PAPER

An Online Self-Constructive Normalized Gaussian Network with Localized Forgetting

Jana BACKHUS[†a)], **Ichigaku TAKIGAWA**[†,††], ***Nonmembers***, **Hideyuki IMAI**[†], **Mineichi KUDO**[†], ***and*** **Masanori SUGIMOTO**[†], ***Members***

SUMMARY In this paper, we introduce a self-constructive Normalized Gaussian Network (NGnet) for online learning tasks. In online tasks, data samples are received sequentially, and domain knowledge is often limited. Then, we need to employ learning methods to the NGnet that possess robust performance and dynamically select an accurate model size. We revise a previously proposed localized forgetting approach for the NGnet and adapt some unit manipulation mechanisms to it for dynamic model selection. The mechanisms are improved for more robustness in negative interference prone environments, and a new merge manipulation is considered to deal with model redundancies. The effectiveness of the proposed method is compared with the previous localized forgetting approach and an established learning method for the NGnet. Several experiments are conducted for a function approximation and chaotic time series forecasting task. The proposed approach possesses robust and favorable performance in different learning situations over all testbeds.

key words: *Normalized Gaussian Networks, dynamic model selection, online learning, chaotic time series forecasting*

1. Introduction

In applications where data samples are received sequentially, incremental learning schemes have to be applied to train neural networks. In truly sequential learning schemes [16], only one data sample is observed at any time and directly discarded after learning. So, no prior knowledge is available on the number of training data or the data distribution. Furthermore, training data are often not independent and identically distributed (i.i.d.) in real world applications. This is a problem when network models are updated in favor of newly arriving training data. If the data distribution is not i.i.d., then networks are prone to forget already learned information. This phenomenon is called negative or in severe cases catastrophic interference when it is not wanted.

Normalized Gaussian networks (NGnet) are feed-forward three layer networks that are related to Radial Basis Function (RBF) neural networks. They have localized learning behavior by partitioning the input space with local units. Only a few units are updated for a newly received data sample. Local model networks are often applied to sequential learning schemes, because their model structure eases the effects of negative interference. NGnets differ from RBF networks in the normalization of the Gaussian activation function. The normalization switches the traditional roles of weights and activities in the hidden layer, and NGnets therefore exhibit better generalization properties [1].

When NGnets are applied to online learning tasks, two major points have to be considered. The learning performance of the NGnet should be robust even in cases where the data distribution is not i.i.d., and since an accurate model size selection at initialization is difficult, the network model should be adapted dynamically. In this paper, we propose an NGnet with localized forgetting that applies self-constructive dynamic model selection for the first time. Our contributions include:

- Revising a previously proposed localized forgetting approach [2].
- Applying dynamic model selection self-constructively.
- Improving model selection in regard to negative interference.
- Dealing with model redundancies by merging units.

The dynamic model selection is applied self-constructively to avoid the necessity of a model initialization and therefore ease application to online learning tasks.

The effectiveness of the proposed method is demonstrated for two experiments: a function approximation task, applying both i.i.d. and non i.i.d. data distributions, and a chaotic time series forecasting task. In regard to the revision of the localized forgetting approach in [2], we compare our proposed method to it, discuss the differences in learning performance, and show that the application of dynamic model selection results in improved learning performance. Also, we compare the proposed method with an established learning method [15] for the NGnet that applies dynamic model selection and show that ours is more robust in different learning environments, especially for non i.i.d. learning tasks. This improved robustness makes our proposed method suitable for online learning tasks.

The rest of the paper is divided as follows: Section 2 explains some basic definitions for the NGnet while the proposed method is explained in Sect. 3 and evaluated in Sect. 4. A conclusion is drawn in Sect. 5.

2. Normalized Gaussian Network and Online EM

2.1 Normalized Gaussian Network

The Normalized Gaussian network (NGnet) is a universal

Manuscript received September 30, 2016.

[†]The authors are with Department of Computer Science and Information Technology, Graduate School of Information Science and Technology, Hokkaido University, Sapporo-shi, 060-0814 Japan.

[††]JST PRESTO, Kawaguchi-shi, 332-0012 Japan.

a) E-mail: jana@main.ist.hokudai.ac.jp

DOI: 10.1587/transfun.E100.A.865

function approximator that was first proposed by Moody and Darken [12]. It transforms an N-dimensional input vector x to a D-dimensional output vector y with

$$y = \sum_{i=1}^{M} N_i(x)\tilde{W}_i\tilde{x}. \tag{1}$$

$$N_i(x) \equiv G_i(x)/\sum_{j=1}^{M} G_j(x) \tag{2}$$

$$G_i(x) \equiv (2\pi)^{-N/2}|\Sigma_i|^{-1/2}\times \exp\left[-\frac{1}{2}(x-\mu_i)'\Sigma_i^{-1}(x-\mu_i)\right]. \tag{3}$$

The model softly partitions the input space into M local units with the normalized Gaussian functions N_i. $\tilde{W}_i \equiv (W_i, b_i)$ is a $D \times (N+1)$-dimensional linear regression matrix with $\tilde{x}' \equiv (x', 1)$ where prime ($'$) denotes a transpose.

2.2 Online EM Algorithm

A stochastic interpretation was first proposed for the NGnet by Xu et al. [19]. The unknown model parameters are estimated by maximum likelihood estimation based on the log-likelihood of the observed in- and output data (x, y). Here, the Expectation-Maximization (EM) algorithm is applied for parameter estimation. An offline approach has been proposed by Xu et al. [19] that was later adopted to an online EM algorithm by Sato and Ishii [15].

The stochastic model is defined by the following probability distribution for a complete event (x, y, i) [19]

$$P(x, y, i|\theta) = (2\pi)^{-\frac{D+N}{2}}\sigma_i^{-D}|\Sigma_i|^{-\frac{1}{2}}M^{-1}\times \exp\left[-\frac{1}{2}(x-\mu_i)'\Sigma_i^{-1}(x-\mu_i) - \frac{1}{2\sigma_i^2}(y-\tilde{W}_i\tilde{x})^2\right], \tag{4}$$

where $\theta \equiv \{\mu_i, \Sigma_i, \sigma_i^2, \tilde{W}_i | i = 1, ..., M\}$ is the set of model parameters that have to be estimated. For the online EM, the parameters are updated with the following E- and M-step.

2.2.1 E (Estimation) Step:

Given the current estimator $\theta(t-1)$, the posterior probability $P_i(t) \equiv P(i|x(t), y(t), \theta(t-1))$ evaluates how likely the i-th unit generates the current observation $(x(t), y(t))$:

$$P_i(t) \equiv P(i|x(t), y(t), \theta(t-1)) = \frac{P(x(t), y(t), i|\theta(t-1))}{\sum_{j=1}^{M} P(x(t), y(t), j|\theta(t-1))}. \tag{5}$$

2.2.2 M (Maximization) Step:

The model parameters θ are updated with

$$\mu_i(t) = \langle\!\langle x\rangle\!\rangle_i(t)/\langle\!\langle 1\rangle\!\rangle_i(t), \tag{6}$$

$$\Sigma_i^{-1}(t) = [\langle\!\langle xx'\rangle\!\rangle_i(t)/\langle\!\langle 1\rangle\!\rangle_i(t) - \mu_i(t)\mu_i'(t)]^{-1}, \tag{7}$$

$$\tilde{W}_i(t) = \langle\!\langle y\tilde{x}'\rangle\!\rangle_i(t)[\langle\!\langle \tilde{x}\tilde{x}'\rangle\!\rangle_i(t)]^{-1}, \tag{8}$$

$$\sigma_i^2(t) = \frac{[\langle\!\langle |y|^2\rangle\!\rangle_i(t) - Tr(\tilde{W}_i(t)\langle\!\langle \tilde{x}y'\rangle\!\rangle_i(t))]}{D\langle\!\langle 1\rangle\!\rangle_i(t)}. \tag{9}$$

The parameter updates (6)–(9) include symbols $\langle\!\langle\cdot\rangle\!\rangle_i(t)$ that denote weighted sums of accumulated observations $(x(t), y(t))$ until the current time step t.

A weighted sum is incrementally updated at each time step t with respect to the posterior probability $P_i(t)$.

$$\langle\!\langle f\rangle\!\rangle_i(t) = \lambda(t)\langle\!\langle f\rangle\!\rangle_i(t-1) + P_i(t)f_t, \tag{10}$$

where $f \equiv f(x, y)$ and $f_t \equiv f(x(t), y(t))$ are used with abbreviations. A time-dependent discount factor $\lambda(t)$ has been introduced to (10). It plays an important role in discarding the effect of old learning results that were employed to an earlier inaccurate estimator. The factor has to be chosen so that $\lambda \to 1$ when $t \to \infty$ to fulfill the Robbins-Monro condition for convergence of stochastic approximations [9].

In the online EM [15], $\langle\!\langle f\rangle\!\rangle_i(t)$ has been defined as a weighted mean by employing a normalization coefficient $\eta(t) = (1 + \lambda(t)/\eta(t-1))^{-1}$. The weighted mean is calculated as follows:

$$\langle\!\langle f\rangle\!\rangle_i(t) = (1-\eta(t))\langle\!\langle f\rangle\!\rangle_i(t-1) + \eta(t)P_i(t)f_t. \tag{11}$$

In the following, we will however omit the normalization by employing a weighted sum (10) as defined in [2].

3. Self-Constructive Normalized Gaussian Network with Localized Forgetting

In this paper, we introduce an improvement for a parameter estimation method with localized forgetting and then discuss the application of dynamic model selection to an NGnet with localized forgetting for the first time.

3.1 Parameter Update with Localized Forgetting

Previously, Celaya and Agostini have proposed a parameter update with localized forgetting [2] to improve learning robustness. The weighted sum (10) employs a discount factor $\lambda(t)$ to forget old learning results over time. The time-dependent discount is a problem when data are not i.i.d., and learning is prone to negative interference. In [2], forgetting is based on weights so that only as much old information is forgotten as new information is received for a unit i. For this, (10) has been changed to

$$\langle\!\langle f\rangle\!\rangle_i(t) = \lambda(t)^{P_i(t)}\langle\!\langle f\rangle\!\rangle_i(t-1) + \frac{1-\lambda(t)^{P_i(t)}}{1-\lambda(t)}f_t. \tag{12}$$

This update method is preferable because of its stable learning performance. We consider it however bad practice that the update with new information f_t in (12) is dependent on discount $\lambda(t)$. $\lambda(t)$ controls forgetting and has been originally introduced to speed up convergence [14]. The

constraint for $\lambda(t)$ is $(0 \leq \lambda(t) \leq 1)$ [15], where $\lambda(t) = 1$ is a special case with no forgetting of old learning results. Yet, the update factor $\frac{1-\lambda(t)^{P_i(t)}}{1-\lambda(t)}$ becomes undefined over all t when $\lambda(t) = 1$, and the NGnet is then unable to learn.

We propose a modified localized forgetting approach by reconsidering its derivation from (10) with additional aspects that have not been considered previously. For our derivation, (10) is rewritten to

$$\langle\!\langle f \rangle\!\rangle_i(t) = \Lambda(P_i(t))\langle\!\langle f \rangle\!\rangle_i(t-1) + \Omega(P_i(t))f_t, \tag{13}$$

where a new forgetting factor $\Lambda(P_i(t))$ and update factor $\Omega(P_i(t))$ have to be determined by considering the following conditions. For a full update, $P_i(t) = 1$ and (13) should reduce to the same as (10), given by $\Lambda(1) = \lambda(t)$ and $\Omega(1) = 1$. For the NGnet, the units divide the input space between each other according to their representation ability of a current sample $(x(t), y(t))$. A full update therefore only happens at a same time step t for a weighted sum $\langle\!\langle f \rangle\!\rangle_M$ that accumulates information over all units M, or when a unit represents the data sample to 100% which rarely happens. Additionally, an update with value f_t should be consistent for a shared weighted sum $\langle\!\langle f \rangle\!\rangle_{i+j}(t-1)$ by unit i and j when updated once with weight $(P_i(t) + P_j(t))$ or twice with weights $P_i(t)$ and $P_j(t)$ at a time step t. For a single update with $(P_i(t)+P_j(t))$, we get

$$\begin{aligned}\langle\!\langle f \rangle\!\rangle_{i+j}(t) = \Lambda(P_i(t) + P_j(t))\langle\!\langle f \rangle\!\rangle_{i+j}(t-1) \\ +\Omega(P_i(t) + P_j(t))f_t.\end{aligned} \tag{14}$$

If $\langle\!\langle f \rangle\!\rangle_{i+j}(t-1)$ is updated twice with $P_i(t)$ and $P_j(t)$, then

$$\langle\!\langle f \rangle\!\rangle_{i+j}(t) = \Lambda(P_i(t))\langle\!\langle f \rangle\!\rangle_{i+j}(t-1) + \Omega(P_i(t))f_t, \tag{15}$$

$$\langle\!\langle f \rangle\!\rangle_{i+j}(t+1) = \Lambda(P_j(t))\langle\!\langle f \rangle\!\rangle_{i+j}(t) + \Omega(P_j(t))f_t. \tag{16}$$

We insert (15) into (16) and get

$$\begin{aligned}\langle\!\langle f \rangle\!\rangle_{i+j}(t+1) = \Lambda(P_j(t))\Lambda(P_i(t))\langle\!\langle f \rangle\!\rangle_{i+j}(t-1) \\ +(\Lambda(P_j(t))\Omega(P_i(t)) + \Omega(P_j(t)))f_t.\end{aligned} \tag{17}$$

(17) is however one step ahead of time and ignores the fact that units divide a current data sample $(x(t), y(t))$ between each other at a time step t. Furthermore, the update $\Omega(P_i(t))f_t$ is already partially forgotten by $\Lambda(P_j(t))$ which should be prevented for updates on the same t. Then, (17) reduces to

$$\begin{aligned}\langle\!\langle f \rangle\!\rangle_{i+j}(t) =&\Lambda(P_j(t))\Lambda(P_i(t))\langle\!\langle f \rangle\!\rangle_{i+j}(t-1) \\ &+ (\Omega(P_i(t)) + \Omega(P_j(t)))f_t.\end{aligned} \tag{18}$$

Finally, we get two functional equations from (14) and (17)

$$\Lambda(P_i(t) + P_j(t)) = \Lambda(P_j(t))\Lambda(P_i(t)), \tag{19}$$

$$\Omega(P_i(t) + P_j(t)) = \Omega(P_i(t)) + \Omega(P_j(t)). \tag{20}$$

For (19), the solution is well known to be of the form $\Lambda(P_i(t)) = c^{P_i(t)}$, where c is determined by using $\Lambda(1) = \lambda(t)$. So, $\Lambda(P_i(t)) = \lambda(t)^{P_i(t)}$ is derived while $\Omega(P_i(t)) = P_i(t)$ is the solution for the update factor. The new stepwise update for the weighted sum is obtained as

$$\langle\!\langle f \rangle\!\rangle_i(t) = \lambda(t)^{P_i(t)}\langle\!\langle f \rangle\!\rangle_i(t-1) + P_i(t)f_t. \tag{21}$$

The new approach complies with all considered dependencies and adds new information f_t independent from $\lambda(t)$. It is then able to update its parameters over all $0 \leq \lambda(t) \leq 1$.

3.2 Model Selection

The selection of an accurate model size is another important problem when applying neural networks. Especially for online learning tasks, it is difficult to choose a good model size because domain knowledge is limited. Furthermore, choosing an accurate model size by hand necessitates extensive trial-and-error studies. An alternative is dynamic adaptation of the network model during learning. Dynamic model adaptation avoids the need to set a static model size in advance and can be achieved with some methods to increase or reduce model complexity. Also, the model can be built from scratch during learning by applying these methods self-constructively. Several works for RBF networks have proposed self-constructive unit adaptation, Platt's Resource Allocating Network (RAN) [13] as well as its extensions (RANEKF [8], MRAN [11]) and the GGAP-RBF network that has been proposed by Huang et al. [6].

3.2.1 Unit Manipulation Mechanisms

Some unit manipulation mechanisms have been introduced for the NGnet [15] and are adapted to use with our localized forgetting approach. The adapted unit manipulation mechanisms include a produce, delete and split mechanism. However, the delete mechanism needs to be revised to be applicable to the local forgetting NGnet. In addition, the manipulation decision of split is changed to improve robustness against negative interference, and a merge mechanism is added to reduce redundancy of units and further improve model compactness.

(1) Produce

$P(x(t), y(t)|\theta(t-1))$ is a probability that indicates how properly the current model parameters $\theta(t-1)$ can estimate the newly received data sample $(x(t), y(t))$. When the probability is smaller than a certain threshold $T_{Produce}$, a new unit is created according to the produce mechanism in [15].

(2) Delete

A weighted sum $\langle\!\langle 1 \rangle\!\rangle_i(t)$ indicates how much the i-th unit has been in charge of the observed data until the current time step t. When the delete mechanism has been first proposed in [15], $\langle\!\langle 1 \rangle\!\rangle_i(t)$ was assumed to be a weighted mean scaled between zero and one by a normalizer $\eta(t) = (1 + \lambda(t)/\eta(t-1))^{-1}$. The normalizer $\eta(t)$ replaces the normalizer $1/T$ used in the offline EM-algorithm, where T is the total number of samples, which is unknown in the online EM-algorithm.

In our approach, $\langle\!\langle 1 \rangle\!\rangle_i(t)$ is however an unscaled

weighted sum and cannot be used directly as a reference. We introduce therefore a local unit update counter as a normalizer to overcome this problem. To take each unit's received discount into account, we define the update counter $c_i^{(update)}$ similarly to $\eta(t)$ stated above, but here each unit i manages its own counter locally. The update counter is incremented by one at every time step where the unit's update is numerically important. In other words, when the update weight $P_i(t) > 10^{-16}$, the update counter is incremented by

$$c_i^{(update)} = 1 + \lambda(t)^{P_i(t)} c_i^{(update)}. \tag{22}$$

A unit i is deleted if $\langle\!\langle 1 \rangle\!\rangle_i(t)/c_i^{(update)} < T_{Delete}$, where T_{Delete} is a delete threshold.

(3) Split

The output variance of a unit i, $\sigma_i^2(t)$, represents the accumulated squared error between the unit's predictions and the real outputs. High variance values are related to the unit being in charge of a too large partition of the input space, and splitting such units can improve learning performance. Our split decision compares $\sigma_i^2(t)$ with the output variances of the other units using a local evaluation where only output variances of some nearest neighbors are considered. When the unit's output variance is considerably bigger than the biggest variance of its neighbors, the unit is split according to the split mechanism in [15].

(4) Merge

A merge manipulation is introduced to reduce redundancies in the network model. Redundancy refers to two network units approximating a similar partition of the input-output-space. For finding possible merge candidates, the grade of overlap between units has to be evaluated. The overlap is calculated here with the Bhattacharyya coefficient (BC) for that a closed form solution exits for two multivariate Gaussian distributions $G_1(\mu_1, \Sigma_1)$ and $G_2(\mu_2, \Sigma_2)$ [5].

$$\begin{aligned} d_B(G_1, G_2) =& \frac{1}{8} \cdot (\mu_1 - \mu_2)' \Sigma^{-1} (\mu_1 - \mu_2) \\ &+ \frac{1}{2} \cdot \log \frac{|\Sigma|}{\sqrt{|\Sigma_1| \cdot |\Sigma_2|}}, \end{aligned} \tag{23}$$

$$BC(G_1, G_2) = \exp\left(-d_B(G_1, G_2)\right). \tag{24}$$

Here, d_B is the Bhattacharyya distance and $\Sigma = (\Sigma_1 + \Sigma_2)/2$. The Bhattacharyya coefficient has the advantage that merge can be applied online based on currently available information of each unit, which is for example not possible for the merge mechanism in [18].

For a similarity $S(i, j)$ between two units i and j, we have to calculate the overlap of the units's input and output distributions. We denote the input distribution of a unit i as $G_i^{input}(\mu_i, \Sigma_i)$ and the output distribution as $G_i^{output}(\tilde{W}_i \tilde{x}, \sigma_i^2 I)$. The similarity $S(i, j)$ is calculated by

$$S(i, j) = BC(G_i^{input}, G_j^{input}) \cdot BC(G_i^{output}, G_j^{output}). \tag{25}$$

If $S(i, j) > T_{Merge}$ for a threshold T_{Merge}, then the units are possible merge candidates. The flow of the merge mechanism is described in the following:

1. Calculate the similarity $S(i, j)$ for all pairs $\{i, j\}$.
2. Choose the pair $\{i_{max}, j_{max}\}$ with maximal similarity.
3. If $S(i_{max}, j_{max}) > T_{Merge}$ then merge units, go to 1.
4. Otherwise, stop routine.

The merge mechanism becomes computationally heavy with increasing model complexity M. Furthermore, merge candidates are not found frequently, and it is sufficient to apply merge in intervals of a few hundred time steps. Yet, the calculation of the output BC depends on input x for the output center $\tilde{W}_i \tilde{x}$. Calculating similarities with current input $x(t)$ is inappropriate when performed in intervals. A possible alternative would be to use the weighted sum $\langle\!\langle \tilde{x} \rangle\!\rangle'_i(t) \equiv (\langle\!\langle x \rangle\!\rangle_i(t), \langle\!\langle 1 \rangle\!\rangle_i(t))$, however preliminary experiments have shown that this approach is overestimating the similarity between output distributions.

We revise the overlap calculation to avoid the inclusion of x in the output center for the BC calculation. A multivariate theorem for Gaussian distributions can be used to conduct an affine transformation of the output distribution. Here, we want to transform the output distribution $y \sim N(\tilde{W}_i \tilde{x}, \sigma_i^2 I)$ so that input x is not included in the center. For convenience, we consider the transformation of the transpose $y' \sim N(\tilde{x}' \tilde{W}'_i, \sigma_i^2 I)$ instead. After affine transformation, the output distribution becomes $\tilde{W}'_i \sim N(\tilde{W}'_i, \sigma_i^2 [\langle\!\langle \tilde{x} \tilde{x}' \rangle\!\rangle_i(t)]^{-1})$, and input x is excluded from the center. But $\tilde{W}'_i$ is an $(N+1) \times D$-dimensional matrix, and the left term of (23) becomes a $D \times D$-dimensional matrix dependent on the output dimension D. Therefore, we update (23) to

$$\begin{aligned} d_B(G_1, G_2) = \frac{1}{8D} \cdot Tr\left((\mu_1 - \mu_2)' \Sigma^{-1} (\mu_1 - \mu_2)\right) \\ + \frac{1}{2} \cdot \log \frac{|\Sigma|}{\sqrt{|\Sigma_1| \cdot |\Sigma_2|}}, \end{aligned} \tag{26}$$

where Tr is the trace of the matrix.

Finally, we have to merge units i and j when $S(i, j) > T_{Merge}$. Again, we consider the units' Gaussian distributions for the input and output space and merge the centers and covariances of input and output distributions in the same matter. New centers μ_{new} and covariances Σ_{new} are calculated by

$$\mu_{new} = \omega_i \mu_i + \omega_j \mu_j, \tag{27}$$

$$\Sigma_{new} = \sum_k^{\{i,j\}} \omega_k \left(\Sigma_k + (\mu_k - \mu_{new})(\mu_k - \mu_{new})'\right), \tag{28}$$

where $\omega_k = \frac{\langle\!\langle 1 \rangle\!\rangle_k(t)}{\langle\!\langle 1 \rangle\!\rangle_i(t) + \langle\!\langle 1 \rangle\!\rangle_j(t)}$ are functioning as weights with $\langle\!\langle 1 \rangle\!\rangle_k(t)$ representing the importance of unit i and j during training until current time step t, calculated with (21).

3.2.2 Self-Constructive Model Adaptation

In online learning schemes, domain knowledge is limited and an accurate model initialization before training is often difficult. Therefore, we build the network model dynamically and self-constructively during learning. The model is initialized with zero units ($M = 0$), and an initial unit is produced upon receiving the first training data sample. Some preset initialization is necessary for the first unit, where parameters are set to

$$\mu_1 = x(t) \tag{29}$$
$$\Sigma_1^{-1} = I_N \tag{30}$$
$$\sigma_1^2 = 0.1 \tag{31}$$
$$\tilde{W}_1 = (0, y(t)). \tag{32}$$

All other units are then created in relation to the first unit with the self-constructive model selection algorithm described below.

if M==0 **then**
  Produce first unit
else
  Compute Probabilities:
  $P(x(t), y(t)|\theta(t-1)) = \sum_{i=1}^{M} P(x(t), y(t), i|\theta(t-1))$
  if $P(x(t), y(t)|\theta(t-1)) < T_{Produce}$ **then**
    Produce new unit
  else
    Update units
    Test delete, split, merge
  end if
end if

4. Experiments

We evaluate the effectiveness of our proposed method, an NGnet with local forgetting (*LF*) that applies self-constructive model selection for the first time, and compare it with two older learning approaches ([2], [15]) for a function and chaotic time series approximation task.

4.1 Preparations

First of all, we discuss several preparative decision steps that have been taken before the experiments.

4.1.1 Compared Methods

In our experiments, we want to evaluate our proposed method in regard to the methods in [2] and [15].

In a first step, we compare our method with [2] to discuss differences between the newly proposed local forgetting update approach in Sect. 3.1 and the previous one [2]. In [2], model selection was yet to be considered. Since it is not applicable without proposing some changes in the delete manipulation, we do not apply dynamic model selection for this comparison. Instead, we compare the two methods with a static model size and identical initialization.

In a second step, we compare our method with an NGnet with global forgetting *GF* that has been proposed in [15] and where dynamic model selection has been considered. Here, we include dynamic model selection for our approach and compare it with two versions of *GF*: one is equal to [15] including a normalization coefficient (*GFnorm*) in its update function as in (11), while the other one employs only a discount factor (*GFdisc*) as in (10). For *GFdisc*, we need a normalizer for the delete mechanisms because of the same reasons described in Chapter 3.2.1 Paragraph (2). For global forgetting, the normalizer is equal to a unit life time counter, because units forget and need to be updated at every time step. Both global forgetting approaches apply dynamic model selection self-constructively in all experiments.

4.1.2 Dynamic Model Selection

Threshold parameters have to be set for the four manipulation mechanisms: produce (*Prod.*), delete (*Del.*), split (*Spl.*) and merge (*Mrg.*). We have conducted many preliminary experiments to find accurate parameters for all compared methods, and the best performing ones were selected separately for each experiment and method.

An overview of the selected parameters can be found in Table 1. We have summarized the two global forgetting methods as *GF* when the same parameters have been selected after extensive testing. For the function approximation task, we test different learning scenarios that are all marked with *FA* in the table. Some split parameters have the form 3× or 10×, which means three or ten times. In those cases, the split threshold is dynamic and set relative to another value. For the *chaotic* experiment, we also apply a dynamic threshold decision for *GFdisc*'s split mechanism, because the static one used in [15] has unstable performance. A unit is then split when its output variance is ten times bigger than the average of the other units' variances.

The produce threshold is set to the same parameter for all compared methods within an experiment. This is reasonable because the same produce mechanism is used by

Table 1 Manipulation parameter settings for all experiments.

Method	Parameter Settings			
	Prod.	Del.	Spl.	Mrg.
FA: Balanced				
LF (Prop.)	0.1	0.05	3×	0.7
GF	0.1	0.00001	0.12	-
FA: Imbalanced				
LF (Prop.)	0.1	0.0001	3×	0.9
GF	0.1	0.00001	0.12	-
FA: Dynamic				
LF (Prop.)	0.1	0.01	3×	0.7
GFdisc	0.1	0.001	0.1	-
Chaotic				
LF (Prop.)	0.00001	0.0001	3×	0.7
GFdisc	0.00001	0.0001	10×	-

all methods and differences in model selection behavior are more dependent on the other manipulations. After testing several values, we set $T_{Produce} = 0.1$ for the *FA* experiments and $T_{Produce} = 0.00001$ for the *chaotic* experiment.

4.1.3 Performance Evaluation

For all experiments, we evaluate the learning performance of the tested methods with the Root Mean Square Error (RMSE). The RMSE is defined by

$$RMSE = \sqrt{\frac{\Sigma_{i=1}^{n_{test}} (y_i - \hat{y})^2}{n_{test}}}, \tag{33}$$

where n_{test} is the number of test data samples.

Additionally, one-tailed permutation tests are conducted for the function approximation task. In the result tables, we present obtained p-values in the column *p-Val* to evaluate the significance of the learning results, comparing *LF* with the other tested methods.

4.1.4 Selection of Discount Factor

The discount factor $\lambda(t)$ that is applied in (21) is scheduled over time t with

$$\lambda(t) = 1 - \frac{1-a}{at+b}, \tag{34}$$

depending on the two parameters a and b. a controls how fast $\lambda(t) \to 1$ and b sets the initial value of λ. An example of a's influence is shown in Fig. 1. In our experiments, we apply several values for a and b to evaluate the effect of different discount schedules on the performance of the compared methods.

We also conduct experiments without forgetting. Here, we apply a discount factor $\lambda(t) = 1$ which is equal to any value b and $a = 1$. Then, *LF* updates (21) reduce to the same as *GFdisc*'s (10) making a fair comparison of their different model selection approaches possible.

4.2 Function Approximation Task

In the first experiment, we consider a commonly used function approximation task ([2], [15], [16]). The function has

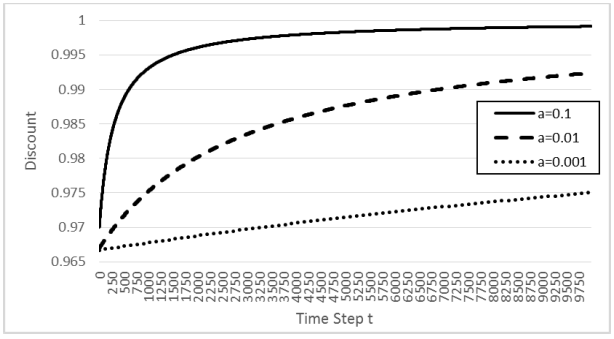


Fig. 1 Discount scheduling for $b = 30$.

the input dimension $N = 2$, the output dimension $D = 1$, and is defined by

$$g(x_1, x_2) = \max\{e^{-10x_1^2}, e^{-50x_2^2}, 1.25e^{-5(x_1^2+x_2^2)}\}. \tag{35}$$

A normally distributed random noise $\epsilon(t) \sim N(0, 0.01)$ is added to each function output $g(x_1(t), x_2(t))$, and $y(t) = g(x_1(t), x_2(t)) + \epsilon(t)$ is obtained as the noisy sample output.

4.2.1 Preparations

We consider three different test cases for the function approximation task. In the following, we discuss how we have obtained training and test data for these cases.

(1) Training Data

We use different input data distributions to evaluate the robustness of the compared methods in a variety of learning scenarios. When data are not i.i.d., neural networks are more prone to negative interference and achieving good performance is difficult. Three test cases are considered

- *Balanced*
- *Imbalanced*
- *Dynamic*

For the *balanced* test case, the training data are i.i.d over the whole input space $(-1 \le x_1, x_2 \le 1)$. For the *imbalanced* test case, we use non-identically distributed data. Here, 95% of the data samples are extracted from a subregion of the input domain with $(0 \le x_1, x_2 \le 0.25)$, and the remaining data are i.i.d in $(-1 \le x_1, x_2 \le 1)$. For these two cases, we obtain 10,000 data samples for each test run. For the *dynamic* test case, the input data distribution is slowly changing for x_1 over time. x_1's sample distribution starts at the interval $[-1, -0.2]$ and ends at the interval $[0.2, 1]$ after 250,000 training samples. x_2 is sampled in $[-1, 1]$ over all t. A similar experiment has been conducted in [2] and [15] to evaluate learning performance in dynamic environments.

In all test cases, we have obtained training data sets for 50 test runs, and the same data sets are applied to all compared methods.

(2) Test Data

Similarly to [2], we have obtained test data from a regular grid formed of 21×21 points in the input domain. These data are used to evaluate the learning performance of the compared methods over the whole input space in all three test cases. Additionally, we have obtained test data in the last input interval of the *dynamic* test case. Here, performance is evaluated by two test sets, representing the whole input space (*RMSE All*) and the last training interval (*RMSE Last*) respectively. For the function approximation task, all presented results are the averages of 50 test runs.

4.2.2 Experiments without Model Selection

In a first step, we compare our proposed method *LF (Prop.)*

Table 2 Balanced test without model selection.

Discount	Init (25)			Init (100)		
	RMSE		p-Val	RMSE		p-Val
	LF(Prop.)	LF ([2])		LF(Prop.)	LF ([2])	
b=30						
a=0.1	**0.0698**	0.0699	0.41	0.0541	0.0541	0.36
a=0.01	**0.0632**	0.0639	0.05	0.0497	0.0497	0.32
a=0.001	**0.0592**	0.0594	0.33	0.0455	0.0455	0.37
b=150						
a=0.1	**0.0713**	0.0714	0.36	**0.0548**	0.0549	0.29
a=0.01	0.0694	0.0694	0.20	0.0538	0.0538	0.36
a=0.001	**0.0687**	0.0688	0.14	0.0534	0.0534	0.4
b=1000						
a=0.1	0.0725	0.0725	0.14	0.0554	0.0554	0.27
a=0.01	0.0723	0.0723	0.11	0.0553	0.0553	0.31
a=0.001	0.0723	0.0723	0.11	0.0553	0.0553	0.32
Discount 1						
a=1.0	**0.0731**	0.2280	0.0	**0.0557**	0.2322	0.0

with the former local forgetting method (*LF [2]*) for the *balanced* test case. This experiment evaluates the effectiveness of the newly proposed update function (21) in regard to the previous one (12). The proposed method's main purpose is to eliminate the inability of *LF [2]* to update its parameters when the discount factor $\lambda(t)$ is one over all t.

Two different static model sizes are tested. A static model with 25 units is equal to the size used in [2], and we add some experiments with 100 units. Performance results are stated in Table 2 with *Init (25)* and *Init (100)* respectively. In both cases, units are initialized on a grid of the input space. For the initialization, (x, y) values are obtained according to the location of a unit on the grid, and the unit's parameters are then set according to (29) - (32).

Experimental results are presented in Table 2, with the best results marked in bold. We have applied different discount schedules for forgetting in the first nine cases, and *LF (Prop.)* has achieved better performance in all cases. Yet, the difference in learning performance is very small and results often in equal performances with *LF [2]* after rounding. For discount factors $\lambda(t) < 1$, the numerical difference is small between the two update methods, although it becomes bigger for larger discounts leading to more visible performance differences for $b = 150$ and $b = 30$. Overall, p-values also show that the superiority of *LF (Prop.)* cannot be claimed in most cases. There are however some exceptions for *Init (25)*. Comparing the results of the two different network sizes, we have noticed that an increase in model size comes with an improvement in learning performance. Interestingly, the bigger network size makes the performance gap between the two compared methods smaller, and obtained p-values show less significance between performance differences.

For the results without forgetting (*Discount 1*) in Table 2, *LF (Prop.)* performs much better than *LF [2]* for both network sizes. The p-values also prove the superiority of our proposal. Here, *LF [2]* was not able to update its network parameters, and its performance is the direct result of the initialization. Therefore, we have achieved the main purpose of the new proposal. Now, updates are possible for the local forgetting approach, even when the discount equals one. This complies with the constraint for the discount factor that has been set to $0 \leq \lambda \leq 1$ in [15].

Table 3 Balanced test without forgetting.

Method	RMSE	Net. Size	Manipulation Counter			
			Prod.	Del.	Spl.	Mrg.
Setting 1						
LF (Prop.)	0.0538	42.82	49.08	0	1.6	8.86
GFdisc	**0.0537**	47.78	44.52	0	2.26	0
(p-Val=0.46)						
Setting 2						
LF (Prop.)	**0.0425**	39.9	62.9	19.74	2.08	6.34
GFdisc	0.0525	42.16	67.94	28.96	2.18	0
(p-Val=0.0)						

4.2.3 Experiments with Model Selection

In a second step, we compare our proposed method *LF (Prop.)* with the global forgetting methods *GFnorm* and *GFdisc*. In the following, we will mainly refer to *LF (Prop.)* as *LF* for convenience.

(1) Balanced Test without Forgetting

Here, *LF* and *GFdisc* are compared with each other for a *balanced* test case without forgetting. Experimental results are presented in Table 3 for two different manipulation parameter settings. For both settings, we have chosen the same parameters as stated for *FA: Balanced* in Table 1 except for delete. The best performing results are marked in bold in the table. We have also included information about the number of manipulation occurrences.

For *Setting 1*, the delete parameter has been set to $T_{Delete} = 0.0001$ so that both methods do not delete units. Comparing the performance of *LF* and *GFdisc*, we can see that they are performing approximately the same. *GFdisc* is slightly better, but the high p-value shows that this difference is not significant. On the other hand, the network size is smaller for *LF* so that it performs equally well with a smaller network size.

For *Setting 2*, we set $T_{Delete} = 0.01$ for *GFdisc* and $T_{Delete} = 0.02$ for *LF* to compare the two methods when they have similar deletion behavior. The improved learning performances show that deleting units has a positive effect for both methods. Yet, while *LF* improves a lot, *GFdisc* is only improving slightly. The difference in learning performance is significant, supported by the p-value equaling zero. This emphasizes the effectiveness of the localized deleting approach applied to *LF*.

Finally, we compare the results of *LF* applying model selection with the results of *LF (Prop.)* without model selection in Table 2. Without model selection, *LF* has achieved $RMSE = 0.0557$ as the best performance. With model selection, *LF* performs better while having less network complexity for both settings presented in Table 3.

Table 4 Balanced test with forgetting.

Discount	LF (Prop.)		GFnorm			GFdisc		
	RMSE	Size	RMSE	Size	p-Val	RMSE	Size	p-Val
b=30								
a=0.1	0.0343	36.26	0.0279	197	0.0	**0.0148**	441.94	0.0
a=0.01	**0.0287**	41.9	0.0458	946.62	0.0	0.0472	1054.82	0.0
a=0.001	**0.0250**	49.6	0.0756	485.5	0.0	0.0827	553.38	0.0
b=150								
a=0.1	0.0358	35	0.0390	87.82	0.0	**0.0182**	134.24	0.0
a=0.01	**0.0338**	35.9	0.0376	480.8	0.0	0.0367	963.64	0.01
a=0.001	**0.0331**	36.52	0.0420	638.96	0.0	0.0429	1037.4	0.0
b=1000								
a=0.1	0.0372	34.1	0.0628	52.24	0.0	**0.0293**	55.86	0.0
a=0.01	0.0368	34.18	0.0511	55.84	0.0	**0.0256**	63.82	0.0
a=0.001	0.0367	34.16	0.0493	56.38	0.0	**0.0257**	65.54	0.0

(2) Balanced Test with Forgetting

For the *balanced* test case with forgetting, we compare *LF* with the two global forgetting approaches *GFnorm* and *GFdisc*. Experimental results are presented in Table 4 with the best results for each discount schedules marked in bold.

Here, *LF* is not the overall superior approach but shows the most robust performance over all discount schedules in regard to learning performance and network size. Also, it performs best for discount schedules where the learning behavior of the global forgetting methods is unstable. On the other hand, *GFdisc* is able to perform best for discount schedules where its learning is stable. This applies for discount schedules that are near to one from the start ($b = 1000$) or reaching one very fast ($a = 0.1$). For higher discounts, the performance of *GFdisc* steadily decreases and network sizes increase, resulting in non robust behavior. The learning behavior of *GFnorm* has the same tendencies as *GFdisc*'s over the different discount schedules. Yet, *GFnorm* is unable to perform best for any discount schedule.

The permutation tests were conducted comparing *LF* with either *GFnorm* or *GFdisc*. Regardless of whether the local or global forgetting method has performed better, the resulting p-values were approximately equal to zero in all cases proving the significance in the performance differences.

Finally, we compare the results of *LF* applying model selection with the results of *LF (Prop.)* without model selection in Table 2. Without model selection, *LF* has achieved learning performances between $RMSE = 0.0725$ and $RMSE = 0.0455$ depending on the network size and discount schedule. For *LF* with model selection, the performance is ranging between $RMSE = 0.0372$ and $RMSE = 0.0287$. So even in the worst performing case, *LF* performs still better and possesses smaller network sizes when model selection is applied. This emphasizes the advantage of applying dynamic model selection to *LF*.

(3) Imbalanced Test without Forgetting

For the *imbalanced* test case without forgetting, experimental results are presented in Table 5 with best results marked in bold. Here, *LF* and *GFdisc* trained NGnets are compared for discount $\lambda = 1$. Except for delete, manipulation parameters are set as stated in Table 1 for *FA: Imbalanced*. For delete, two threshold parameter settings are tested. In *Setting 1*, the threshold is set to $T_{Delete} = 0.0001$ so that no units are deleted, while in *Setting 2* we have relaxed the thresholds to $T_{Delete} = 0.001$ for both methods to show the differences in performance when deleting occurs.

Table 5 Imbalanced test without forgetting.

Method	RMSE	Net. Size	Manipulation Counter			
			Prod.	Del.	Spl.	Mrg.
Setting 1						
LF (Prop.)	**0.0958**	71.3	64.86	0	11.54	6.1
GFdisc	0.0985	67.28	65.84	0	0.44	0
(p-Val=0.01)						
Setting 2						
LF (Prop.)	**0.1036**	65.38	76.94	20.32	13.6	5.84
GFdisc	0.1234	54.16	81.82	29.26	0.6	0
(p-Val=0.0)						

Table 6 Imbalanced test with forgetting.

Discount	LF (Prop.)		GFnorm			GFdisc		
	RMSE	Size	RMSE	Size	p-Val	RMSE	Size	p-Val
b=30								
a=0.1	**0.0941**	76.42	0.1119	106.24	0.0	0.1080	167.14	0.0
a=0.01	**0.0934**	79.44	0.1892	113.58	0.0	0.2214	109.24	0.0
a=0.001	**0.0919**	81.92	0.2855	52.52	0.0	0.3300	49.52	0.0
b=150								
a=0.1	**0.0950**	72.72	0.1163	90.3	0.0	0.1064	159.22	0.0
a=0.01	**0.0951**	73.1	0.1495	110.56	0.0	0.1819	139.86	0.0
a=0.001	**0.0947**	72.66	0.1735	107.56	0.0	0.2026	130.48	0.0
b=1000								
a=0.1	**0.0949**	71.18	0.1460	56.08	0.0	**0.0949**	93.84	0.48
a=0.01	**0.0951**	71	0.1294	60.08	0.0	0.1058	113.96	0.0
a=0.001	**0.0950**	71.06	0.1293	61.1	0.0	0.1076	117.78	0.0

LF shows superior performance for both tested parameter settings with high significance as proved by the low p-values. In *Setting 1*, updates and deletions are the same for both methods so that *LF*'s favorable performance is related to the new split and merge approach. In *Setting 2*, some units have been deleted. This has led to a decrease in learning performance for both methods, but the decrease is visible lower for *LF*. The negative effect of delete is here smaller, likely because of the localized deleting approach applied to *LF*.

(4) Imbalanced Test with Forgetting

For the *imbalanced* test case with forgetting, the experimental results are presented in Table 6. Again, several discount schedules are applied and the best result for each schedule is marked in bold. Because of the imbalanced data distribution, this testbed is more prone to negative interference.

The results show favorable performance for our method *LF* in almost all cases with high significance proven by p-values equaling zero. There is one exception for the discount schedule ($a = 0.1, b = 1000$), where *LF* and *GFdisc* have approximately the same learning performance. While there is no significant difference between the performances, the net-

Table 7 Dynamic input distribution.

Discount/ Method	RMSE All	RMSE Last	Net. Size	Manipulation Counter			
				Prod.	Del.	Spl.	Mrg.
$\lambda = 1$							
LF(Prop.)	**0.0284**	**0.0123**	169.08	115.94	16.94	81.14	12.06
GFdisc (p-Val=0.0)	0.0344	0.0160	99.06	97.4	0	0.66	0
$\lambda = 0.999$							
LF(Prop.)	**0.0323**	0.0112	195.84	141.98	19.28	84.68	12.54
GFdisc (p-Val=0.0)	0.6872	**0.0077**	213.42	1118	906.24	0.66	0

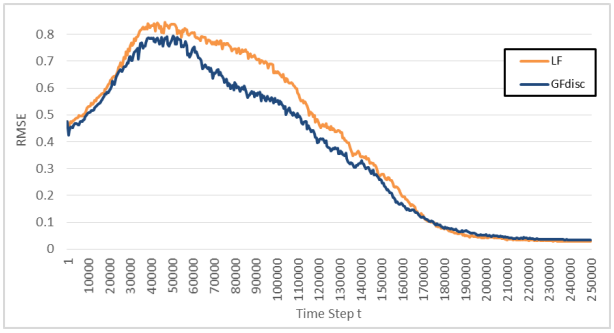


Fig. 2 Dynamic distribution without forgetting.

work size of *LF* is almost 25% smaller and therefore preferable. Additionally, *LF* shows a high stability in learning performance and network sizes over all discount schedules. On the other hand, the learning performances and network sizes of *GFnorm* and *GFdisc* are strongly varying. These results emphasize the robustness of *LF* in negative interference prone testbeds. *LF* is therefore favorable compared with the global forgetting methods.

(5) Dynamic Test without Forgetting

For the *dynamic* test case without forgetting, we train the compared methods with $\lambda(t) = 1$ over all t. Figure 2 presents the evolution of the learning performance during training over the whole input distribution $[-1, 1]$ for *GFdisc* and *LF*. *GFdisc* has better learning performance over a large part of training, but eventually looses out to *LF* in the last quarter. The final performance results are presented in Table 7, and *LF* possesses better approximation capabilities not only for the whole function (*RMSE All*) but also for the last interval $[0.2, 1]$ (*RMSE Last*). The favorable performance of *LF* is also supported by the p-value equaling zero.

(6) Dynamic Test with Forgetting

In the *dynamic* test case with forgetting, a static discount $\lambda(t) = 0.999$ is applied over the whole training time. This is equal to a discount schedule with $a = 0$ and $b = 1000$. We compare *LF* only with *GFdisc* for this testbed, since *GFnorm* was the least competitive for the other two test cases.

In Fig. 3, a huge performance difference is visible for the running error of *LF* and *GFdisc*. At the second time quarter, *GFdisc* seems to be successfully learning and even performs

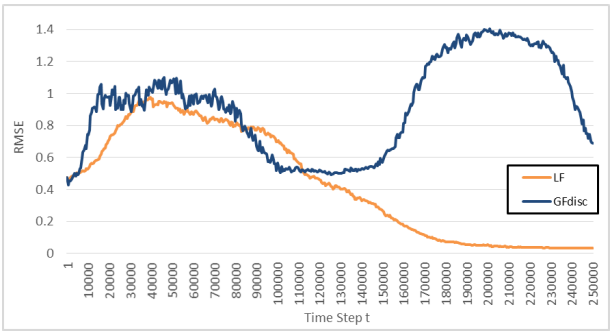


Fig. 3 Dynamic distribution with forgetting.

shortly better than *LF*. However, its performance decreases again. On the other hand, *LF* is able to steadily improve its approximation. The final performance results in Table 7 also show that *LF* is performing much better over the whole function. Interestingly, *GFdisc* performs better for the last training interval *RMSE Last*, but is unable to perform well over the whole function. Reasons for the bad performance of *GFdisc* are global forgetting and likely the high number of unit productions and deletions. *LF* also experiences a performance decrease in comparison with the non-forgetting case but is still able to perform similarly well. These results emphasize again the robustness of *LF*.

The delete thresholds of both methods are equal to the ones in the *dynamic* without forgetting case, so it is interesting to compare the deletion behavior. For *LF*, the number of unit deletions is only slightly increasing for the forgetting case, having a stable deletion behavior. On the other hand, while *GFdisc* has not deleted any units before, the number of deletions increased when discount was applied. This shows the high impact of forgetting on the unit manipulations in *GFdisc* and is likely one reason for its learning instabilities. *LF* is therefore preferable for online learning tasks, especially when data distributions are not i.i.d.

4.3 Chaotic Time Series Approximation Task

In the second experiment, we apply the NGnet to a chaotic time series approximation task to examine the learning performances for a realistic and difficult problem. The Lorenz attractor [10] is applied as testbed and is defined by

$$\begin{aligned} \dot{x} &= a(y - x), \\ \dot{y} &= bx - y - xz, \\ \dot{z} &= xy - cz. \end{aligned} \tag{36}$$

For the commonly used parameters $a = 10, b = 28, c = 8/3$, the attractor has chaotic behavior. Trajectories from the attractor are obtained by fourth order Runge-Kutta integration with integration time step $t_i = 0.001$. The vector notation of the attractor is $\dot{X} = F(X)$ where $X \equiv (x, y, z)'$ and F denotes the vector field. Additionally, some noise can be added to each observation with

$$Y(t) = X(t) + \xi(t), \tag{37}$$

Table 8 Chaotic time series testbed with $b = 150$.

Discount/ Data Noise	b=150								b=1000							
	LF (Prop.)				GFdisc				LF (Prop.)				GFdisc			
	RMSE1	RMSE2	Size	CD	RMSE1	RMSE2	Size	CD	RMSE1	RMSE2	Size	CD	RMSE1	RMSE2	Size	CD
a=0.1																
SD=0.0	**0.1676**	**0.1892**	159	**1.86**	0.1922	0.2298	319	1.63	0.1739	0.1915	148	**1.87**	**0.1410**	**0.1797**	155	1.70
SD=0.1	**0.1679**	**0.1894**	159	**1.86**	0.1696	0.2111	328	1.39	0.1750	0.1927	149	1.86	**0.1407**	**0.1794**	150	**1.87**
SD=0.2	0.1713	0.1921	159	**1.85**	**0.1530**	**0.1760**	307	1.70	0.1766	0.1936	150	1.81	**0.1446**	**0.1738**	158	**1.83**
SD=0.3	**0.1844**	**0.1974**	155	**1.80**	0.2003	0.2139	323	1.78	0.1851	0.1975	154	1.81	**0.1671**	**0.1857**	159	**1.89**
a=0.01																
SD=0.0	**0.1671**	**0.1890**	157	**1.88**	0.5779	0.6212	421	1.13	**0.1737**	**0.1914**	148	**1.87**	0.1885	0.2113	166	1.65
SD=0.1	**0.1678**	**0.1887**	156	**1.86**	0.5511	0.5580	408	0.34	**0.1748**	**0.1926**	149	**1.86**	0.2017	0.2379	170	1.65
SD=0.2	**0.1697**	**0.1910**	159	**1.85**	0.6094	0.6480	425	0.81	0.1765	0.1935	150	1.81	**0.1603**	**0.1909**	171	**1.82**
SD=0.3	**0.1824**	**0.1956**	155	**1.80**	0.6758	0.6848	442	1.25	**0.1849**	**0.1973**	154	**1.81**	0.1857	0.1997	184	0.00
a=0.001																
SD=0.0	**0.1666**	**0.1888**	157	**1.86**	0.7612	0.8457	388	1.13	**0.1737**	**0.1913**	148	**1.87**	0.1797	0.2059	169	1.57
SD=0.1	**0.1672**	**0.1884**	156	**1.87**	0.7746	0.8036	385	0.19	**0.1748**	**0.1926**	149	**1.86**	0.1942	0.2114	173	1.61
SD=0.2	**0.1692**	**0.1905**	160	**1.86**	0.8037	0.8328	383	0.00	0.1765	0.1935	150	**1.82**	**0.1634**	**0.1897**	170	1.80
SD=0.3	**0.1818**	**0.1951**	155	**1.80**	0.8262	0.8468	415	0.60	**0.1849**	**0.1973**	154	**1.81**	0.1867	0.2030	186	0.00

where the noise $\xi(t)$ is a white Gaussian noise with zero mean and some standard deviation SD.

Commonly, a time delay embedding method is used for the application of chaotic time series. According to Takens' theorem [17], the chaotic dynamics can be reconstructed from one dimension, e.g. dimension x. A next state $x(t+P)$ is then defined by the following delay coordinate

$$x(t+P) = \{x(t), x(t-\Delta t), \ldots, x(t-(d-1)\Delta t)\}, \quad (38)$$

where d is an embedding dimension and Δt is an embedding delay. The reconstruction of the chaotic dynamics is dependent on an appropriate choice of d and Δt. For the Lorenz attractor, $\Delta t = 0.15$ and $d = 3$ are sufficient to reconstruct the chaotic dynamics [7]. P is the number of prediction steps ahead and is set to $P = 1$.

4.3.1 Preparations

(1) Training and Test Data

We train the NGnets with 10,000 data samples sequentially obtained from one trajectory of the attractor with sample time step $t_s = 0.01$. Short-term prediction accuracy is evaluated by two test sets. One is noiseless (*RMSE1*) and the other test set added white Gaussian noise with $SD = 0.2$ (*RMSE2*). For both test sets, 10,000 data samples are randomly obtained from several different trajectories.

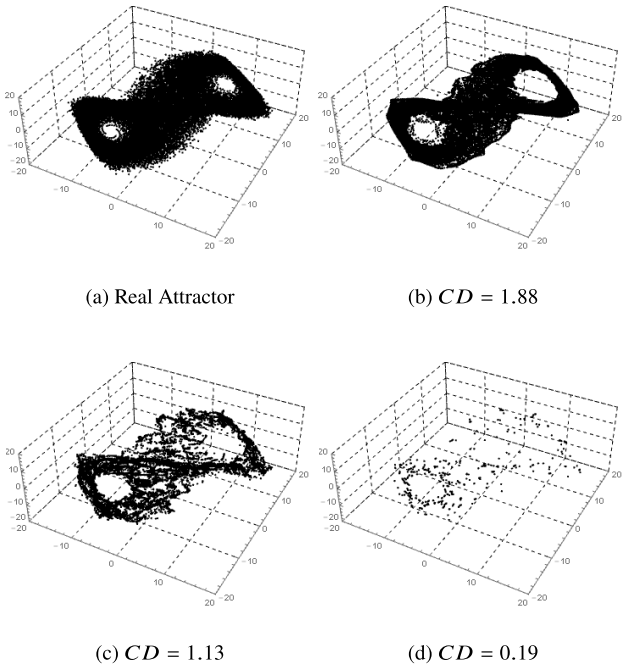
(2) Correlation Dimension (CD)

Additionally, we evaluate the long-term characteristics [3] of the trained NGnet models by obtaining recursively predicted trajectories with 30,000 samples. Recursive means here that the NGnets use their own predicted outputs as the input for the next prediction step. The correlation dimension (CD) [4] is a measure to evaluate the properties of a chaotic time series. CDs are used to evaluate the long-term characteristics of the trained models. For each trained model, we obtain three recursively predicted trajectories from different initial states. Then, CDs are calculated for each trajectory and their average is presented in the test results. The CD of the real Lorenz attractor is $CD_{Original} = 2.05 \pm 0.01$ [4], while for the embedding delay method it is approximately $CD_{Embedded} = 2.021$ [7]. If a calculated CD is similar to the real CD, then the trained model was able to imitate the chaotic characteristics of the Lorenz attractor. We consider CD values over 1.8 as good long-term characteristics.

4.3.2 Experimental Results

The effectiveness of the proposed method *LF* is compared with *GFdisc* for several discount schedules with $b = 150$ and $b = 1000$ in Table 8. Presented results are obtained from test runs with different grades of noise added to the training data, denoted by the noise standard deviation SD. The results show that *LF* has robust short-term performances and long-term characteristics in all test cases. Especially, *LF* shows very good long-term characteristics with no CD below 1.80. There are a few cases where *GFdisc* has better short-term performance or long-term characteristics mainly for $b = 1000$, but it misses the robustness shown by *LF*. Also, better short-term prediction is not necessarily an indicator for good long-term characteristics. For some cases, *LF* has better long-term characteristics even when short-term performance is worse than *GFdisc*'s. This phenomenon has also been noted in [3]. Similarly to the first experiment, *GFdisc* again has difficulties to show stable performance for any discount schedule except ($a = 0.1$, $b = 1000$). It performs especially bad for two out of three discount schedules with $b = 150$. Also, *GFdisc* shows no robustness for its long-term characteristics with only five CD values over 1.8. Overall, *LF* is performing robust in all test cases and its overall performance is clearly better than *GFdisc*.

In Fig. 4(a)–4(d), we show some examples for how different CD values relate to the long-term characteristics. 30,000 data points are presented in all four figures, but Fig. 4(d) seems sparse because the recursive prediction al-



(a) Real Attractor (b) $CD = 1.88$

(c) $CD = 1.13$ (d) $CD = 0.19$

Fig. 4 Real attractor (a) compared with recursive predictions (b)–(d).

ways returns to the same points. Fig. 4(c) looks more like the real attractor but still has some holes. Fig. 4(b) has a compact point distribution in space similar to the real attractor. All learned models have difficulties to represent the inner edges accurately resulting in the difference to the real CD.

The results of the second experiment support the results of the first experiment. *LF* mostly performs better, and shows much more robustness over a large range of different learning situations. It is therefore the preferable method to apply to online learning tasks especially when domain knowledge is limited and negative interference likely a problem.

5. Conclusion

In this paper, we have proposed an online self-constructive NGnet with localized forgetting. Our major contributions are a modification of a previously proposed localized forgetting method and the application of dynamic model selection to it for the first time. In addition, several improvements are made for the dynamic model selection to achieve better robustness of the NGnet in negative interference prone environments and for handling redundancies in the model. The proposed method has been evaluated by two testbeds, a function approximation task and a chaotic time series forecasting task. For the function approximation task, i.i.d. and non i.i.d. data distributions have been tested to show the robustness of the proposed method when prone to negative interference. We have compared our method with a previously proposed localized forgetting method to discuss differences and advantages of applying dynamic model selection. Then, we have compared it with an established method for the NGnet employing global forgetting and dynamic model selection, and results show that our method is able to employ overall robust learning behavior while the global approach's performance is highly dependent on the given learning environment.

Possible future work includes the development of an automatic parameter selection method to further ease the model adaptation of NGnets in online learning tasks, which is especially helpful when domain knowledge is limited.

References

[1] G. Bugmann, "Normalized Gaussian radial basis function networks," Neurocomputing, vol.20, no.1-3, pp.97–110, Aug. 1998.
[2] E. Celaya and A. Agostini, "On-line EM with weight-based forgetting," Neural Comput., vol.27, no.5, pp.1142–1157, May 2015.
[3] M.R. Cowper, B. Mulgrew, and C.P. Unsworth, "Nonlinear prediction of chaotic signals using a normalised radial basis function network," Signal Process., vol.82, no.5, pp.775–789, May 2002.
[4] P. Grassberger and I. Procaccia, "Characterization of strange attractors," Phys. Rev. Lett., vol.50, no.5, pp.346–349, Jan. 1983.
[5] C. Hennig, "Methods for merging Gaussian mixture components," Advances in Data Analysis and Classification, vol.4, no.1, pp.3–34, 2010.
[6] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," IEEE Trans. Neural Netw., vol.16, no.1, pp.57–67, Jan. 2005.
[7] S. Ishii and M.-A. Sato, "Reconstruction of chaotic dynamics by on-line EM algorithm," Neural Networks, vol.14, no.9, pp.1239–1256, Nov. 2001.
[8] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks," Neural Comput., vol.5, no.6, pp.954–975, Nov. 1993.
[9] H.J. Kushner and G.G. Yin, Stochastic Approximation Algorithms and Applications, Springer Verlag, New York, 1997.
[10] E.N. Lorenz, "Deterministic non-periodic flow," J. Atmos. Sci., vol.20, no.2, pp.130–141, March 1963.
[11] Y. Lu, N. Sundararajan, and P. Saratchandran, "A sequential learning scheme for function approximation using minimal radial basis function neural networks," Neural Comput., vol.9, no.2, pp.461–478, Feb. 1997.
[12] J. Moody and C.J. Darken, "Fast learning in networks of locally-tuned processing units," Neural Comput., vol.1, no.2, pp.281–294, June 1989.
[13] J. Platt, "A resource-allocating network for function interpolation," Neural Comput., vol.3, no.2, pp.213–225, June 1991.
[14] M. Sato, "Convergence of on-line EM algorithm," Proc. International Conference on Neural Information Processing, vol.1, pp.476–481, 2000.
[15] M. Sato and S. Ishii, "On-line EM algorithm for the normalized Gaussian network," Neural Comput., vol.12, no.2, pp.407–432, Feb. 2000.
[16] S. Schaal and C.G. Atkeson, "Constructive incremental learning from only local information," Neural Comput., vol.10, no.8, pp.2047–2084, Nov. 1998.
[17] F. Takens, "On the numerical determination of the dimension of an attractor," in Dynamical Systems and Bifurcations, Lecture Notes in Mathematics, vol.1125, pp.99–106, Springer-Verlag, Berlin, 1985.
[18] N. Ueda, R. Nakano, Z. Ghahramani, and G.E. Hinton, "SMEM algorithm for mixture models," Neural Comput., vol.12, no.9, pp.2109–2128, 2000.
[19] L. Xu, M. Jordan and G.E. Hinton, "An alternative model for mixtures of experts," in Advances in Neural Information Processing Systems 7, ed. J.D. Cowan, G. Tesauro and J. Alspector, pp.633–640, MIT Press, Cambridge MA, 1995.

Jana Backhus received her Master of Information Science and Technology in the field of Computer Science at the Graduate School of Information Science and Technology, Hokkaido University in 2014, and is currently working towards the Ph.D. degree at the same institution.

Ichigaku Takigawa received his D.E. from Hokkaido University in 2004 and is currently an Associate Professor at the Department of Computer Science and Information Technology, Graduate School of Information Science and Technology, Hokkaido University. His research interests include machine learning and data mining.

Hideyuki Imai received the D.E. from Hokkaido University in 1999. He is currently an Professor at the Department of Computer Science and Information Technology, Graduate School of Information Science and Technology, Hokkaido University. His research interests include statistical inference.

Mineichi Kudo received his Dr. Eng. degree in Information Engineering from Hokkaido University in 1988. Starting as an instructor at Hokkaido University in 1988, he is a professor since 2001. In 2001, he received the twenty-seventh annual pattern recognition society award together with professor Jack Sklansky. He was elected as a fellow of the International Association for Pattern Recognition on December 10, 2008. His current research interests include design of pattern recognition systems, image processing, data mining and computational learning theory. He is a member of the Pattern Recognition Society and the IEEE.

Masanori Sugimoto received the B.E., M.E., and D.E. degrees from the University of Tokyo, Tokyo, Japan, in 1990, 1992 and 1995, respectively. He is currently a Professor in the Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan. His research interests include acoustic engineering, signal processing, artificial intelligence, and human-computer interaction technologies for designing smart systems and environments.