Learning to imitate stochastic time series in a compositional way by chaos

Jun Namikawa

Brain Science Institute, RIKEN

2-1 Hirosawa, Wako-shi, Saitama, 351-0198 Japan

E-mail `jnamika@bdc.brain.riken.go.jp`

Jun Tani

Brain Science Institute, RIKEN

2-1 Hirosawa, Wako-shi, Saitama, 351-0198 Japan

E-mail `tani@brain.riken.go.jp`

Abstract

This study shows that a mixture of RNN experts model can acquire the ability to generate sequences that are combination of multiple primitive patterns by means of self-organizing chaos. By training of the model, each expert learns a primitive sequence pattern, and a gating network learns to imitate stochastic switching of the multiple primitives via a chaotic dynamics, utilizing a sensitive dependence on initial conditions. As a demonstration, we present a numerical simulation in which the model learns Markov chain switching among some Lissajous curves by a chaotic dynamics. Our analysis shows that by using a sufficient amount of training data, balanced with the network memory capacity, it is possible to satisfy the conditions for embedding the target stochastic sequences into a chaotic dynamical system. It is also shown that reconstruction of a stochastic time series by a chaotic model can be stabilized by adding a negligible amount of noise to the dynamics of the model.

1 Introduction

Recurrent neural networks (RNNs) [3, 12, 20, 29] are abstract models of nervous systems that can perform temporal sequence processing. RNNs have been applied to temporal sequence learning such as sensory-motor sequence patterns [12, 13], symbolic sequences with grammar [3, 20] and continuous spatio-temporal patterns [29]. However, in spite of the considerable amount of RNN research carried out since the mid-1980s, it has been thought that RNNs could not be scaled so as to be capable of learning complex sequence patterns, especially when the sequence patterns to be learned contain long-term dependencies [2]. This is due to the fact that the error signal cannot be propagated effectively in long-time windows of sequences using the back-propagation through time (BPTT) algorithm [22], because of the potential nonlinearity of the RNN dynamics [2]. How to form neural network models that can learn complex temporal sequence patterns has proved to be a challenging problem.

There have been some breakthroughs in approaches to this problem. Echo state networks [10, 11] and a very similar approach, liquid state machines [17], have recently attracted considerable attention. An echo state network possesses a large pool of hidden neurons with fixed random weights, and the pool is capable of rich dynamics. Jaeger [11] demonstrated that an echo state network can successfully learn the Mackey-Glass chaotic time series which is a well-known benchmark system for time series prediction. Long Short-Term Memory (LSTM) proposed by Hochreiter and Schmidhuber [7, 23] has been investigated as a neural network architecture which is more efficient than standard RNN in terms of memory capability. LSTM learns to bridge minimal time lags in long time steps by enforcing constant error flow through "constant error carousels" within special units. These units learn to open and close access to the constant error flow.

Tani and Nolfi [26] investigated the same problems, but from a different angle, focusing on the idea of compositionality for sensory-motor learning. The term compositionality was adopted from the "Principle of Compositionality" [5] in linguistics, which claims that the meaning of a complex expression is determined by the meanings of its constituent expressions and rules used to combine them. This principle, when translated to the context of sensory-motor learning, leads to the assertion that varied and complex patterns can be learned by adaptively combining reusable behavior primitives. Here, acquiring behavior primitives requires a mechanism for autonomously segmenting a continuously experienced sensory-motor flow into reusable chunks. Tani and Nolfi proposed a scheme for hierarchical segmentation of the sensory-motor flow, applying the idea of a mixture of experts [9, 14, 30] to hierarchically organized RNNs. Consider a network consisting of multiple local RNNs, organized in two levels, referred to as the base level and a higher level. At the base level, RNNs called experts competitively learn prediction or generation of specific sensory-motor profiles. As the winner among the experts changes, corresponding to structural changes in the sensory-motor flow, the sensory-motor flow is segmented by switching between winners. Meanwhile, an RNN at the higher level, usually called a gating network, learns the sequence patterns of winner switching with much slower time constants than those at the base level. The higher level learns not for details of sensory-motor patterns but for abstraction of primitive sequences with long-term dependencies.

However, how to practically perform learning at the higher level is a difficult problem, because the gating network is usually a deterministic system, whereas arbitrary composition of primitives with a certain probability distribution brings nondeterminacy to the sequence patterns of winner switching. A possibly easy solution to this problem is to apply the output values of the gating network as a probability of winner switching, or more directly, a stochastic model such as a hidden Markov model (HMM) is used instead of the gating network. Of course, if we use a stochastic model at the higher level, nondeterministic sequences can be learned by the model. Nevertheless, using a stochastic model might remove from the dynamics the stability brought about by the effect of basins of attraction of RNN experts, because stochastic models such as HMMs can operate only with discrete value sequences, so a trajectory jumps discontinuously whenever a winner changes. In particular, when sequences to be learned are continuous, for example, a sensory-motor flow, the stability might be lost by using a stochastic model, because sequence patterns of winner switching are also continuous. An alternative solution, which is the main focus of the current paper, is to imitate nondeterminacy by using the gating network itself. It has been reported that a nondeterministic process can be imitated by a chaotic dynamical system from the view of symbolic dynamics [6, 16]. Moreover, Tani et al. [19, 25] showed that an RNN can acquire the ability to imitate a nondeterministic symbolic process by experimental simulations. The approach imitating a nondeterministic process using the gating network offers the advantage not only of preserving the dynamical stability, but also that of a consistent approach in terms of dynamical systems theory to both the base and higher levels. In addition, by using the neural networks for both levels we could discuss the relation to information processing in brain.

In the current study, we investigate learning for a mixture of experts model focusing on the higher level, and show that the model can learn nondeterministic sequence patterns of winner switching by a self-organizing internal chaos. In the previous study [18], we demonstrated that a mixture of experts model with our proposed learning method was able to learn to segment time series consisting of many primitive patterns, whereas in earlier studies such a series could not be segmented owing to near-miss problems in matching the current sequence pattern to the best expert among others which had acquired similar pattern profiles. The previous study, however, focused on the learning process of segmenting sequences into reusable primitives with respect to spatio-temporal patterns, and so did not deal with higher level learning. This study is therefore focused on higher level learning.

2 Model

A mixture of RNN experts is simply a mixture of experts model for which experts are RNNs. The mixture of RNN experts consists of expert networks together with a gating network (see Figure 1). All experts receive the same input and have the same number of output neurons. The gating network receives the past gate opening values and the input and controls gate opening. The role of each expert is to compute a specific input-output function, and the role of the gating network is to decide which single expert is the winner on each occasion.

The dynamic states of the mixture of RNN experts at time n are updated according to

$$\boldsymbol{u}_n^{(i)} = \big(1 - \epsilon\big)\boldsymbol{u}_{n-1}^{(i)} + \epsilon\big(W_1^{(i)}\boldsymbol{x}_n + W_2^{(i)}\boldsymbol{c}_{n-1}^{(i)} + \boldsymbol{v}_1^{(i)}\big), \tag{1}$$

$$\boldsymbol{c}_n^{(i)} = \tanh(\boldsymbol{u}_n^{(i)}), \tag{2}$$

$$\boldsymbol{y}_n^{(i)} = \tanh(W_3^{(i)}\boldsymbol{c}_n^{(i)} + \boldsymbol{v}_2^{(i)}), \tag{3}$$

$$\boldsymbol{y}_n = \sum_{i=1}^{N} g_n^{(i)}\boldsymbol{y}_n^{(i)}, \tag{4}$$

where N is the number of experts, $\boldsymbol{x}_n$ and $\boldsymbol{y}_n$ are an input and output of the model respectively. For each i, $g_n^{(i)}$, $\boldsymbol{c}_n^{(i)}$ and $\boldsymbol{y}_n^{(i)}$ denote the gate opening value, context and output of the expert network i respectively. Here we assume $g_n^{(i)} \geq 0$ and $\sum_{i=1}^{N} g_n^{(i)} = 1$. The gate opening vector $\boldsymbol{g}_n$ represents the winner-take-all competition among experts to determine the output $\boldsymbol{y}_n$. The gate opening vector $\boldsymbol{g}_n$ is the output of a gating network defined by

$$\boldsymbol{u}_n^g = \big(1 - \epsilon^g\big)\boldsymbol{u}_{n-1}^g + \epsilon^g\big(W_1^g\boldsymbol{g}_{n-\tau} + W_2^g\boldsymbol{x}_{n-\tau} + W_3^g\boldsymbol{c}_{n-1}^g + \boldsymbol{v}_1^g\big), \tag{5}$$

$$\boldsymbol{c}_n^g = \tanh(\boldsymbol{u}_n^g), \tag{6}$$

$$\boldsymbol{b}_n = W_4^g\boldsymbol{c}_n^g + \boldsymbol{v}_2^g, \tag{7}$$

$$g_n^{(i)} = \frac{\exp(b_n^{(i)})}{\sum_{k=1}^{N} \exp(b_n^{(k)})}, \tag{8}$$

where τ represents a feedback time delay. In order to satisfy $g_n^{(i)} \geq 0$ and $\sum_{i=1}^{N} g_n^{(i)} = 1$, equation (8) is given by the soft-max function. Using the sigmoid function denoted by $sigmoid(x) = \frac{1}{1+\exp(-x)}$, the equation (8) can be expressed as

$$g_n^{(i)} = sigmoid\big(b_n^{(i)} - \ln \sum_{k \neq i} \exp(b_n^{(k)})\big). \tag{9}$$

This equation indicates that the output of the gating network is given by the sigmoid function with global suppression.

2.1 Learning Method

2.1.1 Basic procedure

We present the procedure for training the mixture of RNN experts, organized into two phases as follows:

(1) We compute the experts learning together with an estimate $\hat{\boldsymbol{g}}_n$ of the gate opening vector.

(2) Using the estimate $\hat{\boldsymbol{g}}_n$ of gate opening vector as a target, we compute learning for the gating network.

Note that in phase (1) we use the method proposed in [18]. This learning is a maximum likelihood estimation using the gradient descent method. The training procedure progresses with phase (2) after convergence of phase (1).
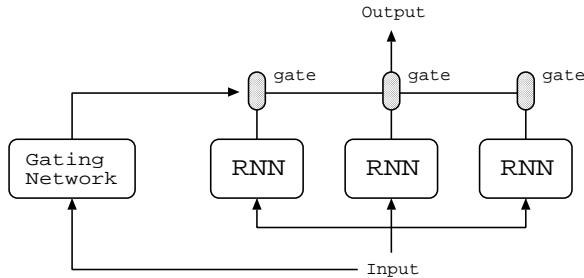
Figure 1: A system comprised of a mixture of RNN experts.

2.1.2 Probability distribution

In order to define the learning method, we assign a probability distribution to the mixture of RNN experts. Here, adjustable parameters of an expert network i and a gating network are denoted by $\vartheta_i = \left(W_1^{(i)}, W_2^{(i)}, W_3^{(i)}, \boldsymbol{v}_1^{(i)}, \boldsymbol{v}_2^{(i)}, \boldsymbol{u}_0^{(i)}\right)$ and $\boldsymbol{\theta}^g = \left(W_1^g, W_2^g, W_3^g, W_4^g, \boldsymbol{v}_1^g, \boldsymbol{v}_2^g, \boldsymbol{u}_0^g\right)$, respectively. An estimate $\hat{\boldsymbol{g}}_n$ for the gate opening vector is described by a variable $\boldsymbol{\beta}_n$ such as

$$\hat{g}_n^{(i)} = \frac{\exp(\beta_n^{(i)})}{\sum_{k=1}^N \exp(\beta_n^{(k)})}. \tag{10}$$

Let $X = (\boldsymbol{x}_n)_{n=1}^T$ be an input sequence, $\boldsymbol{\beta}_n$ and $\boldsymbol{\gamma}$ be parameters, where $\boldsymbol{\gamma}$ is a set of parameters given by $\gamma_i = (\vartheta_i, \sigma_i)$. Given X, $\boldsymbol{\beta}_n$, and $\boldsymbol{\gamma}$, a probability density function (p.d.f.) for the output $\boldsymbol{y}_n$ is defined by

$$p(\boldsymbol{y}_n \mid X, \boldsymbol{\beta}_n, \boldsymbol{\gamma}) = \sum_{i=1}^N \hat{g}_n^{(i)} p(\boldsymbol{y}_n \mid X, \gamma_i), \tag{11}$$

where $p(\boldsymbol{y}_n | X, \gamma_i)$ is given by

$$p(\boldsymbol{y}_n | X, \gamma_i) = \left(\frac{1}{\sqrt{2\pi}\sigma_i}\right)^d \exp(-\frac{||\boldsymbol{y}_n^{(i)} - \boldsymbol{y}_n||^2}{2\sigma_i^2}), \tag{12}$$

d is the output dimension, and $\boldsymbol{y}_n^{(i)}$ is the output from expert i computed by the equations (1), (2) and (3) with parameter ϑ_i. Thus, the output of the model is governed by a mixture of normal distributions. This equation results from the assumption that the observable sequence data is embedded in additive Gaussian noise. It is well known that minimizing the mean square error is equivalent to maximizing the likelihood determined by a normal distribution for learning in a single neural network. Therefore, equation (12) is a natural extension of neural network learning. The details of the derivation of equation (12) are given in [9].

Given a parameter set $\boldsymbol{\theta} = \left((\boldsymbol{\beta}_n)_{n=1}^T, \boldsymbol{\gamma}\right)$ and an input sequence X, the probability of an output sequence $Y = (\boldsymbol{y}_n)_{n=1}^T$ is given by

$$p(Y \mid X, \boldsymbol{\theta}) = \prod_{n=1}^T p(\boldsymbol{y}_n \mid X, \boldsymbol{\beta}_n, \boldsymbol{\gamma}). \tag{13}$$

The function L of a data set $D = (X, Y)$, parameterized by $\boldsymbol{\theta}$, is defined by the product of the likelihood $p(Y \mid X, \boldsymbol{\theta})$ and a prior distribution, which is denoted by

$$L(D, \boldsymbol{\theta}) = p(Y \mid X, \boldsymbol{\theta})\varphi(\boldsymbol{\theta}), \tag{14}$$

4

where $\varphi(\boldsymbol{\theta})$ is the p.d.f. of a prior distribution given by

$$\varphi(\boldsymbol{\theta}) = \prod_{n=1}^{T-1} \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\varsigma} \exp\left(-\frac{(\beta_{n+1}^{(i)} - \beta_n^{(i)})^2}{2\varsigma^2}\right). \tag{15}$$

This equation means that the vector $\boldsymbol{\beta}_n$ is governed by N-dimensional Brownian motion. The prior distribution has the effect of suppressing the change of gate opening values.

Assume that $D^g = (X, (\hat{\boldsymbol{g}}_n)_{n=1}^T)$ is given. We define the likelihood $L(D^g, \boldsymbol{\theta}^g)$ by the multinomial distribution

$$p(\hat{\boldsymbol{g}}_n \mid \boldsymbol{g}_n) = \frac{1}{Z(\hat{\boldsymbol{g}}_n)} \prod_{i=1}^{N} \left(g_n^{(i)}\right)^{\hat{g}_n^{(i)}}, \tag{16}$$

$$L(D^g, \boldsymbol{\theta}^g) = \prod_{n=1}^{T} p(\hat{\boldsymbol{g}}_n \mid \boldsymbol{g}_n), \tag{17}$$

where $Z(\hat{\boldsymbol{g}}_n)$ is the normalization constant, and $\boldsymbol{g}_n$ is given by X and $\boldsymbol{\theta}^g$. Notice that $\boldsymbol{g}_n$ in equation (17) is calculated by the teacher forcing technique [29], namely, using $\hat{\boldsymbol{g}}_{n-\tau}$ instead of $\boldsymbol{g}_{n-\tau}$ in equation (5). In addition, because of the term $\boldsymbol{x}_{n-\tau}$ on the right-hand side of equation (5), $\boldsymbol{g}_n$ cannot be computed if $n \leq \tau$. Then it is assumed that $\boldsymbol{g}_n = \hat{\boldsymbol{g}}_n$ if $n \leq \tau$. If we consider $g_n^{(i)}$ to be the probability of choosing an expert i at time n, maximizing $\ln L(D^g, \boldsymbol{\theta}^g)$ is equivalent to minimizing the Kullback-Leibler divergence

$$D_{\mathrm{KL}}((\hat{\boldsymbol{g}}_n)_{n=1}^T \| (\boldsymbol{g}_n)_{n=1}^T) = \sum_{n=1}^{T} \sum_{i=1}^{N} \hat{g}_n^{(i)} \ln\left(\frac{\hat{g}_n^{(i)}}{g_n^{(i)}}\right). \tag{18}$$

2.1.3 Maximum likelihood estimation

The learning method is to choose the best parameters $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^g$ by maximizing (or integrating over) the likelihood $L(D, \boldsymbol{\theta})$ and $L(D^g, \boldsymbol{\theta}^g)$. More precisely, we use the gradient descent method with a momentum term as the training procedure. The model parameters at learning step t are updated according to

$$\Delta\boldsymbol{\theta}^*(t) = \alpha \frac{\partial \ln L(D^*, \boldsymbol{\theta}^*(t))}{\partial \boldsymbol{\theta}^*(t)} + \eta \Delta\boldsymbol{\theta}^*(t-1) \qquad \text{where } (D^*, \boldsymbol{\theta}^*) \text{ is } (D, \boldsymbol{\theta}) \text{ or } (D^g, \boldsymbol{\theta}^g), \tag{19}$$

$$\boldsymbol{\theta}^*(t+1) = \boldsymbol{\theta}^*(t) + \Delta\boldsymbol{\theta}^*(t), \tag{20}$$

where α is the learning rate and η is a momentum term parameter. For each parameter, the partial differential equations $\frac{\partial \ln L(D, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ are given by

$$\frac{\partial \ln L(D, \boldsymbol{\theta})}{\partial \beta_n^{(i)}} = \frac{\hat{g}_n^{(i)}(p(\boldsymbol{y}_n \mid X, \gamma_i) - p(\boldsymbol{y}_n \mid X, \boldsymbol{\beta}_n, \boldsymbol{\gamma}))}{p(\boldsymbol{y}_n \mid X, \boldsymbol{\beta}_n, \boldsymbol{\gamma})} + \frac{G_n^{(i)}}{\varsigma^2}, \tag{21}$$

$$G_n^{(i)} = \begin{cases} \beta_{n+1}^{(i)} - \beta_n^{(i)} & \text{if } n = 1, \\ -\beta_n^{(i)} + \beta_{n-1}^{(i)} & \text{if } n = T, \\ \beta_{n+1}^{(i)} - 2\beta_n^{(i)} + \beta_{n-1}^{(i)} & \text{otherwise,} \end{cases} \tag{22}$$

$$\frac{\partial \ln L(D, \boldsymbol{\theta})}{\partial \vartheta_i} = \sum_{n=1}^{T} \frac{\hat{g}_n^{(i)} p(\boldsymbol{y}_n \mid X, \gamma_i)}{p(\boldsymbol{y}_n \mid X, \boldsymbol{\beta}_n, \boldsymbol{\gamma})} \left[\frac{-1}{2\sigma_i^2} \frac{\partial}{\partial \vartheta_i} \|\boldsymbol{y}_n^{(i)} - \boldsymbol{y}_n\|^2\right], \tag{23}$$

$$\frac{\partial \ln L(D, \boldsymbol{\theta})}{\partial \sigma_i} = \sum_{n=1}^{T} \frac{\hat{g}_n^{(i)} p(\boldsymbol{y}_n \mid X, \gamma_i)}{p(\boldsymbol{y}_n \mid X, \boldsymbol{\beta}_n, \boldsymbol{\gamma})} \left[-\frac{d}{\sigma_i} + \frac{\|\boldsymbol{y}_n^{(i)} - \boldsymbol{y}_n\|^2}{\sigma_i^3}\right], \tag{24}$$

5

and $\frac{\partial \ln L(D^g, \boldsymbol{\theta}^g)}{\partial \boldsymbol{\theta}^g}$ is also given by

$$\frac{\partial \ln L(D^g, \boldsymbol{\theta}^g)}{\partial \boldsymbol{\theta}^g} = \sum_{n=1}^{T} \sum_{i=1}^{N} \frac{\hat{g}_n^{(i)}}{g_n^{(i)}} \left(\frac{\partial g_n^{(i)}}{\partial \boldsymbol{\theta}^g} \right). \tag{25}$$

Note that $\frac{\partial}{\partial \vartheta_i} ||\boldsymbol{y}_n^{(i)} - \boldsymbol{y}_n||^2$ and $\frac{\partial g_n^{(i)}}{\partial \boldsymbol{\theta}^g}$ can be solved for by the back propagation through time (BPTT) method [22]. In this paper, we assume that the infimum $\bar{\sigma}$ of the parameter σ_i is greater than 0, because if σ_i converges to 0, then $\Delta \theta(t)$ diverges.

We have explained the case in which the training data set D is a single sequence, but the method can be easily extended to learning of several sequences by using the sum of the gradients for each sequence. When several sequences are used as training data, initial states $\boldsymbol{u}_0^{(i)}$ and $\boldsymbol{u}_0^g$, and an estimate $\hat{\boldsymbol{g}}_n$ of the gate opening vector have to be provided for each sequence.

2.1.4 Acceleration of gating network learning

In principle, our training procedure is defined by equations (19) and (20). However, for reasons which will be explained in section 4.1, learning for a gating network sometimes becomes unstable, and so the likelihood $L(D^g, \boldsymbol{\theta}^g)$ often decreases when updating the parameter $\boldsymbol{\theta}^g$ with certain learning rates. Of course, if we use sufficiently small learning rates, the likelihood does not decrease for any learning steps; however, it does require considerable computation time. Hence, in order to accelerate gating network learning practically, we update the learning rate adaptively by means of the following algorithm.

(1) For each learning step, updated parameters are computed by equations (19) and (20) applying the learning rate α, and the rate r defined by

$$r = \frac{D_{\mathrm{KL}}((\hat{\boldsymbol{g}}_n)_{n=1}^{T} || (\boldsymbol{g}'_n)_{n=1}^{T})}{D_{\mathrm{KL}}((\hat{\boldsymbol{g}}_n)_{n=1}^{T} || (\boldsymbol{g}_n)_{n=1}^{T})} \tag{26}$$

is also computed, where $(\boldsymbol{g}_n)_{n=1}^{T}$ and $(\boldsymbol{g}'_n)_{n=1}^{T}$ are sequences of gate opening values corresponding to the current parameters and the updated parameters, respectively.

(2) If $r > r_{\mathrm{th}}$, then α is replaced with $\alpha \alpha_{\mathrm{dec}}$ and we return to (1). Otherwise go to (3).

(3) If $r < 1$, then α is replaced with $\alpha \alpha_{\mathrm{inc}}$. Go to the next learning step.

In this study, we use $r_{\mathrm{th}} = 1.1$, $\alpha_{\mathrm{dec}} = 0.7$ and $\alpha_{\mathrm{inc}} = 1.05$.

2.1.5 Initialization

Here, we describe a policy for initialization of parameters and setting a learning rate in this paper. Every element of matrices and thresholds of a network, either an expert or a gating network, is initialized by choosing randomly from the uniform distribution on the interval $[-\frac{1}{M}, \frac{1}{M}]$, where M is the number of context neurons. Initial states $\boldsymbol{u}_0^{(i)}$ and $\boldsymbol{u}_0^g$ are also initialized randomly from the interval $[-1, 1]$. The parameters $(\boldsymbol{\beta}_n)_{n=1}^{T}$ and $\boldsymbol{\sigma}$ are initialized such that $\boldsymbol{\beta}_n = \boldsymbol{0}$ for each n and $\boldsymbol{\sigma} = \boldsymbol{1}$, respectively. Since the maximum value of $L(D, \boldsymbol{\theta})$ depends on the total length T of training sequences and the dimension d of output neurons, we scale the learning rate α by a parameter $\tilde{\alpha}$ satisfying $\alpha = \frac{1}{Td} \tilde{\alpha}$.

2.2 Feedback loop with time delay

Let us consider the case in which there exists a feedback loop from output to input with a time delay τ, that is to say, the model is an autonomous system. In this case, a training data set $D = (X, Y)$ has to satisfy $\boldsymbol{y}_n = \boldsymbol{x}_{n+\tau}$. At the end of learning, if every output of the model is completely equal to the training data, the model can generate the sequence Y using a feedback loop without external

input data X. In this paper, the dynamics of the model with external input is referred to as open-loop dynamics, and dynamics with self-feedback is also called closed-loop dynamics. In section 3, we consider the case in which the model is an autonomous system.

3 Numerical simulation

We now demonstrate how the mixture of RNN experts learns to generate sequences given as tasks. Our first experiment in section 3.1 is to learn a stochastic finite state automaton by means of a gating network, in order to test whether a gating network can output an estimate of the gate opening vector even if switching of winners is stochastic. We include this test for two reasons. (1) Since a time series of gate opening vectors represents an arbitrary composition of primitives, such a time series is usually a symbolic and nondeterministic sequence. Here a sequence is nondeterministic if the next value of the sequence cannot be uniquely determined from past values. (2) It has been shown that the simple recurrent network (SRN or Elman net) can learn symbolic and nondeterministic sequences such as declarative sentences [3, 4]. The SRN has typically been used to learn to predict the probability of occurrence of the next symbol, and the output values of the SRN represented this probability. For example, if each symbol is to appear next with equal probability, the output values of the SRN are constant, summing to one. However, in the context of the mixture of RNN experts model, the weighted sum of output values of all experts by probability does not represent a correct value. In order to output the correct value, the gating network has to determine a winner among the experts, and the output of the gating network corresponding to the winner should be almost one. This means that the gating network has to learn to output not the probability but stochastic variables instead. It is not trivial to determine whether the gating network can learn this task, because the network is a deterministic dynamical system. At least, if the maximum Lyapunov exponent of the network is a negative value, the output of the network is periodic and so cannot represent any nondeterministic sequences. Basically, a necessary condition to output nondeterministic sequences using a deterministic dynamical system is that the maximum Lyapunov exponent is positive.

On the other hand, it is known that there are chaotic dynamics which resemble nondeterministic sequences, because it is possible to create a correspondence between a hyperbolic dynamical system and a symbolic dynamics by constructing a Markov partition [16]. Tani et al. [19, 25] demonstrated that a nondeterministic symbolic process can be imitated by an RNN trained by the gradient descent method, utilizing initial sensitivity characteristics of a chaotic dynamics. However, they only investigated the case in which the length of training data is about 10. Thus, in our first experiment we show that the gating network can acquire a chaotic dynamics and can generate nondeterministic sequences, even if training data are much longer.

Our second experiment in section 3.2 is to learn Markov chains whose primitives are Lissajous curve patterns. It was shown that a mixture of RNN experts can extract a set of reusable primitives from training data generated by a Markov chain of Lissajous curves [18], but learning for the gating network was not considered. Therefore, we will show that a mixture of RNN experts can generate stochastically combined Lissajous curves by switching winners using a gating network.

In order to evaluate the learning capability of a mixture of RNN experts, we first define some measures of fitness. An error E for expert networks is defined by the mean square error

$$E = \frac{1}{2Td} \sum_{n=1}^{T} ||\boldsymbol{y}_n^{\text{teach}} - \hat{\boldsymbol{y}}_n||^2 \tag{27}$$

for each learning step, where $\boldsymbol{y}_n^{\text{teach}}$ denotes the training data, and $\hat{\boldsymbol{y}}_n = \sum_{i=1}^{N} \hat{g}_n^{(i)} \boldsymbol{y}_n^{(i)}$. An error E^g

for the gating network is defined by the Kullback-Leibler divergence

$$
\begin{aligned}
E^g & = \frac{1}{TN} D_{\mathrm{KL}}((\hat{\boldsymbol{g}}_n)_{n=1}^T || (\boldsymbol{g}_n)_{n=1}^T) \\
& = \frac{1}{TN} \sum_{n=1}^T \sum_{i=1}^N \hat{g}_n^{(i)} \ln \Big(\frac{\hat{g}_n^{(i)}}{g_n^{(i)}} \Big),
\end{aligned}
\tag{28}
$$

where we assume that E and E^g are computed by teacher forcing, i.e., utilizing the training data and estimates of gate opening values as network input. If $E = E^g = 0$, then the mixture of RNN experts can output sequences equivalent to the training data. Since E and E^g are measures for open-loop dynamics, we next define the Kullback-Leibler divergence to evaluate fitness in the closed-loop dynamics.

$$
D_{\mathrm{KL}}^{(m)}((\hat{s}_n)_{n=1}^T || (s_n)_{n=1}^T) = \sum_{\boldsymbol{s} \in \{1, \cdots, N\}^m} p(\boldsymbol{s}|(\hat{s}_n)_{n=1}^T) \ln \frac{p(\boldsymbol{s}|(\hat{s}_n)_{n=1}^T)}{p(\boldsymbol{s}|(s_n)_{n=1}^T)},
\tag{29}
$$

where s_n denotes the index of a current opening gate $s_n \in \{1, \cdots, N\}$ defined by

$$
s_n = \arg \max_i g_n^{(i)}
\tag{30}
$$

and $\hat{s}_n$ is given by equation (30) with respect to $\hat{\boldsymbol{g}}_n$. By using this measure, we can perform an evaluation even if the training data are stochastic time series that are combination of several primitive patterns. In addition, this measure can apply to compare the output with not only training data but also test data. When we evaluate the Kullback-Leibler divergence for the test data, we reconstruct an estimate $\hat{\boldsymbol{g}}_n$ of the gate opening vector for the test data by using the the gradient descent method, and we compare the output of the gating network with $\hat{\boldsymbol{g}}_n$. The Kullback-Leibler divergence by means of test data can be regarded as a measure of the generalization error of the gating network.

3.1 Preliminary experiment: Learning a stochastic finite state automaton

Here, we investigate how a gating network learns time series of gate opening values which form stochastic sequences. As we focus on training of the gating network, we assume that the dimension of the input $\boldsymbol{x}_n$ is $d = 0$. In other words, any effects of experts and the input-output sequence $D = (X, Y)$ are omitted. As a benchmark system, we use a finite state automaton described in Figure 2. This automaton is given by modifying the Reber grammar to an Ergodic model. When we generate a sequence using the automaton, we choose a path randomly from two paths with equal probability. Representation of the symbols by gate opening values is such that each symbol is mapped with a one to one correspondence to an integer from 1 to 7, and if a symbol corresponding to i appears at time n, then $\hat{g}_n^{(i)} = 1$.

As an example, we computed learning for the gating network up to 10000 steps. The number of context neurons, time constant, time delay of a feedback loop, learning rate and momentum were chosen to be 20, $\epsilon^g = 0.1$, $\tau = 1$, $\tilde{\alpha} = 0.01$ and $\eta = 0.9$, respectively. Here we use 4 training sequences of length $T = 500$ (so the total length of training sequences is 2000). In addition, to evaluate the generalization capability of the network, we prepared 4 test sequences of length $T = 500$ separate from the training data, and we computed the Kullback-Leibler divergence for the test data not only the training data. In Figure 3, we depict the error, Kullback-Leibler divergence and Lyapunov exponents for each learning step, where the Kullback-Leibler divergence and Lyapunov exponents are computed for the closed-loop dynamics. The Kullback-Leibler divergence was measured with $m = 3$, because the stochastic process defined by Figure 2 is a second order Markov chain. Since the maximum Lyapunov exponent becomes positive, the gating network acquires a chaotic dynamics by training. Moreover, acquiring the chaotic dynamics is effective for imitation of the stochastic process, because the Kullback-Leibler divergence for both the training data and test data decreases with the increase in the maximum Lyapunov exponent. Figure 4 displays the dynamics of the trained network with a
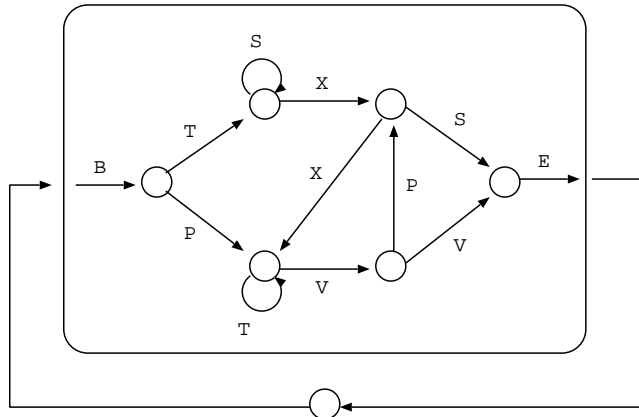
Figure 2: Finite state automaton based on the Reber grammar. If there are two paths, then we randomly choose one with equal probability.

Table 1: The results of training (mean of 50 trials). A percentage in round brackets represents the ratio of trained networks whose maximum Lyapunov exponent is positive. When we evaluated a Lyapunov exponent, we computed 200 sample sequences of $10,000$ time steps for each trained network.

Number of context neurons	Error E^g	Kullback-Leibler divergence $D_{\mathrm{KL}}^{(m)}((\hat{s}_n)_{n=1}^T \| (s_n)_{n=1}^T)$		Lyapunov exponent
		(for training data)	(for test data)	
10	0.144868	2.511791	2.388665	0.203512 (88%)
20	0.010118	0.258037	0.309089	0.215117 (98%)
30	0.005721	0.051233	0.327273	0.160728 (100%)
50	0.004339	0.006338	0.290858	0.116574(100%)

random initial state. It can be seen that the output of the network is almost always 0 or 1. This result indicates that the network learned to output not the probability but stochastic variables. Furthermore, there is a case for which the output of the network does not satisfy the syntax of the Reber grammar (printed as red color symbols over the gate opening values in Figure 4), but the dynamics recovers the correct output patterns.

Table 1 shows the results of training for each number of context neurons. For each case, we computed 50 samples up to 10000 learning steps with different initial conditions, where we use the same parameters as for the above training example without the number of context neurons. In addition, in order to evaluate the generalization capability of trained networks, we prepared test data separate from the training data, and we computed the Kullback-Leibler divergence $D_{\mathrm{KL}}^{(m)}((\hat{s}_n)_{n=1}^T \| (s_n)_{n=1}^T)$ for the test data with random initial states. It can be seen that if there exist sufficient context neurons to learn the training data, the dynamics of some trained networks become chaotic and the Kullback-Leibler divergences between sequences generated by the networks and training/test data become sufficiently small.

3.2 Learning stochastic switching of Lissajous curves

In the current section, we consider two tasks, which are to generate time series stochastically combining Lissajous curves. Assume that for each task, transitions among curves are consonant with orbit continuity. Since we consider a model which has a feedback loop with time delay τ, every training data set $D = (X, Y)$ satisfies $\boldsymbol{y}_n = \boldsymbol{x}_{n+\tau}$.
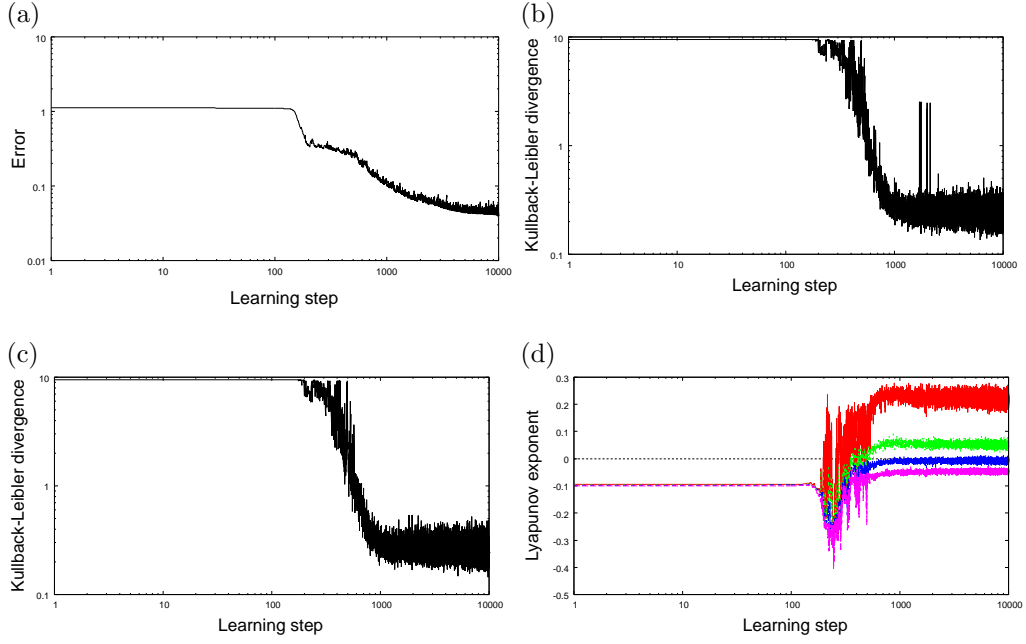
9

Figure 3: (a) The error E^g. (b) Kullback-Leibler divergence $D_{\mathrm{KL}}^{(3)}((\hat{s}_n)_{n=1}^T||(s_n)_{n=1}^T)$ for the training data. (c) Kullback-Leibler divergence $D_{\mathrm{KL}}^{(3)}((\hat{s}_n)_{n=1}^T||(s_n)_{n=1}^T)$ for the test data. (d) Lyapunov exponents up to fourth order.
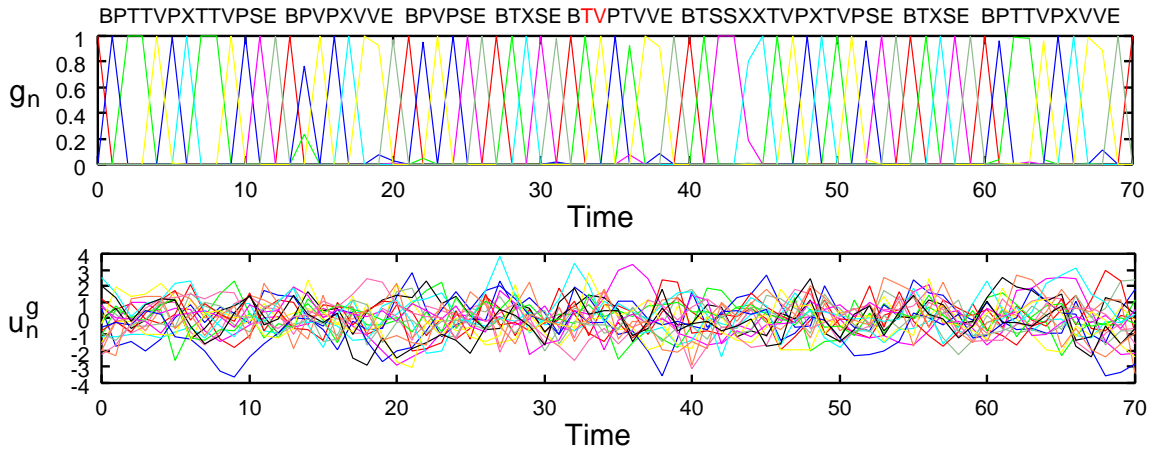

Figure 4: A snapshot of $\boldsymbol{g}_n$ and $\boldsymbol{u}_n^g$ of the trained network with closed-loop dynamics. Each label over gate opening values denotes a symbol corresponding to the current opening gate, and a red label means that the symbol does not satisfy the syntax of the Reber grammar.

10
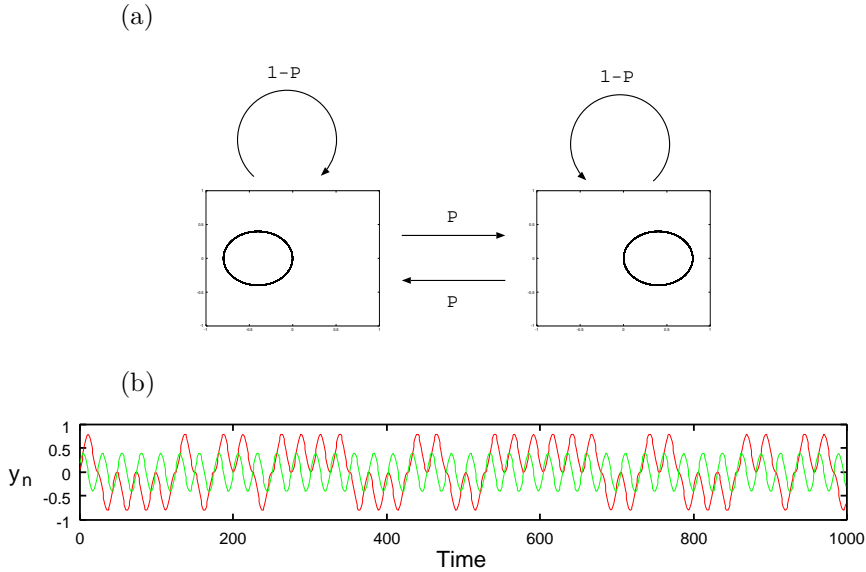
Figure 5: (a) Transition rule of the training data. Each figure represents the trajectory of a Lissajous curve, and P denotes the transition probability between curves. (b) An example of the training data.

3.2.1 Two Lissajous curves case

The first task is to learn 2-dimensional sequences generated by Markov chain switching of 2 Lissajous curves of period 25 (see Figure 5). We use 10 training sequences and 10 test sequences each of length $T = 1000$ and with transition probability $P = 0.5$. The time delay of the feedback loop is $\tau = 1$. The number of experts is $N = 2$. There are 10 context neurons for each expert and 15 for the gating network. The parameter settings are $\epsilon = 0.2$, $\epsilon^g = 0.04$, $\bar{\sigma} = 0.05$, $\varsigma = 10$, $\tilde{\alpha} = 0.01$ and $\eta = 0.9$.

Figure 6 depicts an example of results for training the mixture of RNN experts with this task. Here, we computed learning for experts up to 30000 steps, and learning for the gating network up to 10000 steps. In Figure 6 (c) and (d), Kullback-Leibler divergence drops after 100 steps, though the maximum Lyapunov exponent is not still positive. This result implies that the network acquires periodic behaviors similar to the training data and test data, because a negative maximum Lyapunov exponent indicates periodic dynamics of the network by definition. This task requires acquiring the skill to emulate stochastic behaviors, and so the training is not still sufficient even if Kullback-Leibler divergence is small. In order to measure successful learning, we should compute both the Kullback-Leibler divergence and the maximum Lyapunov exponent. After 1000 steps, the maximum Lyapunov exponent becomes positive together with further decreasing Kullback-Leibler divergence. Figure 7 and 8 depict trajectories of the trained network computed from the closed-loop dynamics. Figure 7 indicates that the trained network can generate trajectories similar to the training data. Moreover, from Figure 6 (e) and Figure 8, it can be observed that the network has acquired stochastic switching between Lissajous curves using a chaotic dynamics.

In Table 2 and Figure 9, to show that the model can acquire probabilities of transition, we compare cases in which the training data are generated by different transition probabilities. Here, we present evaluated results of 30 samples for each case with the same parameters as for the previous training. Table 2 describes transition probabilities between curves and maximum Lyapunov exponents, and Figure 9 also describes probability distributions for the number of repetitions of each curve. These results confirm that the mixture of RNN experts can acquire the transition probabilities of the task.
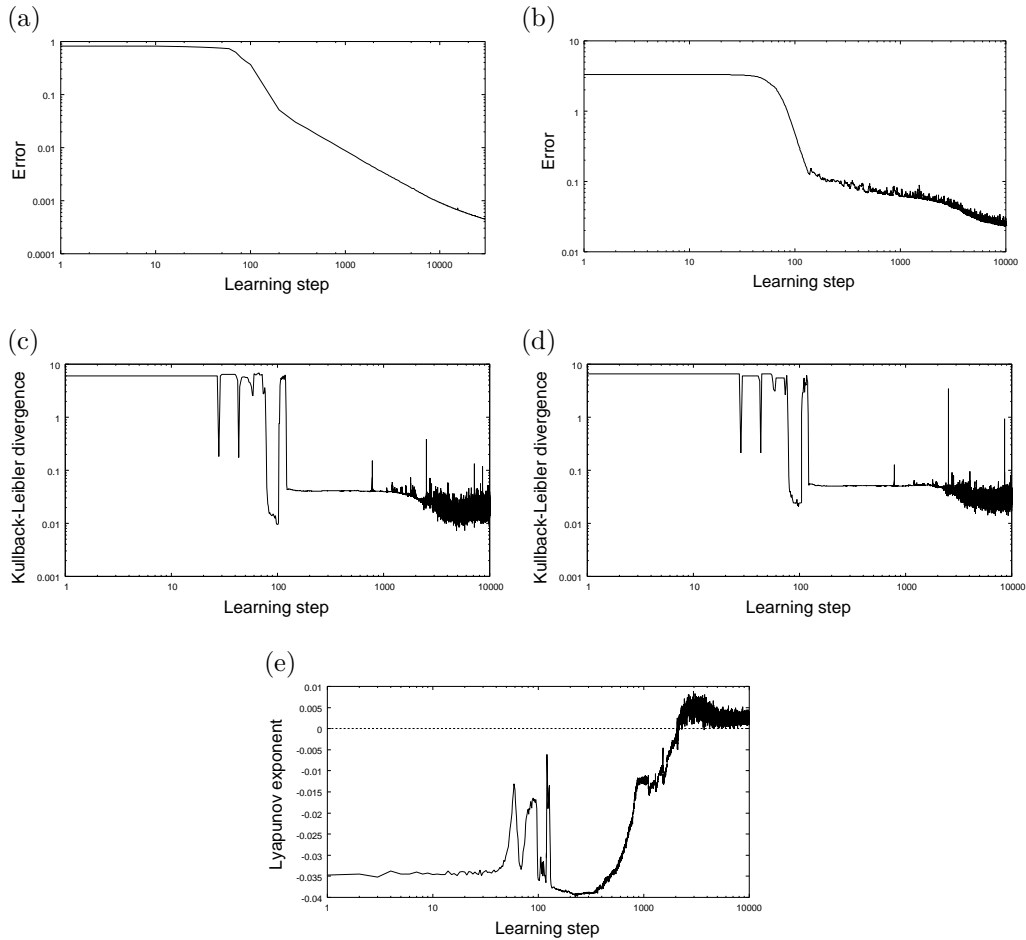
11

(a)

(b)

(c)

(d)

(e)

Figure 6: (a) The error of experts E. (b) The error of the gating network E^g. (c) The Kullback-Leibler divergence $D_{\mathrm{KL}}^{(5)}((\hat{s}_n)_{n=1}^{T}||(s_n)_{n=1}^{T})$ for the training data. (d) The Kullback-Leibler divergence $D_{\mathrm{KL}}^{(5)}((\hat{s}_n)_{n=1}^{T}||(s_n)_{n=1}^{T})$ for the test data. (e) Maximum Lyapunov exponent.

Table 2: The results of training (mean of 30 trials). Here, a probability refers to a transition probability between curves. When we evaluated the probability and the Lyapunov exponent, we computed 50 sample sequences of $100,000$ time steps for each network.

Probability (task)	Probability (trained network)	Lyapunov exponent
0.5	0.451033	0.001076 (93%)
0.25	0.275999	0.002080 (96%)
0.125	0.150866	0.002829 (96%)

12
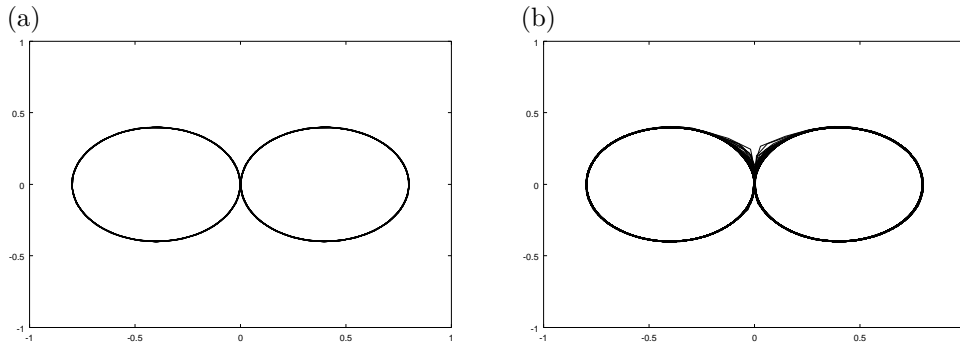
Figure 7: (a) Teaching data. (b) A trajectory of the mixture of RNN experts with closed-loop dynamics (plotted over $10,000$ time steps after convergence of the transients).
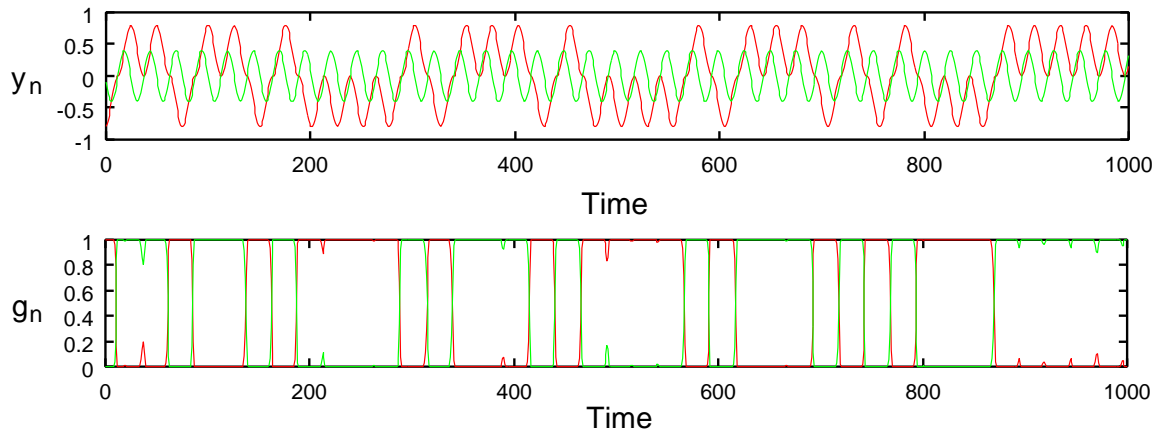

Figure 8: Output $\boldsymbol{y}_n$ and gate opening vector $\boldsymbol{g}_n$ of the mixture of RNN experts with closed-loop dynamics.
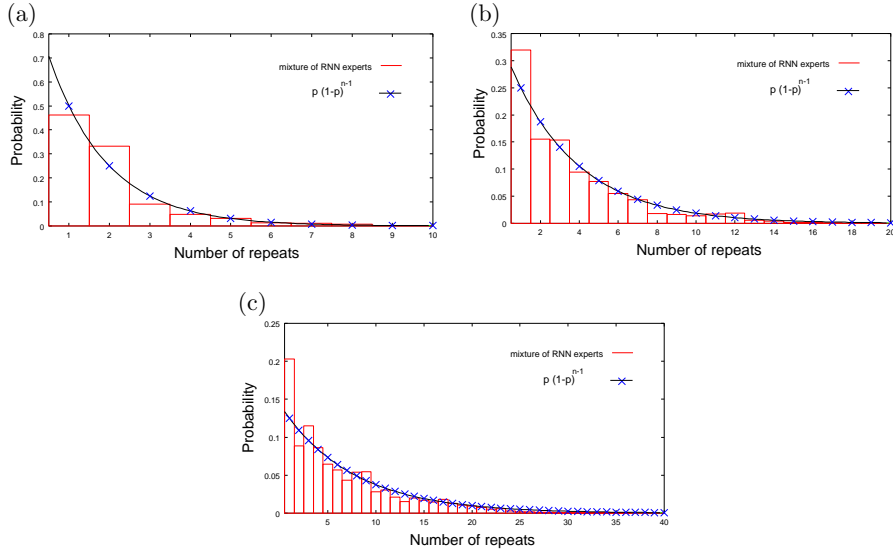
Figure 9: Distributions of the number of repetitions of each Lissajous curve, for each transition probability P of training data. (a) $P = 0.5$, (b) $P = 0.25$, (c) $P = 0.125$. Here, a histogram describes an averaged distribution of a whole set of trained networks, and a line describes the theoretical distribution of training data. When we evaluated the distributions, we computed 50 sample sequences of $100,000$ time steps for each network.

3.2.2 Multiple Lissajous curves case

The second task is to learn time series defined by switching among 9 Lissajous curves of period 32 (see Figure 10). In the task, we use a training sequence of length $T = 10000$ with a time delay of feedback $\tau = 5$, and prepare a test sequence of the same length. We have $N = 24$ experts, each with 10 context neurons, and the gating network has 30 context neurons.

In Figure 11, we display an example of training, where parameters were set as follows: $\epsilon = 0.1$, $\epsilon^g = 0.04$, $\bar{\sigma} = 0.05$, $\varsigma = 1$, $\tilde{\alpha} = 0.01$ and $\eta = 0.9$. It can be seen that the maximum Lyapunov exponent oscillates between positive and negative values after 1000 learning steps; concurrently the Kullback-Leibler divergence changes drastically. This phenomenon implies that properties of the network as a dynamical system are changed by updating parameters, and the network is not structurally stable. In the case, the stability of varying a likelihood (or an error) is not corresponding to that of varying the properties of the model. Actually, in Figure 11 the error decreases monotonically, whereas the maximum Lyapunov exponent oscillates. In general, since the topological property of a dynamical system being not structurally stable is changed by perturbations [1], it is difficult to prevent oscillation of the maximum Lyapunov exponent even when the learning rate is very small. In section 4.3, we will explain and discuss this problem of the structural stability of learning models in detail.

Figures 12 and 13 represent closed-loop dynamics of the trained network with a random initial state, where the network has a positive Lyapunov exponent. It can be seen in Figure 12 (b) that there are two groups of experts; one group represents primitives, and the other represents transition paths between primitives. Furthermore, taking into account the whole set of transition paths, the transition rule for the network described in Figure 12 (b) is similar to that for the training data described in Figure 10 (b).
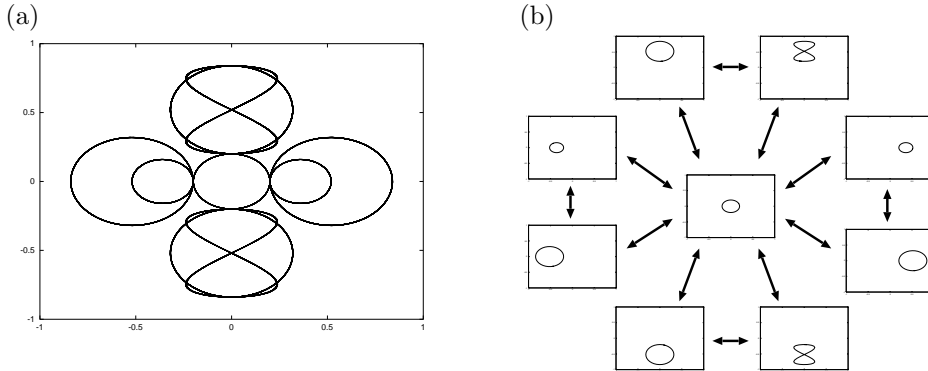
14

Figure 10: (a) Training data generated by switching among 9 Lissajous curves. (b) Each Lissajous curve. Transition among curves is consonant with continuity of the orbit. Keeping a current curve or moving to another curve is chosen randomly with equal probability.

3.3 Summary of results

In the preliminary experiment, we tested the gating network for its capability to learn stochastic sequences given by a finite state automaton with a chaotic dynamics. The result shows that, by increasing the number of context neurons, the Kullback-Leibler divergence can be reduced and the frequency of appearance of chaos also increases.

For the next experiment, we considered the task of learning stochastic time series consisting of Lissajous curves. In the case of two Lissajous curves, a mixture of RNN experts can imitate by chaos not only nondeterministic characteristics but a probability distribution underlying training data. Moreover, the networks can maintain a chaotic dynamics stably at the end of learning. In the case of multiple Lissajous curves, the trained network can generate a similar sequence to the training data, though the maximum Lyapunov exponent is not always positive in the last learning phase. Since this instability characteristic of chaos is due to the structurally unstable dynamics of the network, we will discuss this problem of structural stability in section 4.3.

4 Analysis and Discussion

In this paper, we have investigated learning processes of a gating network contained in the mixture of RNN experts model. In particular, we have focused on how the gating network learns stochastic sequences brought by an arbitrary composition of primitives by self-organizing chaos. The fact that the gating network can learn stochastic time series is important, because it had been considered that it is impossible for RNNs of deterministic dynamical systems to learn nondeterministic time series if the length of training data becomes long [19, 25].

In the following we discuss some general characteristics in the learning of stochastic time series by embedding into chaos. We focus on two essential problems, related to memory capacity and to the structural stability of RNNs, and explain how the problems can be overcome.

4.1 A problem in learning stochastic time series by means of a deterministic but chaotic dynamics

Here, we explain a problem that appears in particular when stochastic time series are learned by a dynamical system. To examine the problem, we consider the difficulty of learning of time series by
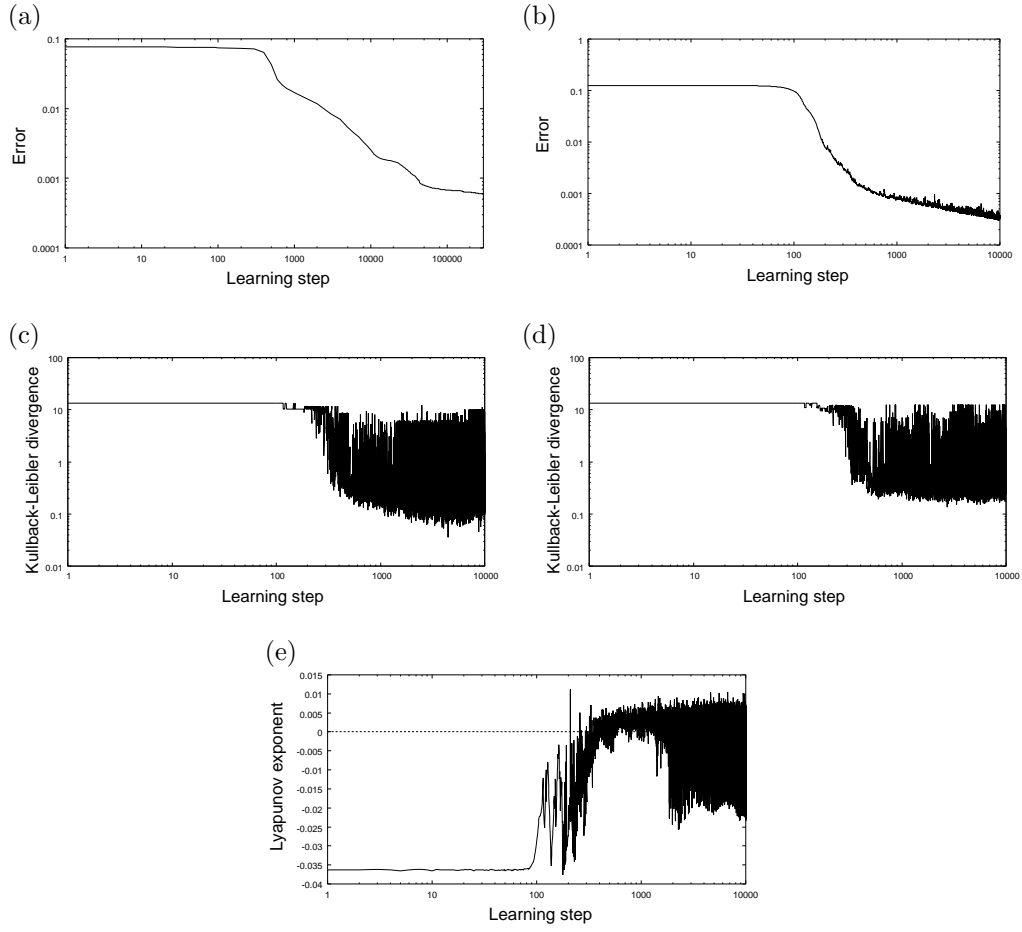
Figure 11: (a) The error of the experts E. (b) The error of the gating network E^g. (c) Kullback-Leibler divergence $D_{\mathrm{KL}}^{(5)}((\hat{s}_n)_{n=1}^T || (s_n)_{n=1}^T)$ for the training data. (d) Kullback-Leibler divergence $D_{\mathrm{KL}}^{(5)}((\hat{s}_n)_{n=1}^T || (s_n)_{n=1}^T)$ for the test data. (e) Maximum Lyapunov exponent.
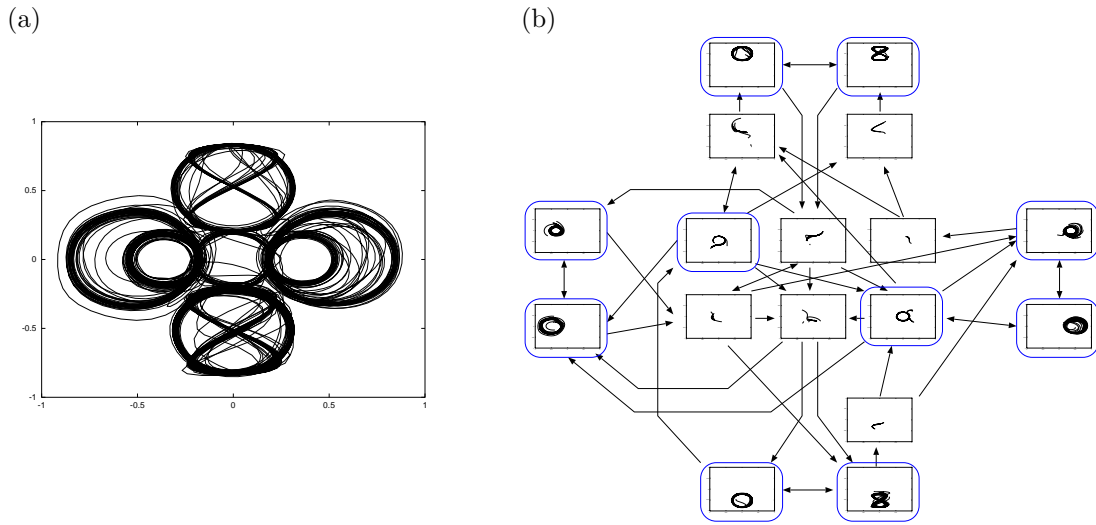
(a) (b)

Figure 12: (a) A trajectory generated by the trained network in the closed-loop dynamics (plotted over 10,000 time steps after convergence of the transients). (b) Output of experts and transition paths. The output of an expert i is plotted if $s_n = i$, namely, if gate i opens at time n. If gate i is never opened, then no output of the expert i is plotted. The graphs enclosed by blue boxes represent Lissajous curves as primitives, and other graphs represent paths connecting curves.
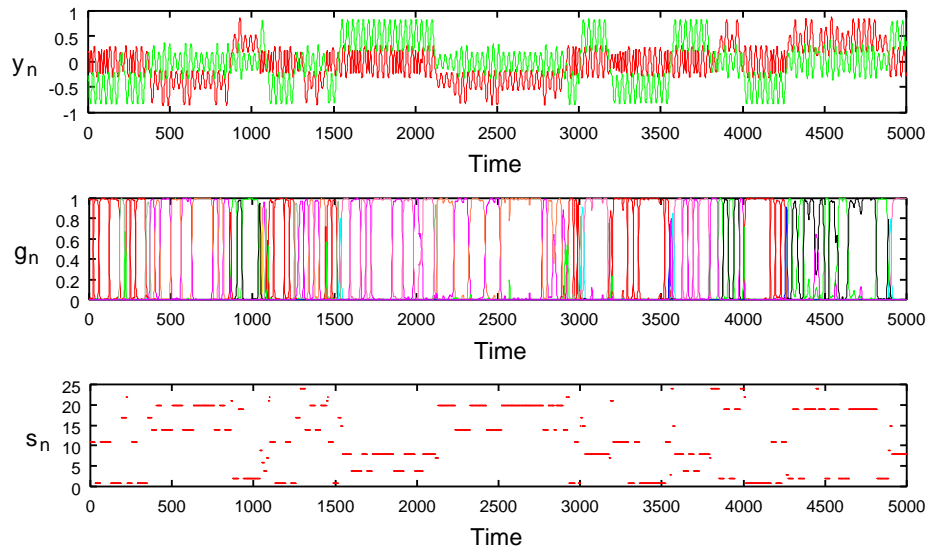

Figure 13: A snapshot of output $\boldsymbol{y}_n$, gate opening vector $\boldsymbol{g}_n$ and index of a current opening gate s_n of the trained network.

RNNs for several categories of target training data sequences.

Firstly, we consider the case in which training data are given by deterministic dynamical systems including chaos, in contrast with the case of nondeterministic time series. If the training data contain all degrees of freedom, then further states are uniquely determined by a current state. Thus, the task of generating the time series is equivalent to the approximation of an input-output function, and it is not necessary for successful learning that the model has internal states with recurrent connections. On the other hand, if the training data contain only a subset of the degrees of freedom, then further states cannot be determined uniquely by a current state. However, by the Takens' embedding theorem, the training data can often be embedded in another Euclidean space, in which time evolution is deterministic, by using the method of delays. Accordingly, even if training data realize only a subset of the degrees of freedom, we can still reduce the task to the approximation of an input-output function by reconstructing the state space. From this fact, the training data can be learned by a model which has a contraction mapping with respect to internal states (e.g., an echo state network), because the contraction mapping plays the role of embedding past time series in the internal states. The successful learning of Mackey-Glass chaotic time series by an echo state network is a remarkable example [11].

By way of contrast, nondeterministic sequences generated by a stochastic process cannot be transformed to deterministic time series by reconstructing the state space with the method of delays. Hence, the training data of nondeterministic sequences cannot be learned in the same way as deterministic sequences. One of the ways to output nondeterministic sequences using a dynamical system model is for the model to exhibit chaotic dynamics with respect to internal states, and for the output of the model to consist of a readout of the internal states. If encoded information of nondeterministic sequences can be embedded in the initial states with enough precision to predict future values, nondeterminacy of the sequences can be imitated by "sensitive dependence on initial conditions", which was discovered by Edward Lorenz in the early 1960s. Since a dynamical system having sensitive dependence on initial conditions can generate various sequences (including training data and test data) by changing initial states, the system can mimic nondeterministic process. For instance, RNNs can learn nondeterministic sequences by utilizing sensitive dependence on initial conditions (e.g., an experimental study in [25] or our simulation in section 3). Learning for such models is, however, often unstable, for the following reason. In order to train such models, a set of adjustable parameters has to include initial states, because information about sequences is embedded in the initial states. If a trained model is governed by a chaotic dynamics, almost any minute change of an initial state brings about a drastic change in the output sequence, due to the initial sensitivity of the chaotic dynamics. Hence this initial sensitivity introduces instability into learning processes.

4.2   Memory capacity of initial states for embedding sequence information

As described in section 4.1, in order for a dynamical systems model utilizing sensitive dependence on initial conditions to successfully learn nondeterministic sequences, it has to be possible to embed information about sequences in the initial states with enough precision to predict future values. However, initial states containing enough information to enable prediction is not a sufficient condition for successful learning, because there is the possibility for trained models to memorize exactly all the training data but not to acquire any chaotic dynamics. We intuitively expect that a sufficient amount of training data will be necessary to force a more general model; i.e., one that has acquired an adequate chaotic dynamics. We will discuss conditions for obtaining a chaotic dynamics taking into account both the amount of training data and the memory capacity of internal states.

To examine conditions for obtaining chaotic dynamics by learning for a mixture of RNN experts, we present additional simulation results. In the following, we refer to the number of training sequences, the step length of each training sequence and the number of context neurons in a gating network as NT, LT, and NC, respectively. Here, we repeat the learning simulation of the task in section 3.2 with the transition probability P set as 0.5, with varying NT, LT and NC. In addition, in order to measure the memory capacity of initial states, we define an error E^{closed} for a closed-loop dynamics

such that

$$E^{\text{closed}} = \frac{1}{2Td} \sum_{n=1}^{T} ||\boldsymbol{y}_n^{\text{teach}} - \boldsymbol{y}_n^{\text{closed}}||^2, \qquad (31)$$

where $\boldsymbol{y}_n^{\text{teach}}$ denotes the training data, and $\boldsymbol{y}_n^{\text{closed}}$ denotes output of the model computed for a closed-loop dynamics with a trained initial state. If E^{closed} becomes sufficiently small, the mixture of RNN experts can output sequences almost the same as the training data, namely, the training data are embedded in initial states. Figure 14 shows the results of training for each parameter setting such as the number of sequences NT, their step length LT and the number of context neurons NC. In this figure, we display the average of the error E^{closed} and the ratio of networks whose maximum Lyapunov exponent is positive, for 10 samples for each parameter setting. The results in Figure 14 (a) clearly indicate that the error E^{closed} depends on both NT and LT, and that E^{closed} decreases with increasing NC. In particular, E^{closed} increases if NT and LT are increased in the case of $NC = 10$. This result is due to the fact that error margins cannot be corrected because of less memory capacity in initial states. The results in Figure 14 (b) indicate that chaos tends to appear if both NT and LT are increased. In addition, by increasing NC, generating chaos tends to require a greater amount of training data. It can be inferred that overabundant memory capacity enables the networks to memorize exactly all training data without extracting an adequate chaotic dynamics from the data. Thus, if networks can exhibit sufficient memory capacity to embed training sequences in initial states, a small number of context neurons tends to bring an adequate chaotic dynamics from the training data. On the other hand, in the case of 10 context neurons, the trained networks did not acquire a chaotic dynamics even if subjected to a great deal of training data. This result implies that acquiring chaos tends to fail if the memory capacity of a network is insufficient to embed training sequences. If a task demands a large amount of training data, a sufficient number of context neurons is necessary to successfully learn the task. Therefore, these results imply that it is necessary to balance a sufficient amount of training data with the memory capacity, in order to embed the target stochastic sequences into chaos.

Finally, we explain the reason that long sequences can be encoded in initial states in our model. It is known that the BPTT method cannot propagate effectively error signals over long-time windows, because information in the error signals decreases exponentially through the sigmoid function [2]. For instance, in studies [19, 25] all training data learned by RNNs were of length less than 100. Our model, however, successfully learns training data whose length is over $10,000$. The difference between the situation in the previous studies and our situation is brought about by the time constant ϵ^g in equation (5) for updating context neural states. Since the internal potential of a context neuron decays slowly by the effect of ϵ^g, our model can keep values of context neurons for long time. The internal potentials of context neurons with time constant ϵ^g play a role similar to that of linear integrators within special units of LSTM [7, 23]. While units of LSTM can retain values without any damping, context neurons with ϵ^g lose information through updating states. Conversely, since the internal potentials of context neurons with ϵ^g are bounded by damping, the potential values do not diverge to infinity, and so the dynamics of our model is more stable in many situations. As potential nonlinearity of a chaotic dynamics often forces divergence on a learning model, our model is desirable when a chaotic dynamics is learned.

4.3   Structural stability of learning models

In the above discussion, we demonstrated that the proposed model can have enough memory capacity to embed long sequences in initial states. However, as we saw in the results of experiments, chaos does not always appear even if both a sufficient amount of training data and memory capacity are given and the error is made sufficiently small. When learning models have to obtain chaotic dynamic, the problem of structural stability of the learning models appears. The notion of structural stability was formulated by Andronov and Pontryagin [1] on the basis of the belief that a model for a natural phenomenon must preserve at least its topological character under perturbation. Mathematically, a dynamical system is
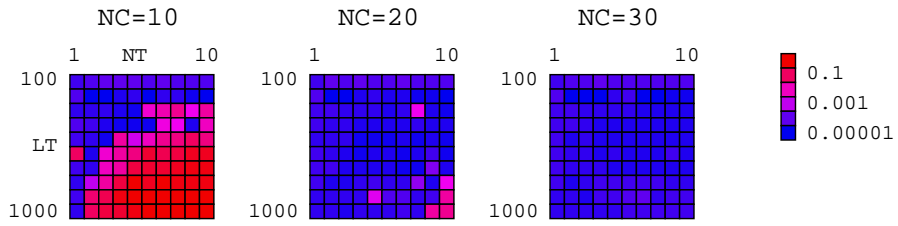
(a) Error E^{closed}
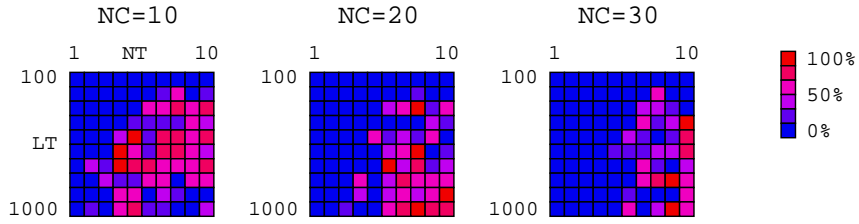
(b) Ratio of positive Lyapunov exponent

Figure 14: Phase plots of the error E^{closed} and the ratio of networks whose maximum Lyapunov exponent is positive. The upper figures display the error E^{closed}, and the lower figures display the ratio of positive Lyapunov exponent. In the figures, NC denotes the number of context neurons in a gating network, along the vertical axis is the length of each training sequence LT, and along the horizontal axis is the number of training sequences NT.

called structurally stable in the C^1 sense if any other dynamical system in the C^1 neighborhood of such a dynamical system is topologically conjugate with the dynamical system concerned. In terms of a learning procedure using the gradient descent method, if a model is structurally stable and the learning rate is sufficiently small, the model preserves its topological character when parameters are changed in the neighborhood of a local minimum. By contrast, in the case when a model is not structurally stable, convergence of a likelihood (or an error) does not correspond to convergence of the topological character of the model, and success or failure of learning cannot be identified until actually computing the time evolution. Therefore, structural stability of the model is preferable from the viewpoint of learning. However, taking into account the whole space of dynamical systems, structurally stable dynamical systems are not dense in this space. Hence a neural network, which is a kind of universal function approximator, is not structurally stable in general. For example, in the simulation described in Figure 11 the network seems not to be structurally stable, while in Figure 3 and Figure 6 the networks might be structurally stable at the end of training. If we use a universal approximator as a learning model, it is difficult to avoid occurrence of a model that is not structurally stable.

Let us consider how to take steps to cope with the appearance of a model that is not structurally stable. One possible way is to add a negligible amount of noise to the dynamics of a trained model. For instance, even if the dynamics of a trained model is not exactly chaotic because it is not structurally stable, the model could acquire a dynamical structure whose neighborhood contained the chaotic dynamics. In this case, by adding low noise, the model can recure chaotic movements by making transverse intersections between stable and unstable manifolds. In addition, if the model is structurally stable, the topological character does not change for low noise by satisfying the shadowing property [21]. As an example, Figure 15 displays trajectories of a network, where the network is trained in section 4.2 and it seems not to be structurally stable. Figure 15 (a) shows a trajectory of the network after $10,000$ learning steps. In this case, the dynamics of the network is a limit cycle and the maximum Lyapunov exponent is negative. In Figure 15 (b), however, the dynamics of the network changes to chaos from a limit cycle after only 10 learning steps. Next, we added Gaussian white noise with variance $\rho^2 = 0.0001$ to the dynamics of the network having a limit cycle, and plotted a trajectory in Figure 15 (c). Consequently, a dynamics similar to that of case (b) can be reconstructed by adding the noise. It is shown that a neighborhood of the network contains a chaotic system, and adding noise restores the network dynamics to the chaotic system while respecting the original topology.

In Figure 16, to evaluate the effect of low noise we display ratios of networks generating nonperiodic output sequences, where the networks were trained in section 4.2, and Gaussian white noise with variance ρ^2 was added to the input x_n for each time step n. Figures 16 (a) and (b) display the results for variances $\rho^2 = 0$ and $\rho^2 = 0.001$, respectively. Nonperiodicity of output sequences implies that the dynamics is chaotic or quasi-periodic. To evaluate periodicity of output sequences, we labeled a left and right circle in output trajectories (see Figure 7) as a symbol L and R, respectively. Forming symbolic sequences like LRLLRRLR $\cdots$ from output trajectories, we computed to determine periodicity of the symbolic sequences over a period of 100. Comparing Figure 7 (a) with (b), it can be seen that the proportion of networks generating nonperiodic sequences is increased by adding the noise. On the other hand, in the region corresponding to a small amount of training data, output is periodic even if noise is added. It turned out that by adding noise, output of a network becomes nonperiodic if the network is trained by using enough training data, whereas output is periodic if we use less training data. In addition, adding noise does not result only in nonperiodic sequences being generated, because there is a region in which output is periodic even if noise is added. Thus, this result implies that in the neighborhood of a trained network subjected to a sufficient amount of data, there exists an appropriate chaotic dynamics. Hence, we can classify trained networks into three classes from the results about their dynamics: (1) those that have a structurally stable and periodic dynamics, (2) those that have a structurally stable and chaotic dynamics and (3) those that are structurally unstable but with a nearby system having a chaotic dynamics. Models with low noise can generate chaotic time series and imitate nondeterministic characteristics underlying the training data not only in case (2) but also in case (3).
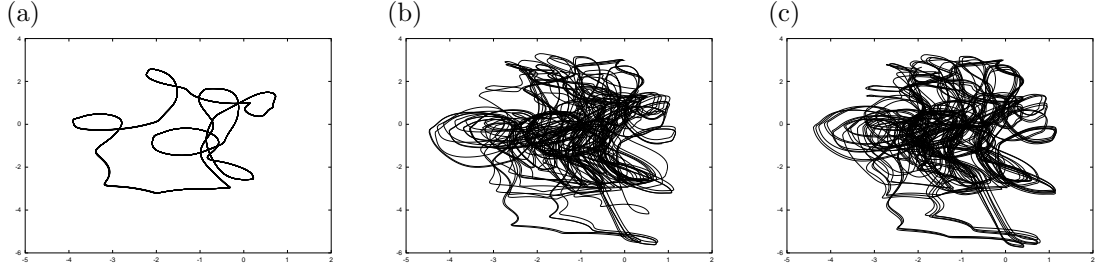
Figure 15: Trajectories of two values of $\boldsymbol{u}_n^g$ of a network with closed-loop dynamics, where the network is trained as in section 4.2 with 10 training sequences, the step length of each sequence being 1000 and 30 context neurons. For each figure, we plotted over $10{,}000$ time steps after convergence of the transients. (a) A trajectory of the network after $10{,}000$ learning steps (the maximum Lyapunov exponent is -0.000054). (b) A trajectory of the network after $10{,}010$ learning steps (the maximum Lyapunov exponent is 0.000995). Dynamics of the network changed to chaos from a limit cycle after only 10 learning steps. (c) A trajectory of the network the same as in case (a), but we added Gaussian white noise with variance $\rho^2 = 0.0001$ to the input x_n for each time step n. Chaotic dynamics appears when the noise is added.
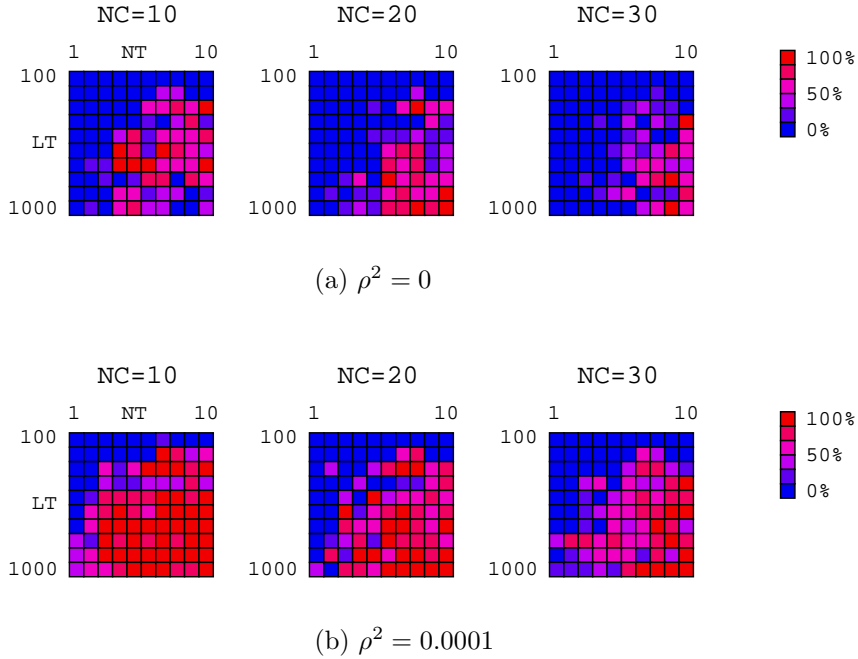

(a) $\rho^2 = 0$

(b) $\rho^2 = 0.0001$

Figure 16: The ratio of networks generating nonperiodic output sequences. Here, Gaussian white noise with variance ρ^2 is added to the input x_n for each time step. (a) Phase plots corresponding to the case of $\rho^2 = 0$, namely, without noise. (b) Phase plots corresponding to the case of $\rho^2 = 0.001$. In these figures, NC denotes the number of context neurons in the gating network, the vertical axis is the length of each training sequence LT, and the horizontal axis is the number of training sequences NT.

22

4.4 Characteristics as high-dimensional chaotic systems

As mentioned in the previous section, the mixture of RNN experts model can estimate a rule to generate desired stochastic sequences as a deterministic chaos instead of memorizing exactly the all training sequences. This fact raises the question of what chaotic dynamical rule is acquired by the model at the end of successful learning. In the following we discuss characteristics of trained models by focusing on chaotic dynamics.

We first explain that the mixture of RNN experts model can acquire a high-dimensional chaotic system through learning. Here, dynamical systems are termed as low-dimensional chaos if the systems have attractors with one positive Lyapunov exponent, because dynamics of them can be described by a limited number of degrees of freedom. By contrast, high-dimensional chaos means that there exist multiple positive Lyapunov exponents (such a situation also called "hyperchaos"). Notice that the definition of mixture of RNN experts being a high-dimensional nonlinear system doesn't infer that the model behaves as high-dimensional chaos, because the system possibly acquires low-dimensional chaos by training. In Figure 3 (d), two or more Lyapunov exponents of the model becomes positive in training. Thus, the model can become a high-dimensional chaotic system when stochastic sequences are trained. On the other hand, the network learnt switching multiple Lissajous curves exhibits intermittent dynamics like chaotic itinerancy [8, 15] that are chaotic itinerant motions among varieties of ordered states (see Figure 13). Moreover, in the simulation for Figure 14 we encountered the networks with long-transient behaviors which are often more than 10 million transient steps even if the networks only have periodic attractors. Since such "super transient" phenomena and chaotic itinerancy are known as typical characteristics of high-dimensional chaos, the networks seem high-dimensional chaotic systems.

The fact that trained networks exhibit chaotic itinerancy is important as a model of brains. The existence of trajectory dynamics such as chaotic itinerancy in animal brains have been reported with the relation to the information processing in the brains [24, 27, 28]. However, in such the argument, there exists a potential problem in terms of dynamical systems theory that chaotic itinerancy might be only observed in critical and structurally stable situations in high-dimensional dynamical systems, and RNNs with asymmetric connections usually allow structurally unstable. Therefore, it has been considered that getting RNNs with chaotic itinerant phenomena is difficult through learning. We, however, have shown that the model could obtain chaotic itinerant phenomena, even if the model could become structurally unstable. The results of our model support a possibility of chaotic itinerant phenomena existing in the brain.

5 Conclusion

This paper has shown that a mixture of RNN experts model can acquire the ability to generate sequences combining multiple primitives. By virtue of training of the model, each expert learns a primitive sequence pattern, and a gating network learns a switching rule between winners to make decisions about which expert to use as a winner on each occasion. Our simulation experiments showed that the model imitates stochastic switching between the multiple primitives in terms of a chaotic dynamics utilizing sensitive dependence on initial conditions. It was also demonstrated that a self-organized chaotic system can reconstruct the probability of primitive switching as observed in the training data. We have also considered two essential problems concerning memory capacity and structural stability of the model. Through analysis of the network dynamics, we have inferred that a sufficient amount of training data, balanced with the memory capacity of the network is a necessary condition for embedding stochastic time series into a chaotic dynamical system. We have also inferred that even if the dynamics of the model is not exactly chaotic because of structural instability, adding a negligible amount of noise into the model can enable stably reconstruction of stochastic time series. The model can be applied to prediction and generation of time-series, especially when the time series to be learned are potentially able to be represented as a composition of several primitives, and there is no required pre-knowledge of the primitives or a composition rule for them.

References

[1] Andronov, A.A., & Pontryagin, L.S. (1937). Systéms grossiers. *Dokl. Acad. Nauk SSSR*, 14, 247–251.

[2] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, 5(2), 157–166.

[3] Elman, J.L. (1990). Finding structure in time. *Cognitive Science*, 14, 179–211.

[4] Elman, J.L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7, 195–224.

[5] Evans, G. (1981). Semantic Theory and Tacit Knowledge. In Holzman, S., & Leich, C., editors, *Wittgenstein: To Follow a Rule*, (pp. 118–137). Routledge and Kegan Paul, London.

[6] Hao, B.L. (1989). *Elementary Symbolic Dynamic and Chaos in Dissipative System*. Singapore: World Scientific.

[7] Hichreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.

[8] Ikeda, K., Otsuka, K., & Matsumoto, K. (1989). Maxwell Bloch turbulence. *Progress of Theoretical Physics (Supplement)*, 99, 295–324.

[9] Jacobs, R.A., Jordan, M.I., Nowlan, S.J., & Hinton, G.E. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3(1), 79–87.

[10] Jaeger, H. (2001). Short term memory in echo state networks. *GMD Report*, 152, 1–60.

[11] Jaeger, H., & Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, (pp. 78–80).

[12] Jordan, M.I. (1986). Attractor Dynamics and Parallelism in a Connectionist Sequential Machine. In *Proc. of Eighth Annual Conference of Cognitive Science Society*, (pp. 531–546). Hillsdale, NJ: Erlbaum.

[13] Jordan, M.I. (1988). Indeterminate motor skill learning problems. In Jeannerod, M., editor, *Attention and Performances, XIII*. Cambridge, MA: MIT Press.

[14] Jordan, M.I., & Jacobs, R.A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2), 181–214.

[15] Kaneko, K., & Tsuda, I. (2003). Chaotic itinerancy. *Chaos: Focus Issue on Chaotic Itinerancy*, 13, 926–936.

[16] Lind, D., & Marcus, B. (1995). *An introduction to symbolic dynamics and coding*. Cambridge University Press, Cambridge.

[17] Maass, W., Natschläger, T., & Markram, H. (2002). A fresh look at real-time computation in generic recurrent neural circuits. Technical report, Technical report, Institute for Theoretical Computer Science, TU Graz.

[18] Namikawa, J., & Tani, J. (2008). A model for learning to segment temporal sequences, utilizing a mixture of RNN experts together with adaptive variance. *Neural Networks*, 21, 1466–1475.

[19] Nishimoto, R., & Tani, J. (2004). Learning to Generate Combinatorial Action Sequences Utilizing the Initial Sensitivity of Deterministic Dynamical Systems. *Neural Networks*, 17, 925–933.

[20] Pollack, J. (1991). The induction of dynamical recognizers. *Machine Learning*, 7, 227–252.

[21] Robinson, C. (1999). *Dynamical Systems, Stability, Symbolic Dynamics, and Chaos, 2nd edition (Studies in Advanced Mathematics)*. CRC Press.

[22] Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by error propagation. In Rumelhart, D.E., & McClelland, J.L., editors, *Parallel Distributed Processing*, (pp. 318–362). Cambridge, MA: MIT Press.

[23] Schmidhuber, J., Gers, F., & Eck, D. (2002). Learning nonregular languages: a comparison of simple recurrent networks and LSTM. *Neural Computation*, 14, 2039–2041.

[24] Skarada, C.A., & Freeman, W.J. (1987). Does the brain make chaos in order to make sense of the world? *Behavioral and Brain Sciences*, 10, 161–165.

[25] Tani, J., & Fukumura, N. (1995). Embedding a Grammatical Description in Deterministic Chaos: an Experiment in Recurrent Neural Learning. *Biological Cybernetics*, 72, 365–370.

[26] Tani, J., & Nolfi, S. (1999). Learning to perceive the world as articulated: an approach for hierarchical learning in sensory-motor systems. *Neural Networks*, 12, 1131–1141.

[27] Tsuda, I. (1991). Chaotic itinerancy as a dynamical basis of hermeneutics in brain and mind. *World Futures: The Journal of General Evolution*, 32, 167–184.

[28] Tsuda, I., & Fujii, H. (2007). Chaos reality in the brain. *Journal of Integrative Neuroscience*, 6(2), 309–326.

[29] Williams, R.J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1, 270–280.

[30] Wolpert, D., & Kawato, M. (1998). Multiple paired forward and inverse models for motor control. *Neural Networks*, 11, 1317–1329.