instance presentations. If the presentation frequency of novel instances is low, the sleep learning is completed during the awake phase. However, if the sleep learning takes a long time and the F'-Net's hidden units reach the upper limit, the system stops learning new instances and focuses on the sleep learning for the S-Net.

Even if such a case occurs, the system can reduce the actual out-of-service period compared with the original system (see Fig. 6). In the following section, this extended ILS is called 'ILS2.'

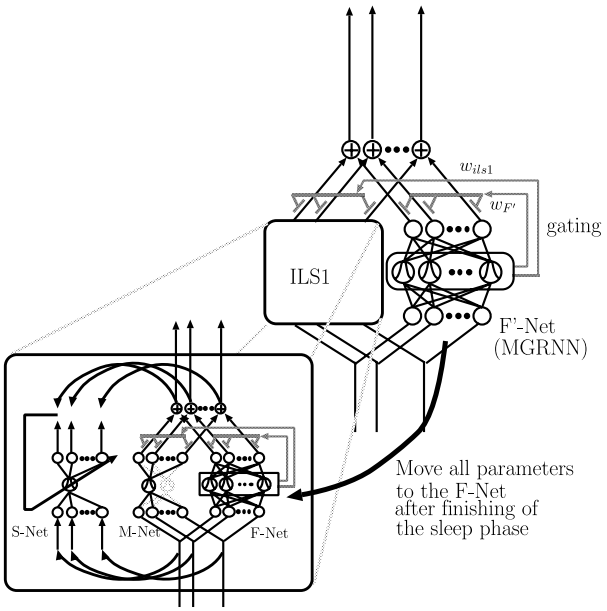

Fig. 5 Extended system: ILS2.

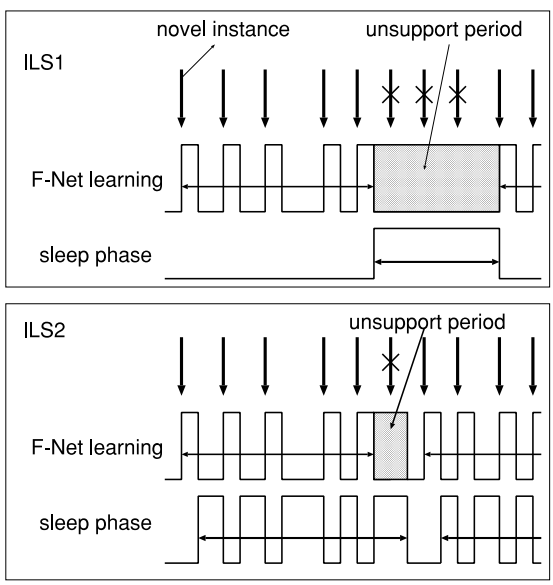

Fig. 6 Examples of ILS1 and ILS2 time charts. (When maximum number of F-Net hidden units is 5.)

7. Examples of System Behaviors

This section presents examples of ILS1 and ILS2 behaviors.

For simplicity, the desired outputs have been set to one-dimensional input and one-dimensional output.

We prepared non-independent samples of a sine curve.

$$y(k) = \sin[x(k)],$$
$$x(k) = 2k\pi/N \quad (k = 1, \cdots, N) \tag{12}$$

where N denotes the number of input variations. To generate non-independent instances, we increased k in steps of 1. During each awake phase, 20 instances were presented to the F-Net. N was set to 100.

Figure 7 shows an example of ILS1 behavior after the 2nd and 3rd awake and sleep phases. We can see that after the 3rd awake phase, the F-Net approximates outputs in the area of the novel instances, and the output curve for the M-Net not only extended the previous curve, but also smoothly followed the new F-Net's curve.

Figure 8 shows the mean square error for all samples and the number of hidden units of the network after each awake and sleep phase. From this figure, we can see that the mean square error immediately after the sleep phase is usually smaller than the error after the awake phase. This means the generalization ability of the system is improved by the S-Net learning, during which the S-Net with fewer redundant hidden units learns the new instances interleaved with pseudo instances. This strategy avoids over-fitting to a small number of instances and also saves resources.

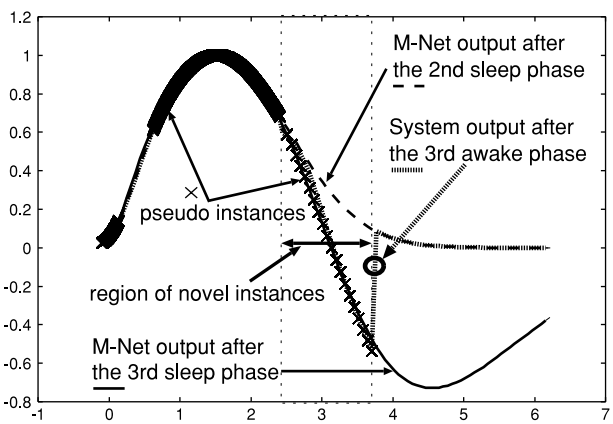

Fig. 7 Examples of ILS1 behavior for non-independent instances: outputs from each network in 2nd and 3rd learning phases.

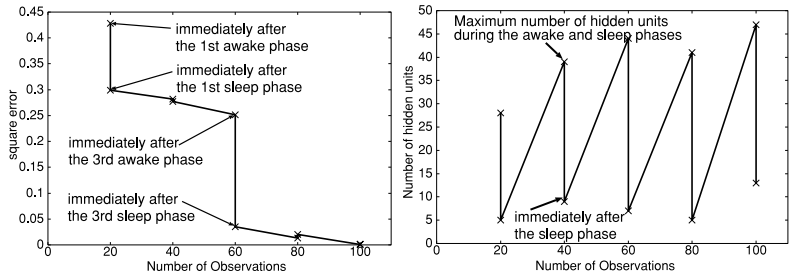

Fig. 8 Mean square error after awake and sleep phases (left) and the number of hidden units during the learning (right).

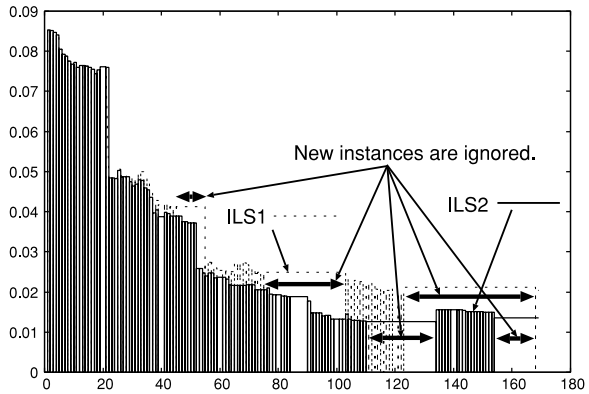

Fig. 9 ILS2 and ILS1 behaviors: mean square error for servo dataset. All samples of servo dataset were presented to the ILS1 and ILS2 one by one at intervals of 1000 msec. Each vertical line indicates the ILS1 or ILS2 learned the new sample at that point. Note that, in the early steps of the learning, ILS2 learned almost all of samples presented whereas ILS1 occasionally ignored some new samples. In the latter steps of the learning, ILS2's S-Net learning period became long because the number of S-Net hidden units was increased. As a result, the F'-Net in ILS2 became full and ILS2 also ignored some new samples.

We constructed the ILS2 algorithm as a java-thread class so that the S- and F'-Net learning could be executed in parallel. Therefore, the S-Net pseudo rehearsal was executed by the corresponding thread class while the F-, M- and F'-Net performed the recognition and learning for the current inputs. To verify that ILS2 supported the learning of new novel instances during the sleep phase, we compared ILS1 and ILS2 behaviors on the servo dataset of UCI Machine Learning Repository. The upper limits of the number of hidden units in F-Net (of ILS1) and F'-Net (of ILS2) were set to 20. All samples of servo dataset were presented to the ILS1 and ILS2 one by one at intervals of 1000 msec, and the system performances were evaluated by the mean square error for the all samples. If system could not support a new sample, the sample passed through the network without any learning process.

Figure 9 shows the mean square errors of ILS1 and ILS2. We can see that ILS2's mean square error was less than ILS1's in the latter steps of the learning. This means that the generalization ability of ILS2 is better than that of ILS1 because the number of instances for the learning of ILS1 is less than that of ILS2. Thus, the generalization ability of ILS2 was better than that of ILS1 because ILS1 had fewer learning instances than ILS2 had.

8. Benchmark Tests

We compared the performance of ILS1 with those of other learning systems on several datasets for regression. In particular, we compared the resources they used to attain their generalization ability.

(1) Competitors

We chose GRNN-editing, EFuNN [11], RFWR [12] and AOSVR [23] as the competitors. Note that GRNN-editing corresponds to nearest neighbor editing for regression. Therefore, it is a generalized regression neural network that prunes redundant units with the nearest neighbor editing method.

(2) Pre-determined parameter

ILS1: Overlap ratio for F-Net $\lambda_F = 0.5$, overlap ratio for S-Net $\lambda_S = 2.5$ and pruning threshold for S-Net $\delta = 0.4$ and other parameters listed in Table A· 1.

GRNN-editing: Variance of radial function $\sigma = 1$, threshold for appending a new unit $\theta_{add} = 0.7$ and error threshold for nearest neighbor editing $\epsilon_{pruning} = 0.01$.

RFWR: Learning parameters for linear models $\lambda = 1$, initial diagonal values for matrix $M = \boldsymbol{I}$, growth threshold $w_{new} = 0.1$, and pruning threshold $w_{prune} = 0.9$.

EFuNN: The fuzzy membership function is constructed from a Gaussian radial function such as RBF hidden units: initial variance is 0.005, aggregation error threshold $E = 0.0001$ and maximum radius $R_{max} = 1$.

AOSVR: The variance of each radial function is 0.5, insensitive width $\varepsilon = 0.1$ and coefficient $C = 10$.

(3) Datasets

We chose the following datasets from the UCI Machine Learning Repository[†]: servo, housing, CPU-performance,

[†]http://www.ics.uci.edu/~mlearn/MLRepository.html

Table 1 Datasets used in the experiment and their corresponding network size.

Name	Size	Network-size
Mackay-Glass	500	4 - 1
CPU-performance	263	6 - 1
Mpg	392	7 - 1
Servo	167	12 - 1
Housing	500	13 - 1

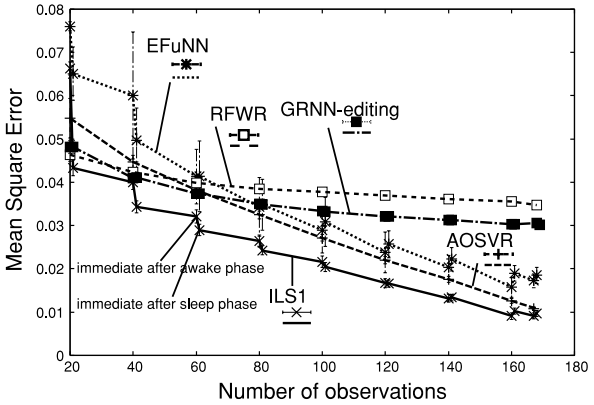

Fig. 10 Mean square error vs number of observations in servo dataset: The error bars indicate 95% confidence intervals.

MPG, and Mackay-Glass. Table 1 lists the network sizes for these datasets. Note that we extracted consecutive 500 samples from 1500 Mackay-Glass-chaos time series samples for the learning.

(4) Benchmark tests

ILS1 learned 20 instances in each awake phase, and its performance was evaluated with the mean error immediately after both awake- and sleep- phases. GRNN-editing and EFuNN learned instances in the same manner, whereas AOSVR and RFWR learned them one by one.

Each instance was presented to the systems only once (one-pass learning). We evaluated the performance of ILS1 immediately after every awake and sleep phase with $E_{all}(t) = \sum_k^{\text{all samples}} SqureErr(\boldsymbol{x}_k, \boldsymbol{\theta}(t))/\text{Sample-Size}$. Note that $E_{all}(t)$ reflects both the ratio of forgetting and the generalization ability. $E_{all}(t)$ was also used for evaluating the other systems.

We also evaluated the number of hidden units. In particular, because the number of hidden units varied during learning, we evaluated the maximum and the minimum number of hidden units. The test was repeated 50 times while changing the order of instances, and the performances were averaged over the 50 trials. 95% confidence intervals were also calculated. In the case of Mackay-Glass chaos, the 500 consecutive samples were selected by randomly changing the starting point every 50 trials.

Figures 10 and 11 plot examples of the mean square error of the systems versus the number of observations for the servo and housing datasets. We can see that the mean square error of ILS1 was smaller than the others over the whole

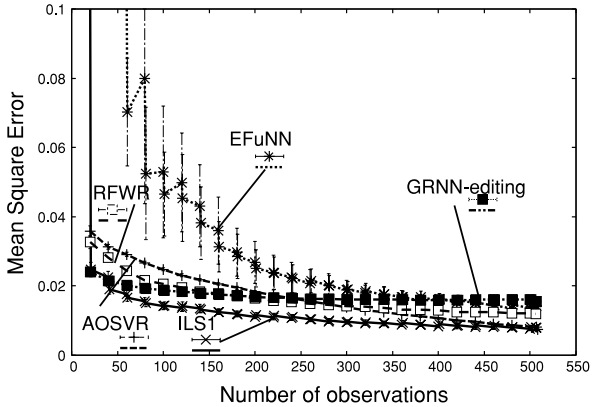

Fig. 11 Mean square error vs number of observations in housing dataset: The error bars indicate 95% confidence intervals.

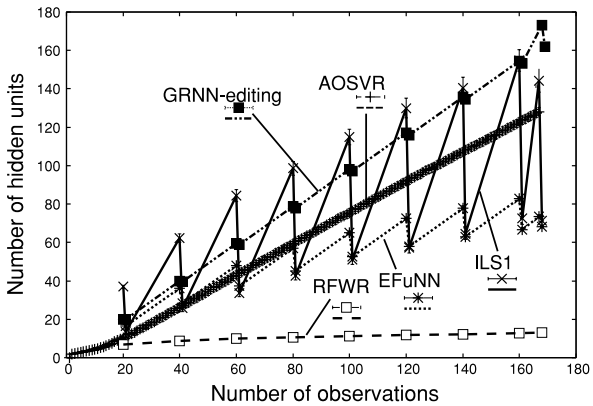

Fig. 12 Mean number of hidden units vs number of observations in servo dataset: The error bars indicate 95% confidence intervals.

range. This means ILS1 is suitable for performing incremental learning and recognition simultaneously. In particular, MSE immediately after the sleep phase was usually less than it was after the last awake phase[†]. Thus, the generalization ability of ILS1 was improved by the adjacent model selection.

Figures 12 and 13 plot the number of hidden units versus the number of observations, for the servo and housing datasets. We can see that the minimum number of hidden units of ILS1 was less than those of EFuNN, GRNN-editing and AOSVR. Although RFWR had fewer hidden units than ILS1 had, its mean square error was larger[††].

The system performances for the other datasets were similar to those of the servo and housing datasets. Therefore, for almost all of the learning steps, the mean square error of ILS1 was less than those of the others and the number of hidden units after the sleep phase was also less than

[†]Note that, in the early steps of the learning, the 95% confidence intervals of the mean square error before and after sleep phase are obviously separated.

[††]Note that the number of parameters in RFWR is proportional to $m \times (n^2 + o)$, but that of ILS1 is $m \times (n + o)$, where m, n and o are the number of hidden units, dimension of input vector and number of outputs.

those of the other systems.

The most important aspect is the compactness of the system. Therefore, it is desirable that the learning machine yield a low error rate with few parameters. The most compact learning machine would not only reduce the required resources but would also improve the generalization ability. To compare the compactnesses of these systems, we calculated the following information criterion (AIC) for the mean square error and number of parameters after the learning:

$$AIC = < Err >_{train} + \frac{P}{n}, \tag{13}$$

where $< Err >_{train}$ denotes mean square error to the training samples, P and n denote the number of parameters and number of training samples. Note that the AIC was originally intended as a suitability measure for various models having the same architecture. However, we used it to compare the compactnesses of the different architectures. Therefore, the model with the smallest AIC was considered to be the most compact model that represents the training data. Table 2 shows the AIC values of the five systems after learning on each dataset. We can see that ILS1 is the best model for servo, housing, cpu-performance and mpg datasets but RFWR is the best for the Mackay-Glass chaos time series.

(5) Number of parameters vs Mean square error

To compare ILS1 with the other learning methods more precisely, we plotted the mean square error of the learned samples versus the number of parameters under various pruning or growth conditions. Each system repeated the same learning tasks as the first benchmark test under various pruning or growth conditions, and the mean square error and mean parameters were plotted. The pruning and growth conditions were varied as follows. For ILS1, the pruning threshold δ was varied over the interval $[0.3, 0.5]$; for GRNN-editing, the pruning error threshold was varied over the interval $[0.05, 0.1]$; for RFWR, the growing threshold $\varepsilon_{pruning}$ was varied over the interval $[0.01, 0.1]$; for EfuNN, the aggregation error threshold E was varied over the interval $[0.001, 0.1]$; for AOSVR, the insensitive width ε was varied over the interval $[0.1, 0.5]$.

Figures 14, 15, 16, 17, 18 plot the mean square error

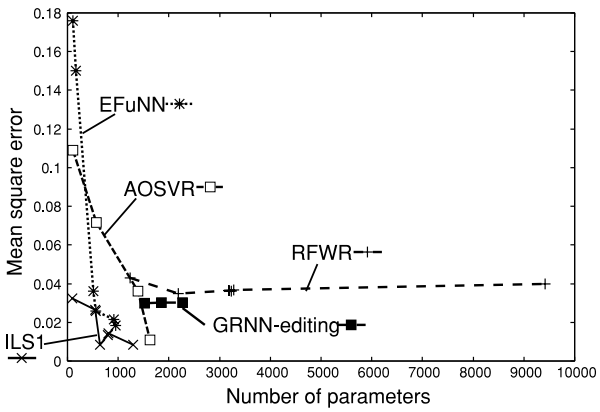

Fig. 14 Mean square error vs number of hidden units in servo dataset.

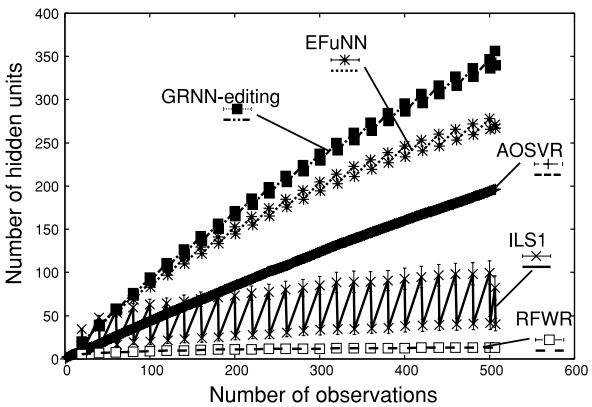

Fig. 13 Mean number of hidden units vs number of observations in housing dataset: The error bars indicate 95% confidence intervals.

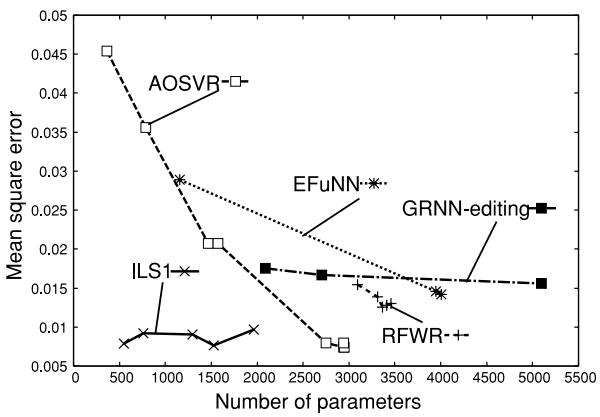

Fig. 15 Mean square error vs number of hidden units in housing dataset.

Table 2 AIC after the learning of each dataset. The model of the smallest value is the best model in terms of compactness and error. See text for detailed explanations.

	ILS1	RFWR	EFuNN	GRNN-editing	AOSVR
servo	3.87	19.75	5.74	12.41	9.79
housing	1.09	6.93	8.03	8.83	5.5
cpu-performance	0.253	1.33	2.18	0.94	0.54
mpg	0.222	0.52	3.34	2.83	2.3
Mackay-Glass	0.260	0.19	1.45	1.81	0.34

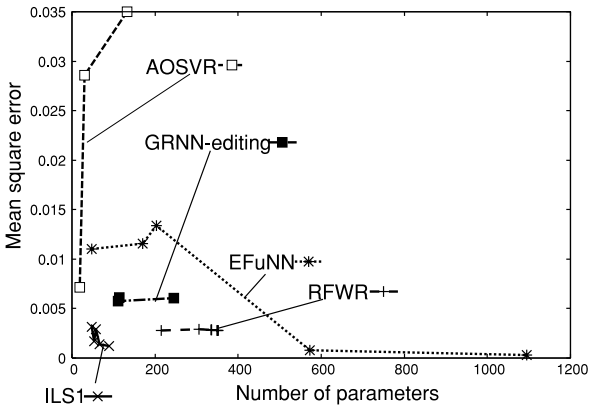

Fig. 16 Mean square error vs number of hidden units in cpu-performance dataset.

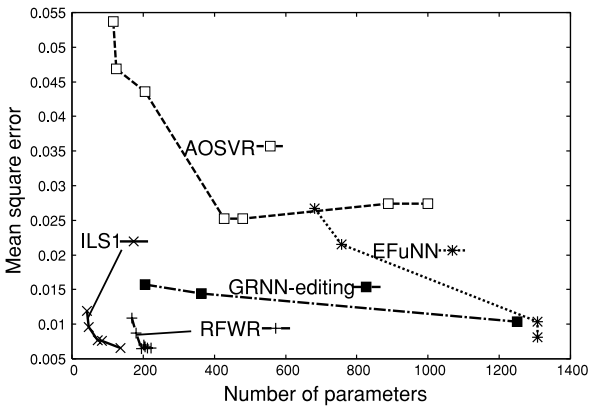

Fig. 17 Mean square error vs number of hidden units in mpg dataset.

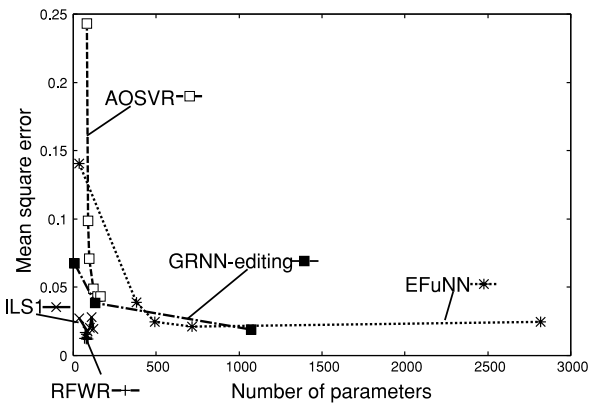

Fig. 18 Mean square error vs number of hidden units in Mackay-Glass chaos.

versus the mean number of parameters over 50 trials for the servo, housing, cpu-performance, mpg, and the Mackay-Glass-chaos time series. Note that the coordinates of the mean square error and number of parameters at the lower left of the figures indicate that the learning system achieved effective learning. We can see the ILS1 data points are always at the lower left. Note that the RFWR performed slightly better than ILS1 in the case of the Mackay-Glass chaos dataset.

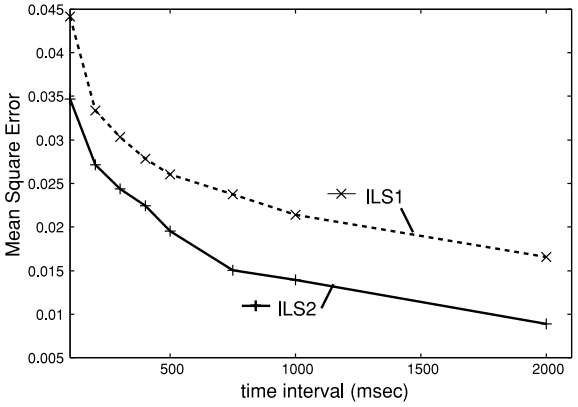

Fig. 19 ILS1 vs ILS2 for servo dataset.

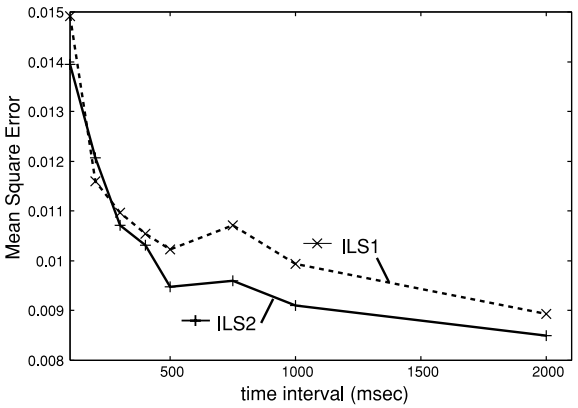

Fig. 20 ILS1 vs ILS2 for housing dataset.

(6) ILS1 vs ILS2

We also compared ILS1 and ILS2 performances using Housing and Servo datasets. The learning samples were presented to ILS1 and ILS2 at interval τ msec even if the sleep phase is not finished, where τ is 100, 200, 300, 400, 500, 750, 1000 and 2000, respectively. This test was performed by VT64 Workstation2000 (2 × Opteron 1.4 GHz). The upper limits of the number of hidden units in F-Net (of ILS1) and F'-Net (of ILS2) were set to 20. Through this experiment, we examine the mean square error versus τ. The datasets used were the same as the above benchmark tests. The test was repeated 50 times and the results were averaged over the trials.

Figures 19 and 20 show the mean square error after the presentation of whole samples versus τ. We can see that ILS2 is superior to ILS1. This means that the number of learning samples of ILS2 is larger than that of ILS1. If τ is large enough, ILS1 performances will close to those of ILS2.

9. Discussion and Conclusion

We presented a learning system that simultaneously

achieves quick adaptation and model selection by using two learning phases: awake and sleep. The experimental results revealed that the system outperformed other learning systems in solving function approximation problems. The main aim of this work was to devise a model selection method to reduce the total computational cost, which in turn would improve the generalization ability, and the benchmark test results proved that we achieved our aim. That is, the mean square error of ILS after the sleep phase was better than after the awake phase in the early steps of the learning. This is because the compact RBF network that was created right after the sleep phase could interpolate the outputs for unseen inputs. This is different from the case of table look up methods such as GRNN.

The main difference between our system and systems such as EFuNN and RFWR is its the network structure. Our system uses the traditional RBF as the learning machine, whereas the others use GRNN or an normalized Gaussian network-like architecture. Specifically, our system does not normalize the output values of the network of the RBF by using the sum of Gaussian outputs, whereas the other methods do. By using the RBF architecture, the resultant network structure becomes slimmer than the others, as was shown experimentally. ILS1 requires that several networks, e.g., the M- and F-Net, have correct pseudo-samples because the RBF (S-Net) outputs during the model selection dramatically change; the system has to keep the previous learning result in another network.

The most important result is that the system supports the learning of non-independent distributed instances without forgetting. To support non-independent instances, learning methods during the awake phase should not contain online gradient-based parameter modifications such as probabilistic gradient descent learning. This is because a method such as probabilistic gradient learning needs the iid instances to approximate accurate gradient descents on the error surface, but there is no guarantee that the actual samples are iid samples.

This system performs learning during the sleep phase. During the sleep phase, the network structure is rebuilt to be simpler and the parameters are refined to improve their generalization ability. Although ILS1 is a learning method inspired by biological learning behaviors rather than the model of biological sleep, the design principle is similar to that of the model of the Hippocampus: the interleaved learning proposed by McClelland et al. [28]. In interleaved learning, the network learns not only the new instances but also old instances stored in a buffer. This learning strategy reduces interferences due to learning new instances. Similarly, during the sleep phase of ILS1, the S-Net relearn pseudo-instances generated not only by the F-Net but also by the M-Net. Note that the F-Net outputs correspond to new instances but M-Net outputs correspond to learned old instances. Norman et al. [29] extends McClelland's model to reduce the interference even if the old instances are lost from the Hippocampus. They predict that the learning of new instances stored in the Hippocampus is achieved during non-REM sleep, whereas the rehearsal for old instances is achieved during REM sleep. They proposed a model of REM sleep: the oscillating learning algorithm , which is also for reducing the interference due to the learning during non-REM sleep. Thus, in terms of the learning strategy, these models for the Hippocampus focus only to how to reduce the interference due to new learning. On the other hand, our model ILS1 focus not only on the reduction of interference but also on the model-selection. This is the essential difference between ILS1 and the Hippocampus models [28], [29].

Similar behaviors can be found in biological learning systems. For example, Amis and Daniel [30] showed that the timing and structure of the activity in a motor cortex elicited by the playback of a song during sleep matched the activity elicited during a daytime recital of the song by the subject. They showed that the timing and structure of the activity in a motor cortex elicited by the playback of song during sleep matches the activity during daytime singing. They said that their data suggested that sensory-motor correspondences are stored during singing but do not modify behavior and that the off-line comparison of rehearsed motor output and predicted sensory feedback is used to adaptively shape motor output.

The benchmark test results also revealed that the maximum resources used during the learning processes is sometimes larger than those used by the GRNN. The maximum quantity of resources used during the learning depends on the dataset and if it contains redundant input dimensions, the number of hidden units will greatly increase. To overcome this problem, we modified the system so that it would ignore redundant dimensions [31].

Acknowledgments

This work was supported by a Grant-in-Aid, No.13870262 Scientific Research, from the Japanese Ministry of Education, Science and Culture.

References

[1] J.E. Moody, The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems, pp.847–854, Morgan Kaufmann Publishers, 1992.
[2] G.E. Hinton and D. van Camp, "Keeping neural networks simple by minimizing the description length of the weights," Sixth ACM Conference on Computational Learning Theory, Santa Cruz, pp.5–13, July 1993.
[3] N. Murata, S. Yoshizawa, and S. Amari, "Network information criterion: Determining the number of hidden units for an artificial neural network model," IEEE Trans. Neural Netw., vol.5, no.6, pp.865–872, Nov. 1994.
[4] P. van de Laar and T. Heskes, "Pruning using parameter and neuronal metrics," Neural Comput., vol.11, pp.977–993, 1999.
[5] D.F. Specht, "A general regression neural network," IEEE Trans. Neural Netw., vol.2, no.6, pp.568–576, Nov. 1991.
[6] D. Tomandl and A. Schober, "A modified generalized regression neural network (MGRNN) with a new efficient training algorithm as a robust "black-box"-tool for data analysis," Neural Networks, vol.14, pp.1023–1034, 2001.
[7] R.O. Duda, P.E. Hart, and D.G. Stork, Pattern Classification, Second

ed., Wiley-Interscience Publication, 2001.
[8] T. Poggio and F. Girosi, "Networks for approximation and learning," Proc. IEEE International Conference on Neural Networks, vol.78, no.9, pp.1481–1497, Sept. 1990.
[9] L. Yingwei, N. Sundararajan, and P. Saratchandran, "A sequential learning scheme for function approximation using minimal radial basis function neural networks," Neural Comput., vol.9, pp.461–478, 1997.
[10] M. Ishikawa, "Structural learning with forgetting," Neural Netw., vol.9, no.3, pp.509–521, 1996.
[11] N. Kasabov, "Evolving fuzzy neural networks for supervised/unsupervised online knowledge-based learning," IEEE Trans. Syst. Man Cybern., vol.31, no.6, pp.902–918, Dec. 2001.
[12] S. Schaal and C.G. Atkeson, "Constructive incremental learning from only local information," Neural Comput., vol.10, no.8, pp.2047–2084, Nov. 1998.
[13] T. Hoya and J.A. Chambers, "Heuristic pattern correction scheme using adaptively trained generalized regression neural networks," IEEE Trans. Neural Netw., vol.12, no.1, pp.91–100, Jan. 2001.
[14] I. Hayashi, H. Maeda, and J.R. Williamson, "A formulation of receptive field type input layer for tam network using gabor function," IEEE International Conference on Fuzzy Systems, #1335, July 2004.
[15] J. Moody and C.J. Darken, "Fast learning in neural networks of locally-tuned processing units," Neural Comput., vol.1, pp.281–294, 1989.
[16] R.M. French, "Pseudo-recurrent connectionist networks: An approach to the "sensitivity stability" dilemma," Connection Science, vol.9, no.4, pp.353–379, 1997.
[17] B. Ans and S. Roussert, "Neural networks with a self-refreshing memory: Knowledge transfer in sequential learning tasks without catastrophic forgetting," Connection Science, vol.12, no.1, pp.1–19, 2000.
[18] K. Yamauchi, S. Itho, and N. Ishii, "Combination of fast and slow learning neural networks for quick adaptation and pruning redundant cells," IEEE SMC'99, 1999 IEEE System, Man and Cybernetics Conference, vol.III, pp.390–395, Oct. 1999.
[19] K. Yamauchi, S. Ito, and N. Ishii, "Wake-sleep learning method for quick adaptation and reduction of redundant cells," ICONIP 2000 7th International Conference on Neural Information Processing, vol.1, pp.559–564, Nov. 2000.
[20] K. Yamauchi, "Sequential learning and model selection with sleep," ICONIP 2001 8th International Conference on Neural Information Processing, ed. L. Zhang and F. Gu, vol.1, pp.205–210, Fudan University Press, Nov. 2001.
[21] K. Yamauchi and N. Kobayashi, "Incremental learning with sleep-learning of noiseless datasets," International Conference on Neural Information Processing ICONIP2002, vol.1, pp.398–403, Nov. 2002.
[22] K. Yamauchi and J. Hayami, "Sleep learning," IEEE World Congress on Computational Intelligence (WCCI2006), pp.6295–6302, July 2006.
[23] J. Ma, J. Theiler, and S. Perkins, "Accurate on-line support vector regression," Neural Comput., vol.15, pp.2683–2703, 2003.
[24] C. Burges, "Geometory and invariance in kernel based methods," in Advances in Kernel Methods — Support Vector Learning, ed. B. Schölkoph, C. Burges, and A. Smola, pp.144–152, MIT Press, 1999.
[25] M. Umano, Y. Hosoya, Y. Uno, K. Seta, and M. Okada, "Incremental learning of fuzzy rules by fuzzy neural network with forgetting facility," 19th Fuzzy System Symposium, pp.665–668, Japan Society for Fuzzy Theory and Intelligent Informatics, 2003.
[26] K. Yamauchi, N. Yamaguchi, and N. Ishii, "Incremental learning methods with retrieving interfered patterns," IEEE Trans. Neural Netw., vol.10, no.6, pp.1351–1365, Nov. 1999.
[27] H. Akaike, "A new look at the statistical model identification," IEEE Trans. Autom. Control, vol.AC-19, no.6, pp.716–723, Dec. 1974.
[28] J.L. McClelland, R.C. O'Reilly, and B.L. McNaughton, "Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory," Psychological Review, vol.102, no.3, pp.419–457, 1995.
[29] K.A. Norman, E.L. Newman, and A.J. Perotte, "Methods for reducing interference in the complementary learning systems mode: Ocillating inhibition and autonomous memory rehearsal," Neural Netw., vol.18, no.9, pp.1212–1228, Nov. 2005.
[30] A.S. Dave and D. Margoliash, "Song replay during sleep and computational rules for sensorymotor vocal learning," Sciense, vol.290, pp.812–816, Oct. 2000.
[31] K. Yamauchi, "Incremental learning and dimension selection with sleep," International Conference on Neural Information Processing ICONIP2004, LNCS3316, pp.489–494, Nov. 2004.

Appendix: Determination of Parameters

The parameters used are listed in Table A· 1.

Many parameters should be set to a certain value which is determined empirically. To determine the parameters for better performance, we fixed a part of parameter to certain values and then optimized the remained parameters through preliminary test. The optimized parameters were the overlap ratio for the S-Net, λ_S, and the remove-threshold, δ, which are the relatively sensitive parameters for the system performance. Normally, these two parameters should be adjusted to each target dataset. However, according to the preliminarily test, the best combination of these parameters for the five datasets were almost the same: $(\lambda_S, \delta) = (2.5, 0.4)$. Thus, we fixed the two parameters to those values for all benchmark tests.

The three parameters, λ_F, λ_S and δ, are especially important and affect the generalization ability of the system directly.

The overlap ratio for the F-Net, λ_F, was set to 0.5, which represents the smoothness of the interpolated surface between instances. If λ_F is small, the F-Net behavior closes to that of 1-Nearest Neighbor. As described in one way to determine the kernel width 0.5× min distance, which is according to the MGRNN heuristic [6], λ_F is set to 0.5, with

Table A· 1 Parameters used. (N_y denotes number of output dimensions)

Parameters	Explanation	Value
A	Smoothness of $w_F(\boldsymbol{x})$	100
θ_d	threshold for the trusted input region	0.95
ε	learning speed	0.001
ϵ_E	Allocation threshold for F-Net	0.0001
λ_F	Overlap ratio for F-Net	0.5
λ_S	**Overlap value for S-Net**	**2.5**
ϵ_{max}	Allocation threshold of S-Net	1
ϵ_{min}	Allocation threshold of S-Net	0.01
e'_{rmse}	Allocation threshold of S-Net	$0.01N_y$
δ	**Pruning threshold**	**0.4**
M	Consecutive observations	10
T	Buffer size for calculating MSE	100
N_o	Number of pseudo instance per cell	200
d_{min}	Threshold for $w_F(\boldsymbol{x})$	0.5
ϵ_o	Threshold for checking whether each hidden unit is activated or not	0.00001

Table A· 2 Datasets used for preliminary tests.

Name	Size	Network-size	Class type
CPU-performance	340	5 - 1	Continuous
MPG	392	7 - 1	Continuous
Servo	167	12 - 1	Continuous
Housing	500	13 - 1	Continuous

Table A· 3 Suitability vs λ_S to each dataset.

Dataset	$\lambda_S = 1$	$\lambda_S = 1.5$	$\lambda_S = 2$	$\lambda_S = 2.5$	$\lambda_S = 3$
Housing	0.032	0.027	0.023	0.0021	0.03
Servo	0.019	0.011	0.001	0.01	0.014
MPG	0.0022	0.0018	0.0046	0.0018	0.003
CPU-Performance	0.008	0.0081	0.013	0.008	0.011
Mackay glass	0.018	0.018	0.028	0.020	0.023
Mean suitability	0.016	0.013	0.016	0.012	0.016

which the slope of the regression curve in the middle between the nearest two samples equals the slope of the connecting line. To do this, the kernel width of the old F-Net unit, which is the nearest to the new allocated hidden unit, needs to be the same width as that of the new kernel.

The S-Net's overlap ratio and remove-threshold, λ_S and δ, were determined through two preliminary tests. The first test was to determine the overlap ratio, λ_S, whereas the second test was to determine the remove-threshold, δ. In these two tests, the system performance the four datasets listed in Table A· 2 were evaluated with varying λ_S and δ.

In the first test, all parameters except for λ_S were fixed at the values in Table A· 1. Note that δ was also fixed at a certain value 0.5 in this test. λ_S was varied during the interval [0.5,3] and suitability measure [1] was calculated to evaluate performance corresponding to each value of λ_S.

The suitability measure used was similar to AIC

$$Suitability = < Err >_{train} + w\frac{P}{n}, \qquad (A\cdot 1)$$

where $< Err >_{train}$ is the mean square error to train samples, w is an weight, P is the number of system parameters and n is the number of training samples. We set $w = 0.002$ for the evaluation. Note that Eq. (A· 1) shows a small value when the system yields small error to the learning samples with a small number of parameters. User should set the weight w appropreately according to how much he attends to the balance between the error and number of parameters.

Table A· 3 denotes the suitability to each dataset after finishing all learning phases. This test was executed in the same manner as the experiment. Learning was repeated 10 times changing the order of presentation of the dataset. The result was averaged over 10 trials. We can see that the optimum λ_S depends on the dataset but optimum values are distributed around $\lambda_S = 2.5$. To evaluate the averaged optimum value, we calculated the normalized suitability, which was normalized by the maximum AIC in the datasets. We then averaged all normalized AIC for each λ_S. The last line of Table A· 3 denotes mean normalized suitabilitiess. We can see that the mean value becomes minimum when $\lambda_S = 2.5$.

The second test was to determine of δ and here, λ_S was set to 2.5, which is the optimum value determined in the first test, and δ was varied during the interval $[0.3, 0.9]$. Table A· 4 denotes the suitability to each dataset. We can see the optimum values also depend on the dataset, but these are distributed around 0.4. To evaluate the averaged value, we calculated the averaged value in the same manner as the first test. The last line of Table A· 4 denotes the mean value. We can see the optimal value of δ is about 0.4.

Table A· 4 Suitability vs δ to each dataset.

Dataset	$\delta = 0.3$	$\delta = 0.4$	$\delta = 0.5$	$\delta = 0.7$	$\delta = 0.9$
Servo	0.024	0.021	0.024	0.033	0.039
Housing	0.014	0.010	0.011	0.014	0.017
CPU-performance	0.0019	0.0016	0.0019	0.0016	0.0031
MPG	0.0072	0.0076	0.008	0.0034	0.012
Mackay-glass	0.018	0.020	0.021	0.028	0.028
Mean suitability	0.013	0.012	0.013	0.016	0.02

Koichiro Yamauchi received the B.E. and M.E. degrees in 1989 and 1991, respectively, in electrical and computer engineering from the Nagoya Institute of Technology, Japan and the Ph.D. degree in engineering science from the Osaka University Japan, in 1994. From April 1994 to October 2000, he was at the Department of Intelligence and Computer Science of the Nagoya Institute of Technology, where he worked on construction of incremental learning algorithms of artificial neural networks, case based reasoning systems using neural networks and sensor fusion systems. From November 2000 to 2003, he has been with Hokkaido University, where he was an Associate Professor of the Graduate School of Engineering. Since 2004, he is an Associate Professor of Graduate School of Information Science and Technology of Hokkaido University. His current research interests includes incremental learning of model-based learning systems, computational modeling of biological memory systems and construction of virtual mind on the computer.

Jiro Hayami received his B.I.S. degree from Hokkaido University in 2005 and enrolled in the Graduate School of Information Science and Technology. His current research interests include online learning, model selection and variable reduction.