ARTICLE Communicated by Christopher Buckley

A Novel Predictive-Coding-Inspired Variational RNN Model for Online Prediction and Recognition

Ahmadreza Ahmadi
ar.ahmadi62@gmail.com
Okinawa Institute of Science and Technology, Okinawa, Japan 904-0495, and School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, 305-701, Republic of Korea

Jun Tani*
tani1216jp@gmail.com
Okinawa Institute of Science and Technology, Okinawa, Japan 904-0495

This study introduces PV-RNN, a novel variational RNN inspired by predictive-coding ideas. The model learns to extract the probabilistic structures hidden in fluctuating temporal patterns by dynamically changing the stochasticity of its latent states. Its architecture attempts to address two major concerns of variational Bayes RNNs: how latent variables can learn meaningful representations and how the inference model can transfer future observations to the latent variables. PV-RNN does both by introducing adaptive vectors mirroring the training data, whose values can then be adapted differently during evaluation. Moreover, prediction errors during backpropagation—rather than external inputs during the forward computation—are used to convey information to the network about the external data. For testing, we introduce error regression for predicting unseen sequences as inspired by predictive coding that leverages those mechanisms. As in other variational Bayes RNNs, our model learns by maximizing a lower bound on the marginal likelihood of the sequential data, which is composed of two terms: the negative of the expectation of prediction errors and the negative of the Kullback-Leibler divergence between the prior and the approximate posterior distributions. The model introduces a weighting parameter, the meta-prior, to balance the optimization pressure placed on those two terms. We test the model on two data sets with probabilistic structures and show that with high values of the meta-prior, the network develops deterministic chaos through which the randomness of the data is imitated. For low values, the model behaves as a random process. The network performs best on intermediate values and is able to capture the latent probabilistic structure with good generalization. Analyzing the meta-prior's impact on the

*Corresponding author.

Neural Computation **31, 2025–2074** (2019)
https://doi.org/10.1162/neco_a_01228

network allows us to precisely study the theoretical value and practical benefits of incorporating stochastic dynamics in our model. We demonstrate better prediction performance on a robot imitation task with our model using error regression compared to a standard variational Bayes model lacking such a procedure.

1 Introduction

Predictive coding has attracted considerable attention in cognitive neuroscience as a neuroscientific model unifying possible neuronal mechanisms of prediction, recognition, and learning (Rao & Ballard, 1999; Lee & Mumford, 2003; Clark, 2015; Friston, 2018). Predictive coding suggests that, first agents predict future perception through a top-down internal process. Then, prediction errors are generated by comparing the actual perception and the predicted ones. These errors are propagated through a bottom-up process to update agents' internal states such that the error is minimized and the actual perceptual inputs are recognized. Learning may then be achieved by optimizing the internal model.

Tani and colleagues (Tani & Nolfi, 1999; Tani & Ito, 2003; Tani, Ito, & Sugita, 2004) have investigated neural networks, which may be considered analogous to the predictive-coding framework, especially for learning temporal patterns in robotic experiments. They used recurrent neural networks (RNNs) (Elman, 1990; Jordan, 1997; Hochreiter & Schmidhuber, 1997) since RNNs are capable of learning long-term dependencies in temporal patterns. However, their predictive ability is limited in real-world applications where high uncertainty is involved. This limitation is mainly because conventional RNNs are able to predict only perceptual inputs deterministically.

To help solve this, Murata and colleagues (Murata, Namikawa, Arie, Sugano, & Tani, 2013; Murata et al., 2017) proposed a stochastic RNN. In this RNN, the uncertainty in data is estimated by the mean and variance of a gaussian distribution in the output layer via learning. The hidden layers remained deterministic, however, because there was no known way to do backpropagation through random variables. This therefore limited the network from fully extracting the probabilistic structures of the target data during learning.

To work around this limitation, Kingma and Welling (2013), in their work on variational Bayes autoencoders (VAEs), developed a technique called the *reparameterization trick*, which allows backpropagating errors through hidden layers with random variables, thus allowing for internal stochasticity in neural networks.

Kingma and Welling (2013) used this method in an autoencoder in order to approximate a posterior distribution of latent variables. The variational Bayes (VB) approach optimizes the network by maximizing a variational lower bound on the marginal likelihood of the data, and the prior distribution is sampled from a standard normal gaussian. This lower bound is

composed of two terms: the negative of the prediction error and the negative of the Kullback-Leibler (KL) divergence between the approximate posterior and prior distributions.

Various RNNs have been proposed based on the VAE. The first variational Bayes RNNs proposed sampling the prior distribution from a standard normal gaussian at each time step (Fabius & van Amersfoort, 2014; Bayer & Osendorfer, 2014). Later, Chung et al. (2015) proposed a VAE RNN, the variational RNN (VRNN), which used a conditional prior distribution derived from the state variables of an RNN to account for temporal dependencies within the data. Since then, various attempts have been made to the approximate posterior of the VRNN. Some recent studies proposed approximate posteriors that had more similar structures to the true posterior by considering future dependencies on sequential data by using two RNNs—one forward and one backward (Fraccaro, Sønderby, Paquet, & Winther, 2016; Goyal, Sordoni, Côté, Ke, & Bengio, 2017; Shabanian, Arpit, Trischler, & Bengio, 2017). Another issue targets VRNN-based models: they have a tendency to ignore the stochasticity introduced by their random variables and to rely only on deterministic states. To remedy this, there have been several attempts to force the latent variables to learn meaningful information in the approximate posteriors (Bowman et al., 2015; Karl, Soelch, Bayer, & van der Smagt, 2016; Goyal et al., 2017).

This article proposes a novel network model, referred to as the predictive-coding-inspired variational RNN (PV-RNN), that integrates ideas from recent variational RNNs and predictive coding. In this model, the prior distribution is computed using conditional parameterization similar to Chung et al. (2015), whereas the posterior is approximated using a new adaptive vector $\boldsymbol{A}$, which forces the latent variables to represent meaningful information. This new vector also provides the approximate posterior with the future dependency information via backpropagation through time (BPTT; Werbos, 1974; Rumelhart, Hinton, & Williams, 1985) without a backward RNN. All model variables and $\boldsymbol{A}$ are optimized by maximizing a variational lower bound on the marginal likelihood of the data.

Our model also incorporates a process inspired by the predictive coding framework, error regression, which is used online during testing in our experiments after learning is finished. During error regression, the model constantly makes predictions, and the resulting prediction errors are backpropagated up the network hierarchy to update the internal states $\boldsymbol{A}$ of the model in order to maximize both negative terms of the lower bound.

Many studies have assumed that the brain may use predictive coding to minimize a free energy or maximize a lower bound on surprise (Friston, 2005, 2010; Hohwy, 2013; Clark, 2015). By incorporating features inspired by predictive coding principles, our model may be considered to be more consistent with the ideas of computational neuroscience than other VAE-based models. While most models propagate inputs through the network during the forward computation, our model only propagates prediction errors through backpropagation through time (BPTT).

One important motivation in this study is to clarify how uncertainty or probabilistic structure hidden in fluctuating temporal patterns can be learned and then internally represented in the latent variables of an RNN. Importantly, randomness hidden in sequences can be accounted for by either deterministic chaos or a stochastic process. Therefore, if we consider that we may observe sensory data with only finite resolution (Crutchfield, 1992), then if the original dynamics are chaotic, the symbolic dynamics observed through Markov partitioning involved with the coarse-graining may be ergodic, generating stochasticity (Sinai, 1972). Inversely, if a deterministic RNN acts as a generative model to reconstruct such stochastic sequences through learning, the RNN may do so by embedding the sequences into internal, deterministic chaos by leveraging initial sensitivity (Tani & Fukumura, 1995). An interesting question, however, is, If a generative model contains an adaptive mechanism to estimate first-order statistics—as in our proposed model and other variational Bayes RNNs—how may the components of deterministic and stochastic dynamics be used to account for observed stochasticity in the model's output?

To examine this question, we introduce a variable, the meta-prior, that weights the minimization of the divergence between the posterior and the prior against that of the prediction error in the computation of the variational lower bound. We investigate how the meta-prior influences development of different types of information processing in the model. First, we conduct a simulation experiment using a simple probabilistic finite state machine (PFSM) and observe how different settings of the meta-prior affect representation of uncertainty in the latent state of the model. Next, we examine how different representations of latent states in the model can lead to the development of purely deterministic dynamics, random processes, or something in between these two extremes. In particular, we examine how generalization capabilities correlate with such differences.

Next, we consider a more complex setup, where the data embed multitimescale information and the network features multiple layers, each with its own time constant. This allows the model to deal with fluctuating temporal patterns that consist of sequences of hand-drawn primitives with probabilistic transitions among them. We conduct simulation experiments to examine if the multiple-layer model exhibits qualitatively the same ability as the one of the previous experiment to extract the latent probabilistic structures of such compositionally organized sequence data.

Finally, we evaluate the performance of the proposed model in a real-world setting by conducting a robotic experiment. On a task where a robot learns to imitate another, imitation performance is compared between PV-RNN with error regression for posterior inference, and VRNN, which uses a variational autoencoder. This experiment aims to evaluate our hypothesis that the posterior inference calculated through error regression provides better estimates than an autoencoder-based model.

2 Model

We now describe in detail the generative and inference models, as well as the learning procedure. The generative model produces predictions based on the latent state of the network. Conversely, the inference model, given an observation, estimates what should be the latent state in order to produce the observation. The learning process concerns itself with discovering good values for the learnable variables of both the generative and inference models.

2.1 Generative Model. As with many published variational Bayes models, the generative model P_θ of PV-RNN is an RNN with stochastic latent variables. Here, θ denotes the learnable variables of the generative model, which is illustrated in Figure 1A by black lines. The variables θ are distributed among the components $\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{d}$ of the generative model, as $\theta_X, \theta_Z, \theta_d$. $\boldsymbol{Z}$ and $\boldsymbol{d}$ are the stochastic and deterministic latent states, respectively, and $\boldsymbol{X}$ is the generated prediction. For a prediction $\boldsymbol{X}_{1:T} = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_T)$, the generative model factorizes as

$$\begin{aligned} & P_\theta(\boldsymbol{X}_{1:T}, \boldsymbol{Z}_{1:T}, \boldsymbol{d}_{1:T} \mid \boldsymbol{Z}_0, \boldsymbol{d}_0) \\ &= P_{\theta_X}(\boldsymbol{X}_{1:T} \mid \boldsymbol{d}_{1:T}, \boldsymbol{Z}_{1:T}) P_{\theta_Z}(\boldsymbol{Z}_{1:T} \mid \boldsymbol{d}_{1:T}, \boldsymbol{Z}_0) P_{\theta_d}(\boldsymbol{d}_{1:T} \mid \boldsymbol{Z}_{1:T}, \boldsymbol{d}_0) \\ &= \prod_{t=1}^{T} P_{\theta_X}(\boldsymbol{X}_t \mid \boldsymbol{d}_t, \boldsymbol{Z}_t) P_{\theta_Z}(\boldsymbol{Z}_t \mid \boldsymbol{d}_{t-1}) P_{\theta_d}(\boldsymbol{d}_t \mid \boldsymbol{d}_{t-1}, \boldsymbol{Z}_t) \end{aligned} \tag{2.1}$$

The initial values of $\boldsymbol{Z}$ and $\boldsymbol{d}$ at time step zero, $\boldsymbol{Z}_0$ and $\boldsymbol{d}_0$, are set to zero in our experiments. The latent state $\boldsymbol{d}_t$ is recursively computed using an RNN model:

$$\boldsymbol{d}_t = f_{\theta_d}(\boldsymbol{d}_{t-1}, \boldsymbol{Z}_t). \tag{2.2}$$

In this article, we use a multiple timescale recurrent neural network (MTRNN) (Yamashita & Tani, 2008) as f_{θ_d}, but any type of RNN, such as long short term memory (LSTM) or gated recurrent units (GRUs) could be used instead. MTRNNs are a type of RNNs composed of several hierarchical layers, with each layer using a different time constant. The internal dynamic of an MTRNN model is computed as

$$\begin{aligned} \boldsymbol{h}_t^k &= (1 - \frac{1}{\tau_k})\boldsymbol{h}_{t-1}^k + \frac{1}{\tau_k}(\boldsymbol{W}_{dd}^{kk}\boldsymbol{d}_{t-1}^k + \boldsymbol{W}_{dz}^{kk}\boldsymbol{Z}_t^k + \boldsymbol{W}_{dd}^{kk+1}\boldsymbol{d}_{t-1}^{k+1} + \boldsymbol{W}_{dd}^{kk-1}\boldsymbol{d}_{t-1}^{k-1}) \\ \boldsymbol{d}_t^k &= \tanh(\boldsymbol{h}_t^k), \end{aligned} \tag{2.3}$$

where $\boldsymbol{h}_t^k$ is the vector of the internal state values of the kth context layer at time t, $\boldsymbol{W}_{dd}^{kk}$ is the matrix of the connectivity weights from the $\boldsymbol{d}$ units in

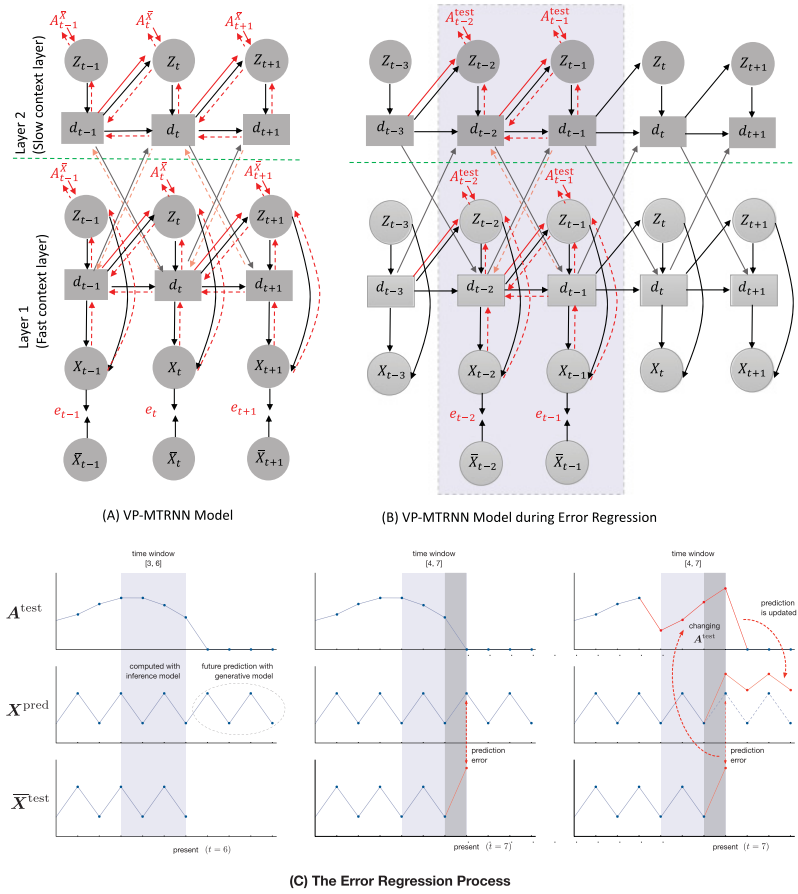

Figure 1: (A) The generative and inference models of PV-RNN in an MTRNN setting, (B) the error regression graph during tests, and (C) the error regression process. In panels A and B, black lines represent the generative model and red lines show the inference model, with solid red lines showing the feedforward computations of the inference model and dashed red lines showing the BPTT that is used to update $A^{\overline{X}}$ in panel A and A^{test} in panel B. The gray area in panel B represents a two-step temporal window of the immediate past in which $A^{\text{test}}_{t-2:t-1}$ is modified to maximize the lower bound. Panel C illustrates the error regression process. At $t = 6$, predictions are generated (left) after observing $\overline{X}^{\text{test}}_{1:6}$. The three-time step time window is slid by one time step to [4, 7] (middle; now, $t = 7$), and an error is observed between the prediction $X^{\text{pred}}_{4:7}$ and the target value $\overline{X}^{\text{test}}_{4:7}$. The lower bound is computed, and backpropagation is performed; $A^{\text{test}}_{4:7}$ is then optimized, and the prediction $X^{\text{pred}}_{4:7}$ is updated (right). This backpropagation/optimization/prediction cycle can be repeated multiple times before moving on to the [5, 8] time window.

the kth context layer to itself, $\boldsymbol{W}_{dz}^{kk}$ the connectivity weights from $\boldsymbol{Z}$ to $\boldsymbol{d}$ in layer k, $\boldsymbol{W}_{dd}^{kk+1}$ is the matrix of the connectivity weights from the $\boldsymbol{d}$ units of the $k+1$th context layer to the ones in the kth layer, and, similarly, $\boldsymbol{W}_{dd}^{kk-1}$ is the matrix for the one coming from layer $k-1$, and τ is the time constant. Bias terms are not shown in equation 2.3 for clarity. In this article, we consider networks with no more than three layers. Also, as is common with MTRNNs, the lower layer will have a faster time constant than the higher layer. In that context, we refer to the lowest layer, with the fastest time constant, as the fast layer, and symmetrically, to the highest layer, with the slowest time constant, as the slow layer. The slow (highest) layer does not have any layer above it, and so, obviously, in equation 2.3, the term $\boldsymbol{W}_{dd}^{kk+1}\boldsymbol{d}_{t-1}^{k+1}$ is removed. The same thing applies for the fast (lowest) layer and the term $\boldsymbol{W}_{dd}^{kk-1}\boldsymbol{d}_{t-1}^{k-1}$. Figure 1A shows the PV-RNN model implemented with a two-layer MTRNN. We extended the original MTRNN model (Yamashita & Tani, 2008) by adding stochastic units $\boldsymbol{Z}$ to each layer. Each layer communicates only with the layer above and the one below to create a hierarchical structure.

Finally, the prior distribution $P_{\theta_Z}(\boldsymbol{Z}_t \mid \boldsymbol{d}_{t-1})$ is a gaussian with a diagonal covariance matrix, which depends on $\boldsymbol{d}_{t-1}$. Priors depending on the previous state were used in Chung et al. (2015), and it outperformed the independent standard gaussian prior used in STORN (Bayer & Osendorfer, 2014),

$$P_{\theta_Z}(\boldsymbol{Z}_t \mid \boldsymbol{d}_{t-1}) = \mathcal{N}(\boldsymbol{Z}_t; \boldsymbol{\mu}_t^{(p)}, \boldsymbol{\sigma}_t^{(p)}) \quad \text{where} \quad [\boldsymbol{\mu}_t^{(p)}, \log \boldsymbol{\sigma}_t^{(p)}] = f_{\theta_Z}^{(p)}(\boldsymbol{d}_{t-1}), \tag{2.4}$$

where $f_{\theta_Z}^{(p)}$ denotes a one-layer feedforward neural network and $\boldsymbol{\mu}_t^{(p)}$ and $\boldsymbol{\sigma}_t^{(p)}$ are the mean and standard deviation of $\boldsymbol{Z}_t$. We use the reparameterization trick (Kingma & Welling, 2013) such that the latent value $\boldsymbol{Z}$ in both posterior and prior are reparameterized as $\boldsymbol{Z} = \boldsymbol{\mu} + \boldsymbol{\sigma} * \boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon}$ is sampled from $\mathcal{N}(0, I)$. In this study, $P_{\theta_X}(\boldsymbol{X}_t \mid \boldsymbol{d}_t^1, \boldsymbol{Z}_t^1)$ is obtained by a one-layer feedforward model $f_{\theta_X}^{(x)}$.

One peculiar detail about the generative model is that it does not accept any external inputs. Indeed, the generative model, unlike many other variational Bayes RNN models, generates sequences based on the latent state exclusively. Rather than using external inputs, the PV-RNN model propagates the errors between the predictions and the observations via backpropagation through time. To understand this clearly, we need to explain the inference model first.

2.2 Inference Model. Based on the generative model, the true posterior distribution of $\boldsymbol{Z}_t$ depends on $\boldsymbol{X}_{t:T}$, which can be verified using d-separation

(Geiger, Verma, & Pearl, 1990). Computing the true posterior is intractable, so an inference model is designed to compute an approximate posterior.

To compute $\boldsymbol{Z}_t$, the network considers the deterministic state of the network during the previous time step, $\boldsymbol{d}_{t-1}$. In all other variational Bayes RNNs, $\boldsymbol{d}$ units are fed training patterns directly, but in our case, we removed those inputs to force $\boldsymbol{d}$ not to ignore $\boldsymbol{Z}$. We need another method to feed the network with information specific to the current pattern. To that end, for a training sequence of T time steps $\overline{\boldsymbol{X}}_{1:T}$, we introduce the adaptive vectors $\boldsymbol{A}^{\overline{X}}_{1:T}$. For each time step $\overline{\boldsymbol{X}}_t$ of $\overline{\boldsymbol{X}}$, we have a corresponding vector $\boldsymbol{A}^{\overline{X}}_t$. This vector is specific to the sequence $\overline{\boldsymbol{X}}$. In other words, the model is going to have $T \times N_{\overline{X}}$ adaptive vectors like this, with $N_{\overline{X}}$ the number of training sequences.

Each $\boldsymbol{A}^{\overline{X}}_t$ is going to be adapted through BPTT, and the changes made through BPTT will depend on the prediction errors between $\boldsymbol{X}$ and $\overline{\boldsymbol{X}}$ from T to t, $\boldsymbol{e}_{t:T}$. Naturally, the other learning variables of the network $\theta_X, \theta_Z, \theta_d$, and ϕ (see equation 2.5 for ϕ) will also be affected during BPTT by the information contained in $\boldsymbol{e}_{1:T}$. But those variables are trained on all training patterns. Only $\boldsymbol{A}^{\overline{X}}$ will be specifically trained on the prediction errors relative to $\overline{\boldsymbol{X}}$. As such, $\boldsymbol{A}^{\overline{X}}_t$ is able to specifically capture information about the future time steps $\overline{\boldsymbol{X}}_{t:T}$ of the training sample and their existing dependencies with the current time step t. Then, during inference, $\boldsymbol{A}^{\overline{X}}_t$ and $\boldsymbol{d}_{t-1}$ are combined to compute the mean and standard deviation $\boldsymbol{\mu}^{(q)}_t$ and $\boldsymbol{\sigma}^{(q)}_t$ that define the distribution from which $\boldsymbol{Z}_t$ will be drawn. It is to this mechanism that we will be referring in the rest of the article when we claim that we do not directly feed the external inputs to the network during the forward computation; instead, the prediction errors, and thus information about future observations, are propagated through the network via BPTT. The idea to convey information about future observations is also present in variational Bi-LSTMs (Fraccaro et al., 2016; Goyal et al., 2017; Shabanian et al., 2017), although they use a backward RNN for this purpose, and therefore a feedforward mechanism, rather than backpropagation, as we do here.

The approximate posterior is obtained as

$$q_\phi(\boldsymbol{Z}_t \mid \boldsymbol{d}_{t-1}, \boldsymbol{e}_{t:T}) = \mathcal{N}(\boldsymbol{Z}_t; \boldsymbol{\mu}^{(q)}_t, \boldsymbol{\sigma}^{(q)}_t) \quad \text{where}$$
$$[\boldsymbol{\mu}^{(q)}_t, \log \boldsymbol{\sigma}^{(q)}_t] = f^{(q)}_\phi(\boldsymbol{d}_{t-1}, \boldsymbol{A}^{\overline{X}}_t), \tag{2.5}$$

where $f^{(q)}_\phi$ is a one-layer feedforward network, and ϕ denotes the posterior parameters. Detailed computations of $\boldsymbol{A}^{\overline{X}}_t$ in $\boldsymbol{\mu}^{(q)}_t$ and $\log \boldsymbol{\sigma}^{(q)}_t$ are given in appendix A.

Using $\boldsymbol{A}^{\overline{X}}_t$ vectors in our model presents another advantage. In all other variational Bayes RNNs, $\boldsymbol{d}$ units are fed the training patterns directly, and

the network can solely rely on $\boldsymbol{d}$ to regenerate the training pattern, ignoring $\boldsymbol{Z}$ during learning and making it largely irrelevant in the computation (Bowman et al., 2015; Karl et al., 2016; Kingma et al., 2016; Chen et al., 2016; Zhao, Zhao, & Eskenazi, 2017; Goyal et al., 2017). In our proposed model, if $\boldsymbol{d}$ ignores $\boldsymbol{Z}$, then it has no access to pattern-specific information. This is one reason why $\boldsymbol{A}_t^{\overline{X}}$ vectors target $\boldsymbol{Z}_t$ and not $\boldsymbol{d}_t$: to avoid ignoring $\boldsymbol{Z}_t$ during training. On top of that, in our implementation, $\boldsymbol{Z}_t$ has a 10 times smaller dimension than $\boldsymbol{d}_t$, making it more efficient for $\boldsymbol{A}_t^{\overline{X}}$ to target $\boldsymbol{Z}_t$ than $\boldsymbol{d}_t$. One might wonder if rather than introducing new latent vectors $\boldsymbol{A}_t^{\overline{X}}$, we might have directly replaced $\boldsymbol{Z}_t$ by $\boldsymbol{A}_t^{\overline{X}}$ during the posterior computation. We did not do this for two reasons. First, we wanted to keep the structure of the prior and posterior as close as possible. Second, we assumed that providing the information about the past $\boldsymbol{d}_{t-1}$ to the posterior computation of $\boldsymbol{Z}_t$ would be beneficial in some context. This assumption is tested in appendix G.

2.3 Learning Process. To learn the variables θ and ϕ of the generative and inference models, we need to define a loss function. For variational Bayes neural networks, it has been shown that models' variables can be jointly learned by maximizing a lower bound on the marginal likelihood of training data (Kingma & Welling, 2013; Bayer & Osendorfer, 2014; Chung et al., 2015; Fraccaro et al., 2016; Goyal et al., 2017). We maximize a lower bound because maximizing the marginal likelihood directly is intractable. We now derive the lower bound.

Based on equation 2.1, the marginal likelihood or evidence can be expressed as

$$\begin{aligned} &P_\theta(\boldsymbol{X}_{1:T}|\boldsymbol{Z}_0, \boldsymbol{d}_0) \\ &= \int\int \prod_{t=1}^{T} [P_{\theta_X}(\boldsymbol{X}_t \mid \boldsymbol{d}_t, \boldsymbol{Z}_t) P_{\theta_Z}(\boldsymbol{Z}_t \mid \boldsymbol{d}_{t-1}) P_{\theta_d}(\boldsymbol{d}_t \mid \boldsymbol{d}_{t-1}, \boldsymbol{Z}_t)] d\boldsymbol{Z}_{1:T}\, d\boldsymbol{d}_{1:T}. \end{aligned} \tag{2.6}$$

Given $\boldsymbol{d}_{t-1}$ and $\boldsymbol{Z}_t$, the value of $\boldsymbol{d}_t$ is deterministic. Therefore, if we denote the value of the variable $\boldsymbol{d}_t$ as $\tilde{\boldsymbol{d}}_t$ (equal to $f_{\theta_d}(\boldsymbol{d}_{t-1}, \boldsymbol{Z}_t)$, as per equation 2.2), $P_{\theta_d}(\boldsymbol{d}_t \mid \boldsymbol{d}_{t-1}, \boldsymbol{Z}_t)$ is a Dirac distribution centered on $\tilde{\boldsymbol{d}}_t$. By replacing $P_{\theta_d}(\boldsymbol{d}_t \mid \boldsymbol{d}_{t-1}, \boldsymbol{Z}_t)$ by the Dirac delta function $\delta(\boldsymbol{d}_t - \tilde{\boldsymbol{d}}_t)$ in equation 2.6, we can remove the integral over $\boldsymbol{d}$:

$$P_\theta(\boldsymbol{X}_{1:T}|\boldsymbol{Z}_0, \boldsymbol{d}_0) = \int \prod_{t=1}^{T} \left[P_{\theta_X}(\boldsymbol{X}_t \mid \tilde{\boldsymbol{d}}_t, \boldsymbol{Z}_t) P_{\theta_Z}(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1})\right] d\boldsymbol{Z}_{1:T}. \tag{2.7}$$

If we factorize the integral over time and take the logarithm of the marginal likelihood, we will have

$$\log P_\theta(\boldsymbol{X}_{1:T}|\boldsymbol{Z}_0, \boldsymbol{d}_0) = \log \prod_{t=1}^{T} \left[\int P_{\theta_X}(\boldsymbol{X}_t \mid \tilde{\boldsymbol{d}}_t, \boldsymbol{Z}_t) P_{\theta_Z}(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1}) d\boldsymbol{Z}_t \right]$$

$$= \sum_{t=1}^{T} \log \left[\int P_{\theta_X}(\boldsymbol{X}_t \mid \tilde{\boldsymbol{d}}_t, \boldsymbol{Z}_t) P_{\theta_Z}(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1}) d\boldsymbol{Z}_t \right]. \quad (2.8)$$

We now multiply the inside of the integral by $1 = \frac{q_\phi(\boldsymbol{Z}_t|\tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T})}{q_\phi(\boldsymbol{Z}_t|\tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T})}$ in order to obtain an expectation form. Also, this introduces the inference model into equations that were generative-model-only so far, allowing for the joint optimization of both models:

$$\log P_\theta(\boldsymbol{X}_{1:T}|\boldsymbol{Z}_0, \boldsymbol{d}_0)$$

$$= \sum_{t=1}^{T} \log \underbrace{\left[\int q_\phi(\boldsymbol{Z}_t|\tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T}) \frac{P_{\theta_Z}(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1})}{q_\phi(\boldsymbol{Z}_t|\tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T})} P_{\theta_X}(\boldsymbol{X}_t \mid \tilde{\boldsymbol{d}}_t, \boldsymbol{Z}_t) d\boldsymbol{Z}_t \right]}_{E_{q_\phi(\boldsymbol{Z}_t|\tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T})}\left[\frac{P_{\theta_Z}(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1})}{q_\phi(\boldsymbol{Z}_t|\tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T})} P_{\theta_X}(\boldsymbol{X}_t \mid \tilde{\boldsymbol{d}}_t, \boldsymbol{Z}_t) \right]}. \quad (2.9)$$

Since a logarithm is a concave function, we can apply Jensen's inequality: $\log(E[X]) \geq E[\log(X)]$:

$$\log P_\theta(\boldsymbol{X}_{1:T}|\boldsymbol{Z}_0, \boldsymbol{d}_0)$$

$$= \sum_{t=1}^{T} \log \left[\int q_\phi(\boldsymbol{Z}_t|\tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T}) \frac{P_{\theta_Z}(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1})}{q_\phi(\boldsymbol{Z}_t|\tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T})} P_{\theta_X}(\boldsymbol{X}_t \mid \tilde{\boldsymbol{d}}_t, \boldsymbol{Z}_t) d\boldsymbol{Z}_t \right]$$

$$\geq \underbrace{\sum_{t=1}^{T} \int q_\phi(\boldsymbol{Z}_t|\tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T}) \log \left[\frac{P_{\theta_Z}(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1})}{q_\phi(\boldsymbol{Z}_t|\tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T})} P_{\theta_X}(\boldsymbol{X}_t \mid \tilde{\boldsymbol{d}}_t, \boldsymbol{Z}_t) \right] d\boldsymbol{Z}_t}_{L(\theta,\phi)\text{: Variational Evidence Lower Bound}}. \quad (2.10)$$

Now the variational evidence lower bound (ELBO) $L(\theta, \phi)$ can be maximized instead of the logarithm of the marginal likelihood $P_\theta(\boldsymbol{X}_{1:T}|\boldsymbol{Z}_0, \boldsymbol{d}_0)$ in order to optimize the learning variables of the generative model and the approximate posterior. This formula for maximizing the lower bound is equivalent to the principle of free energy minimization provided by Friston

(2005). $L(\theta, \phi)$ can be rewritten as

$$\begin{aligned} L(\theta,\phi) &= \sum_{t=1}^{T}\Bigg(\int q_\phi(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T}) \log P_{\theta_X}(\boldsymbol{X}_t \mid \tilde{\boldsymbol{d}}_t, \boldsymbol{Z}_t) d\boldsymbol{Z}_t \\ &\quad - \int q_\phi(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T}) \log \frac{q_\phi(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T})}{P_{\theta_Z}(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1})} d\boldsymbol{Z}_t\Bigg) \\ &= \sum_{t=1}^{T}\Bigg(E_{q_\phi(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T})}[\log P_{\theta_X}(\boldsymbol{X}_t \mid \tilde{\boldsymbol{d}}_t, \boldsymbol{Z}_t)] \\ &\quad - KL[q_\phi(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T}) \parallel P_{\theta_Z}(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1})]\Bigg), \end{aligned} \tag{2.11}$$

where the first term on the right-hand side is the expected log likelihood under $q_\phi(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T})$ or the negative of the expected prediction error (Kingma & Welling, 2013), and the second term is the negative Kullback-Leibler (KL) divergence between the posterior and prior distributions of the latent variables. Only the summation over time is shown in this equation, but the lower bound is also summed over the number of training samples. We divided the first term by the dimension of $\boldsymbol{X}$ and the second term by the dimension of $\boldsymbol{Z}$ during experiments. The KL divergence is computed analytically as

$$\begin{aligned} &KL[q_\phi(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T}) \parallel P_{\theta_Z}(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1})] \\ &= \log \frac{\boldsymbol{\sigma}_t^{(p)}}{\boldsymbol{\sigma}_t^{(q)}} + \frac{(\boldsymbol{\mu}_t^{(p)} - \boldsymbol{\mu}_t^{(q)})^2 + (\boldsymbol{\sigma}_t^{(q)})^2}{2(\boldsymbol{\sigma}_t^{(p)})^2} - \frac{1}{2}, \end{aligned} \tag{2.12}$$

which is simply the KL divergence between two gaussian distributions. The detailed derivation of the KL divergence is in appendix B.

The variables of the prior θ_Z are optimized through the KL divergence term, whereas variables of the posterior ϕ are optimized through both terms. We can exploit this asymmetry. By weighting the two terms differently, we can increase or decrease the explicit optimization pressure on the learning variables corresponding to the prior or the posterior. To that end, we introduce a weighting parameter, the meta-prior w, in the lower bound (see equation 2.11) to regulate the strength of the KL divergence, producing:

$$\begin{aligned} L_w(\theta,\phi) &= \sum_{t=1}^{T}(E_{q_\phi(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T})}[\log P_{\theta_X}(\boldsymbol{X}_t \mid \tilde{\boldsymbol{d}}_t, \boldsymbol{Z}_t)] \\ &\quad - w \cdot KL[q_\phi(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1}, \boldsymbol{e}_{t:T}) \parallel P_{\theta_Z}(\boldsymbol{Z}_t \mid \tilde{\boldsymbol{d}}_{t-1})]). \end{aligned} \tag{2.13}$$

In the experiments, all model variables and $\boldsymbol{A}$ are optimized in order to maximize the lower bound using ADAM (Kingma & Ba, 2014). We use the same parameter setting for the ADAM optimizer as the original paper: $\alpha = 0.001$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ in training. In both experiments, the latent units $\boldsymbol{Z}$ were 10 times smaller than the number of deterministic units $\boldsymbol{d}$.

2.4 Error Regression. Testing the network on unseen training sequences is not straightforward: it does not accept any input during the forward computation. It does, however, propagate errors during backpropagation, so we leverage this mechanism during testing.

While training the inference model, we created sequences of adaptive vectors $\boldsymbol{A}^{\overline{X}}_{1:T}$—one for each training observation. The purpose was to capture the relevant information about the training observation into $\boldsymbol{A}^{\overline{X}}_{1:T}$ and train the other weights of the network, θ and ϕ, to use this information to make useful predictions. Another way to understand this is that $\boldsymbol{A}^{\overline{X}}_{1:T}$ are building good representations that the rest of the network, shared among all training sequences, learns to use. In that sense, the adaptive vectors $\boldsymbol{A}^{\overline{X}}_{1:T}$ on one side, and the weights θ and ϕ on the other side, are fulfilling vastly different roles. And as we will see, once the training is done, the values of $\boldsymbol{A}^{\overline{X}}_{1:T}$ are no longer needed to process unseen testing sequences.

When processing an unseen testing sequence, the weights θ and ϕ are fixed, and the adaptive vectors $\boldsymbol{A}^{\overline{X}}_{1:T}$ are unavailable. We initialize the adaptive vector $\boldsymbol{A}^{\text{test}}_{1:T'}$ to zero values; we are going to optimize $\boldsymbol{A}^{\text{test}}_{1:T'}$ online, during the processing of $\overline{\boldsymbol{X}}^{\text{test}}$, to maximize our ability to predict it. This online optimization is done incrementally, inside a time window of size m. The process is illustrated in Figure 1C.

Using the m (and, for now, zero-valued) $\boldsymbol{A}^{\text{test}}_{1:m}$ values, we can generate $\boldsymbol{Z}^{\text{pred}}_{1:m}$ using the inference model q_ϕ (see equation 2.5) and compute $\boldsymbol{d}^{\text{pred}}_{1:m}$ using equation 2.3. The prediction $\boldsymbol{X}^{\text{pred}}_{1:m}$ can also be computed using $\boldsymbol{Z}^{\text{pred}}_{1:m}$ and $\boldsymbol{d}^{\text{pred}}_{1:m}$. $\boldsymbol{X}^{\text{pred}}_{1:m}$ is then compared to $\overline{\boldsymbol{X}}^{\text{test}}_{1:m}$, and the resulting prediction errors $\boldsymbol{e}_{1:m}$ are backpropagated through the network to update the values of $\boldsymbol{A}^{\text{test}}_{1:m}$. The update is done the same way the network is trained—by computing the lower bound and using BPTT—except that the variables θ and ϕ are fixed and are not modified. The new values of $\boldsymbol{A}^{\text{test}}_{1:m}$ are used to generate a new prediction $\boldsymbol{X}^{\text{pred}}_{1:m}$, and a new optimization cycle can occur. The number of optimization cycles of $\boldsymbol{A}^{\text{test}}_{1:m}$ for a given time window can depend on reaching a given error threshold, be fixed beforehand, or, in a real-time context, depend on the available computational time. Next, the time window is slid to $[2, m+1]$, and $\boldsymbol{A}^{\text{test}}_{2:m+1}$ are used to generate $\boldsymbol{X}^{\text{pred}}_{2:m+1}$ and are optimized. Importantly, only the part of $\boldsymbol{A}^{\text{test}}_{1:T'}$ inside the time window—here $\boldsymbol{A}^{\text{test}}_{2:m+1}$—is optimized. In particular, $\boldsymbol{A}^{\text{test}}_{1}$ is now fixed. After the optimization of $\boldsymbol{A}^{\text{test}}_{2:m+1}$, the time window moves to $[3, m+2]$ and so on.

At any point in this process, for a time window $[t-m, t-1]$, the prediction steps outside the time window $\boldsymbol{X}_t, \boldsymbol{X}_{t+1}, \boldsymbol{X}_{t+2}, \ldots$ can be generated by computing $\boldsymbol{Z}_t, \boldsymbol{Z}_{t+1}, \boldsymbol{Z}_{t+2}, \ldots$ using the generative model (see equation 2.4), which does not depend on the values of $\boldsymbol{A}_t^{\text{test}}, \boldsymbol{A}_{t+1}^{\text{test}}, \boldsymbol{A}_{t+2}^{\text{test}}, \ldots$ which are, at this point, zero. The predictions $\boldsymbol{X}_t, \boldsymbol{X}_{t+1}, \boldsymbol{X}_{t+2}, \ldots$ correspond to unobserved parts of the testing sequence at this point and therefore are the model's prediction of the future. These additional predictions have no impact on the BPTT process of error regression.

Finally, we note that the optimization can begin before the time window is at full size and start with time windows $[1, 1], [1, 2], \ldots, [1, m], [2, m+1]$ and so on. Additionally, the optimization does not need to happen at every time step and can, for instance, be triggered every 10 time steps, with time windows $[1, 10], [1, 20], \ldots, [1, m], [11, m+10], [21, m+20], \ldots$ (assuming here that m is a multiple of 10).

The error regression process was implemented in deterministic RNNs, and it was shown how it could help the generalization capability of those models (Tani & Ito, 2003; Murata et al., 2017; Ahmadi & Tani, 2017b). This testing process through error regression bears similarities to, and is inspired by, predictive coding. Predictive coding proposes that the brain is continually making predictions about incoming sensory stimuli and that error between the prediction and the real stimuli is propagated back up through the layers of the processing hierarchy. Those error signals are then used to update the internal state of the brain, with impacts on future predictions. Our network goes through similar stages during error regression: predictions are made ($\boldsymbol{X}^{\text{pred}}$), compared to actual observations ($\overline{\boldsymbol{X}}^{\text{test}}$), and the errors ($\boldsymbol{e}_{1:m}$) are backpropagated to update the internal state of the network ($\boldsymbol{A}_{1:m}^{\text{test}}$). To be very clear, our network is not a model of the brain; it does not claim to explain any existing neurological data or make any useful predictions about animal brains. We are merely drawing inspiration from the predictive coding ideas to design new machine learning networks. In particular, in neurological models of predictive coding (Rao & Sejnowski, 2000), each layer makes an independent prediction and propagates the error signal to the upper processing layer only. In our network, the prediction error from the raw sensory data is backpropagated through the entire network hierarchy. This is deliberate, because we use BPTT: we adapted the ideas of predictive coding to the classical tools of recurrent neural networks.

2.5 Related Work. RNNs are widely used to model temporal sequences due to their ability to capture long dependencies in data. However, a deterministic RNN $\boldsymbol{d}_t = f(\boldsymbol{d}_{t-1}, \overline{\boldsymbol{X}}_{t-1})$ can have problems when modeling stochastic sequences with a high signal-to-noise ratio (Chung et al., 2015). In an attempt to solve this problem, Bayer and Osendorfer (2014) introduced a model called STORN by inserting a set of independent latent variables (sampled from a fixed distribution) into the RNN model. Later,

the VRNN model was proposed using conditional prior parameterization (Chung et al., 2015). In their model, the prior distribution is obtained using a nonlinear transformation of the previous hidden state of the forward network as $[\boldsymbol{\mu}_t^{(p)}, \log(\boldsymbol{\sigma}_t^{(p)})] = f^{(p)}(\boldsymbol{d}_{t-1})$. VRNN outperformed STORN by using this type of conditional prior. However, in VRNN, the posterior is inferred at each time step without using information from future observations. A posterior inferred in such a way would be different from the true posterior. Later, this issue was considered by using two RNNs: a forward RNN and a backward one. The backward RNN was used in the posterior to transfer future observations for the current prediction (Fraccaro et al., 2016; Goyal et al., 2017; Shabanian et al., 2017). As explained, our model manages this by updating $\boldsymbol{A}^{\overline{X}}$ through backpropagation of the error signal.

Recent studies of generative models show that extracting a meaningful latent representation can be difficult when using a powerful decoder. The $\boldsymbol{d}$ units ignore the latent variables $\boldsymbol{Z}$ and capture most of the entropy in the data distribution (Goyal et al., 2017). Many researchers have addressed this issue by either weakening the decoder or annealing the KL divergence term during training (Bowman et al., 2015; Karl et al., 2016; Kingma et al., 2016; Chen et al., 2016; Zhao et al., 2017). In a recent attempt, the authors of *Z*-forcing also proposed an auxiliary training signal for latent variables alone, which forces the latent variables to reconstruct the state of the backward RNN (Goyal et al., 2017). This method introduces an additional generative model and, as a result, an additional cost on the lower bound. Comparatively, our model captures information about external inputs in $\boldsymbol{A}^{\overline{X}}$ and the information flows to $\boldsymbol{d}$ through $\boldsymbol{Z}$, rendering the model unable to ignore its latent variables.

Therefore, in our model, those two issues, capturing future dependencies and avoiding having the network ignore its latent states, are addressed with the same mechanism: the adaptive vectors $\boldsymbol{A}^{\overline{X}}$.

Introducing adjustable parameters in the lower bound has been studied for variational Bayes neural networks previously. KL annealing does this (Bowman et al., 2015), linearly increasing the weight of the KL-divergence term from 0 to 1 during the training process to avoid ignoring the latent variables and to improve convergence. Higgins et al. (2017) showed that the degree of the disentanglement in latent representations of VAE models can be improved by strengthening the importance of the KL divergence term in the lower bound. The generative factors in an image of a dog, for example, can be its color, size, and breed. Disentangling the generative factors in the model can be beneficial, as it creates latent units sensitive to the changes in a single generative factor while being relatively invariant to changes in other factors (Bengio, Courville, & Vincent, 2013). Our model considers weighting the KL divergence term for a purpose different from KL annealing or disentanglement: to influence the balance between a deterministic and a stochastic representation of the data in the model.

The current study is a continuation of our previous work (Ahmadi & Tani, 2017a) that proposed a predictive-coding variational Bayes RNN and studied the effect of weighting the KL divergence term. This model, however, was composed of only the latent variables $\boldsymbol{Z}$ and did not use the deterministic units $\boldsymbol{d}$; it also used a prior distribution with a fixed mean and standard deviation that has been shown not to be plausible for models that deal with time-series data (Chung et al., 2015). This led us to consider separating stochastic and deterministic states in the current model. It allows us to have a conditional prior.

Separating deterministic and stochastic states provides an additional advantage: it allows having a number of $\boldsymbol{Z}$ units significantly smaller than $\boldsymbol{d}$ units. In our test, having 10 times more $\boldsymbol{d}$ units than $\boldsymbol{Z}$ units was the best balance between performance and computational time; the number of $\boldsymbol{A}$ units was always the same as the number of $\boldsymbol{Z}$ units. We use this ratio in all our experiments.

3 Simulation Experiments

We conducted simulation experiments to examine how learning in the proposed model depends on the meta-prior w. The first experiment investigates how the proposed model could learn to extract the latent probabilistic structure from discrete (0 or 1) data sequences generated from a simple probabilistic finite state machine (PFSM) under different settings of the meta-prior w. The purpose of this relatively simple experiment is to conduct a detailed analysis of the underlying mechanism of the PV-RNN when embedding the latent probabilistic structure of the data into mixtures of deterministic and stochastic dynamics. In the second experiment, a more complex situation is considered where the model is required to extract latent probabilistic structures from continuous sequence patterns (movement trajectories). For this purpose, trajectory data were generated by considering probabilistic switching of primitive movement patterns based on another predefined PFSM in which each primitive was generated with fluctuations in amplitude, velocity, and shape. Again, we examined how the performance depends on the meta-prior w.

3.1 Experiment 1. The PFSM shown in Figure 2A was used as the target generator. Transitions from s_1 to s_2 and s_2 to s_3 were deterministically determined with 1 and 0 as output, respectively. However, the transitions from s_3 to s_1 were randomly sampled with 30% and 70% probabilities for output 0 and 1, respectively. Ten target sequence patterns, of 24 time steps each, were generated and provided to the PV-RNN as training data. Each model had only one context layer consisting of 10 $\boldsymbol{d}$ units and a single $\boldsymbol{Z}$ unit. The time constant τ for all $\boldsymbol{d}$ units was set to 2.0. The output of the network, $\boldsymbol{X}_{1:T}$, was discretized during testing, with outputs less than 0.5 assigned to zero and the ones equal to or larger than 0.5 assigned to one. Finding an

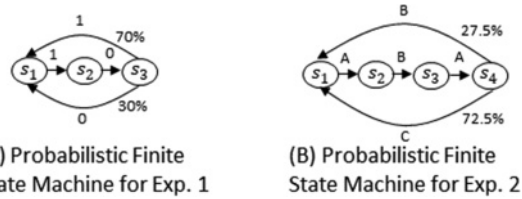

Figure 2: The probabilistic finite state machines used to generate training patterns for PV-RNNs in the (A) first and (B) second experiments.

adequate range of w at the beginning of an experiment depends on the network parameter settings, the data set, and the task. For this experiment, the most interesting behavior was observed in the range [0.0001, 0.1]. For w set to larger values such as 0.5 and 1.0, the networks showed the same qualitative behavior to the network with w set to 0.1. Training was conducted on seven models with the different meta-prior w set to 0.1, 0.05, 0.025, 0.015, 0.01, 0.001, and 0.0001, respectively. In this experiment, we used an MTRNN to be consistent with our other experiments. However, it is possible to do this experiment with a simple RNN as well. Similar results were obtained by using a simple RNN and are shown in appendix D.

After training for 500,000 epochs, given a training sequence $\overline{\boldsymbol{X}}$, the learned value of $\boldsymbol{A}_1^{\overline{X}}$ is fed to the network, generating $\boldsymbol{Z}_1$ via equation 2.5. Then the remaining latent states $\boldsymbol{Z}_{2:T}$ and the output $\boldsymbol{X}_{1:T}$ are generated from the generative model (see equation 2.4). The purpose is to study if providing $\boldsymbol{A}_1^{\overline{X}}$ is enough for the trained network to regenerate $\overline{\boldsymbol{X}}$ accurately. We refer to this procedure as *target regeneration*.

Figure 3 compares one target sequence pattern and its corresponding regeneration by the PV-RNN model trained with different values of the meta-prior. For large values of w, the network reproduced the training pattern accurately. As the value of w decreases, divergences appear earlier and earlier, and for low values even the deterministic steps show errors.

For a given reconstruction, one can compute the diverging step as the time t of the first difference between the target and the reconstruction. If both target and reconstruction are identical, the diverging step is equal to the length of the reconstruction. For each training pattern, we compute the diverging step 10 times and compute the mean of all results to obtain the average diverging step (ADS) over the training data set.

To characterize the deterministic nature of the network behavior, we compute the variance of the divergence (VD), which shows diversity among sequence patterns regenerated from the same value of $\boldsymbol{A}_1^{\overline{X}}$. For a given value of $\boldsymbol{A}_1^{\overline{X}}$, we ran the regeneration 50 times and computed the mean variance (across all 50 runs and all time steps) of the generated $\boldsymbol{X}$ before discretization.

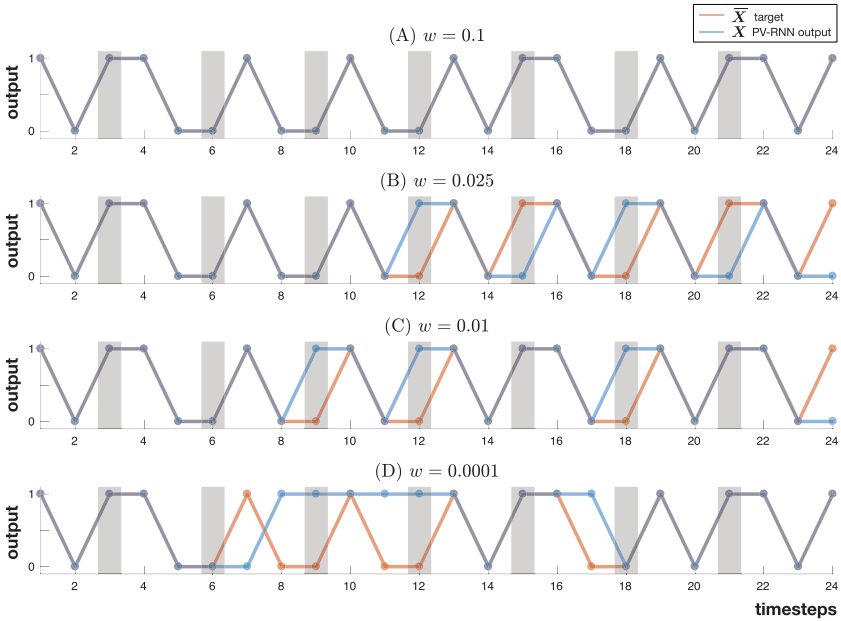

Figure 3: Larger values of the meta-prior translate into better reconstruction of the training patterns. The four graphs show a training pattern (in orange) and its reconstruction by PV-RNN (in blue) for different values of w. Overlapping sections are in dark gray. For $w = 0.1$, the target sequence is completely regenerated. When w is equal to 0.025 and 0.01, all deterministic steps are correctly reproduced, but regenerated patterns begin to diverge at the 11th and 8th step, respectively. When w is set to 0.0001, even the deterministic transition rules fail to be reproduced, and the signals diverge at the 6th time step.

ADS and VD for different values of w are shown in Table 1. ADS decreases while VD increases as w decreases. For $w = 0.1$, VD is near zero; the network reproduces the same pattern with little variation, and the behavior developed can be regarded as deterministic. The relatively high value of VD for $w = 0.0001$, however, points to highly stochastic dynamics.

In Table 1, we examine the ability of the network to extract the latent probabilistic structure from the data by computing the KL divergence between the probability distributions of sequences of length 12 generated by the PFSM, $P(\overline{\boldsymbol{X}}_{t:t+11})$ and the one generated by the PV-RNN, $P(\boldsymbol{X}_{t:t+11})$ (thus characterizing how similar they are). To compute the probability distribution $P(\boldsymbol{X}_{t:t+11})$, we set $\boldsymbol{A}_1$ randomly and generate a sequence of 50,000 steps using the generative model. We refer to this as *free generation*. We consider the distribution of the 49,989 sequences of length 12 $\boldsymbol{X}_{t:t+11}$ present in the sequence and compute their distribution. For the probability distribution

Table 1: Evaluation of the Regeneration and Generalization Capabilities of the PV-RNN Model Trained with Different Values for w.

	Meta-Prior w						
	0.1	0.05	0.025	0.015	0.01	0.001	0.0001
Average diverging step (ADS)	**22**	19	14	12	11	9	8
Variance of divergence (VD)	**0.00003**	0.00155	0.0480	0.0499	0.0618	0.134	0.172
KL divergence of test phase	5.040	2.276	**0.0684**	0.120	0.148	1.0679	5.607

Notes: The average diverging step (ADS) and variance of divergence (VD) measures point to better reconstruction performance when w is high. However, taking into account the KL divergence between the probabilistic distribution of the generated pattern $P(\boldsymbol{X}_{t:t+11})$ and the one of the training data, $P(\overline{\boldsymbol{X}}_{t:t+11})$ paints another picture: the network best captures the probabilistic structure of the data for an average value of w. The bold numbers in rows 1 and 2 show the model with the best regeneration capability. The bold numbers in row 3 shows the model with best generalization capability.

$P(\overline{\boldsymbol{X}}_{t:t+11})$, we concatenate the 10 training sequences in one sequence of 240 time steps (which is a valid output sequence of the PFSM) and compute the distribution of the 229 sequences of the length 12 we could extract from it. The resulting KL divergence measures from those two distributions for all PV-RNN models show that average values of the meta-prior capture the underlying transition probability of the PFSM from the training data best.

Figure 4 displays the mean and variance of the latent state during regeneration of a target sequence for w equal to 0.1 and 0.025 and confirms this analysis. With $w = 0.1$, the network possesses deterministic dynamics that amount to rote learning. With $w = 0.025$, the network distinguishes between deterministic and probabilistic states, and captures the probabilistic structure in its internal dynamics. Plots showing the cases of w set to 0.001 and 0.0001 are provided in Figure 11 in appendix H. With $w = 0.0001$, the value of sigma becomes high even for the deterministic case; the network does not distinguish anymore between deterministic and probabilistic states, and behaves as a random process.

Figure 5 illustrates the generated output, the mean $\boldsymbol{\mu}^{(p)}$, and the standard deviation $\boldsymbol{\sigma}^{(p)}$ of the $\boldsymbol{Z}$ unit for PV-RNNs trained with w equal to 0.1 and 0.025 from time steps 20,002 to 20,040. The behaviors of $\boldsymbol{\mu}^{(p)}$ and $\boldsymbol{\sigma}^{(p)}$ in both cases are similar to those shown in Figure 4. In the case of w set to 0.025, the network was most successful at extracting the latent probabilistic structure from the data by detecting both the uncertain and deterministic states in the sequence. The same figure for w equal to 0.001 and 0.0001 is shown in Figure 12 in appendix H. The transition rules defined in the PFSM were mostly broken for the case with the minimum w value (0.0001) and frequently for the case of w equal to 0.001 as the model wrongly estimated the uncertainty as high even for the deterministic states.

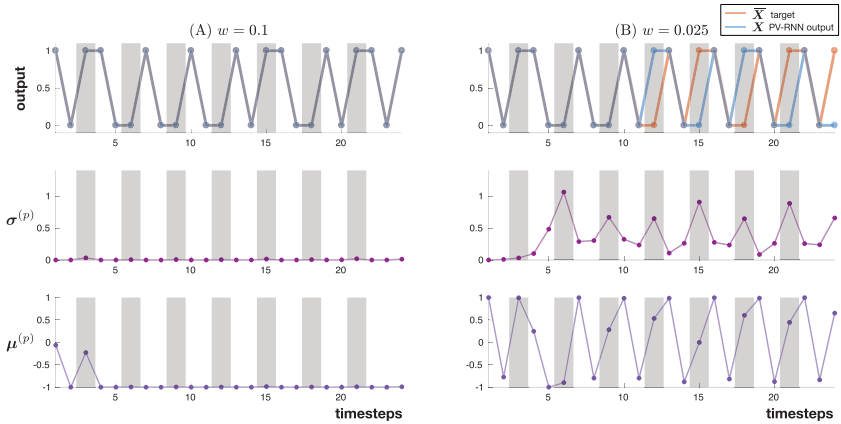

Figure 4: A high meta-prior forces the network into deterministic dynamics. With an average value of w, the probabilistic structure of the data is captured. The mean $\boldsymbol{\mu}^{(p)}$ (middle row) and variance $\boldsymbol{\sigma}^{(p)}$ (bottom row) of the latent state during regeneration of a given $\overline{\boldsymbol{X}}$ (top row) for w equal to 0.1 and 0.025 are shown. With $w = 0.1$, the $\boldsymbol{\sigma}^{(p)}$ is near zero. It amounts to rote learning by the network of the training pattern. $\boldsymbol{\mu}^{(p)}$, on the other hand, varies only during the first few time steps, suggesting that the information identifying which training pattern to regenerate is transferred to the network early on, and thereafter, the value of $\boldsymbol{Z}$ is disregarded. With $w = 0.025$, the variance $\boldsymbol{\sigma}^{(p)}$ is much larger overall and significantly higher for the probabilistic states (gray bars). This suggests that PV-RNN with w set to 0.025 is capable of discriminating between deterministic and probabilistic steps in the sequence. This effect is reflected in $\boldsymbol{\mu}^{(p)}$ as well, with most deterministic states having a $\boldsymbol{\mu}^{(p)}$ close to either 1 or -1 and probabilistic states mostly confined to the range $[0, 0.75]$, the asymmetry over the range possible range ($[-1, 1]$) possibly even reflecting the 70%/30% difference in transition probability.

We observed that the deterministic network developed with w set to 0.1 generated nonperiodic output patterns. This can be roughly seen in Figure 5A. We assumed that deterministic chaos or transient chaos developed in this learning condition. To confirm this, the Lyapunov exponents were computed using the method in Alligood, Sauer, and Yorke (1996). Interestingly, the largest Lyapunov exponent was positive. We evaluated this by generating patterns (using free generation) for 50,000 steps twice: once as usual and once with the random variable generating the value of the $\boldsymbol{Z}$ unit, $\boldsymbol{\varepsilon}_{1:50000}$, set to zero (so that the value of $\boldsymbol{\sigma}^{(p)}$ is irrelevant). This was done to verify that the noise generation was not affecting the value of the Lyapunov exponent. In both cases, the largest Lyapunov exponents were positive (around 0.1). The method for computing Lyapunov exponents is described in appendix C.

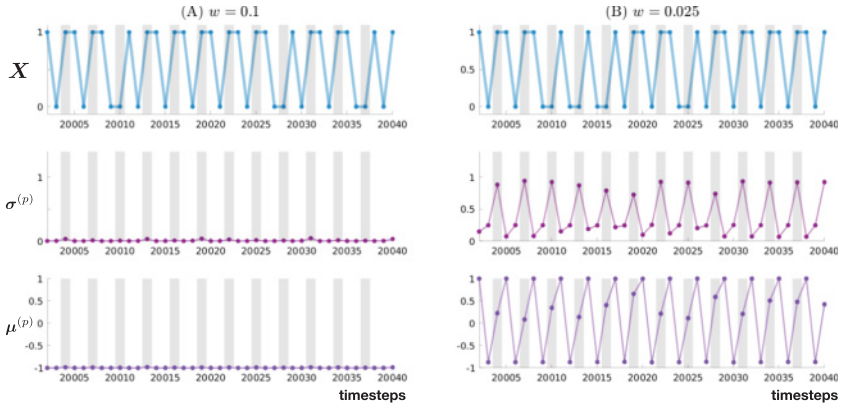

Figure 5: The generated output, the mean $\boldsymbol{\mu}^{(p)}$, and the standard deviation $\boldsymbol{\sigma}^{(p)}$ from time steps 20,002 to 20,040 of two PV-RNNs trained with the meta-prior w set to 0.1 (A) and 0.025 (B). Gray bars show the time steps corresponding to uncertain states.

The results of this experiment can be summarized as follows. It was shown that different types of internal dynamics can be developed in the current model depending on the value of the meta-prior w used during training on stochastic sequences. When w is set to a large value (0.1), deterministic dynamics were generated by minimizing $\boldsymbol{\sigma}^{(p)}$ in the prior to nearly 0 for all time steps. The deterministic aspect of the developed dynamics was further confirmed by observing that they generated the least diversity when generation was run multiple times starting from the same initial $\boldsymbol{A}_1$. The finding of the maximum Lyapunov exponent of the dynamics as a positive value confirmed that those dynamics developed into deterministic chaos. It was also found that the average diverging steps (ADS) became larger when w was set to a larger value: each training target sequence was captured exactly for relatively long time steps, in a fashion akin to rote learning.

On the other hand, decreasing w generated stochastic dynamics, even approaching the random process for low values of the meta-prior, as evidenced by the increase of diversity in sequences generated from the same latent initial state. It was found, however, that the best generalization in learning took place with w set to an intermediate value. The analysis of the latent variable in this condition revealed that low values of w translated into high values of $\boldsymbol{\sigma}^{(p)}$ for probabilistic and deterministic state transitions. For intermediate values of the meta-prior, however, high values of $\boldsymbol{\sigma}^{(p)}$ were mostly observed for probabilistic state transition, indicating that the model did discriminate between the two in that case. To understand why this is the case, one must observe that the KL divergence term of equation 2.13 acts as

a pressure for $\boldsymbol{\sigma}^{(p)}$ to be close to $\boldsymbol{\sigma}^{(q)}$ and $\boldsymbol{\mu}^{(p)}$ to be close to $\boldsymbol{\mu}^{(q)}$ and for the posterior and prior distributions to be similar to one another.

When w is small, the pressure that the KL divergence term has on the backpropagation process is small to almost nonexistent. Therefore, the pairs $\boldsymbol{\sigma}^{(p)}$, $\boldsymbol{\sigma}^{(q)}$ and $\boldsymbol{\mu}^{(p)}$, $\boldsymbol{\mu}^{(q)}$ are free to be uncorrelated. The other term of equation 2.13, the reconstruction error, puts learning pressure on $\boldsymbol{\sigma}^{(q)}$ and $\boldsymbol{\mu}^{(q)}$. Therefore, there is little learning pressure on the prior distribution, and it mostly stays close to its initialization values. In our implementation, those values are random, and therefore the network acts as a random process when the $\boldsymbol{Z}$ states are generated by the generative model.

When w is high, the pressure is high for the posterior and prior distributions to be similar. Deterministic states are easier for the network to learn, and therefore both the $\boldsymbol{\sigma}^{(q)}$ and $\boldsymbol{\sigma}^{(p)}$ can converge to small values so as to reduce both the KL divergence term and the reconstruction error term of equation 2.13. Probabilistic states take longer to learn. Looking at the closed-form solution of the KL divergence term, equation 2.12, one way to reduce the KL divergence between the posterior and prior distributions is to increase $\boldsymbol{\sigma}^{(p)}$ when $\boldsymbol{\mu}^{(q)}$ and $\boldsymbol{\mu}^{(p)}$ are different. And this is the temporary solution that the network seems to be using, when looking at the evolution of $\boldsymbol{\sigma}^{(p)}$ in Figure 14 in appendix H. Eventually the network makes $\boldsymbol{\sigma}^{(q)}$ and $\boldsymbol{\sigma}^{(p)}$ converge to zero in order to minimize the KL divergence further.

For the network with w set to an intermediate value, the pressure is less for the posterior and prior distributions to be similar. $\boldsymbol{\sigma}^{(q)}$ and $\boldsymbol{\sigma}^{(p)}$ do not converge to zero, and the network seems to stay in the intermediate solution.

3.2 Experiment 2. In this experiment, the PV-RNN was required to extract latent probabilistic structures from observed continuous sequence data (movement trajectories). We generated 48,400-time-step data sequences and one of length 6400 time steps were generated using the PFSM depicted in Figure 2B, where the primitive pattern A, B, and C corresponded to a circle, a Figure 8, and a triangle, respectively. The sequences were based on human hand-drawn patterns with naturally varying amplitude, velocity, and shape. One such sequence can be seen in Figure 13 in appendix H. Sixteen of the 48,400-step sequences were used to train the model, while the 32 remaining ones were reserved for testing. The details of the generation are in appendix E.

For this experiment, the most interesting behavior was observed with w in the range $[1.0 \times 10^{-3}, 0.01 \times 10^{-3}]$. To avoid excessive notation in the following text, we introduce $w' = w \times 10^3$, so that when w evolves in the range $[1.0 \times 10^{-3}, 0.01 \times 10^{-3}]$, w' evolves in $[1.0, 0.01]$.

Six PV-RNN models were trained with w' set to 1.0, 0.5, 0.25, 0.15, 0.1, and 0.01. Each model had three context layers consisting of 80 $\boldsymbol{d}$ units and 8 $\boldsymbol{Z}$ units for the fast context (FC) layer, 40 $\boldsymbol{d}$ units and 4 $\boldsymbol{Z}$ units for the

middle context (MC) layer, and 20 $\boldsymbol{d}$ units and 2 $\boldsymbol{Z}$ units for the slow context (SC) layer. The time constants of FC, MC, and SC units were set to 2, 4, and 8, respectively. Training ran for 250,000 epochs in each case. We also conducted experiments using two context layers for two PV-RNN models with w' set to 0.25. The results are in appendix F.

During testing, the capability of the generative model to reproduce the training patterns was evaluated through target regeneration. For this purpose, target patterns were regenerated by providing the initial latent state $\boldsymbol{A}_1$ with the value obtained during training, as we did in experiment 1. The latent states $\boldsymbol{Z}_{2:T}$ and $\boldsymbol{X}_{1:T}$ were computed by the generative model. Figure 6 illustrates how the regeneration is affected by different values of the meta-prior, and Table 2 shows the ADS for those values, that is, in the continuous case, the time step at which the mean square error between the target and the generated pattern exceeded a threshold (0.01) over 10 repetitions of each training sequence, as well as the mean activity of the variance $\boldsymbol{\sigma}^{(p)}$ for the whole training set. We obtain results in accordance with the ones of experiment 1: the PV-RNN model trained with the largest meta-prior value ($w' = 1.0$) exhibits deterministic dynamics, while the one trained with low values w' approaches random process behavior.

To test the generalization capabilities of the models, the prediction performance using error regression was evaluated. The test pattern of length 6400 steps was given to each PV-RNN model to make predictions from 1 to 5 steps ahead. The size of the time window was set to 50, and $\boldsymbol{A}^{\text{test}}_{t-50:t-1}$ was optimized 30 times at every time step. The time window was continuously sliding one step forward at a time to generate the whole sequence $\boldsymbol{X}_{1:6400}$. The MSE between the test pattern and the generated output for all prediction steps is given in Table 3. The PV-RNN trained with w' set to 0.25 outperforms other models in all cases except for the one-step-ahead prediction where $w' = 0.1$ has a small lead. Two-dimensional visualizations of the test for one-step- and five-step-ahead predictions with w' set to 1.0, 0.25, and 0.1 are shown in Figure 7. As expected, predicting five steps ahead is challenging for all models. However, in this case, the network with w' set to 0.25 performs best at preserving the structure of the target. When w' is set to 0.1, the network predicting five steps ahead generates a quite noisy pattern. The structure looks qualitatively wrong in some areas for both cases of prediction when w' is set to 1.0.

Previous performance measures focus on the model's ability to quantitatively reproduce or predict each time step. To characterize the ability of the model to qualitatively reproduce and predict the correct patterns, we designed an experiment using error regression with a longer but fixed time window. Contrary to previous experiments with error regression when the time window would gradually grow to full size m and then slide over the whole test sequence, here we consider one time window starting at time step 1 and ending at time step 200 (included). In particular, we do not consider time windows $[1, 1]$, $[1, 2]$, $\ldots$, $[1, 199]$. One thousand optimization

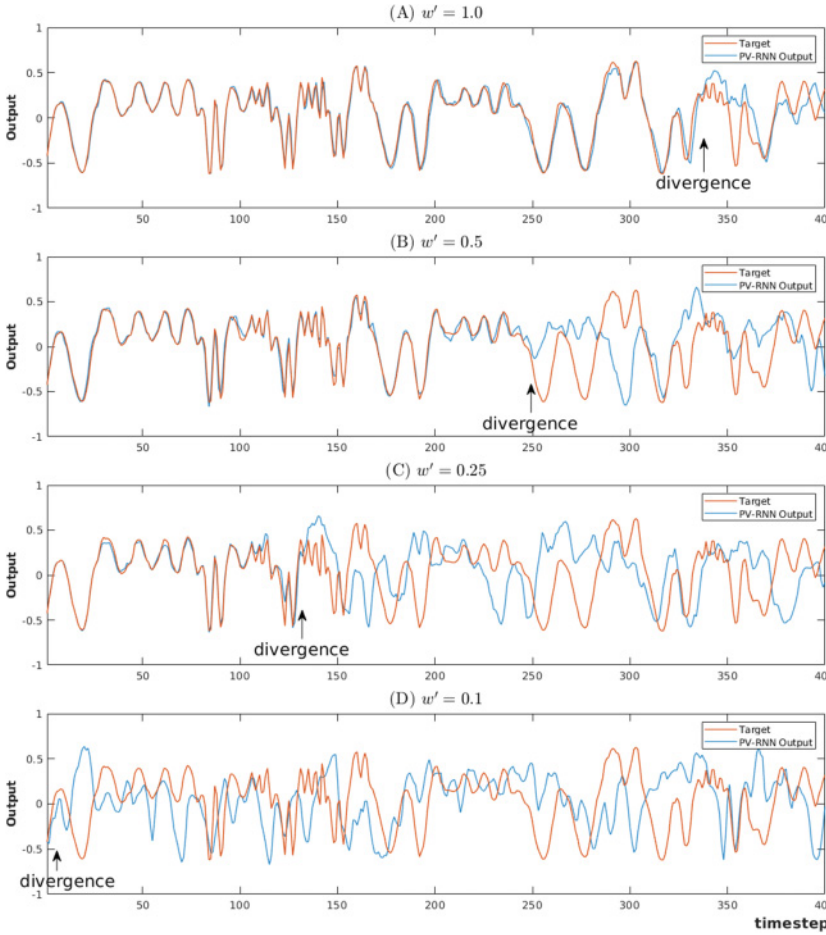

Figure 6: PV-RNN regenerates training patterns better when the meta-prior has a high value. The four graphs show one dimension (y, here) of a training pattern (in orange) and the output regenerated by PV-RNN (in blue), obtained by bootstrapping the value of $\boldsymbol{A}_1$ with the one obtained during training, and computing predictions $\boldsymbol{X}_{1:T}$ using the generative model exclusively for the remaining time steps. Black arrows point to the diverging steps in which the regenerated output diverges from the target pattern.

steps of $\boldsymbol{A}^{test}_{1:200}$ are performed in this error regression time window. Then the generative model is used to generate 200 additional steps, producing $\boldsymbol{X}_{201:400}$. This predicted 2D output is then analyzed and labeled by a human with the three primitive pattern types A, B, or C, and compared to the ground truth of the corresponding testing pattern. Figure 8 shows one

Table 2: Evaluation of the Regeneration Capability of the PV-RNN Model Trained with Different Values for w.

	w'					
	1.0	0.5	0.25	0.15	0.1	0.01
ADS	**343**	229	103	17	4	1
Mean of Variance	**0.0007**	0.0015	0.0039	0.005	0.021	0.1126

Notes: High meta-prior translates into deterministic dynamics, while low values produce random-process-like behavior. When $w' = 1.0$, the divergence starts from the 343th step driven by low stochasticity, and as w' becomes smaller, the divergence starts earlier. When $w' = 0.01$, the divergence starts immediately after the onset and the variance is high. The bold numbers show the model with the best regeneration capacity.

Table 3: MSE between the Target and Look Ahead Prediction.

	w'					
	1.0	0.5	0.25	0.15	0.1	0.01
1-step prediction	0.0101	0.00726	0.00418	0.00376	**0.00341**	0.00834
2-steps prediction	0.0171	0.0127	**0.00907**	0.00918	0.0116	0.0229
3-steps prediction	0.0222	0.0183	**0.0140**	0.0152	0.0205	0.0378
4-steps prediction	0.0279	0.0233	**0.0189**	0.0212	0.0301	0.0497
5-steps prediction	0.0325	0.0274	**0.0234**	0.0270	0.0375	0.0578

Notes: The best predictions are produced by the model trained with an intermediate value of the meta-prior ($w' = 0.25$). The table shows the MSE between the unseen test targets and the one-step- to five-steps-ahead generated predictions. The networks with minimum MSE are shown in bold numbers.

instance of the labeling of a test pattern, and illustrates the effects of different values of the meta-prior.

If a model produces the right primitive after the error regression window, it has 1-primitive prediction capability. If it produces the correct two primitive in the right order, it has 2-primitive prediction capability, and so on. Primitive reproduction is not rated on how long it lasts or if it would coincide temporally with the test pattern. Table 4 shows the aggregated prediction performance of each PV-RNN model over the 32 patterns of the testing data set. The prediction capability is best when w' is set to 0.25. This indicates that generalization in predicting at the primitive sequence level can be achieved to the highest degree by extracting the latent probabilistic structure in primitive sequences adequately when w' balances well the two optimization terms of the lower bound. Previous work with MTRNN models has shown that higher layers (middle and slow) can learn the transitions between primitive patterns, while the lowest (fast) layer learns detailed information about primitive patterns (Yamashita & Tani, 2008;

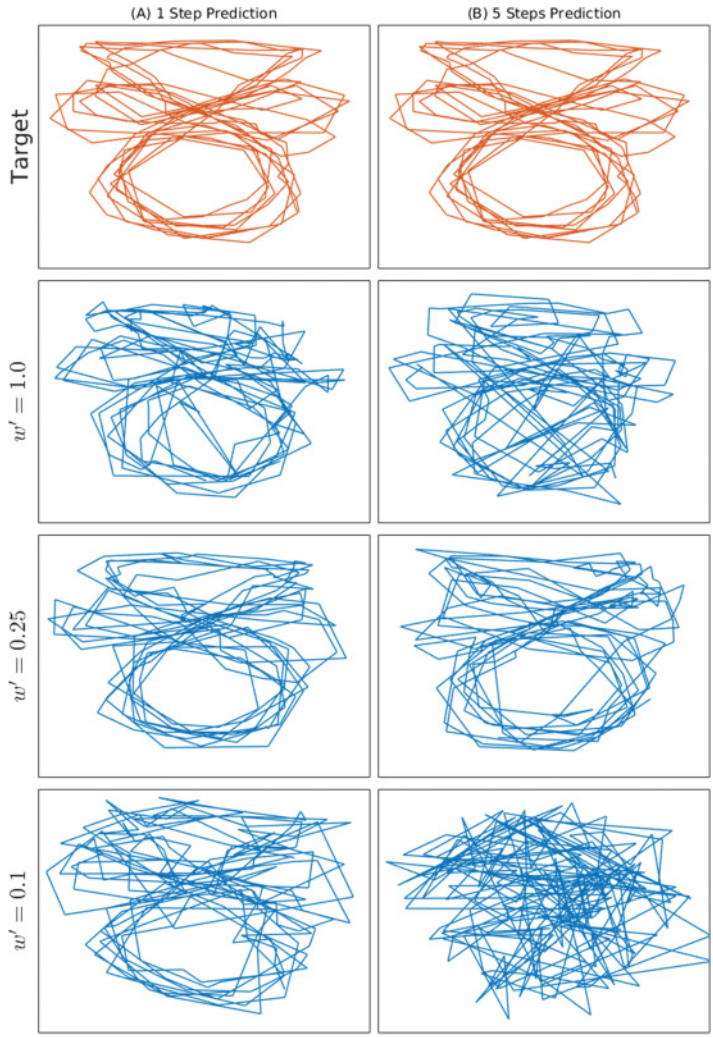

Figure 7: Only the $w' = 0.25$ case retains good qualitative behavior for both one-step- and five-step-ahead predictions. Target and prediction outputs of PV-RNNs with w' set to 1.0, 0.25, and 0.1 during error regression when predictions are made (A) one step ahead and (B) five steps ahead. Only the first 200 steps are shown to retain clarity.

Ahmadi & Tani, 2017b). Here, it is probable that PV-RNN was able to predict long-term primitive sequences by using the slow timescale dynamics developed in the higher layers of the network hierarchy.

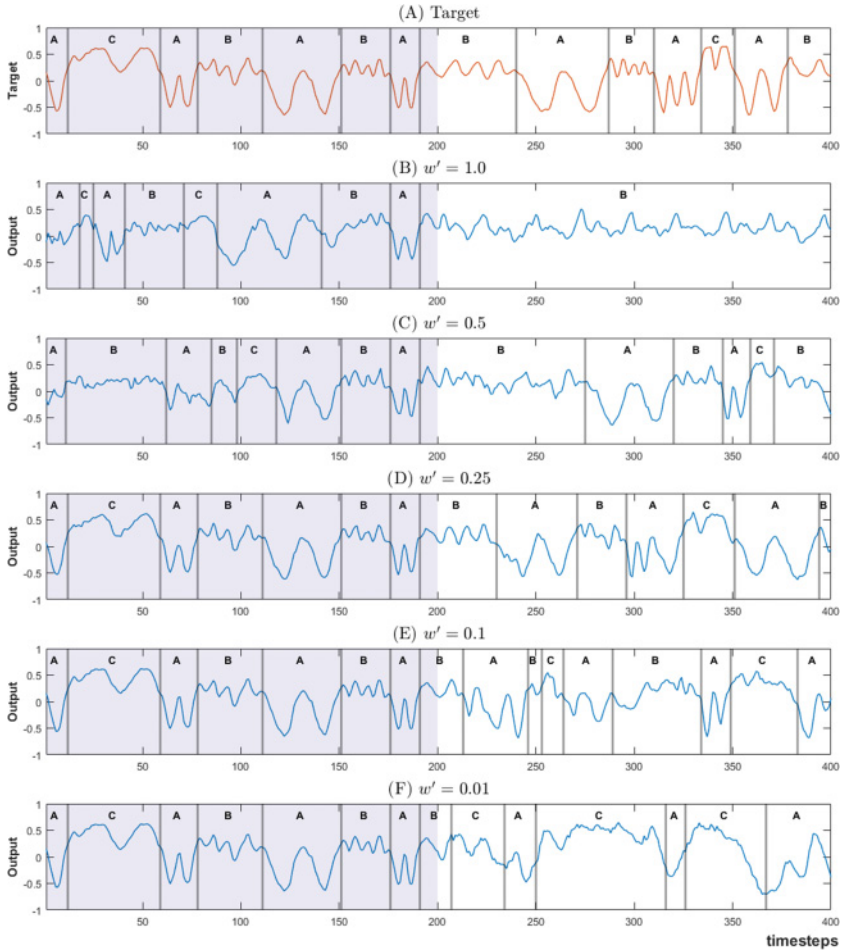

Figure 8: $w' = 0.25$ produces the most faithful qualitative reproduction of sequence of primitives in terms of both order and timing. It is the only one that reproduces the correct deterministic part of the target sequence $*, A, B, A, *, A, B$ (as well as producing possible stochastic steps, that is, not producing B on a stochastic step). In all these graphs, error regression is done once over the [1200] time window, for 1000 optimization steps (shaded area). Then the generative model produces the remaining 200 time steps, which are labeled to one of the three A, B, or C primitives, based on their similarity to them. While here, only the y dimension is displayed, the labeling was done on the 2D signal. Gray bars display perceived transition primitive patterns (and actual ones for the target pattern).

Table 4: Percent Accuracy for Primitive Prediction for Models Trained with Different Values for w.

	w'					
	1.0	0.5	0.25	0.15	0.1	0.01
1-primitive prediction (%)	81.25	90.63	**100**	93.75	90.63	81.25
2-primitive prediction (%)	62.50	78.13	**84.38**	68.75	59.38	37.50
3-primitive prediction (%)	40.63	53.13	**59.38**	37.50	21.88	9.38

Notes: Intermediate values of the meta-prior reproduce best the sequence of primitives. The table shows the percent accuracy for one, two, and three primitive prediction for models trained with different values for w'. The network with the best accuracy is shown in bold numbers.

One issue remains unclear in Figure 8: Why did the predictability worsen even though the reconstruction becomes better when w' goes from 0.25 to 0.01? To answer this, we compare the divergence between the posterior and the prior in the regression window of the $w' = 0.25$ and $w' = 0.01$ models. Figure 9 shows the target patterns, the reconstructed outputs, the MSE between the target patterns and reconstructed outputs, the mean of posterior $\boldsymbol{\mu}_t^{(q)}$, and the mean of prior $\boldsymbol{\mu}_t^{(p)}$ for the middle layer on the same test sequence. The reconstruction is more effective with w' set to 0.01, but the activities of the prior and posterior differ much more, as the activity of the mean of the middle layer indicates. Low values of the meta-prior lead to a low optimization pressure on the KL divergence term between the prior and posterior of the lower bound, and thus less pressure for the prior and posterior to coincide, and thus poor learning for the prior (the activity of the mean of the prior for $w' = 0.01$ is poor), leading to poor prediction capabilities. Instead the optimization pressure concentrates on the reconstruction error term, leading to a lower MSE for the reconstruction. This analysis strongly suggests that a good balance between minimizing the reconstruction error and minimizing the KL divergence between the posterior and the prior by setting the meta-prior in an intermediate range is the best way to ensure the best performance of the network across a range of tasks.

One question that may arise is how to find the optimal meta-prior at the beginning of the training. We do not have a good answer for that, as the optimal value depends on the task and the network topology. In the experiments of this article, we conducted a simple grid search; this method does not guarantee finding the optimal value and is time-consuming. We may employ optimization techniques such as evolutionary algorithms, or consider the meta-prior as one of the training parameters of the network, and optimize it through backpropagation. Our preliminary experiments with learning the meta-prior did not show any satisfactory results, and the network did not converge. We have left this issue as future work.

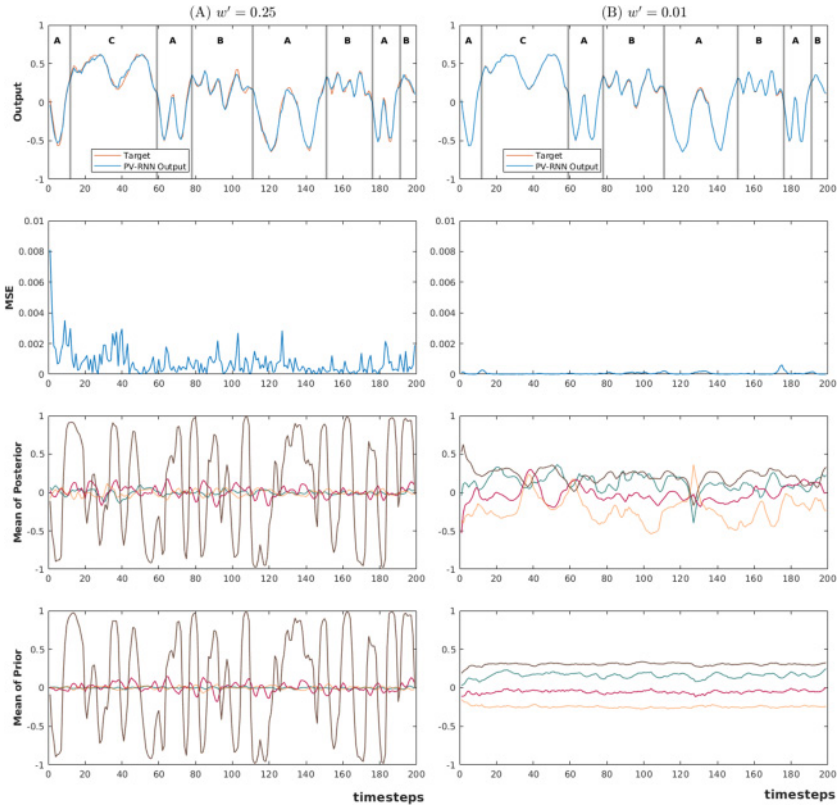

Figure 9: While the reconstruction is better for $w' = 0.01$ (lower MSE), the prior and posterior activity of the middle layer are significantly more different than for $w' = 0.25$, and the prior activity in particular seems to be constant. Here, we take a closer look at the activity of the $w' = 0.25$ and $w' = 0.01$ models presented in Figure 8. Here, we consider only the part happening during the time window [1200]. Only the y dimension of the 2D patterns is illustrated in the first row of the graphs; however, the MSE (second row) is computed for both dimensions. The two bottom rows show the mean $\boldsymbol{\mu}_t^{(q)}$ (posterior) and $\boldsymbol{\mu}_t^{(p)}$ (prior) of the middle layer.

4 Robot Experiment

As explained in the previous section, PV-RNN was able to deal with probabilistic patterns during simulation experiments. We also conducted a robotics experiment involving synchronous imitation between two robots. This experiment allows us to do several things at once. One is to provide a more realistic test case, in higher dimensions (12), with complex

perceptual noise source (e.g., hardware variations, motor noise, temperature, small synchronization discrepancies, and variations introduced when capturing training sequences from human movement). Another is to consider a context where actions need to be performed. The action space and sensory space are different in this experiment, leading us to adapt and apply the model in new ways. A final one is to compare PV-RNN with VRNN (Chung et al., 2015) and thus to contrast the effectiveness of error regression versus an autoencoder-based approach. For this, synchronous imitation is considered to be an ideal task because it involves predicting both future perceptual sequences to compensate possible perceptual delay and recognizing others intention via posterior inference.

4.1 Experimental Settings. We used two identical OP2 humanoid robots placed face-to-face. One robot was the demonstrator and the other the imitator. To generate the training data, 15 movement primitives of 200 time steps, different from one another, were designed. A human then executed each primitive on both robots at the same time, in mirror fashion, such that the movement of the left arm of one robot was executed on the right arm of the other (see Figure 10A). The imitator was then given training sequences composed of proprioception data—the joint angles of its own arms during movement, $\overline{\boldsymbol{X}}_t^{Pro}$—and exteroception data—the XYZ coordinate of the hand tip of the demonstrator robot, from the perspective of the imitator robot, $\overline{\boldsymbol{X}}_t^{Ext}$.

The testing data were generated by a human, creating a movement sequence by repeating a movement primitive a few times (seven cycles on average) and then switching at random to another primitive, so that all 15 primitives were used. The human strived to produce qualitatively the same movement, but not quantitatively: speed, amplitude and shape could differ. The resulting testing sequence is 4641 time steps long.

During testing, the demonstrator would play the prerecorded testing sequence, and the imitator robot would receive, at each time step t, the corresponding exteroception data $\overline{\boldsymbol{X}}_t^{Ext}$, but not, crucially, the $\overline{\boldsymbol{X}}_t^{Pro}$ proprioception data. The imitator would use the PV-RNN model to make predictions about both the exteroception and the proprioception data. The proprioception predictions would be sent to the PID controller to be executed on the imitator robot, while the exteroception predictions would be compared with the actual observed one, $\overline{\boldsymbol{X}}_t^{Ext}$, and the resulting error would be propagated through the network to perform error regression. We insist that the errors propagated through the network are only relative to $\overline{\boldsymbol{X}}_t^{Ext}$, as the target proprioception sequence is unavailable during testing. An important challenge in this setup is the switching, in the sequence, between different primitive patterns; the imitator must be able to recognize when they happen and update the internal state of the network, $\boldsymbol{A}^{\text{test}}$, appropriately (see Figure 10B). The same training and testing data were used on the same setup to train

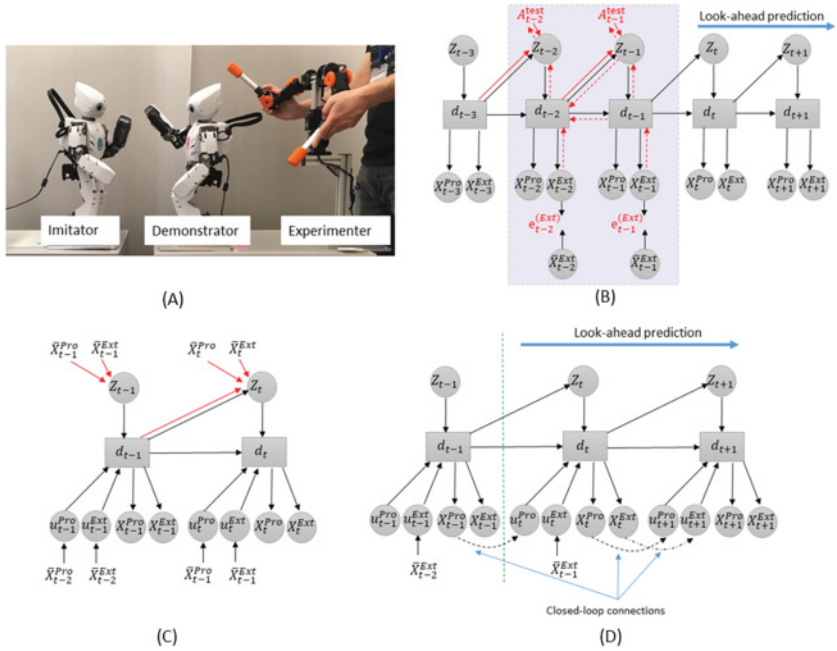

Figure 10: Robotic experiment of synchronous imitation. One OP2 robot imitates the other OP2 teleoperated by the experimenter (A), graph of computation and information flow using PV-RNN during testing (B), VRNN during training (C), and VRNN during testing (D). The red lines in panels B and C show the inference models of PV-RNN and VRNN, and the dashed lines in panel D show how the closed-loop generation is computed. The models in panels B–D are depicted with only one context layer for clarity. $\boldsymbol{u}_t^{Pro}$ and $\boldsymbol{u}_t^{Ext}$ represent the proprioception and exteroception input units at time step t that are fed by $\overline{\boldsymbol{X}}_{t-1}^{Pro}$ and $\overline{\boldsymbol{X}}_{t-1}^{Ext}$, respectively.

a VRNN model; Figures 10C and 10D show how VRNN was used in the training and testing, respectively.

To fairly compare the PV-RNN and VRNN, the same MTRNN-type network layer structure was used for both networks. The internal dynamic of the MTRNN used in VRNN was computed as in equation 2.3, but for the lowest layer, there is an added term $\boldsymbol{W}_{du}\boldsymbol{u}_t$ to feed the input $\boldsymbol{u}_t$ to the network. $\boldsymbol{u}_t$ is composed of $\boldsymbol{u}_t^{Pro}$ and $\boldsymbol{u}_t^{Ext}$. In that case, equation 2.3 becomes

$$\begin{aligned}\boldsymbol{h}_t^1 &= \left(1-\frac{1}{\tau_1}\right)\boldsymbol{h}_{t-1}^1 + \frac{1}{\tau_1}\left(\boldsymbol{W}_{dd}^{1,1}\boldsymbol{d}_{t-1}^1 + \boldsymbol{W}_{dz}^{1,1}\boldsymbol{Z}_t^1 + \boldsymbol{W}_{dd}^{1,2}\boldsymbol{d}_{t-1}^2 + \boldsymbol{W}_{du}\boldsymbol{u}_t\right)\\ \boldsymbol{d}_t^1 &= \tanh(\boldsymbol{h}_t^1). \end{aligned} \tag{4.1}$$

In variational Bayes autoencoder RNNs (Chung et al., 2015; Fraccaro et al., 2016; Goyal et al., 2017; Shabanian et al., 2017), the previous time step target $\overline{\boldsymbol{X}}_{t-1}$ is provided as the current input to predict $\boldsymbol{X}_t$ in the output. As can be seen in Figure 10C, during training, $\boldsymbol{u}_t^{Pro} = \overline{\boldsymbol{X}}_{t-1}^{Pro}$ and $\boldsymbol{u}_t^{Ext} = \overline{\boldsymbol{X}}_{t-1}^{Ext}$. The look-ahead predictions for multiple steps were conducted by closed-loop generation (Tani, 1996) wherein both prediction of the proprioception and exteroception at a particular future time step are obtained by feeding the prediction outputs of them at the previous time step into the inputs at this step. Formally, during testing, for one-step-ahead prediction, we retain $\boldsymbol{u}_t^{Ext} = \overline{\boldsymbol{X}}_{t-1}^{Ext}$, but the proprioception data come from the prediction of the network itself, such that $\boldsymbol{u}_t^{Pro} = \boldsymbol{X}_{t-1}^{Pro}$. For two-steps-or-more-ahead predictions, we do not have access to exteroception targets. Therefore, both the proprioception and exteroception data come from the prediction of the network itself, such that $\boldsymbol{u}_{t+1:T}^{Pro} = \boldsymbol{X}_{t:T-1}^{Pro}$ and $\boldsymbol{u}_{t+1:T}^{Ext} = \boldsymbol{X}_{t:T-1}^{Ext}$.

The approximate posterior of the VRNN is obtained as (Chung et al., 2015)

$$q_\phi(\boldsymbol{Z}_t \mid \boldsymbol{d}_{t-1}, \overline{\boldsymbol{X}}_t) = \mathcal{N}(\boldsymbol{Z}_t; \boldsymbol{\mu}_t^{(q)}, \boldsymbol{\sigma}_t^{(q)}) \quad \text{where}$$
$$[\boldsymbol{\mu}_t^{(q)}, \log \boldsymbol{\sigma}_t^{(q)}] = f_\phi^{(q)}(\boldsymbol{d}_{t-1}, \overline{\boldsymbol{X}}_t), \tag{4.2}$$

where $f^{(q)}$, $\overline{\boldsymbol{X}}$, and ϕ denote a one layer feedforward neural network, the target, and the posterior variables, respectively. This means that in the VRNN (during training), the current time step target (observation) is directly provided to the approximate posterior, whereas in the PV-RNN, not only the current time step target but also future ones are indirectly provided to the approximate posterior through BPTT. The prior computation of VRNN is the same as equation 2.4.

We used the same number of $\boldsymbol{d}$ and $\boldsymbol{Z}$ units for the VRNN and PV-RNN models. Each model had three context layers consisting of 120 $\boldsymbol{d}$ units and 12 $\boldsymbol{Z}$ units for the fast context (FC) layer, 60 $\boldsymbol{d}$ units and 6 $\boldsymbol{Z}$ units for the middle context (MC) layer, and 30 $\boldsymbol{d}$ units and 3 $\boldsymbol{Z}$ units for the slow context (SC) layer. Time constants of FC, MC, and SC units were set to 2, 10, and 50, respectively. In each model case, the training ran for 50,000 epochs.

For testing, we performed prediction from one to five steps ahead. The motor controllers of the robot receive the last proprioception predictions as control targets: if we do one-step-ahead predictions, the motor controllers will receive $\boldsymbol{X}_t^{Pro}$ at time t. If we do five-step-ahead predictions, they will receive $\boldsymbol{X}_{t+4}^{Pro}$ at time t. This would allow correction for perceptual and processing delay. For error regression in PV-RNN, the size of the time window m was set to 10, and 100 regression steps were performed at each time step. Not providing VRNN with any proprioception information led to poor performance. To remedy this, we provided VRNN with 20 steps of proprioception information at the beginning of the test sequence. PV-RNN does not have access to any proprioception information.

Table 5: MSE between the Target and Look-Ahead Prediction.

	Number of Prediction Steps				
	1 Step	2 Steps	3 Steps	4 Steps	5 Steps
PV-RNN	0.00254	0.00259	0.00339	0.00395	0.00473
VRNN	0.00641	0.00702	0.00779	0.00852	0.00943

Notes: PV-RNN outperforms VRNN regardless of the number of prediction steps. The table shows the MSE between the target and look-ahead prediction with different time steps ahead for PV-RNN and VRNN. The network with minimum MSE is shown in bold numbers.

Synchronized imitation with PV-RNN could not be done online because the error regression with 100 optimization steps costs about twice the computational time (205 ms) than real time would allow. Therefore, in the case of using PV-RNN, the test was conducted on prerecorded data sequence of exteroception. Although the VRNN case could be performed in real time, we also used the prerecorded target sequences to ensure the fairest comparison.

4.2 Experimental Results. For PV-RNN and VRNN, we compared the performance for different values of the meta-prior w: 1.0, 0.5, 0.25, 0.2, and 0.1. We present here the results for the best value of w for each model, 0.5 for PV-RNN and 0.25 for VRNN.

The mean square error (MSE) between the target (exteroception and proprioception targets) and the predicted outputs (exteroception and proprioception outputs) with different look-ahead step length is shown for both cases of using PV-RNN and VRNN in Table 5.

In both PV-RNN and VRNN, the error increases as the model predicts more steps ahead, as is expected. PV-RNN consistently outperforms VRNN, showing the effectiveness of error regression for prediction performance.

We recorded two videos—one-step-(Video 1) and five-steps-ahead (Video 2) predictions—of the movement patterns of three robots. One robot (middle) was the demonstrator robot, and the others were the imitator robots: one controlled by the PV-RNN (left) and one by the VRNN (right). PV-RNN convincingly outperforms VRNN in the video of the one-step-ahead prediction. The PV-RNN robot synchronously imitates the target robot, whereas the VRNN does not always perform a smooth, or correct, imitation. One-step-ahead prediction (50 ms) seems to be enough to overcome the perceptual delay in this setting, as delay is difficult to observe between demonstrator and imitator. In the five-step-ahead prediction, PV-RNN still shows better prediction performance than VRNN. However, it also fails to imitate the target robot for several movements and exhibits brusk changes of speed, accelerating and slowing down around the target pattern.

Looking at the video of a successful imitation frame by frame, one can see that the imitator robot movements seem to be ahead of the target one.

5 Discussion

This article examines how uncertainty or probabilistic structure hidden in observed temporal patterns can be captured in an RNN through learning. To that end, it proposes a novel, predictive-coding-inspired variational Bayes RNN. Our model possesses three main features that distinguish it from the existing variational Bayes RNN. The first is the use of a weighting parameter, the meta-prior, between the two terms of the lower bound to control the optimization pressure. The second is propagating errors through backpropagation instead of propagating inputs during the forward computation. And the third is the error regression procedure during testing, performing online optimization of the internal state of the network during prediction.

The idea of weighting the KL divergence term of the lower bound has been employed before, most notably in KL annealing (Bowman et al., 2015). However in this article, it is used for a different purpose. Through two experiments, the first one involving a finite state machine and the second one continuous temporal patterns composed of probabilistic transitions between a set of hand-generated movement, we showed that by changing the value of the meta-prior, we could achieve either a deterministic or a random process behavior. The deterministic behavior reconstructed training sequences well and imitated the stochasticity of the data through deterministic chaos but could not generalize to unseen testing sequences. The random process behavior could neither reconstruct training sequences nor generalize to unseen ones; its behavior was dominated by noise. The best value of the meta-prior could be found between those two extremes, where we showed that the network displayed both good reconstruction and generalization capability. It can be summarized that although probabilistic temporal patterns can be imitated by either deterministic chaos or stochastic process, as suggested by the ergodic theory of chaos (Crutchfield, 1992), the best representation can be developed by mixing the deterministic and stochastic dynamics via iterative learning of the proposed RNN model.

We employed the idea of propagating errors instead of propagating inputs to address two important issues in variational Bayes RNNs: how to provide the future dependency information to the latent states and how to avoid ignoring the latent states during learning. We addressed both issues by introducing an adaptive vector $\boldsymbol{A}^{\overline{X}}$ in the inference model. $\boldsymbol{A}^{\overline{X}}$ is optimized through BPTT and captures the future dependencies of the external observation. This was verified in both simulated experiments, as providing the first time step $\boldsymbol{A}_1^{\overline{X}}$ was sufficient to reconstruct the training sequences. Furthermore, because information from $\boldsymbol{A}^{\overline{X}}$ flows to $\boldsymbol{Z}$ and the information

from $\boldsymbol{Z}$ flows through the deterministic states $\boldsymbol{d}$, which are ultimately responsible for the output sequences, the network is forced to construct good representations in the latent state $\boldsymbol{Z}$. This phenomenon was shown in the first experiment when looking at the activity of mean $\boldsymbol{\mu}^{(p)}$ and variance $\boldsymbol{\sigma}^{(p)}$ of $\boldsymbol{Z}$.

Finally, we used an error regression procedure during testing for making predictions about unseen sequences. In the second experiment, the PV-RNN, based on a network architecture with multiple layers, was evaluated for look-ahead prediction in primitive sequences. We found that relatively long sequences of primitive transitions were successfully predicted despite some discrepancies when w was set to an adequate intermediate value. This, however, required a window of a sufficient length (200 steps) and a high number of iterations for error regression (1000 iterations). This suggests that a good balance between minimizing reconstruction error and divergence between the prior and posterior distributions can result in accurate prediction of future primitive sequences.

Furthermore, the error regression procedure we employed bears a lot of similarities to the predictive coding principle. In particular, it shares the same processing cycle: making predictions, propagating prediction errors through the network hierarchy, and updating its internal state online to improve future predictions. Some important differences exist with predictive-coding implementations closer to the neurobiology of the brain (Rao & Sejnowski, 2000); in our model, errors are propagated globally rather than locally. This is deliberate, to take advantage of the canonical tools of variational Bayes RNNs.

Other autoencoder-based variational Bayes RNNs infer the latent variable at each time step through a recurrent mapping of the hidden state of the previous step, fed with inputs with the current time step (Fabius & van Amersfoort, 2014; Bayer & Osendorfer, 2014; Chung et al., 2015). In the robotic experiment for an imitation learning task, we showed that our model outperforms VRNN (Chung et al., 2015). This demonstrated the effectiveness of the error regression.

The learning vector $\boldsymbol{A}^{\overline{X}}$ addressed two issues but introduces another. Indeed, as described in section 2.2, the dimension of $\boldsymbol{A}^{\overline{X}}$ increases linearly with the length and the number of training samples. It seems to preclude working with large data sets as a naive implementation might exceed any reasonable available memory. However, with each vector $\boldsymbol{A}^{\overline{X}}$ corresponding to a specific training pattern, it can be dynamically loaded and unloaded into memory whenever needed during training, resulting in a memory requirement equal to the one of the largest training batch. Furthermore, the trained values of the $\boldsymbol{A}^{\overline{X}}$ vectors are not needed for predicting unseen testing sequences and can be discarded entirely. If one wishes to perform reconstruction of the training patterns, only the first time step $\boldsymbol{A}_1^{\overline{X}}$ is needed. Finally, although we did not do it in this article, $\boldsymbol{A}^{\overline{X}}$ vectors may not be

needed for every time step and we could consider having $A^{\overline{X}}$ vectors every 10 steps, for instance. The implications are not necessarily trivial, and this is a subject of ongoing study.

While the memory requirement of the model may not be a fundamental problem, perhaps a more serious issue lies in the computational requirement of the error regression process. Indeed, compared to most models that only need forward computations for evaluation, our model still needs to backpropagate and perform optimization. This can severely hamper its ability to be deployed on a variety of platforms. It can also be an issue for real-time robotics, as was the case in our robotic experiment. We are currently investigating ways to reduce the computational burden of error regression.

An intriguing consideration is that the current results showing that the generalization capability of PV-RNN depend on the setting of the meta-prior w bears parallels to observational data about autism spectrum disorders (ASD) and may suggest possible accounts of its underlying mechanisms. ASD is a wide-ranging pathology including deficits in communication, abnormal social interactions, and restrictive or repetitive interests and behaviors (DiCicco-Bloom & Crabtree, 2006). Recently, there has been an emerging view suggesting that deficits in low-level sensory processing may cascade into higher-order cognitive competency, such as in language and communication (Stevenson et al., 2014; Lawson, Rees, & Friston, 2014; Robertson & Baron-Cohen, 2017. Van de Cruys et al. (2014) have suggested that ASD might be caused by overly strong top-down prior potentiation to minimize prediction errors (thus increasing precision) in perception, which can enhance capacities for rote learning while resulting in the loss of the capacity to generalize what is learned, a common ASD symptom.

This account by Van de Cruys et al. (2014) corresponds to some extent to the situation of PV-RNN when learning with w set to a larger value. PV-RNN is able to exactly reconstruct complex training sequences by embedding them in deterministic dynamics, while being unable to generalize to unseen sequences. With a larger value of w, the optimization pressure on the KL divergence term, and therefore on the prior to be similar to the posterior, is stronger. This pushes the network to reduce uncertainty in the network (low $\boldsymbol{\sigma}$), increasing the precision of the predictions, resulting in a stronger top-down prior. Such a phenomenon could explain how ASD patients in social contexts might frequently suffer from overamplified error in predicting behaviors of others; this results from overestimated precision in prediction due to overfitting in learning. For this reason, such patients may tend to indulge in their own repetitive behaviors that generate a tolerable amount of error. Then, mechanisms akin to the inability of adapting w adequately may result in the pathology. Lawson et al. (2014) propose that maladaptation of precision itself in hierarchical message passing in the brains may contribute to many features of autistic perception in a study using a hierarchical Bayesian model built on the predictive coding framework. If a neural mechanism indeed exists that corresponds to adapting w,

study about the precise feedback mechanisms to regulate w within an optimal range could be an important research question for ASD. It could also shed light on how the brain handles various cognitive tasks using statistical inference.

Various robotics applications would be interesting for future study. One of the main features of the proposed model is that it can learn the latent probabilistic structure of data; this should translate into high performance at learning skilled behaviors through supervised teaching, such as manual manipulation of objects. Indeed, a crucial component in generating skilled behavior is that precision in movement control must change depending on the situation during task execution. For example, when an individual is grasping an object, the precision during reaching can be low, but it must become much higher at the moment of contact with the object to establish a good grasp. It is highly likely that PV-RNN can learn to extract such statistical structure by inferring the necessary precision in generating movement trajectories from the set of training trajectories.

Another interesting direction for future study would be the introduction of goal-directed planning mechanisms into the current model. Arie, Endo, Arakaki, Sugano, and Tani (2009) showed that the deterministic MTRNN model can generate goal-directed plans. It does that by starting from a desired goal state for a future time step and backpropagating the error to infer the necessary context state of the current time step to achieve this goal. From this inferred initial context state, a proprioception sequence (joint angles of the robot) can be predicted and used to directly specify what actions the robot should undertake. Furthermore, Butz, Bilkey, Humaidan, Knott, and Otte (2019) recently proposed, a retrospective and prospective inference schEme (REPRISE), which can infer both retrospectively for past contextual event states and prospectively for future optimal motor command sequences satisfying some given goal states. REPRISE is built on their previous work, which included the prospective phase using an RNN (Otte, Zwiener, & Butz, 2017). In the new proposed model, an RNN is augmented with contextual neurons in order to encode continuous sensorimotor dynamics into sequences of discrete events. The contextual neurons are adapted during the retrospective phase in order to minimize the loss between predicted and actual sensory information. Then the motor commands in the future time steps are adapted via BPTT during the prospective phase in order to minimize the discrepancies between predicted future states and desired goal states. Such a mechanism could also be realized in PV-RNN by inferring the optimal adaptive vector $\boldsymbol{A}$ sequence accounting for both past sensory experience and the specified future goal states. Future study should examine how the extended PV-RNN can perform goal-directed planning tasks, including online replanning ones, compared to existing models such as REPRISE Butz et al. (2019).

The robotic experiment presented in this article was limited in many ways. We are now working on a two-robot setup where, rather than one

demonstrator and one imitator, there are two imitators imitating each other. In the context of active inference (Friston, Daunizeau, & Kiebel, 2009; Friston, Daunizeau, Kilner, & Kiebel, 2010; Pezzulo, Rigoli, & Friston, 2015; Baltieri & Buckley, 2017), an agent interacting with an environment has two choices when its predictions do not agree with its observations: either modify its internal state to produce predictions that better align with observations, or perform an adequate intervention in the environment to make observations better correspond to the predictions. In other words, when the world does not fit our expectations, we can change our expectations or change the world by acting adequately on it. In a context where a demonstrator performs for an imitator, the imitator has no choice but to change its internal state when prediction errors occur. But in a situation with two imitators, robot A may learn how robot B responds to its own actions, especially when robot A's actions generate prediction errors for robot B. If robot B is only updating its internal state, robot A might end up continuously performing interventions on robot B's behavior rather than changing its own internal state. The most interesting case should happen when both robots are able to learn to predict the consequences of their own actions on the other robots *and* perform interventions to influence one another's actions. The circular causality developed between those two may lead to ambiguity in determining which one drives the other, which one demonstrates, and which one imitates, with possibly continuous switching between those roles. The early results we obtained on such a setup are encouraging. Studies examining such aspects could greatly contribute to understanding the underlying mechanisms of social cognition and how to engineer the autonomous development of collaborative actions among multiple agents.

6 Conclusion

We proposed a predictive-coding-inspired variational Bayes RNN to capture the stochasticity of time-series data. To that end, the cost function of the network is composed of two terms: the expected prediction error and the KL divergence (measuring how similar two distributions are) between the posterior and prior distributions. The relative importance of those two terms is weighted by a parameter w, the meta-prior.

First, in a simple task, we demonstrated that increasing the value of the meta-prior w, and therefore the optimization pressure on the KL divergence term in the cost function, led to the model's behaving increasingly deterministically, leading to development of deterministic chaos, with low generalization capabilities. Conversely, lowering the value of w led to a network's behaving increasingly stochastically. Stochastic models are better at generalization, but at the extreme, they turn into random generators that disregard the structure of the input data.

The best behavior is found when the value of w is between those two extremes: the network was able to achieve the best generalization capability

with an intermediate value of the meta-prior. We confirmed this observation on more complex tasks using both hand-drawn patterns and robotic motions and on a more complex model using a higher number of context layers.

Our approach provides interesting solutions to two issues that variational Bayes RNNs typically have: latent variables are ignored during training, and they do not have access to information about the future dependencies of the data. Our network solves those two issues by avoiding feeding inputs to the network during the forward computation, preferring to propagate prediction errors during BPTT to dedicated latent variables.

These variables are leveraged when predicting unseen testing sequences: we use an error regression procedure during evaluation, which performs online optimization of the internal state of the network based on observed prediction errors. Therefore, the predictions are constantly reevaluated as new external data become observable. We have shown that our model outperforms the VRNN model (Chung et al., 2015) on a robotic imitation task.

Appendix A: Posterior Computation

$$q_\phi(\boldsymbol{Z}_t \mid \boldsymbol{d}_{t-1}, \boldsymbol{e}_{t:T}) = \mathcal{N}(\boldsymbol{Z}_t; \boldsymbol{\mu}_t^{(q)}, \boldsymbol{\sigma}_t^{(q)})$$
$$\text{where} \quad [\boldsymbol{\mu}_t^{(q)}, \log \boldsymbol{\sigma}_t^{(q)}] = f^{(q)}(\boldsymbol{d}_{t-1}, \boldsymbol{A}_t^{\overline{\boldsymbol{X}}}) \tag{A.1}$$

The equation for $\boldsymbol{\mu}$ and $\log \boldsymbol{\sigma}$ of posterior can be written as

$$\begin{cases} \boldsymbol{\mu}_t^k = \tanh\left(\boldsymbol{W}_{\mu d}^{kk} \tilde{\boldsymbol{d}}_{t-1}^k + \boldsymbol{A}_{\mu,t}^{\overline{\boldsymbol{X}},k}\right) \\ \log \boldsymbol{\sigma}_t^k = \boldsymbol{W}_{\sigma d}^{kk} \tilde{\boldsymbol{d}}_{t-1}^k + \boldsymbol{A}_{\sigma,t}^{\overline{\boldsymbol{X}},k} \end{cases}, \tag{A.2}$$

where $\boldsymbol{\mu}_t^k$ is the vector of the mean values of the kth context layer at time t, $\boldsymbol{W}_{\mu d}^{kk}$ is the matrix of the connectivity weights from the $\boldsymbol{d}$ units in the kth context layer to $\boldsymbol{\mu}$ units in the same context layer, and $\boldsymbol{W}_{\sigma d}^{kk}$ is the matrix of the connectivity weights from the $\boldsymbol{d}$ units in the kth context layer to the $\boldsymbol{\sigma}$ units in the same context layer. The notation (q) was omitted from the equation for simplicity. $\boldsymbol{A}_{\mu,t}^{\overline{\boldsymbol{X}},k}$ and $\boldsymbol{A}_{\sigma,t}^{\overline{\boldsymbol{X}},k}$ are obtained as follows,

$$\begin{cases} \boldsymbol{A}_{\mu,t}^{\overline{\boldsymbol{X}},k} = \boldsymbol{A}_{\mu,t}^{\overline{\boldsymbol{X}},k} + \alpha \frac{\partial L}{\partial \boldsymbol{A}_{\mu,t}^{\overline{\boldsymbol{X}},k}} \\ \boldsymbol{A}_{\sigma,t}^{\overline{\boldsymbol{X}},k} = \boldsymbol{A}_{\sigma,t}^{\overline{\boldsymbol{X}},k} + \alpha \frac{\partial L}{\partial \boldsymbol{A}_{\sigma,t}^{\overline{\boldsymbol{X}},k}} \end{cases}, \tag{A.3}$$

where α is the learning rate. Based on equation A.2, we can rewrite equation A.3 to have the derivatives with respect to mean and standard deviation values as

$$\begin{cases} A_{\mu,t}^{\overline{X},k} = A_{\mu,t}^{\overline{X},k} + \alpha \left(1 - \tanh^2 \left(W_{\mu d}^{kk} \tilde{\boldsymbol{d}}_{t-1}^k + A_{\mu,t}^{\overline{X},k}\right)\right) \left(\dfrac{\partial L}{\partial \boldsymbol{\mu}_t^k}\right) \\ A_{\sigma,t}^{\overline{X},k} = A_{\sigma,t}^{\overline{X},k} + \alpha \dfrac{\partial L}{\partial \log \boldsymbol{\sigma}_t^k} \end{cases}. \tag{A.4}$$

Appendix B: KL Divergence

We let each posterior and prior distribution be a gaussian with a diagonal covariance matrix, so:

$$KL[q_\phi(\boldsymbol{Z}_t) \parallel P_{\theta_Z}(\boldsymbol{Z}_t)] = E_{q_\phi}[\log q_\phi(\boldsymbol{Z}_t)] - E_{q_\phi}[\log P_{\theta_Z}(\boldsymbol{Z}_t)], \tag{B.1}$$

$$q_\phi(\boldsymbol{Z}_t) = \frac{1}{\sqrt{2\pi(\boldsymbol{\sigma}_t^q)^2}} e^{\frac{-(\boldsymbol{Z}_t - \boldsymbol{\mu}_t^q)^2}{2(\boldsymbol{\sigma}_t^q)^2}}, \tag{B.2}$$

$$P_{\theta_Z}(\boldsymbol{Z}_t) = \frac{1}{\sqrt{2\pi(\boldsymbol{\sigma}_t^p)^2}} e^{\frac{-(\boldsymbol{Z}_t - \boldsymbol{\mu}_t^p)^2}{2(\boldsymbol{\sigma}_t^p)^2}}. \tag{B.3}$$

For simplicity, we removed parentheses from p and q. Based on equations B.1 to B.3,

$$E_{q_\phi}[\log q_\phi(\boldsymbol{Z}_t)] = E_{q_\phi}\left[-\frac{1}{2}\log 2\pi - \frac{1}{2}\log(\boldsymbol{\sigma}_t^q)^2 + \frac{-(\boldsymbol{Z}_t - \boldsymbol{\mu}_t^q)^2}{2(\boldsymbol{\sigma}_t^q)^2}\right], \tag{B.4}$$

$$\begin{aligned} E_{q_\phi}[\log P_{\theta_Z}(\boldsymbol{Z}_t)] &= -\frac{1}{2} E_{q_\phi}\left[\log 2\pi + \log(\boldsymbol{\sigma}_t^p)^2 + \frac{(\boldsymbol{Z}_t - \boldsymbol{\mu}_t^p)^2}{(\boldsymbol{\sigma}_t^p)^2}\right], \\ &= -\frac{1}{2} E_{q_\phi}\left[\log 2\pi + \log(\boldsymbol{\sigma}_t^p)^2 + \frac{(\boldsymbol{Z}_t)^2 + 2\boldsymbol{Z}_t\boldsymbol{\mu}_t^p - (\boldsymbol{\mu}_t^p)^2}{(\boldsymbol{\sigma}_t^p)^2}\right]. \end{aligned} \tag{B.5}$$

Variance and $E[\boldsymbol{Z}_t^2]$ can be written as

$$\boldsymbol{\sigma}_t^2 = E[(\boldsymbol{Z}_t - \boldsymbol{\mu}_t)^2], \quad E[\boldsymbol{Z}_t^2] = \boldsymbol{\mu}_t^2 + \boldsymbol{\sigma}_t^2, \tag{B.6}$$

so equations B.4 and B.5 can be rewritten as

$$\begin{aligned} E_{q_\phi}[\log q_\phi(\boldsymbol{Z}_t)] &= -\frac{1}{2}\left(\log 2\pi + \log(\boldsymbol{\sigma}_t^q)^2 + \frac{(\boldsymbol{\sigma}_t^q)^2}{(\boldsymbol{\sigma}_t^q)^2}\right) \\ &= -\frac{1}{2}\left(\log 2\pi + \log(\boldsymbol{\sigma}_t^q)^2 + 1\right), \end{aligned} \tag{B.7}$$

$$E_{q_\phi}[\log P_{\theta_Z}(\boldsymbol{Z}_t)] = -\frac{1}{2}\Bigg(\log 2\pi + \log (\boldsymbol{\sigma}_t^p)^2 + \frac{(\boldsymbol{\mu}_t^q)^2 + (\boldsymbol{\sigma}_t^q)^2 - 2\boldsymbol{\mu}_t^q \boldsymbol{\mu}_t^p + (\boldsymbol{\mu}_t^p)^2}{(\boldsymbol{\sigma}_t^p)^2} \Bigg). \tag{B.8}$$

Now, equation B.1 can be rewritten as

$$\begin{aligned} KL[q_\phi(\boldsymbol{Z}_t) \parallel P_{\theta_Z}(\boldsymbol{Z}_t)] &= -\frac{1}{2}\Bigg(\log (\boldsymbol{\sigma}_t^q)^2 + 1 - \log (\boldsymbol{\sigma}_t^p)^2 \\ &\quad - \frac{(\boldsymbol{\mu}_t^q)^2 + (\boldsymbol{\sigma}_t^q)^2 - 2\boldsymbol{\mu}_t^q \boldsymbol{\mu}_t^p + (\boldsymbol{\mu}_t^p)^2}{(\boldsymbol{\sigma}_t^p)^2} \Bigg) \\ &= \log \frac{\boldsymbol{\sigma}_t^{(p)}}{\boldsymbol{\sigma}_t^{(q)}} + \frac{(\boldsymbol{\mu}_t^{(p)} - \boldsymbol{\mu}_t^{(q)})^2 + (\boldsymbol{\sigma}_t^{(q)})^2}{2(\boldsymbol{\sigma}_t^{(p)})^2} - \frac{1}{2}. \end{aligned} \tag{B.9}$$

Appendix C: Lyapunov Exponent Computation

For simplicity, let us consider a PV-RNN consisting of 2 $\boldsymbol{d}$ units and 1 $\boldsymbol{Z}$ unit. We need to first compute Jacobian matrices at each time step as

$$J_t = \begin{bmatrix} \frac{\partial \boldsymbol{Z}_{t+1,1}}{\partial \boldsymbol{Z}_{t,1}} & \frac{\partial \boldsymbol{Z}_{t+1,1}}{\partial \boldsymbol{d}_{t,1}} & \frac{\partial \boldsymbol{Z}_{t+1,1}}{\partial \boldsymbol{d}_{t,2}} \\ \frac{\partial \boldsymbol{d}_{t+1,1}}{\partial \boldsymbol{Z}_{t,1}} & \frac{\partial \boldsymbol{d}_{t+1,1}}{\partial \boldsymbol{d}_{t,1}} & \frac{\partial \boldsymbol{d}_{t+1,1}}{\partial \boldsymbol{d}_{t,2}} \\ \frac{\partial \boldsymbol{d}_{t+1,2}}{\partial \boldsymbol{Z}_{t,1}} & \frac{\partial \boldsymbol{d}_{t+1,2}}{\partial \boldsymbol{d}_{t,1}} & \frac{\partial \boldsymbol{d}_{t+1,2}}{\partial \boldsymbol{d}_{t,2}} \end{bmatrix}.$$

It can be noted that $\boldsymbol{X}$ does not exist in the Jacobian matrices because in the generative model, $\boldsymbol{X}_{1:T}$ are not given to the context layers. We can now resort to the approximation of the image ellipsoid $J_T J_{T-1} \dots J_1 U$ of the unit sphere by a computational algorithm. (More details can be seen in Alligood et al., 1996.) Here, the computation of the first-largest Lyapunov exponent is given. Let us start with an orthonormal basis $r = [1.0 \;\; 0.0 \;\; 0.0]^T$, and use the Gram-Schmidt orthogonalization procedure, so we have algorithm 1, where $\|.\|$ and LE denote Euclidean length and the first largest Lyapunov exponent, respectively. T was 50,000 in our experiments.

Appendix D: Experiment 1 with a Simple RNN

We conducted experiment 1 again using the same data sets and network parameter settings with the exception of the time constant. We set the time

Algorithm 1: Lyapunov Exponent Computation.

$LE = 0.0$

for <t = 1 to T> **do**

$y_t = J_t\, r$

$r = \frac{y_t}{\|\mathbf{y_t}\|}$

$LE \mathrel{+}= \log \|\mathbf{y_t}\|$

end for

$LE = \frac{LE}{T}$

Table 6: Evaluation of the Regeneration and Generalization Capabilities of the PV-RNN Model Trained with Different Values for w.

	Meta-Prior w						
	0.1	0.05	0.025	0.015	0.01	0.001	0.0001
Average diverging step (ADS)	**23**	19	15	12	11	8	7
KL divergence of test phase	6.8589	1.6515	**0.1106**	0.1306	0.359	0.8758	2.8845

Notes: The average diverging step (ADS) points to better reconstruction performance when w is high. However, taking into account the KL divergence between the probabilistic distribution of the generated pattern $P(\boldsymbol{X}_{t:t+11})$ and the one of the training data $P(\overline{\boldsymbol{X}}_{t:t+11})$ paints another picture: the network best captures the probabilistic structure of the data for an average value of w. The bold number in the first row shows the model with the best regeneration capability. The bold numbers in the second row shows the model with best generalization capability.

constant in equation 2.3 to 1.0, which removes the leaky integrator and the ***d*** unit becomes a simple RNN. ADS and KL divergence of the test phase for different values of w are shown in Table 6. The results are in line with the ones shown in section 3.1 where the MTRNN with time constant 2.0 was used.

Appendix E: Experiment 2 Data Set

The data set for experiment 2 data sequences was generated in three stages as follows.

First, three different 2D movement primitive patterns and a PFSM of defining the transition probability among those patterns were prepared. Then a human subject was asked to draw patterns in 2D using a tablet

device by sequentially concatenating the primitive patterns by following the transition probability defined in the PFSM of Figure 2B. Each drawing of hand-drawing primitive patterns necessarily contains fluctuations in amplitude, velocity, and shape. Primitive patterns A, B, C were circles, a rotated figure eight, and triangles similar to those depicted in the first row of Figure 7. Each of them is a cyclic pattern with periodicity 2. The PFSM adopted in this experiment is shown in Figure 2B. The total number of A, B, C primitive patterns generated was 160 (4458 time steps). The branching after s_4 by generating either a primitive B or C was randomly chosen by the human. We measured the conditional probabilities in the data after the generation, and they were $P(B|ABA) = 0.275\%$ and $P(C|ABA) = 0.725\%$.

Next, a target generator was built using the human-generated data for the purpose of producing training and testing patterns used for evaluating the PV-RNN. An MTRNN was used as the target generator by training it with using the human-generated data as the teaching target sequences. After the training, the closed-loop operation of the MTRNN (feeding next step inputs with current step prediction outputs) generated sample sequence patterns while adding gaussian noise with zero mean and constant $\boldsymbol{\sigma}$ of 0.05 to the internal state of each context unit at each time step. This makes the outputs of the network stochastic while preserving the probabilistic structure, not necessarily exactly the same as the one in the training patterns prepared. More details and implementations of this target generator MTRNN can be seen in Ahmadi and Tani (2017a). Due to the noise inserted into the internal dynamics of the MTRNN, the output patterns were noisier and fluctuated more than the human-generated patterns, and those patterns could have different numbers of cycles than two. Finally, three groups of patterns were sampled from the MTRNN-generated output patterns, one consisting of 16 sequence patterns, each with a 400 step-length for the training of the PV-RNN, another comprising 1 sequence patterns with a 6400-step length for the first test phase of the PV-RNN, and the last one consisting of 32 sequence patterns, each with a 400-step length for the second test phase of the PV-RNN. The main reason that the target generator was used instead of using human-generated trajectory data was that significantly more target data were used (up to 128 sequences) while designing the model, more than could be reasonably created using human generation. The target generator, MTRNN, can effortlessly generate as many instances of patterns as one needs.

Appendix F: Experiment 2 with Two Context Layers

Two PV-RNN models were trained with w' set to 0.25. The first model had two context layers consisting of 80 $\boldsymbol{d}$ units and 8 $\boldsymbol{Z}$ units for the fast context (FC) layer and 40 $\boldsymbol{d}$ units and 4 $\boldsymbol{Z}$ units for the slow context (SC) layer. The time constants of FC and SC units were set to 2 and 4, respectively. The

Table 7: MSE between the Unseen Test Targets and the One-Step- to Five-Steps-Ahead Generated Predictions for Models with Two Context Layers.

	Number of Prediction Steps				
	1-Step Prediction	2-Steps Prediction	3-Steps Prediction	4-Steps Prediction	5-Steps Prediction
First model	0.00439	0.00912	0.014	0.0183	0.02236
Second model	0.0041	0.00865	0.013	0.0178	0.0223

Table 8: Accuracy for One, Two, and Three Primitive Prediction for Models with Two Context Layers.

	Number of Prediction Primitives		
	One	Two	Three
First model	90.62%	75%	46.87%
Second model	100	81.25	50

second model had two context layers consisting of 90 $\boldsymbol{d}$ units and 9 $\boldsymbol{Z}$ units for the fast context (FC) layer and 50 $\boldsymbol{d}$ units and 5 $\boldsymbol{Z}$ units for the slow context (SC) layer. The time constants of FC and SC units were set as in the first model. Training ran for 300,000 epochs in each case. The first model is equivalent to removing the slow context layer from the PV-RNN model of section 3.2 with w' set to 0.25. Moreover, the summations of $\boldsymbol{d}$ units and $\boldsymbol{Z}$ units in the second model and the PV-RNN model of section 3.2 with w' set to 0.25 are equal, although the second model has 3640 more learnable weights.

The prediction performance using error regression was evaluated for both models as in section 3.2. Tables 7 and 8 show the error regression results of the one-step- to five-steps-ahead predictions and the one-primitive to three-primitives predictions, respectively. Both networks show similar results to the network with three context layers (see Table 3) for predicting each time step. However, it can be seen by comparing Table 4 with Table 8 that the network with three context layers outperforms the networks with two context layers for predicting correct patterns.

Appendix G: Experiment 2 with an Alternate Inference Model

We examined how providing the past $\boldsymbol{d}_{t-1}$ to the inference model could be beneficial by deleting $\boldsymbol{d}_{t-1}$ information. The new approximate posterior is

Table 9: MSE between the Unseen Test Targets and the One-Step- to Five-Steps-Ahead Generated Predictions for an Inference Model without $\boldsymbol{d}_{t-1}$ Information.

	Number of Prediction Steps				
	1-Step Prediction	2-Steps Prediction	3-Steps Prediction	4-Steps Prediction	5-Steps Prediction
Alternate inference model	0.0105	0.02116	0.02977	0.03414	0.03911

obtained as

$$q_\phi(\boldsymbol{Z}_t \mid \boldsymbol{e}_{t:T}) = \mathcal{N}(\boldsymbol{Z}_t; \boldsymbol{\mu}_t^{(q)}, \boldsymbol{\sigma}_t^{(q)}) \quad \text{where} \quad [\boldsymbol{\mu}_t^{(q)}, \log \boldsymbol{\sigma}_t^{(q)}] = f_\phi^{(q)}(A_t^{\overline{X}}) \quad \text{(G.1)}$$

A PV-RNN model was trained with w' set to 0.25. Other network parameters were exactly the same as the PV-RNN models in section 3.2. Table 9 shows the error regression results of the one-step- to five-steps-ahead predictions. By comparing these results with the error regression results of the PV-RNN with w' set to 0.25 shown in Table 3, it can be seen that the model presented here significantly underperforms (it is almost twice as bad) the model with $\boldsymbol{d}_{t-1}$ information.

Appendix H: Additional Figures

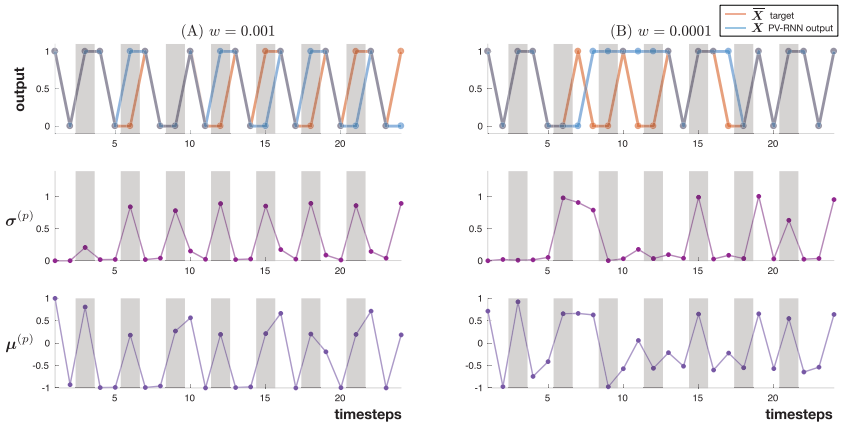

Figure 11: The target and the regenerated outputs, the mean $\boldsymbol{\mu}^{(p)}$, and the standard deviation $\boldsymbol{\sigma}^{(p)}$ of two PV-RNNs trained with meta-prior w set to 0.01×10^{-1} (A) and 0.001×10^{-1} (B). The gray bars show the time steps corresponding to uncertain states.

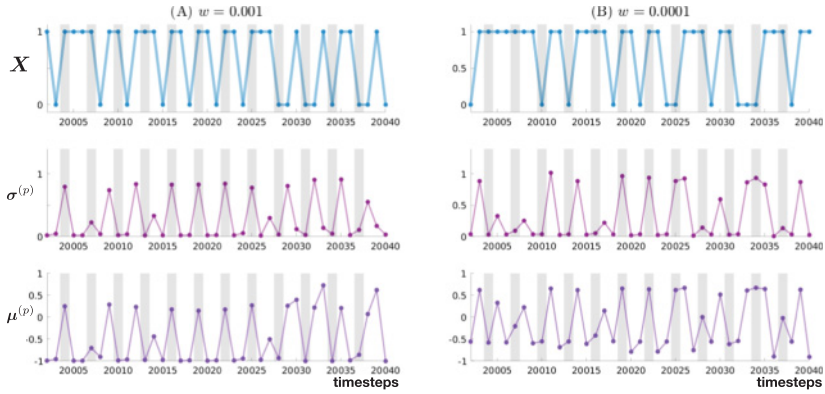

Figure 12: The generated output, the mean $\mu^{(p)}$, and the standard deviation $\sigma^{(p)}$ from time steps 20,002 to 20,040 of two PV-RNNs trained with the meta-prior w set to 0.01×10^{-1} (A) and 0.001×10^{-1} (B). The gray bars show the time steps corresponding to uncertain states.

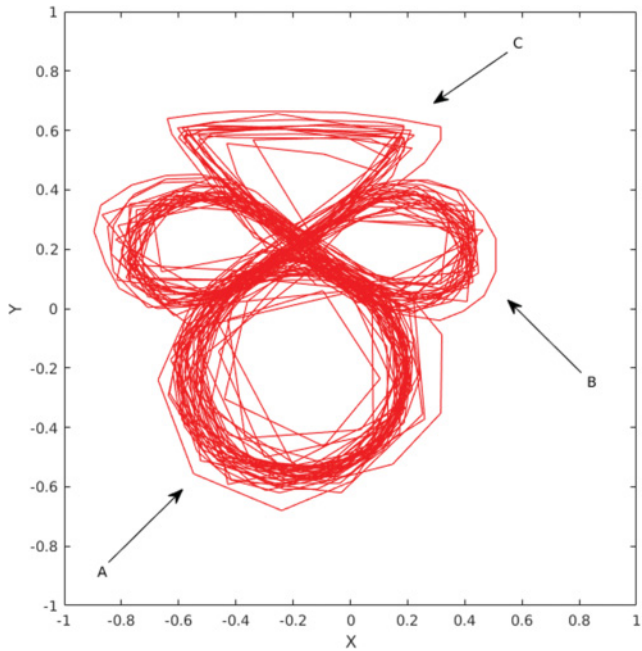

Figure 13: The A, B, and C patterns generated based on PFSM shown in Figure 2B. The whole pattern is 1000 time steps, including 20 of pattern A, 13 of pattern B, and 7 of pattern C: Each pattern has two cycles. The patterns are not identical and contain fluctuations in amplitude, velocity, and shape.

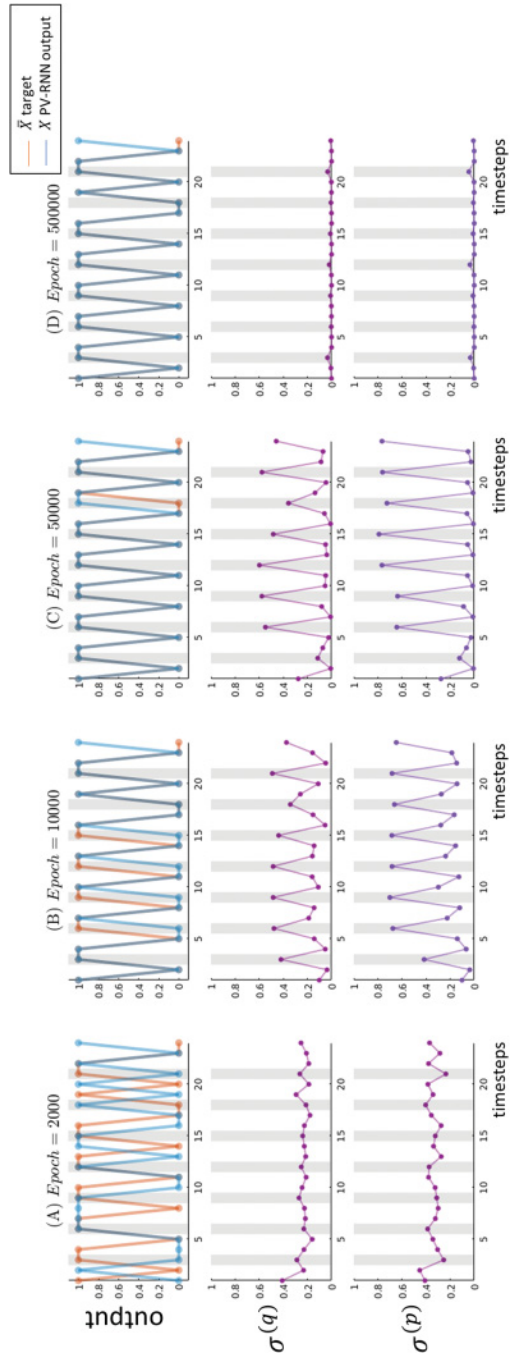

Figure 14: At the early stage of the training, $\boldsymbol{\sigma}^{(q)}$ and $\boldsymbol{\sigma}^{(p)}$ are large for both the probabilistic states (gray bars) and the deterministic states; the transition rules defined in the PFSM are mostly broken. After 10,000 epochs, $\boldsymbol{\sigma}^{(q)}$ and $\boldsymbol{\sigma}^{(p)}$ are significantly higher for the probabilistic states than the deterministic ones. Therefore, the reconstruction of the training patterns is successfully done on the deterministic states. At the final stage of the training, $\boldsymbol{\sigma}^{(q)}$ and $\boldsymbol{\sigma}^{(p)}$ are near zero and the target sequence is completely regenerated. These results show that the network reconstructs the deterministic states of the training patterns first.

Acknowledgments

We give special thanks to people who helped us with this study. First and foremost, we are particularly grateful for great assistance and insightful advice given by Fabien Benureau for improving the content and language of the article. We sincerely express our appreciation to Tom Burns, Nadine Wirkuttis, Wataru Ohata, Takazumi Matsumoto, and Siqing Hou for their great help in improving this work as well.

References

Ahmadi, A., & Tani, J. (2017a). Bridging the gap between probabilistic and deterministic models: A simulation study on a variational Bayes predictive coding recurrent neural network model. In *Proceedings of the International Conference on Neural Information Processing* (pp. 760–769). Berlin: Springer.

Ahmadi, A., & Tani, J. (2017b). How can a recurrent neurodynamic predictive coding model cope with fluctuation in temporal patterns? Robotic experiments on imitative interaction. *Neural Networks*, *92*, 3–16.

Alligood, K. T., Sauer, T. D., & Yorke, J. A. (1996). *Chaos*. Berlin: Springer.

Arie, H., Endo, T., Arakaki, T., Sugano, S., & Tani, J. (2009). Creating novel goal-directed actions at criticality: A neuro-robotic experiment. *New Mathematics and Natural Computation*, *5*(1), 307–334.

Baltieri, M., & Buckley, C. L. (2017). An active inference implementation of phototaxis. In *Proceedings of the European Conference on Artificial Life* (vol. 14, pp. 36–43). Cambridge, MA: MIT Press.

Bayer, J., & Osendorfer, C. (2014). *Learning stochastic recurrent networks*. arXiv:1411.7610.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., & Bengio, S. (2015). *Generating sentences from a continuous space*. arXiv:1511.06349.

Butz, M. V., Bilkey, D., Humaidan, D., Knott, A., & Otte, S. (2019). Learning, planning, and control in a monolithic neural event inference architecture. *Neural Networks*, *117*, 135–144.

Chen, X., Kingma, D. P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., . . . Abbeel, P. (2016). *Variational lossy autoencoder*. arXiv:1611.02731.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., & Bengio, Y. (2015). A recurrent latent variable model for sequential data. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems, 28* (pp. 2980–2988). Red Hook, NY: Curran.

Clark, A. (2015). *Surfing uncertainty: Prediction, action, and the embodied mind*. New York: Oxford University Press.

Crutchfield, J. P. (1992). Semantics and thermodynamics. In M. Casdagli & S. Eubank (Eds.), *Nonlinear modeling and forecasting* (p. 317). Reading, MA: Addison-Wesley.

DiCicco-Bloom, B., & Crabtree, B. F. (2006). The qualitative research interview. *Medical Education*, *40*(4), 314–321.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211.

Fabius, O., & van Amersfoort, J. R. (2014). *Variational recurrent auto-encoders*. arXiv:1412.6581.

Fraccaro, M., Sønderby, S. K., Paquet, U., & Winther, O. (2016). Sequential neural models with stochastic layers. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems*, *29* (pp. 2199–2207). Red Hook, NY: Curran.

Friston, K. (2005). A theory of cortical responses. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, *360*(1456), 815–836.

Friston, K. (2010). The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience*, *11*(2), 127.

Friston, K. (2018). Does predictive coding have a future? *Nature Neuroscience*, *21*(8), 1019.

Friston, K. J., Daunizeau, J., & Kiebel, S. J. (2009). Reinforcement learning or active inference? *PloS One*, *4*(7), e6421.

Friston, K. J., Daunizeau, J., Kilner, J., & Kiebel, S. J. (2010). Action and behavior: A free-energy formulation. *Biological Cybernetics*, *102*(3), 227–260.

Geiger, D., Verma, T., & Pearl, J. (1990). Identifying independence in Bayesian networks. *Networks*, *20*(5), 507–534.

Goyal, A., Sordoni, A., Côté, M.-A., Ke, N., & Bengio, Y. (2017). Z-forcing: Training stochastic recurrent networks. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. V. N. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems*, *30* (pp. 6713–6723). Red Hook, NY: Curran.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., . . . Lerchner, A. (2017). Beta-vae: Learning basic visual concepts with a constrained variational framework. In *Proceedings of the International Conference on Learning Representations*.

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

Hohwy, J. (2013). *The predictive mind*. New York: Oxford University Press.

Jordan, M. I. (1997). Serial order: A parallel distributed processing approach. In J. W. Donahoe & V. P. Dorsel (Eds.), *Advances in psychology* (vol. 121, pp. 471–495). Amsterdam: Elsevier.

Karl, M., Soelch, M., Bayer, J., & van der Smagt, P. (2016). *Deep variational Bayes filters: Unsupervised learning of state space models from raw data*. arXiv:1605.06432.

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv:1412.6980.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., & Welling, M. (2016). Improved variational inference with inverse autoregressive flow. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems, 29* (pp. 4743–4751). Red Hook, NY: Curran.

Kingma, D. P., & Welling, M. (2013). *Auto-encoding variational Bayes*. arXiv:1312.6114.

Lawson, R. P., Rees, G., & Friston, K. J. (2014). An aberrant precision account of autism. *Frontiers in Human Neuroscience*, *8*, 302.

Lee, T. S., & Mumford, D. (2003). Hierarchical Bayesian inference in the visual cortex. *Journal of the Optical Society of America*, *20*(7), 1434–1448.

Murata, S., Yamashita, Y., Arie, H., Ogata, T., Sugano, S., & Tani, J. (2017). Learning to perceive the world as probabilistic or deterministic via interaction with others: A neuro-robotics experiment. *IEEE Trans. Neural Netw. Learning Syst.*, *28*(4), 830–848.

Murata, S., Namikawa, J., Arie, H., Sugano, S., & Tani, J. (2013). Learning to reproduce fluctuating time series by inferring their time-dependent stochastic properties: Application in robot learning via tutoring. *IEEE Transactions on Autonomous Mental Development*, *5*(4), 298–310.

Otte, S., Zwiener, A., & Butz, M. V. (2017). Inherently constraint-aware control of many-joint robot arms with inverse recurrent models. In *Proceedings of the International Conference on Artificial Neural Networks* (pp. 262–270). Berlin: Springer.

Pezzulo, G., Rigoli, F., & Friston, K. (2015). Active inference, homeostatic regulation and adaptive behavioural control. *Progress in Neurobiology*, *134*, 17–35.

Rao, R. P., & Ballard, D. H. (1999). Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, *2*(1), 79.

Rao, R. P., & Sejnowski, T. J. (2000). Predictive sequence learning in recurrent neocortical circuits. In S. A. Solla, T. K. Leen, & K.-R. Müller (Eds.), *Advances in neural information processing systems*, *12* (pp. 164–170). Cambridge, MA: MIT Press.

Robertson, C. E., & Baron-Cohen, S. (2017). Sensory perception in autism. *Nature Reviews Neuroscience*, *18*(11), 671.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation*. La Jolla: University of California, San Diego, Institute for Cognitive Science.

Shabanian, S., Arpit, D., Trischler, A., & Bengio, Y. (2017). *Variational Bi-LSTMs*. arXiv:1711.05717.

Sinai, Y. G. (1972). Gibbs measures in ergodic theory. *Russian Mathematical Surveys*, *27*(4), 21.

Stevenson, R. A., Siemann, J. K., Schneider, B. C., Eberly, H. E., Woynaroski, T. G., Camarata, S. M., & Wallace, M. T. (2014). Multisensory temporal integration in autism spectrum disorders. *Journal of Neuroscience*, *34*(3), 691–697.

Tani, J. (1996). Model-based learning for mobile robot navigation from the dynamical systems perspective. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *26*(3), 421–436.

Tani, J., & Fukumura, N. (1995). Embedding a grammatical description in deterministic chaos: An experiment in recurrent neural learning. *Biological Cybernetics*, *72*(4), 365–370.

Tani, J., & Ito, M. (2003). Self-organization of behavioral primitives as multiple attractor dynamics: A robot experiment. *IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans*, *33*(4), 481–488.

Tani, J., Ito, M., & Sugita, Y. (2004). Self-organization of distributedly represented multiple behavior schemata in a mirror system: Reviews of robot experiments using RNNPB. *Neural Networks*, *17*(8–9), 1273–1289.

Tani, J., & Nolfi, S. (1999). Learning to perceive the world as articulated: An approach for hierarchical learning in sensory-motor systems. *Neural Networks*, *12*(7-8), 1131–1141.

Van de Cruys, S., Evers, K., Van der Hallen, R., Van Eylen, L., Boets, B., de Wit, L., & Wagemans, J. (2014). Precise minds in uncertain worlds: Predictive coding in autism. *Psychological Review*, *121*(4), 649.

Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences. PhD diss., Harvard University.

Yamashita, Y., & Tani, J. (2008). Emergence of functional hierarchy in a multiple timescale neural network model: A humanoid robot experiment. *PLoS Computational Biology*, *4*(11), e1000220.

Zhao, T., Zhao, R., & Eskenazi, M. (2017). *Learning discourse-level diversity for neural dialog models using conditional variational autoencoders*. arXiv:1703.10960.

Received November 4, 2018; accepted June 24, 2019.