Deep Explicit Duration Switching Models for Time Series

Abdul Fatir Ansari[2*†] **Konstantinos Benidis**[1*] **Richard Kurle**[1] **Ali Caner Türkmen**[1]

Harold Soh[2] **Alexander J. Smola**[1] **Yuyang Wang**[1] **Tim Januschowski**[1]

[1]Amazon Research [2]National University of Singapore

Correspondence to: abdulfatir@u.nus.edu, {kbenidis, kurler}@amazon.com

Abstract

Many complex time series can be effectively subdivided into distinct regimes that exhibit persistent dynamics. Discovering the switching behavior and the statistical patterns in these regimes is important for understanding the underlying dynamical system. We propose the Recurrent Explicit Duration Switching Dynamical System (RED-SDS), a flexible model that is capable of identifying both state- and time-dependent switching dynamics. State-dependent switching is enabled by a recurrent state-to-switch connection and an explicit duration count variable is used to improve the time-dependent switching behavior. We demonstrate how to perform efficient inference using a hybrid algorithm that approximates the posterior of the continuous states via an inference network and performs exact inference for the discrete switches and counts. The model is trained by maximizing a Monte Carlo lower bound of the marginal log-likelihood that can be computed efficiently as a byproduct of the inference routine. Empirical results on multiple datasets demonstrate that RED-SDS achieves considerable improvement in time series segmentation and competitive forecasting performance against the state of the art.

1 Introduction

Time series forecasting plays a key role in informing industrial and business decisions [17, 24, 8], while segmentation is useful for understanding biological and physical systems [40, 45, 34]. State Space Models (SSMs) [16] are a powerful tool for such tasks—especially when combined with neural networks [42, 12, 13]—since they provide a principled framework for time series modeling. One of the most popular SSMs is the Linear Dynamical System (LDS) [5, 43], which models the dynamics of the data using a continuous latent variable, called *state*, that evolves with Markovian linear transitions. The assumptions of LDS allow for exact inference of the states [27]; however, they are too restrictive for real-world systems that often exhibit piecewise linear or non-linear hidden dynamics with a finite number of operating modes or *regimes*. For example, the power consumption of a city may follow different hidden dynamics during weekdays and weekends. Such data are better explained by a Switching Dynamical System (SDS) [1, 21], an SSM with an additional set of latent variables called *switches* that define the operating mode active at the current timestep.

Switching events can be classified into time-dependent or state-dependent [33]. Historically, emphasis was placed on the former, which occurs after a certain amount of time has elapsed in a given regime. While in a vanilla SDS switch durations follow a geometric distribution, more complex long-term

*Equal contribution.

†Work done during an internship at Amazon Research.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

temporal patterns can be captured using explicit duration models [40, 9]. As a recent alternative to time-dependency, recurrent state-to-switch connections [35] have been proposed that capture state-dependent switching, i.e., a change that occurs when the state variable enters a region that is governed by a different regime. For added flexibility, these models can be used in conjunction with transition/emission distributions parameterized by neural networks [25, 19, 13, 30]. Recent works, e.g., [13, 30], proposed hybrid inference algorithms that exploit the graphical model structure to perform approximate inference for some latent variables and conditionally exact inference for others.

Despite these advances in representation and inference, modeling complex real-world temporal phenomena remains challenging. For example, state-of-the-art state-dependent models (e.g., [13]) lack the capacity to adequately capture time-dependent switching. Empirically, we find this hampers their ability to learn parsimonious segmentations when faced with complex patterns and long-term dependencies (see Fig. 1 for an example). Conversely, time-dependent switching models are "open-loop" and unable to model state-conditional behavioral transitions that are common in many systems, e.g., in autonomous or multi-agent systems [35]. Intuitively, the suitability of the switching model largely depends on the underlying data-generating process; city power consumption may be better modeled via time-dependent switching, whilst the motion of a ball bouncing between two walls is driven by its state. Indeed, complex real-world processes likely involve both types of switching behavior.

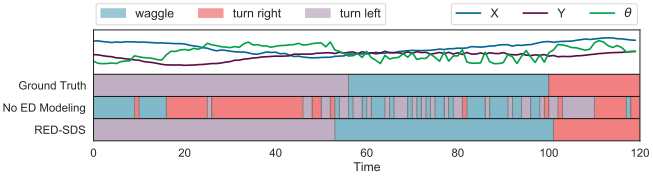


Figure 1: Segments (colored bars at the bottom) inferred by a baseline with no Explicit Duration (ED) modeling vs. our RED-SDS for a time series from the dancing bees dataset (top). The baseline struggles to learn long-term temporal patterns, particularly during the "waggle" phase of the bee dance.

Motivated by this gap, we propose the Recurrent Explicit Duration Switching Dynamical System (RED-SDS) that captures *both* state-dependent and time-dependent switching. RED-SDS combines the recurrent state-to-switch connection with explicit duration models for switches. Notably, RED-SDS allows the incorporation of inductive biases via the hyperparameters of the duration models to better identify long-term temporal patterns. However, this combination also complicates inference, especially when using neural networks to model the underlying probability distributions. To address this technical challenge, we propose a hybrid algorithm that (i) uses an inference network for the continuous latent variables (states) and (ii) performs efficient exact inference for the discrete latent variables (switches and counts) using a forward-backward routine similar to Hidden Semi-Markov Models [48, 9]. The model is trained by maximizing a Monte Carlo lower bound of the marginal log-likelihood that can be efficiently computed by the inference routine.

We evaluated RED-SDS on two important tasks: segmentation and forecasting. Empirical results on segmentation show that RED-SDS is able to identify both state- and time-dependent switching patterns, considerably outperforming benchmark models. For example, Fig. 1 shows that RED-SDS addresses the oversegmentation that occurs with an existing strong baseline [13]. For forecasting, we illustrate the competitive performance of RED-SDS with an extensive evaluation against state-of-the-art models on multiple benchmark datasets. Further, we show how our model is able to simplify the forecasting problem by breaking the time series into different meaningful regimes without any imposed structure. As such, we manage to learn appropriate duration models for each regime and extrapolate the learned patterns into the forecast horizon consistently.

In summary, the key contributions of this paper are:

- RED-SDS, a novel non-linear state space model which combines the recurrent state-to-switch connection with explicit duration models to flexibly model switch durations;
- an efficient hybrid inference and learning algorithm that combines approximate inference for states with conditionally exact inference for switches and counts;
- a thorough evaluation on a number of benchmark datasets for time series segmentation and forecasting, demonstrating that RED-SDS can learn meaningful duration models, identify both state- and time-dependent switching patterns and extrapolate the learned patterns consistently into the future.

2 Background: switching dynamical systems

Notation. Matrices, vectors and scalars are denoted by uppercase bold, lowercase bold and lowercase normal letters, respectively. We denote the sequence $\{\mathbf{y}_1, \ldots, \mathbf{y}_T\}$ by $\mathbf{y}_{1:T}$, where $\mathbf{y}_t$ is the value of $\mathbf{y}$ at time t. In our notation, we do not further differentiate between random variables and their realizations.

Switching Dynamical Systems (SDS) are hybrid SSMs that use discrete "switching" states z_t to index one of K base dynamical systems with continuous states $\mathbf{x}_t$. The joint distribution factorizes as

$$p(\mathbf{y}_{1:T}, \mathbf{x}_{1:T}, z_{1:T}) = \prod_{t=1}^{T} p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{x}_{t-1}, z_t)p(z_t|z_{t-1}), \tag{1}$$

where $p(\mathbf{x}_1|\mathbf{x}_0, z_1)p(z_1|z_0) = p(\mathbf{x}_1|z_1)p(z_1)$ is the initial (continuous and discrete) state prior. The base dynamical systems have continuous state transition $p(\mathbf{x}_t|\mathbf{x}_{t-1}, z_t)$ and continuous or discrete emission $p(\mathbf{y}_t|\mathbf{x}_t)$ that can both be linear or non-linear.

The discrete transition $p(z_t|z_{t-1})$ of vanilla SDS is parametrized by a stochastic transition matrix $\mathbf{A} \in \mathbb{R}^{K\times K}$, where the entry $a_{ij} = \mathbf{A}(i, j)$ represents the probability of switching from state i to state j. This results in an "open loop" as the transition only depends on the previous switch which inhibits the model from learning state-dependent switching patterns [35]. Further, the state duration (also known as the *sojourn time*) follows a geometric distribution [9], where the probability of staying in state i for d steps is $\rho_i(d) = (1 - a_{ii})a_{ii}^{d-1}$. This *memoryless* switching process results in frequent regime switching, limiting the ability to capture consistent long-term time-dependent switching patterns. In the following, we briefly discuss two approaches that have been proposed to improve the state-dependent and time-dependent switching capabilities in SDSs.

Recurrent SDS. Recurrent SDSs (e.g., [6, 35, 7, 30]) address state-dependent switching by changing the switch transition distribution to $p(z_t|\mathbf{x}_{t-1}, z_{t-1})$—called the state-to-switch recurrence—implying that the switch transition distribution changes at every step and the sojourn time no longer follows a geometric distribution. This extension complicates inference. Furthermore, the first-order Markovian recurrence does not adequately address long-term time-dependent switching.

Explicit duration SDS. Explicit duration SDSs are a family of models that introduce additional random variables to explicitly model the switch duration distribution. Explicit duration variables have been applied to both HMMs and SDSs with Gaussian linear continuous states; the resulting models are referred to as Hidden Semi-Markov Models (HSMMs) [38, 48], and Explicit Duration Switching Linear Gaussian SSMs (ED-SLGSSMs) [9, 40, 10], respectively. Several methods have been proposed in the literature for modeling the switch duration, e.g., using decreasing or increasing count, and duration-indicator variables. In the following, we briefly describe modeling switch duration using increasing count variables and refer the reader to Chiappa [9] for details.

Increasing count random variables c_t represent the *run-length* of the currently active regime and can either increment by 1 or reset to 1. An increment indicates that the switch variable z_t is copied over to the next timestep whereas a reset indicates a regular Markov transition using the transition matrix $\mathbf{A}$. Each of the K switches has a distinct duration distribution ρ_k, a categorical distribution over $\{d_{\min}, \ldots, d_{\max}\}$, where $d_{\min}$ and $d_{\max}$ delimit the number of steps before making a Markov transition. Following [40, 9], the probability of a count increment is given by

$$v_k(c) = 1 - \frac{\rho_k(c)}{\sum_{d=c}^{d_{\max}} \rho_k(d)}. \tag{2}$$

The transition of count c_t and switch z_t variables is defined as

$$p(c_t|z_{t-1} = k, c_{t-1}) = \begin{cases} v_k(c_{t-1}) & \text{if} \quad c_t = c_{t-1} + 1 \\ 1 - v_k(c_{t-1}) & \text{if} \quad c_t = 1 \end{cases}, \tag{3}$$

$$p(z_t = j|z_{t-1} = i, c_t) = \begin{cases} \delta_{z_t=i} & \text{if} \quad c_t > 1 \\ \mathbf{A}(i, j) & \text{if} \quad c_t = 1 \end{cases}, \tag{4}$$

where δ_{cond} denotes the delta function which takes the value 1 only when cond is true.

Although SDSs with explicit switch duration distributions can identify long-term time-dependent switching patterns, the switch transitions are not informed by the state—inhibiting their ability to

model state-dependent switching events. Furthermore, to the best of our knowledge, SDSs with explicit duration models have only been studied for Gaussian linear states [10, 9, 40].

3 Recurrent explicit duration switching dynamical systems

In this section we describe the Recurrent Explicit Duration Switching Dynamical System (RED-SDS) that combines both state-to-switch recurrence and explicit duration modeling for switches in a single non-linear model. We begin by formulating the generative model as a recurrent switching dynamical system that explicitly models the switch durations using increasing count variables. We then discuss how to perform efficient inference for different sets of latent variables. Finally, we discuss how to estimate the parameters of RED-SDS using maximum likelihood.

3.1 Model formulation

Consider the graphical model in Fig. 2 (a); the joint distribution of the counts $c_t \in \{1, \dots, d_{\max}\}$, the switches $z_t \in \{1, \dots, K\}$, the states $\mathbf{x}_t \in \mathbb{R}^m$, and the observations $\mathbf{y}_t \in \mathbb{R}^d$, conditioned on the control inputs $\mathbf{u}_t \in \mathbb{R}^c$, factorizes as

$$p_\theta(\mathbf{y}_{1:T}, \mathbf{x}_{1:T}, z_{1:T}, c_{1:T} | \mathbf{u}_{1:T}) = p(\mathbf{y}_1|\mathbf{x}_1) p(\mathbf{x}_1|z_1, \mathbf{u}_1) p(z_1|\mathbf{u}_1) \cdot \left[\prod_{t=2}^{T} p(\mathbf{y}_t|\mathbf{x}_t) p(\mathbf{x}_t|\mathbf{x}_{t-1}, z_t, \mathbf{u}_t) p(z_t|\mathbf{x}_{t-1}, z_{t-1}, c_t, \mathbf{u}_t) p(c_t|z_{t-1}, c_{t-1}, \mathbf{u}_t) \right]. \tag{5}$$

Similar to [40, 9], we consider increasing count variables c_t to incorporate explicit switch durations into the model, i.e., c_t can either increment by 1 or reset to 1 at every timestep and represent the run-length of the current regime. A self-transition is allowed after the exhaustion of $d_{\max}$ steps for flexibility. In the subsequent discussion we omit the control inputs $\mathbf{u}_t$ for clarity of exposition.

We model the initial prior distributions in Eq. (5) for the respective discrete and continuous case as

$$p(z_1) = \mathrm{Cat}(z_1; \boldsymbol{\pi}), \tag{6}$$

$$p(\mathbf{x}_1|z_1) = \mathcal{N}(\mathbf{x}_1; \boldsymbol{\mu}_{z_1}, \boldsymbol{\Sigma}_{z_1}), \tag{7}$$

where Cat denotes a categorical and $\mathcal{N}$ a multivariate Gaussian distribution. The transition distributions for the discrete variables (count and switch) are given by

$$p(c_t|z_{t-1}, c_{t-1}) = \begin{cases} v_{z_{t-1}}(c_{t-1}) & \text{if} \quad c_t = c_{t-1} + 1 \\ 1 - v_{z_{t-1}}(c_{t-1}) & \text{if} \quad c_t = 1 \end{cases}, \tag{8}$$

$$p(z_t|\mathbf{x}_{t-1}, z_{t-1}, c_t) = \begin{cases} \delta_{z_t = z_{t-1}} & \text{if} \quad c_t > 1 \\ \mathrm{Cat}(z_t; \mathcal{S}_\tau(f_z(\mathbf{x}_{t-1}, z_{t-1}))) & \text{if} \quad c_t = 1 \end{cases}, \tag{9}$$

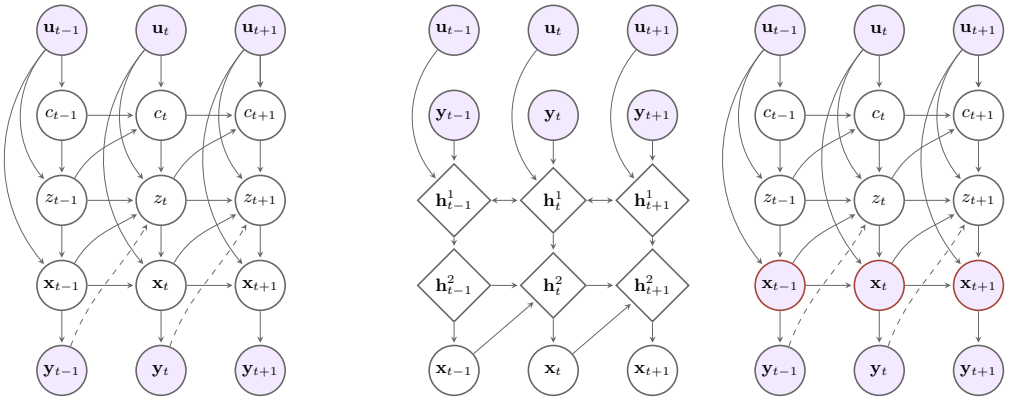
where $\mathcal{S}_\tau$ is the tempered softmax function (cf. Section 3.3) with temperature τ, and f_z can be a linear function or a neural network. The probability of a count increment v_k for a switch k is defined via the duration model ρ_k as in Eq. (2). The continuous state transition and the emission are given by

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}, z_t) = \mathcal{N}(\mathbf{x}_t; f_x^\mu(\mathbf{x}_{t-1}, z_t), f_x^\Sigma(\mathbf{x}_{t-1}, z_t)), \tag{10}$$

$$p(\mathbf{y}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{y}_t; f_y^\mu(\mathbf{x}_t), f_y^\Sigma(\mathbf{x}_t)), \tag{11}$$

where f_x^μ, f_x^Σ, f_y^μ, f_y^Σ are again linear functions or neural networks.

The model is general and flexible enough to handle both state- and time-dependent switching. The switch transitions $z_{t-1} \to z_t$ are conditioned on the previous state $\mathbf{x}_{t-1}$ which ensures that the switching events occur in a "closed loop". The switch duration models ρ_k provide flexibility to stay long term in the same regime, allowing to better capture time-dependent switching. We use increasing count variables to incorporate switch durations into our model as they are more amenable to the case when the count transitions depend on the control $\mathbf{u}_t$. For instance, decreasing count variables, another popular option [11, 36, 9], *deterministically* count down from the sampled segment duration length to 1. This makes it difficult to condition the switch duration model on the control inputs. In contrast, increasing count variables increment or reset probabilistically at every timestep.



(a) Generative Model (b) Inference

Figure 2: **(a)** Forward generative model of RED-SDS. **(b)** Left: Approximate inference for the states $\mathbf{x}_t$ using an inference network. $\mathbf{h}^1_t$ is given by a non-causal network and $\mathbf{h}^2_t$ is given by a causal RNN. Right: Exact inference for switch z_t and count c_t variables given pseudo-observations (highlighted in red) of $\mathbf{x}_t$ provided by the inference network. (Shaded) circles represent (observed) random variables, diamonds represent deterministic nodes, and dashed lines represent optional connections.

3.2 Inference

Exact inference is intractable in SDSs and scales exponentially with time [32]. Various approximate inference procedures have been developed for traditional SDSs [14, 21, 6], while more recently inference networks have been used for amortized inference for all or a subset of latent variables [25, 28, 13, 30]. Particularly, Dong et al. [13] used an inference network for the states and performed exact HMM-like inference for the switches, conditioned on the states. We take a similar approach and use an inference network for the continuous latent variables (states) and perform conditionally exact inference for the discrete latent variables (switches and counts) similar to the forward-backward procedure for HSMM [48, 9]. We define the variational approximation to the true posterior $p(\mathbf{x}_{1:T}, z_{1:T}, c_{1:T}|\mathbf{y}_{1:T})$ as $q(\mathbf{x}_{1:T}, z_{1:T}, c_{1:T}|\mathbf{y}_{1:T}) = q_\phi(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})p_\theta(z_{1:T}, c_{1:T}|\mathbf{y}_{1:T}, \mathbf{x}_{1:T})$ where ϕ and θ denote the parameters of the inference network and the generative model respectively.

Approximate inference for states. The posterior distribution of the states, $q_\phi(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$, is approximated using an inference network. We first process the observation sequence $\mathbf{y}_{1:T}$ using a non-causal network such as a bi-RNN or a Transformer [46] to simulate smoothing by incorporating both past and future information. The non-causal network returns an embedding of the data $\mathbf{h}^1_{1:T}$ which is then fed to a causal RNN that outputs the posterior distribution $q_\phi(\mathbf{x}_{1:T}|\mathbf{y}_{1:T}) = \prod_t q(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{h}^1_{1:T})$. See Fig. 2 (b) for an illustration of the inference procedure.

Exact inference for counts and switches. Inference for the switches $z_{1:T}$ and the counts $c_{1:T}$ can be performed exactly conditioned on states $\mathbf{x}_{1:T}$ and observations $\mathbf{y}_{1:T}$. Samples from the approximate posterior $\tilde{\mathbf{x}}_{1:T} \sim q(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$ are used as pseudo-observations of $\mathbf{x}_{1:T}$ to infer the posterior distribution $p_\theta(z_{1:T}, c_{1:T}|\mathbf{y}_{1:T}, \tilde{\mathbf{x}}_{1:T})$. A naive approach to infer this distribution is by treating the pair (c_t, z_t) as a "meta switch" that takes $Kd_{\max}$ possibles values and perform HMM-like forward-backward inference. However, this results in a computationally expensive $O(TK^2d^2_{\max})$ procedure that scales poorly with $d_{\max}$. Fortunately, we can pre-compute some terms in the forward-backward equations by exploiting the fact that the count variable can only increment by 1 or reset to 1 at every timestep. This results in an $O(TK(K + d_{\max}))$ algorithm that scales gracefully with $d_{\max}$ [9]. The forward α_t and backward β_t variables, defined as

$$\alpha_t(z_t, c_t) = p(\mathbf{y}_{1:t}, \mathbf{x}_{1:t}, z_t, c_t), \tag{12}$$
$$\beta_t(z_t, c_t) = p(\mathbf{y}_{t+1:T}, \mathbf{x}_{t+1:T}|\mathbf{x}_t, z_t, c_t), \tag{13}$$

can be computed by modifying the forward-backward recursions used for the HSMM [9] to handle the additional observed variables $\mathbf{x}_{1:t}$. We refer the reader to Appendix A.1 for the exact derivation.

3.3 Learning

The parameters $\{\phi, \theta\}$ can be learned by maximizing the evidence lower bound (ELBO):

$$\begin{aligned}\mathcal{L}_{\text{ELBO}} &= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})p(z_{1:T},c_{1:T}|\mathbf{y}_{1:T},\mathbf{x}_{1:T})}\left[\log \frac{p(\mathbf{y}_{1:T},\mathbf{x}_{1:T},z_{1:T},c_{1:T},)}{q(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})p(z_{1:T},c_{1:T}|\mathbf{y}_{1:T},\mathbf{x}_{1:T})}\right] \\ &= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})}\left[\log \frac{p(\mathbf{y}_{1:T},\mathbf{x}_{1:T})}{q(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})}\right].\end{aligned} \tag{14}$$

The likelihood term $p(\mathbf{y}_{1:T}, \mathbf{x}_{1:T})$ can be computed using the forward variable $\alpha_T(z_T, c_T)$ by marginalizing out the switches and the counts,

$$p(\mathbf{y}_{1:T}, \mathbf{x}_{1:T}) = \sum_{z_T, c_T} \alpha_T(z_T, c_T), \tag{15}$$

and the entropy term $-\mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})}\left[\log q(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})\right]$ can be computed using the approximate posterior $q(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$ output by the inference network. The ELBO can be maximized via stochastic gradient ascent given that the posterior $q(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$ is reparameterizable.

We note that Dong et al. [13] used a lower bound for the likelihood term in Switching Non-Linear Dynamical Systems (SNLDS); however, it can be computed succinctly by marginalizing out the discrete random variable (i.e., the switch in SNLDS) from the forward variable α_T, similar to Eq. (15). Using our objective function, we observed that the model was less prone to posterior collapse (where the model ends up using only one switch) and we did not require the additional ad-hoc KL regularizer used in Dong et al. [13]. Please refer to Appendix B.4 for a brief discussion on the likelihood term in SNLDS.

Temperature annealing. We use the tempered softmax function $\mathcal{S}_\tau$ to map the logits to probabilities for the switch transition $p(z_t|\mathbf{x}_{t-1}, z_{t-1}, c_t = 1)$ and the duration models $\rho_k(d)$ which is defined as

$$\mathcal{S}_\tau(\mathbf{o})_i = \frac{\exp\left(\frac{o_i}{\tau}\right)}{\sum_j \exp\left(\frac{o_j}{\tau}\right)}, \tag{16}$$

where $\mathbf{o}$ is a vector of logits. The temperature τ is deterministically annealed from a high value during training. The initial high temperature values soften the categorical distribution and encourage the model to explore all switches and durations. This prevents the model from getting stuck in poor local minima that ignore certain switches or longer durations which might explain the data better.

4 Related work

The most relevant components of RED-SDS are recurrent state-to-switch connections and the explicit duration model, enabling both for state- and time-dependent switching. Additionally, RED-SDS allows for efficient approximate inference (analytic for switches and counts), despite parameterizing the various conditional distributions through neural networks. Existing methods address only a subset of these features as we discuss in the following.

The most prominent SDS is the Switching Linear Dynamical System (SLDS), where each regime is described by linear dynamics and additive Gaussian noise. A major focus of previous work has been on efficient approximate inference algorithms that exploit the Gaussian linear substructure (e.g., [21, 49, 14]). In contrast to RED-SDS, these models lack recurrent state-to-switch connections and duration variables and are limited to linear regimes.

Previous work has addressed the state-dependent switching by introducing a connection to the continuous state of the dynamical system [6, 35, 7, 30]. The additional recurrence complicates inference w.r.t. the continuous states; prior work uses expensive sampling methods in order to approximate the corresponding integrals [6] or as part of a message passing algorithm for joint inference of states and parameters [35]. On the other hand, ARSGLS [30] avoids sampling the continuous states by using conditionally linear state-to-switch connections and softmax-transformed Gaussian switch variables. However, both the ARSGLS and the related KVAE [19] can be interpreted as an SLDS with "soft" switches that interpolate linear regimes continuously rather than truly discrete states. This makes them less suited for time series segmentation compared to RED-SDS. Contrary to

the aforementioned models, RED-SDS allows non-linear regimes described by neural networks and incorporates a discrete explicit duration model without complicating inference w.r.t. the continuous states, since closed-form expressions are used for the discrete variables instead. Using amortized variational inference for continuous variables and analytic expressions for discrete variables has been proposed previously for segmentation in SNLDS [13]. RED-SDS extends this via an additional explicit duration variable that represents the run-length for the currently active regime.

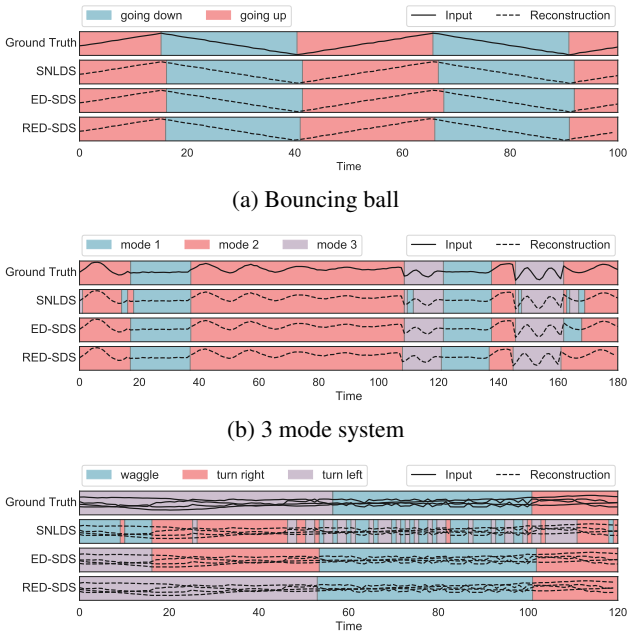
Explicit duration variables have previously been proposed for changepoint detection [2, 3] and segmentation [10, 26]. For instance, BOCPD [2] is a Bayesian online changepoint detection model with explicit duration modeling. RED-SDS improves upon BOCPD by allowing for segment labeling rather than just detecting changepoints. The HDP-HSMM [26] is a Bayesian non-parametric extension to the traditional HSMM. Recent work [11, 36] has also combined HSMM with RNNs for amortized inference. These models—being variants of HSMM—do not model the latent dynamics of the data like RED-SDS. Chiappa and Peters [10] proposed approximate inference techniques for a variant of SLDS with explicit duration modeling. In contrast, RED-SDS is a more general non-linear model that allows for efficient amortized inference—closed-form w.r.t. the discrete latent variables.

5 Experiments

In this section, we present empirical results on two prominent time series tasks: segmentation and forecasting. Our primary goals were to determine if RED-SDS (a) can discover meaningful switching patterns in the data in an unsupervised manner, and (b) can probabilistically extrapolate a sequence of observations, serving as a viable generative model for forecasting. In the following, we discuss the main results and relegate details to the appendix.

5.1 Segmentation

We experimented with two instantiations of our model: `RED-SDS` (complete model) and `ED-SDS`, the ablated variant without state-to-switch recurrence. We compared against the closely related `SNLDS` [13] trained with a modified objective function. The original objective proposed in [13] suffered from training difficulties: it resulted in frequent posterior collapse and was sensitive to the cross-entropy regularization term. Our version of `SNLDS` can be seen as a special case of `RED-SDS` with $d_{\max} = 1$, i.e., without the explicit duration modeling (cf. Appendix B.4). We also conducted preliminary experiments on soft-switching models: `KVAE` [19] and `ARSGLS` [30]. However, these models use a continuous interpolation of the different operating modes which cannot always be correctly assigned to a single discrete mode, hence we do not report these unfavorable findings here (cf. Appendix B.4). For all models, we performed segmentation by taking the most likely value of the switch at each timestep from the posterior distribution over the switches. As the segmentation labels are arbitrary and may not match the ground truth labels, we evaluated the models using multiple metrics: frame-wise segmentation accuracy (after matching the labelings using the Hungarian algorithm [29]), Normalized Mutual Information (NMI) [47], and Adjusted Rand Index (ARI) [23] (cf. Appendix B.2).



(a) Bouncing ball

(b) 3 mode system

(c) Dancing bees

Figure 3: Qualitative segmentation results on the bouncing ball, 3 mode system, and dancing bees datasets. Background colors represent the different operating modes.

Table 1: Quantitative results on segmentation tasks. Accuracy, NMI, and ARI denote the frame-wise segmentation accuracy, the Normalized Mutual Information, and the Adjusted Rand Index metrics respectively (higher values are better). Mean and standard deviation are computed over 3 independent runs.

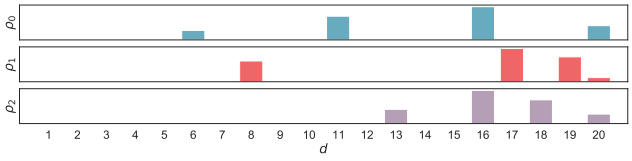
		bouncing ball	3 mode system	dancing bees	dancing bees(K=2)
Accuracy	SNLDS	**0.97±0.00**	0.82±0.08	0.44±0.01	0.63±0.02
	ED-SDS (ours)	0.95±0.00	0.97±0.00	0.56±0.06	0.79±0.09
	RED-SDS (ours)	**0.97±0.00**	**0.98±0.00**	**0.73±0.10**	**0.91±0.04**
NMI	SNLDS	**0.83±0.01**	0.63±0.08	0.10±0.04	0.05±0.02
	ED-SDS (ours)	0.71±0.00	0.89±0.01	0.28±0.02	0.31±0.17
	RED-SDS (ours)	0.81±0.00	**0.91±0.01**	**0.48±0.07**	**0.60±0.09**
ARI	SNLDS	**0.90±0.01**	0.67±0.11	0.10±0.03	0.07±0.02
	ED-SDS (ours)	0.81±0.01	0.93±0.00	0.27±0.04	0.36±0.19
	RED-SDS (ours)	0.88±0.00	**0.95±0.01**	**0.53±0.11**	**0.68±0.11**

We conducted experiments on three benchmark datasets: bouncing ball, 3 mode system, and dancing bees to investigate different segmentation capabilities of the models. We refer the reader to Appendix B.1 for details on how these datasets were generated/preprocessed. For all the datasets, we set the number of switches equal to the number of ground truth operating modes.

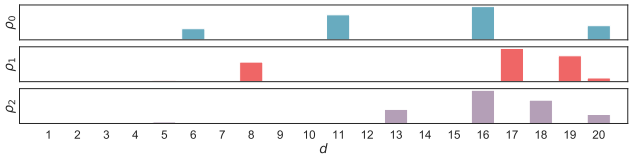
Bouncing ball. We generated the bouncing ball dataset similar to [13], which comprises univariate time series that encode the location of a ball bouncing between two fixed walls with a constant velocity and elastic collisions. The underlying system switches between two operating modes (going up/down) and the switching events are completely governed by the state of the ball, i.e., a switch occurs only when the ball hits a wall. As such, the switching events are best explained by state-to-switch recurrence. All models are able to segment this simple dataset well as shown qualitatively in Fig 3 (a) and quantitatively in Table 1. We note that despite the seemingly qualitative equivalence, models with state-to-switch recurrence perform best quantitatively. RED-SDS learns to ignore the duration variable by assigning almost all probability mass to shorter durations (cf. Appendix B.5), which is intuitive since the recurrence best explains this dataset.

3 mode system. We generated this dataset from a switching linear dynamical system with 3 operating modes and an explicit duration model for each mode (shown in Fig. 4 (a)). We study this dataset in the context of time-dependent switching—the operating mode switches after a specific amount of time elapses based on its duration model. Both ED-SDS and RED-SDS learn to segment this dataset almost perfectly as shown in Fig. 3 (b) and Table 1 owing to their ability to explicitly model switch durations. In contrast, SNLDS fails to completely capture the long-term temporal patterns, resulting in spurious short-term segments as shown in Fig. 3 (b). Moreover, RED-SDS is able to recover the duration models associated with the different modes (Fig. 4). These results demonstrate that explicit duration models can better identify the time-dependent switching patterns in the data and can leverage prior knowledge about the switch durations imparted via the $d_{\min}$ and $d_{\max}$ hyperparameters.

Dancing bees. We used the publicly-available dancing bees dataset [40]—a challenging dataset that exhibits long-term temporal patterns and has been studied previously in the context of time series segmentation [41, 39, 18]. The dataset comprises trajectories of six dancer honey bees performing the waggle dance. Each trajectory consists of the 2D coordinates and the heading angle of a bee at every timestep with three possible types of motion: waggle, turn right, and turn left. Fig. 3 (c) shows that RED-SDS is able to segment the complex long-term motion patterns quite well. In contrast, ED-SDS identifies the long segment durations but often infers the mode inaccurately while SNLDS strug-



(a) True duration model



(b) Learned duration model

Figure 4: The ground truth duration model for the 3 mode system dataset (top) and the duration model learned by RED-SDS (bottom). The x-axis represents the durations from 1 to 20 and the y-axis represents the duration probabilities of the 3 modes $\rho_0(d), \rho_1(d)$, and $\rho_2(d)$.

Table 2: CRPS metrics (lower is better). Mean and standard deviation are computed over 3 independent runs. The method achieving the best result is highlighted in **bold**.

	exchange	solar	electricity	traffic	wiki
DeepAR	0.019±0.002	0.440±0.004	**0.062±0.004**	0.138±0.001	0.855±0.552
DeepState	0.017±0.002	0.379±0.002	0.088±0.007	0.131±0.005	0.338±0.017
KVAE-MC	0.020±0.001	0.389±0.005	0.318±0.011	0.261±0.016	0.341±0.032
KVAE-RB	0.018±0.001	0.393±0.006	0.305±0.022	0.221±0.002	0.317±0.013
RSGLS-ISSM	0.014±0.001	**0.358±0.001**	0.091±0.004	0.206±0.002	0.345±0.010
ARSGLS	0.022±0.001	0.371±0.007	0.154±0.005	0.175±0.008	**0.283±0.006**
RED-SDS (ours)	**0.013±0.001**	0.419±0.010	0.066±0.002	**0.129±0.002**	0.318±0.006

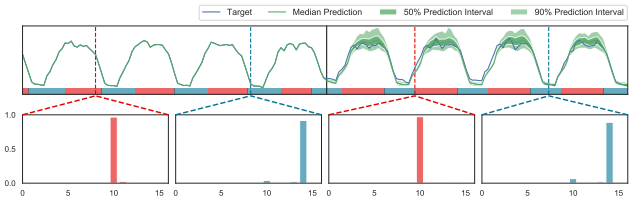
gles to learn the long-term motion patterns resulting in oversegmentation. This limitation of SNLDS is particularly apparent in the "waggle" phase of the dance which involves rapid, shaky motion. We also observed that sometimes ED-SDS and RED-SDS combined the turn right and turn left motions into a single switch, effectively segmenting the time series into regular (turn right and turn left) and waggle motion. This results in another reasonable segmentation, particularly in the absence of ground-truth supervision. We thus reevaluated the results after combining the turn right and turn left labels into a single label and present these results under `dancing bees(K=2)` in Table 1. Empirically, RED-SDS significantly outperforms ED-SDS and SNLDS on both labelings of the dataset. This suggests that real-world phenomena are better modeled by a combination of state- and time-dependent modeling capacities via state-to-switch recurrence and explicit durations, respectively.

5.2 Forecasting

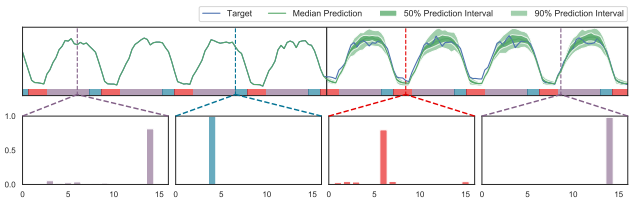
We evaluated RED-SDS in the context of time series forecasting on 5 popular public datasets available in GluonTS [4], following the experimental set up of [30]. The datasets have either hourly or daily frequency with various seasonality patterns such as daily, weekly, or composite. In Appendix C.1 we provide a detailed description of the datasets. We compared RED-SDS to closely related forecasting models: ARSGLS and its variant RSGLS-ISSM [30]; KVAE-MC and KVAE-RB, which refer to the original KVAE [19] and its Rao-Blackwellized variant (as described in [30]) respectively; DeepState [42]; and DeepAR [44], a strong *discriminative* baseline that uses an autoregressive RNN (cf. Appendix C.4 for a discussion on these baselines).

We used data prior to a fixed forecast date for training and test the forecasts on the remaining unseen data; the probabilistic forecasts are conditioned on the training range and computed with 100 samples for each method. We used a forecast window of 150 days and 168 hours for datasets with daily and hourly frequency, respectively. We evaluated the forecasts using the *continuous ranked probability score* (CRPS) [37], a proper scoring rule [22] (cf. Appendix C.2). The results are reported in Table 2; RED-SDS compares favorably or competitively to the baselines on 4 out of 5 datasets.

Figure 5 illustrates how RED-SDS can infer meaningful switching patterns from the data and extrapolate the learned patterns into the future. It perfectly reconstructs the past of the time series and segments it in an interpretable manner without an imposed



(a) $K = 2$



(b) $K = 3$

Figure 5: Segmentation and forecasting on an `electricity` time series for (a) $K = 2$ and (b) $K = 3$ switches. The black vertical line indicates the start of forecasting. The plots at the second row of each figure indicate the duration model at the timestep marked by the corresponding vertical dashed lines.

seasonality structure, e.g., as used in `DeepState` and `RSGLS-ISSM`. The same switching pattern is consistently predicted into the future, simplifying the forecasting problem by breaking the time series into different regimes with corresponding properties such as trend or noise variance. Further, the duration models at several timesteps (the duration model is conditioned on the control $\mathbf{u}_t$) indicate that the model has learned how long each regime lasts and therefore avoids oversegmentation which would harm the efficient modeling of each segment. Notably, the model learns meaningful regime durations that sum up to the 24-hour day/night period for both $K = 2$ and $K = 3$ switches. Thus, `RED-SDS` brings the added benefit of interpretability—both in terms of the discrete operating mode and the segment durations—while obtaining competitive quantitative performance relative to the baselines.

6 Conclusion and future work

Many real-world time series exhibit prolonged regimes of consistent dynamics as well as persistent statistical properties for the durations of these regimes. By explicitly modeling both state- and time-dependent switching dynamics, our proposed RED-SDS can more accurately model such data. Experiments on a variety of datasets show that RED-SDS—when equipped with an efficient inference algorithm that combines amortized variational inference with exact inference for continuous and discrete latent variables—improves upon existing models on segmentation tasks, while performing similarly to strong baselines for forecasting.

One current challenge of the proposed model is that learning interpretable segmentation sometimes requires careful hyperparameter tuning (e.g., $d_{\min}$ and $d_{\max}$). This is not surprising given the flexible nature of the neural networks used as components in the base dynamical system. A promising future research direction is to incorporate simpler models that have a predefined structure, thus exploiting domain knowledge. For instance, many forecasting models such as DeepState and RSGLS-ISSM parametrize classical level-trend and seasonality models in a non-linear fashion. Similarly, simple forecasting models with such structure could be used as base dynamical systems along with more flexible neural networks. Another interesting application is semi-supervised time series segmentation. For timesteps where the correct regime label is known, it is straightforward to condition on this additional information rather than performing inference; this may improve segmentation accuracy while providing an inductive bias that corresponds to an interpretable segmentation.

Funding disclosure

This work was funded by Amazon Research. This research is supported in part by the National Research Foundation Singapore under its AI Singapore Programme (Award Number: AISG-RP-2019-011) to H. Soh.

References

[1] G Ackerson and K Fu. On state estimation in switching environments. *IEEE transactions on automatic control*, 15(1), 1970.

[2] Ryan Prescott Adams and David JC MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007.

[3] Diego Agudelo-España, Sebastian Gomez-Gonzalez, Stefan Bauer, Bernhard Schölkopf, and Jan Peters. Bayesian online detection and prediction of change points. *arXiv preprint arXiv:1902.04524*, 2019.

[4] Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C Maddix, Syama Rangapuram, David Salinas, Jasper Schulz, et al. Gluonts: Probabilistic and neural time series modeling in python. *Journal of Machine Learning Research*, 21(116), 2020.

[5] Yaakov Bar-Shalom and Xiao-Rong Li. Estimation and tracking- principles, techniques, and software. *Norwood, MA: Artech House, Inc, 1993.*, 1993.

[6] David Barber. Expectation correction for smoothed inference in switching linear dynamical systems. *The Journal of Machine Learning Research*, 7, 2006.

[7] Philip Becker-Ehmck, Jan Peters, and Patrick Van Der Smagt. Switching linear dynamics for variational Bayes filtering. In *ICML*, 2019.

[8] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Bernie Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, et al. Neural forecasting: Introduction and literature overview. *arXiv preprint arXiv:2004.10240*, 2020.

[9] Silvia Chiappa. Explicit-duration markov switching models. *arXiv preprint arXiv:1909.05800*, 2019.

[10] Silvia Chiappa and Jan Peters. Movement extraction by detecting dynamics switches and repetitions. In *NeurIPS*, 2010.

[11] Hanjun Dai, Bo Dai, Yan-Ming Zhang, Shuang Li, and Le Song. Recurrent hidden semi-markov model. In *ICLR*, 2017.

[12] Emmanuel de Bézenac, Syama Sundar Rangapuram, Konstantinos Benidis, Michael Bohlke-Schneider, Richard Kurle, Lorenzo Stella, Hilaf Hasson, Patrick Gallinari, and Tim Januschowski. Normalizing kalman filters for multivariate time series analysis. In *NeurIPS*, 2020.

[13] Zhe Dong, Bryan A Seybold, Kevin P Murphy, and Hung H Bui. Collapsed amortized variational inference for switching nonlinear dynamical systems. In *ICML*, 2020.

[14] Arnaud Doucet, Nando de Freitas, Kevin P. Murphy, and Stuart J. Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *UAI*, 2000.

[15] Dheeru Dua, Casey Graff, et al. Uci machine learning repository. 2017.

[16] James Durbin and Siem Jan Koopman. *Time series analysis by state space methods*. Oxford university press, 2012.

[17] Fotios Petropoulos et. al. Forecasting: theory and practice, 2020.

[18] Emily B. Fox, Erik B. Sudderth, Michael I. Jordan, and Alan S. Willsky. Nonparametric bayesian learning of switching linear dynamical systems. In *NIPS*, 2008.

[19] Marco Fraccaro, Simon Kamronn, Ulrich Paquet, and Ole Winther. A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *NeurIPS*, 2017.

[20] Jan Gasthaus, Konstantinos Benidis, Yuyang Wang, Syama Sundar Rangapuram, David Salinas, Valentin Flunkert, and Tim Januschowski. Probabilistic forecasting with spline quantile function rnns. In *AISTATS*, 2019.

[21] Zoubin Ghahramani and Geoffrey E Hinton. Variational learning for switching state-space models. *Neural computation*, 12(4), 2000.

[22] Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477), 2007.

[23] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1), 1985.

[24] Tim Januschowski and Stephan Kolassa. A classification of business forecasting problems. *Foresight: The International Journal of Applied Forecasting*, (52), 2019.

[25] M. Johnson, D. Duvenaud, Alexander B. Wiltschko, Ryan P. Adams, and S. Datta. Composing graphical models with neural networks for structured representations and fast inference. In *NeurIPS*, 2016.

[26] Matthew J Johnson and Alan Willsky. The hierarchical dirichlet process hidden semi-markov model. *arXiv preprint arXiv:1203.3485*, 2012.

[27] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[28] Taesup Kim, Sungjin Ahn, and Yoshua Bengio. Variational temporal abstraction. In *NeurIPS*, 2019.

[29] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[30] Richard Kurle, Syama Sundar Rangapuram, Emmanuel de Bezenac, Stephan Gunnemann, and Jan Gasthaus. Deep Rao-blackwellised particle filters for time series forecasting. In *NeurIPS*, 2020.

[31] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2018.

[32] Uri N Lerner. *Hybrid Bayesian networks for reasoning about complex systems*. PhD thesis, Citeseer, 2002.

[33] Daniel Liberzon. *Switching in systems and control*. Springer Science & Business Media, 2003.

[34] Scott Linderman, Annika Nichols, David Blei, Manuel Zimmer, and Liam Paninski. Hierarchical recurrent state space models reveal discrete and continuous dynamics of neural activity in c. elegans. *bioRxiv*, 2019.

[35] Scott W Linderman, Andrew C Miller, Ryan P Adams, David M Blei, Liam Paninski, and Matthew J Johnson. Recurrent switching linear dynamical systems. *arXiv preprint arXiv:1610.08466*, 2016.

[36] Hao Liu, Lirong He, Haoli Bai, Bo Dai, Kun Bai, and Zenglin Xu. Structured inference for recurrent hidden semi-markov model. In *IJCAI*, 2018.

[37] James E Matheson and Robert L Winkler. Scoring rules for continuous probability distributions. *Management science*, 22(10), 1976.

[38] Kevin P Murphy. Hidden semi-markov models (HSMMs). *Unpublished notes*, 2, 2002.

[39] Sang Min Oh, James M Rehg, and Frank Dellaert. Parameterized duration mmodeling for switching linear dynamic systems. In *CVPR*, 2006.

[40] Sang Min Oh, James M Rehg, Tucker Balch, and Frank Dellaert. Learning and inferring motion patterns using parametric segmental switching linear dynamic systems. *International Journal of Computer Vision*, 77(1-3), 2008.

[41] Sangmin Oh, James M. Rehg, Tucker R. Balch, and Frank Dellaert. Learning and inference in parametric switching linear dynamic systems. *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, 2:1161–1168 Vol. 2, 2005.

[42] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. *NeurIPS*, 2018.

[43] Sam Roweis and Zoubin Ghahramani. A unifying review of linear gaussian models. *Neural computation*, 11(2), 1999.

[44] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3), 2020.

[45] Anuj Sharma, Robert Johnson, Florian Engert, and Scott Linderman. Point process latent variable models of larval zebrafish behavior. In *NeurIPS*, 2018.

[46] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

[47] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *The Journal of Machine Learning Research*, 11, 2010.

[48] Shun-Zheng Yu. Hidden semi-markov models. *Artificial intelligence*, 174(2), 2010.

[49] Onno Zoeter and Tom Heskes. Change point problems in linear dynamical systems. *The Journal of Machine Learning Research*, 6, 2005.

A Model: additional details

A.1 Forward-backward algorithm

As mentioned in Section 3.2, inference for the discrete latent variables, i.e., the counts $c_{1:T}$ and the switches $z_{1:T}$, can be performed exactly conditioned on states $\mathbf{x}_{1:T}$ and observations $\mathbf{y}_{1:T}$. We first define the forward α_t and backward β_t variables as

$$\alpha_t(z_t, c_t) = p(\mathbf{y}_{1:t}, \mathbf{x}_{1:t}, z_t, c_t), \tag{17}$$
$$\beta_t(z_t, c_t) = p(\mathbf{y}_{t+1:T}, \mathbf{x}_{t+1:T} | \mathbf{x}_t, z_t, c_t). \tag{18}$$

The smoothed posterior over the switches and counts, $p(z_t, c_t | \mathbf{y}_{1:T}, \mathbf{x}_{1:T})$, can be computed using α_t and β_t as

$$\begin{aligned}\gamma_t(z_t, c_t) = p(z_t, c_t | \mathbf{y}_{1:T}, \mathbf{x}_{1:T}) &\propto p(\mathbf{y}_{1:T}, \mathbf{x}_{1:T}, z_t, c_t) \\ &= p(\mathbf{y}_{t+1:T}, \mathbf{x}_{t+1:T} | \mathbf{x}_t, z_t, c_t) p(\mathbf{y}_{1:t}, \mathbf{x}_{1:t}, z_t, c_t) \\ &= \beta_t(z_t, c_t) \alpha_t(z_t, c_t).\end{aligned} \tag{19}$$

In the following, we derive the recursions for α_t and β_t, similar to the forward-backward algorithm for HSMMs [9, Chapter 3]:

$$\begin{aligned}\alpha_1(z_1, c_1) &= \delta_{c_1=1} p(\mathbf{y}_1, \mathbf{x}_1 | z_1) p(z_1), \\ \alpha_t(z_t, c_t) &= p(\mathbf{y}_{1:t}, \mathbf{x}_{1:t}, z_t, c_t) \\ &= \sum_{z_{t-1}} \sum_{c_{t-1}} p(\mathbf{y}_{1:t}, \mathbf{x}_{1:t}, z_{t-1}, z_t, c_{t-1}, c_t) \\ &= \sum_{z_{t-1}} \sum_{c_{t-1}} \Big[p(\mathbf{y}_t, \mathbf{x}_t | \mathbf{x}_{t-1}, z_t) p(z_t | \mathbf{x}_{t-1}, z_{t-1}, c_t) \\ &\qquad \cdot p(c_t | z_{t-1}, c_{t-1}) p(\mathbf{y}_{1:t-1}, \mathbf{x}_{1:t-1}, z_{t-1}, c_{t-1}) \Big] \\ &= \sum_{z_{t-1}} \sum_{c_{t-1}} p(\mathbf{y}_t, \mathbf{x}_t | \mathbf{x}_{t-1}, z_t) p(z_t | \mathbf{x}_{t-1}, z_{t-1}, c_t) p(c_t | z_{t-1}, c_{t-1}) \alpha(z_{t-1}, c_{t-1}) \\ &= p(\mathbf{y}_t, \mathbf{x}_t | \mathbf{x}_{t-1}, z_t) \Bigg[\delta_{\substack{z_{t-1}=z_t \\ c_t>1 \\ c_{t-1}=c_t-1}} v_{z_{t-1}}(c_{t-1}) \alpha(z_{t-1}, c_{t-1}) \\ &\qquad + \delta_{c_t=1} \sum_{z_{t-1}} p(z_t | \mathbf{x}_{t-1}, z_{t-1}, c_t) \sum_{c_{t-1}} (1 - v_{z_{t-1}}(c_{t-1})) \alpha(z_{t-1}, c_{t-1}) \Bigg],\end{aligned}$$

$$\begin{aligned}\beta_T(z_T, c_T) &= 1, \\ \beta_t(z_t, c_t) &= p(\mathbf{y}_{t+1:T}, \mathbf{x}_{t+1:T} | \mathbf{x}_t, z_t, c_t) \\ &= \sum_{z_{t+1}, c_{t+1}} p(\mathbf{y}_{t+1:T}, \mathbf{x}_{t+1:T}, z_{t+1}, c_{t+1} | \mathbf{x}_t, z_t, c_t) \\ &= \sum_{z_{t+1}, c_{t+1}} \Big[p(\mathbf{y}_{t+2:T}, \mathbf{x}_{t+2:T} | \mathbf{x}_{t+1}, z_{t+1}, c_{t+1}) p(\mathbf{y}_{t+1}, \mathbf{x}_{t+1} | \mathbf{x}_t, z_{t+1}) \\ &\qquad \cdot p(z_{t+1} | \mathbf{x}_t, z_t, c_{t+1}) p(c_{t+1} | z_t, c_t) \Big] \\ &= \delta_{\substack{z_{t+1}=z_t \\ c_{t+1}=c_t+1}} \beta_{t+1}(z_{t+1}, c_{t+1}) p(\mathbf{y}_{t+1}, \mathbf{x}_{t+1} | \mathbf{x}_t, z_{t+1}) v_{z_t}(c_t) \\ &\quad + \delta_{\substack{c_t \geq d_{\min} \\ c_{t+1}=1}} (1 - v_{z_t}(c_t)) \sum_{z_{t+1}} \beta_{t+1}(z_{t+1}, c_{t+1}) p(\mathbf{y}_{t+1}, \mathbf{x}_{t+1} | \mathbf{x}_t, z_{t+1}) p(z_{t+1} | \mathbf{x}_t, z_t, c_{t+1}).\end{aligned}$$

A.2 RED-SDS instantiation

In this section, we present our instantiation of RED-SDS, providing the general structure of the architecture and the functions f_z, f_x^μ, f_x^Σ, f_y^μ, f_y^Σ (cf. Section 3.1). For specific implementation details we refer the reader to Appendices B.3 and C.3 for segmentation and forecasting, respectively.

Control embedding network. Control inputs can be either static, i.e., constant for all timesteps per time series (e.g., time series ID), or time-dependent (e.g., time embedding or other covariates). We denote the raw control input as $\mathbf{u}_{\text{raw}} = [\mathbf{u}^{\text{static}}, \mathbf{u}^{\text{time}}]$, where $\mathbf{u}^{\text{static}}$ and $\mathbf{u}^{\text{time}}$ correspond to the static and time-dependent control features, respectively.

In our forecasting experiments, we consider static features that are natural numbers (corresponding to time series IDs). The static features are first passed through an embedding layer. The embedding of the static features is then concatenated with the time-dependent features and the result is fed into a single hidden layer MLP that outputs the control input $\mathbf{u}_t \in \mathbb{R}^c$. This process is described by the following operations:

$$\mathbf{u}_t = f_u\left(g_{\text{emb}}^u(\mathbf{u}_t^{\text{static}}), \mathbf{u}_t^{\text{time}}\right), \tag{20}$$

where g_{emb}^u denotes the embedding layer and f_u is the MLP.

Inference network. The observations $\mathbf{y}_{1:T}$ are first passed through an embedding network g_{emb}^y, that can either be a bi-RNN or a Transformer, which outputs an embedding $\mathbf{h}_{1:T}^1$ of the observations. The embedding is then fed into a causal RNN g_{rnn} along with the control inputs $\mathbf{u}_{1:T}$ which outputs hidden states $\mathbf{r}_{1:T}$. A single hidden layer MLP g_{fc} then maps the hidden states $\mathbf{r}_{1:T}$ to the parameters of the Gaussian distribution $q(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{h}_{1:T}^1)$. This inference network is described by the following operations:

$$\mathbf{h}_{1:T}^1 = g_{\text{emb}}^y(\mathbf{y}_{1:T}), \tag{21}$$

$$\mathbf{r}_t = g_{\text{rnn}}(\mathbf{x}_{t-1}, \mathbf{r}_{t-1}, \mathbf{u}_t, \mathbf{h}_t^1), \tag{22}$$

$$\mathbf{x}_t \sim \mathcal{N}\left(\mathbf{x}_t; g_{\text{fc}}^\mu(\mathbf{r}_t), g_{\text{fc}}^\Sigma(\mathbf{r}_t)\right), \tag{23}$$

where $\mathbf{x}_0$ and $\mathbf{r}_0$ are zero vectors.

Duration network. When no control input is available, the duration model is a learnable matrix $\mathbf{P} \in \mathbb{R}^{K \times (d_{\max} - d_{\min} + 1)}$, fixed across all timesteps; the k-th row represents the logits of the duration model ρ_k for the k-th switch. When control inputs are used, the duration model depends on the control input and is no longer fixed across timesteps. The control input $\mathbf{u}_t$ (see Eq. 20) is fed into a single hidden layer MLP which outputs a (time-varying) matrix $\mathbf{P}_t \in \mathbb{R}^{K \times d_{\max}}$, i.e.,

$$\mathbf{P}_t = f_d(\mathbf{u}_t). \tag{24}$$

where $\mathbf{P}_t$ has the same structure as $\mathbf{P}$. The first $d_{\min} - 1$ columns are masked and the final duration models ρ_k are obtained by applying the tempered softmax function to the rows of $\mathbf{P}_t$,

$$\rho_k(d)_t = \text{Cat}(d; \mathcal{S}_{\tau_\rho}(\mathbf{P}_t^{k,:})), \tag{25}$$

where $\mathbf{P}_t^{k,:}$ denotes the k-th row of $\mathbf{P}_t$ and $\mathcal{S}_{\tau_\rho}$ denotes the tempered softmax function with temperature τ_ρ.

Discrete transition network. The pseudo-observations of the states $\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$, sampled from the inference network, along with the control inputs $\mathbf{u}_2, \ldots, \mathbf{u}_T$, are passed to the neural network f_z (for the case when $c_t = 1$), a single hidden layer MLP, to model the discrete transition distribution,

$$p(z_t|\mathbf{x}_{t-1}, z_{t-1}, c_t, \mathbf{u}_t) = \begin{cases} \delta_{z_t = z_{t-1}} & \text{if} \quad c_t > 1 \\ \text{Cat}(z_t; \mathcal{S}_{\tau_z}(f_z(\mathbf{x}_{t-1}, z_{t-1}, \mathbf{u}_t))) & \text{if} \quad c_t = 1 \end{cases}, \tag{26}$$

where $\mathcal{S}_{\tau_z}$ denotes the tempered softmax function with temperature τ_z. The network f_z takes $\mathbf{x}_{t-1}$ and $\mathbf{u}_t$ as input and outputs a matrix $\tilde{\mathbf{A}}_t \in \mathbb{R}^{K \times K}$. Each row of the matrix $\tilde{\mathbf{A}}_t \in \mathbb{R}^{K \times K}$ is normalized using $\mathcal{S}_{\tau_z}$ to obtain the stochastic transition matrix $\mathbf{A}_t$ where each row represents a categorical distribution that can be indexed by z_{t-1}.

Continuous transition network. The continuous transition network f_x is a linear function or a single hidden layer MLP that models the continuous transition distribution

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}, z_t, \mathbf{u}_t) = \mathcal{N}\left(\mathbf{x}_t; f_x^{\mu}(\mathbf{x}_{t-1}, z_t, \mathbf{u}_t), f_x^{\Sigma}(\mathbf{x}_{t-1}, z_t, \mathbf{u}_t)\right). \tag{27}$$

The function f_x takes $\mathbf{x}_{t-1}$ and $\mathbf{u}_t$ as input and outputs the parameters of the Gaussian distribution. The dependence on z_t is realized by using separate functions f_x^k for the K unique values of the switch z_t.

Emission network. The emission network f_y is a linear function or an MLP with two hidden layers that models the emission distribution

$$p(\mathbf{y}_t|\mathbf{x}_t) = \mathcal{N}\left(\mathbf{y}_t; f_y^{\mu}(\mathbf{x}_t), f_y^{\Sigma}(\mathbf{x}_t)\right). \tag{28}$$

A.3 Applications

In this section, we describe how to perform time series segmentation and generate probabilistic forecasts using RED-SDS.

A.3.1 Segmentation

We perform time series segmentation by labeling every timestep with the most likely switch. This is done by first computing the posterior distribution $\gamma_t(z_t, c_t)$ (Eq. 19) for each timestep and then obtaining the most likely switch $\hat{k}_t$ by marginalizing out the count variable c_t as follows

$$\hat{k}_t = \underset{j}{\operatorname{argmax}} \sum_{d=d_{\min}}^{d_{\max}} \gamma_t(z_t = j, c_t = d). \tag{29}$$

A.3.2 Forecasting

We generate probabilistic forecasts by generating multiple future sample paths. Let $\mathbf{y}_{1:T}$ be an input time series and τ the forecast horizon. We begin by generating M state samples from the variational posterior $q(\mathbf{x}_T|\mathbf{y}_{1:T})$ and the corresponding M switch-count pairs from the posterior over the switches and counts $p(z_T, c_T|\mathbf{y}_{1:T}, \hat{\mathbf{x}}_{1:T})$. These M triplets of state, switch, and count are then used to unroll the generative model into the future for τ timesteps generating M sample paths (forecasts). Algorithm 1 describes the steps involved to generate one such sample path.

Algorithm 1 RED-SDS future unrolling

INPUT
Time series $\mathbf{y}_{1:T}$, forecast horizon τ
SAMPLE STATES AT T
Continuous state sample from the variational posterior $\hat{\mathbf{x}}_T \sim q(\mathbf{x}_T|\mathbf{y}_{1:T})$
Discrete state and duration samples $\hat{z}_T, \hat{c}_T \sim p(z_T, c_T|\mathbf{y}_{1:T}, \hat{\mathbf{x}}_{1:T}) = \gamma_T(z_T, c_T)$, where $\gamma_T(z_T, c_T)$ is computed using the forward-backward algorithm
Set $\hat{\mathbf{y}}_T = \mathbf{y}_T$
UNROLL IN FORECAST HORIZON
for $t = T+1 : T+\tau$ **do**
$\hat{c}_t \sim p(c_t|\hat{z}_{t-1}, \hat{c}_{t-1})$
$\hat{z}_t \sim p(z_t|\hat{\mathbf{x}}_{t-1}, \hat{z}_{t-1}, \hat{c}_t)$
$\hat{\mathbf{x}}_t \sim p(\mathbf{x}_t|\hat{\mathbf{x}}_{t-1}, \hat{z}_t)$
$\hat{\mathbf{y}}_t \sim p(\mathbf{y}_t|\hat{\mathbf{x}}_t)$
end for
RETURN
Predictive samples $\hat{\mathbf{y}}_{T+1:T+\tau}$

B Details on segmentation experiments

B.1 Datasets

Bouncing ball. The bouncing ball dataset comprises univariate time series $y_{1:T}$, with $y_t \in \mathbb{R}$, that encode the location of a ball bouncing between two fixed walls with constant absolute velocity and elastic collisions between the ball and the wall(s). The distance between the walls was set to 10 with the initial location of the ball randomly generated between the walls. The initial velocity was sampled from the uniform distribution $\mathcal{U}(-0.5, 0.5)$ and its sign was flipped every time the ball went beyond 0 or 10. The final observation was generated by adding $\epsilon \sim \mathcal{N}(0, 0.1^2)$ to the ball's location. The ground truth label was assigned based on the sign of the velocity. We generated 100000 and 1000 time series of 100 timesteps for the train and the test datasets respectively.

3 mode system. The 3 mode system dataset comprises univariate time series generated from a switching linear dynamical system with 3 modes and an explicit duration model for each mode. The dimensionality of the state variables $\mathbf{x}_t$ was set to 2. The initial switch and state distributions were given by

$$p(z_1) = \mathcal{U}_{\text{cat}}\{1, 2, 3\}, \tag{30}$$

$$p(\mathbf{x}_1|z_1) = \mathcal{N}\left(\begin{bmatrix}2\\0\end{bmatrix}, 0.01\mathbf{I}\right), \tag{31}$$

where $\mathcal{U}_{\text{cat}}$ denotes the uniform categorical distribution.

The transition distribution of the increasing count variables was given by

$$p(c_t|z_{t-1}=k, c_{t-1}) = \begin{cases} v_k(c_{t-1}) & \text{if} \quad c_t = c_{t-1}+1 \\ 1 - v_k(c_{t-1}) & \text{if} \quad c_t = 1 \end{cases}, \tag{32}$$

where the probability of a count increment v_k for the switch k is defined as in Eq. (2) via the duration models ρ_k, given by

$$\rho_1(d) = \text{Cat}\left(\left[\tfrac{2}{17} \quad 0 \quad 0 \quad 0 \quad 0 \quad \tfrac{5}{17} \quad 0 \quad 0 \quad 0 \quad 0 \quad \tfrac{7}{17} \quad 0 \quad 0 \quad 0 \quad \tfrac{3}{17}\right]\right), \tag{33}$$

$$\rho_2(d) = \text{Cat}\left(\left[0 \quad 0 \quad \tfrac{1}{4} \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad \tfrac{2}{5} \quad 0 \quad \tfrac{3}{10} \quad \tfrac{1}{20}\right]\right), \tag{34}$$

$$\rho_3(d) = \text{Cat}\left(\left[0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad \tfrac{3}{17} \quad 0 \quad 0 \quad \tfrac{7}{17} \quad 0 \quad \tfrac{5}{17} \quad 0 \quad \tfrac{2}{17}\right]\right), \tag{35}$$

with $d_{\min} = 6$ and $d_{\max} = 20$.

The switch transition distribution was given by

$$p(z_t = j|z_{t-1} = i, c_t) = \begin{cases} \delta_{z_t=i} & \text{if} \quad c_t > 1 \\ \begin{bmatrix} 0.1 & 0.2 & 0.7 \\ 0.3 & 0.5 & 0.2 \\ 0.3 & 0.3 & 0.4 \end{bmatrix}_{(i,j)} & \text{if} \quad c_t = 1 \end{cases}. \tag{36}$$

The state transition distribution was given by

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}, z_t = k) = \mathcal{N}(\mathbf{A}_k\mathbf{x}_{t-1} + \mathbf{b}_k, 0.01\mathbf{I}), \tag{37}$$

where the $\mathbf{A}$s and $\mathbf{b}$s were defined as follows

$$\mathbf{A}_1 = 0.99\begin{bmatrix}\cos(0) & -\sin(0) \\ \sin(0) & \cos(0)\end{bmatrix}, \tag{38}$$

$$\mathbf{b}_1 = \mathbf{0}, \tag{39}$$

$$\mathbf{A}_2 = 0.99\begin{bmatrix}\cos(\pi/8) & -\sin(\pi/8) \\ \sin(\pi/8) & \cos(\pi/8)\end{bmatrix}, \tag{40}$$

$$\mathbf{b}_2 = 0.25\boldsymbol{\epsilon}_2, \tag{41}$$

$$\mathbf{A}_3 = 0.99\begin{bmatrix}\cos(\pi/4) & -\sin(\pi/4) \\ \sin(\pi/4) & \cos(\pi/4)\end{bmatrix}, \tag{42}$$

$$\mathbf{b}_3 = 0.25\boldsymbol{\epsilon}_3, \tag{43}$$

with $\boldsymbol{\epsilon}_2$, $\boldsymbol{\epsilon}_3$ sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

The emission distribution was given by

$$p(\mathbf{y}_t|\mathbf{x}_t, z_t = k) = \mathcal{N}\left(\mathbf{c}_k^\top \mathbf{x}_t + d_k, 0.04\mathbf{I}\right), \tag{44}$$

with $\mathbf{c}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $d_k \sim \mathcal{U}_{\text{cat}}\{0, 1, 2\}$.

The ground truth label was assigned based on z_t. We generated 10000 and 500 time series of 180 timesteps for the train and the test datasets, respectively.

Dancing bees. We used the publicly-available[3] dancing bees dataset [40], which comprises tracks of six dancer honey bees that were obtained using a vision-based tracker. The time series consist of the 2D coordinates (x_t, y_t) and the heading angle (θ_t) of the honey bee at every timestep. Each timestep is labeled as one of the three dance types: waggle, turn right, and turn left. The six long time series have lengths 1058, 1125, 1054, 757, 609 and 814. We first standardized the 2D coordinates (x_t, y_t) for each time series by subtracting the mean and dividing by the standard deviation to get the normalized coordinates $(\hat{x}_t, \hat{y}_t)$. We then constructed 4-dimensional time series with elements $[\hat{x}_t, \hat{y}_t, \sin(\theta_t), \cos(\theta_t)]$. Each time series was split into chunks of 120 timesteps starting at every changepoint in the ground truth label (discarding terminal chunks less than 120 timesteps). We used time series $\{1, 3, 4, 5, 6\}$ for training and the longest time series $\{2\}$ for testing, resulting in 84 and 21 data points in the train and test datasets, respectively.

B.2 Segmentation metrics

The segment label ordering assigned by a model such as RED-SDS—trained without label supervision—is arbitrary and may not match with the ground truth label ordering. Thus, we evaluated the segmentation performance using three metrics that are independent of remappings of the predicted labels:

(a) Frame-wise segmentation accuracy: This metric computes the percentage of predicted labels that match the ground truth labels. The predicted labels were first remapped using the Hungarian algorithm [29] which finds the mapping that maximizes the accuracy. We used `linear_sum_assignment` from `scipy` to find the optimal mapping.

(b) Normalized Mutual Information (NMI) [47]: NMI computes the mutual information between two labelings, normalized to lie between 0 (no mutual information) and 1 (perfect correlation). The metric is independent of the actual values of labels, i.e., a remapping of the labels won't change the score. We used `normalized_mutual_info_score` from `scikit-learn` to compute the NMI score.

(c) Adjusted Rand Index (ARI) [23]: ARI computes a similarity measure between two labelings which is adjusted for chance. The metric lies between -1 to 1 where a random labeling has a score close to 0 and a score of 1 indicates a perfect match. Like NMI, ARI is independent of the actual values of the labels. We used `adjusted_rand_score` from `scikit-learn` to compute the ARI score.

B.3 Training and hyperparameter details

In this section, we discuss the training and hyperparameter details for the segmentation experiments.

Training parameters. We trained all the datasets with a fixed batch size of 32 for 20000 training steps. We used the Adam optimizer for the gradient updates with 10^{-5} weight-decay and clipped the gradient norm to 10. The learning rate was warmed-up linearly from a lower value in $\{5 \times 10^{-5}, 1 \times 10^{-4}\}$ to $\{2 \times 10^{-4}, 5 \times 10^{-3}\}$ for the first $\{2000, 1000\}$ steps after which a cosine decay follows for the remaining time steps with a decay rate of 0.99.

Network types. We experimented with linear and non-linear functions for the continuous transition f_x and emission f_y functions. The linear function implies multiplication with a transformation matrix while the non-linear function was an MLP with ReLU non-linearity. For the inference embedding

[3] `https://www.cc.gatech.edu/~borg/ijcv_psslds/`

Table 3: Network architectures for the different components of the model. `Linear` denotes a linear layer without bias, `MLP` $[a_1, \ldots, a_l]$ denotes an l-hidden-layer MLP with hidden units $a_1, \ldots, a_l$ and ReLU non-linearity, `biGRU` $[b]$ denotes a single-layer bidirectional GRU with b hidden units, and `RNN` $[c]$ denotes a single-layer RNN with c hidden units.

Network	Datasets		
	`bouncing ball`	`3 mode system`	`dancing bees`
Discrete Transition (f_z)	`MLP` $[4 \times 2^2]$	`MLP` $[4 \times 3^2]$	`MLP` $[4 \times 3^2]$
Continuous Transition (f_x)	`MLP` [32]	`MLP` [32]	`Linear`
Emission Network (f_y)	`MLP` [8, 32]	`MLP` [8, 32]	`Linear`
Inference Embedder (g^y_{emb})	`biGRU` [4]	`biGRU` [4]	`biGRU` [16]
Causal RNN (g_{rnn})	`RNN` [16]	`RNN` [16]	`RNN` [16]
Parameter Network (g_{fc})	`MLP` [32]	`MLP` [32]	`MLP` [32]

Table 4: Temperature annealing details.

(a) Annealing schedules of the switch (τ_z) and duration (τ_ρ) temperatures.

Hyperparameter	Datasets		
	`bouncing ball`	`3 mode system`	`dancing bees`
Initial Temp.	τ_z, τ_ρ: 10	τ_z, τ_ρ: 10	τ_z: 100, τ_ρ: 10
Min Temp.	τ_z, τ_ρ: 1	τ_z, τ_ρ: 1	τ_z: 10, τ_ρ: 1
Decay Rate	0.99	0.99	0.95
Begin Decay Step	1000	1000	5000
Decay Every (steps)	50	50	100

(b) ✓ indicates that the temperature is annealed using the schedule in (a) and ✗ indicates that the temperature is kept fixed.

Model	Variable	Datasets		
		`bouncing ball`	`3 mode system`	`dancing bees`
`SNLDS`	Switch	✓	✓	✓
`ED-SDS`	Switch	✗	✗	✓
	Duration	✓	✓	✓
`RED-SDS`	Switch	✗	✗	✓
	Duration	✗	✓	✓

network g^y_{emb}, we used a single layer bidirectional GRU. The causal RNN g_{rnn} was a single layer forward RNN and the parameter network g_{fc} was a single hidden layer MLP. Table 3 summarizes the network architectures of the different model components for the three datasets.

Temperature Annealing. As discussed in Section 3.3, we annealed the temperature parameter of the tempered softmax function (Eq. 16) to encourage the model to explore all switches and durations during the initial training phase. We decayed the temperature parameter exponentially from a initial value to a minimum value and report the details on annealing schedules in Table 4.

Duration. We used the $(d_{\min}, d_{\max})$ hyperparameters to impart prior knowledge to the model about the switch durations. These parameters were set to $(1, 20)$ for the bouncing ball dataset. For the 3 mode system and dancing bees datasets, we used larger values of $(5, 20)$ and $(20, 50)$ respectively to encourage the model to learn longer switch durations as exhibited by these datasets.

Number of switches. The number of switches K plays an important role in the model, particularly for segmentation. For all the datasets considered in our experiments, the number of operating modes is known; therefore, we set K to the number of ground truth operating modes.

Dimensionality of state $\mathbf{x}_t$. The dimensionality of the state variables $\mathbf{x}_t$ was tuned from the set $\{2, 4\}$ for the univariate datasets (bouncing ball and 3 mode system) and was set to 8 for the multivariate dataset (dancing bees).

B.4 Baseline models

B.4.1 SNLDS

In this section, we discuss the objective function of SNLDS proposed by Dong et al. [13] and the modified version used in our experiments. As mentioned in the main text, SNLDS can be viewed as an ablated version of RED-SDS without the explicit duration models, i.e., with $d_{\min} = d_{\max} = 1$.

The ELBO for SNLDS is given by

$$\text{ELBO} = \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})p(z_{1:T}|\mathbf{y}_{1:T},\mathbf{x}_{1:T})}\left[\log \frac{p(\mathbf{y}_{1:T},\mathbf{x}_{1:T},z_{1:T})}{q(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})p(z_{1:T}|\mathbf{y}_{1:T},\mathbf{x}_{1:T})}\right] \tag{45}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})}\left[\log \frac{p(\mathbf{y}_{1:T},\mathbf{x}_{1:T})}{q(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})}\right]. \tag{46}$$

To estimate the gradients of $\log p(\mathbf{y}_{1:T},\mathbf{x}_{1:T})$ in Eq. (46), Dong et al. [13] proposed auto-differentiating through the following expression,

$$\mathbb{E}_{p(z_{1:T}|\mathbf{y}_{1:T},\mathbf{x}_{1:T})}\left[\log p(\mathbf{y}_{1:T},\mathbf{x}_{1:T},z_{1:T})\right] = \sum_{t=2}^{T}\sum_{j,k}\xi_t(j,k)\left[\log B_t(k)A_t(j,k)\right] + \sum_{k}\gamma_1(k)\left[\log B_1(k)\pi_k\right], \tag{47}$$

where

$$\pi(k) = p(z_1 = k), \tag{48}$$

$$\gamma(k) = p(z_t = k|\mathbf{y}_{1:T},\mathbf{x}_{1:T}), \tag{49}$$

$$\xi(j,k) = p(z_t = k, z_{t-1} = j|\mathbf{y}_{1:T},\mathbf{x}_{1:T}), \tag{50}$$

$$B_t(k) = p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{x}_{t-1}, z_t = k), \tag{51}$$

$$A_t(j,k) = p(z_t = j|\mathbf{y}_{t-1}, z_{t-1} = k). \tag{52}$$

However, this makes the model prone to state collapse, i.e., the model ends up only using a single switch, as also noted by [13]. This led them to add an ad-hoc cross-entropy regularizer to the objective function that discourages state collapse during the initial phase of training. The regularizer minimizes the KL divergence between a uniform prior on the switches and the smoothed posterior over switches,

$$\mathcal{L}_{\text{CE}} = \sum_{t=1}^{T}\mathcal{D}_{KL}(p_{\text{prior}}(z_t)\|\gamma(z_t)), \tag{53}$$

where $p_{\text{prior}}(z_t) = \prod_j \left[\frac{1}{K}\right]^{\mathbf{1}[j=z_t]}$.

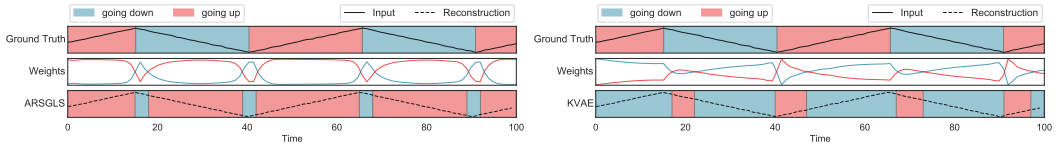
During our initial experiments with SNLDS, we observed frequent state collapse when using the approximation in Eq. (47) and the training was sensitive to the annealing schedule of $\mathcal{L}_{\text{CE}}$. We further noted that the likelihood term $p(\mathbf{y}_{1:T},\mathbf{x}_{1:T})$ in Eq. (46) can be computed exactly (same as in RED-SDS) using the SNLDS forward variable $\alpha_T(z_T) = p(\mathbf{y}_{1:T},\mathbf{x}_{1:T},z_T)$ as

$$p(\mathbf{y}_{1:T},\mathbf{x}_{1:T}) = \sum_{z_T}\alpha_T(z_T). \tag{54}$$

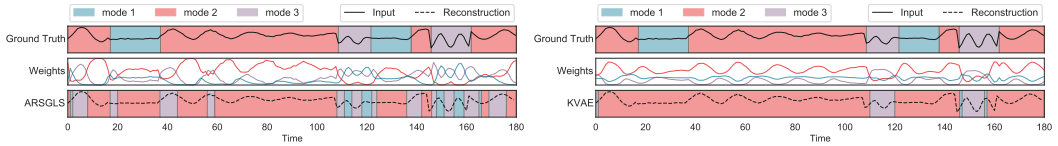
Therefore, we used Eq. (54) for $p(\mathbf{y}_{1:T},\mathbf{x}_{1:T})$ in our experiments which was less prone to state collapse and didn't require the sensitive KL regularization term (Eq. 53).

B.4.2 Soft-switching models

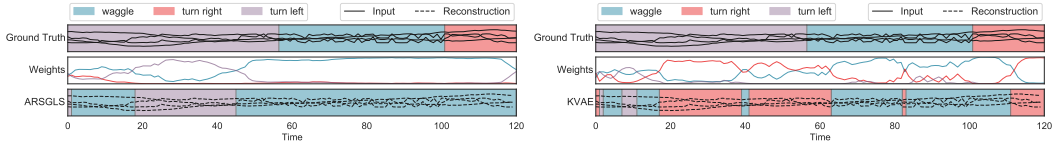
We conducted experiments using two soft-switching models: ARSGLS and KVAE. The ARSGLS extends the classical SLDS by conditionally linear state-to-switch connections and an auxiliary



(a) Bouncing ball



(b) 3 mode system



(c) Dancing bees

Figure 6: Qualitative segmentation results on the bouncing ball, 3 mode system, and dancing bees datasets for the soft-switching models: ARSGLS (left) and KVAE (right). Background colors represent the different operating modes.

variable along with a neural network (decoder) that allows it to model more complex non-linear emission models and multivariate observations. However, the switch variables in ARSGLS are Gaussian rather than categorical, such that the recurrent state-to-switch connection does not break the computational tractability of the conditionally linear dynamical system. Instead of "indexing" a distinct base LDS at each timestep, the parameters are predicted as a weighted average of the matrices of the base LDSs. These combination weights are predicted by a neural network with a softmax transformation which takes the Gaussian switch variables as input.

Similarly, the KVAE uses an LSTM (followed by an MLP) to predict the weights for averaging the base matrices, where the LSTM takes the latent variables embedded by a variational autoencoder as inputs.

As both ARSGLS and KVAE are trained using a continuous interpolation of operating modes, they cannot always correctly assign a single discrete operating mode to every timestep during test time. Fig. 6 shows qualitative segmentation results for ARSGLS and KVAE trained on the bouncing ball, 3 mode system, and dancing bees datasets along with the combination weights at each timestep (middle of each plot). The segment labels were assigned by taking the $\arg\max$ of the combination weights. Although the combination weights follow some interpretable patterns (e.g., in the bouncing ball and 3 mode system datasets), they cannot assign each timestep to a single mode. Thus, both ARSGLS and KVAE perform poorly on these segmentation tasks.

B.5 Additional results

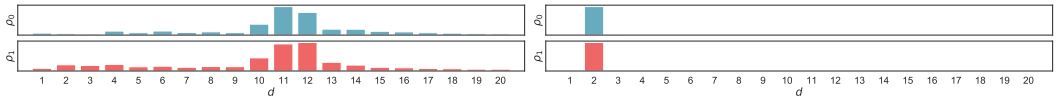
Table 5 presents segmentation results on the bouncing ball and 3 mode system datasets with the dimenionality of the state $\mathbf{x}_t$ equal to 2 (ground truth) and 4. All the three models perform similarly for $\dim(\mathbf{x}_t) = 2$ and $\dim(\mathbf{x}_t) = 4$ on the bouncing ball dataset. On the 3 mode system dataset, RED-SDS performs better with $\dim(\mathbf{x}_t) = 4$ than $\dim(\mathbf{x}_t) = 2$.

Fig. 7 shows the duration models learned by ED-SDS and RED-SDS. On the bouncing ball dataset, RED-SDS assigns all the probability mass to shorter durations indicating that the state-to-switch recurrence is more informative about the switching behavior in this dataset. In contrast, ED-SDS lacks state-to-switch recurrence and assigns probability mass to longer durations. On the 3 mode system dataset, both ED-SDS and RED-SDS recover the ground truth duration model. On the dancing

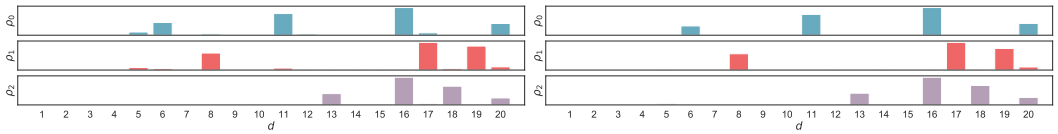
bees dataset, both ED-SDS and RED-SDS assign probability mass to longer durations indicating the existence of long-term temporal patterns.

Table 5: Quantitative results on the bouncing ball and 3 mode system datasets with different values of $\dim(\mathbf{x}_t)$.

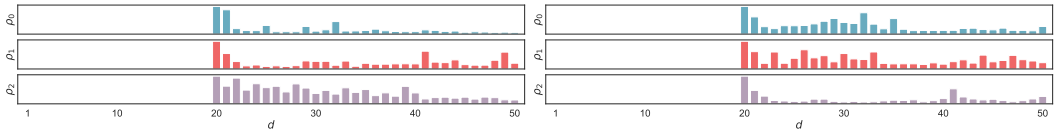
Metric	Model	$\dim(\mathbf{x}_t)$	Datasets	
			`bouncing ball`	`3 mode system`
Accuracy	`SNLDS`	2	**0.97±0.00**	0.82±0.14
		4	**0.97±0.00**	0.82±0.08
	`ED-SDS` (ours)	2	0.95±0.00	0.97±0.00
		4	0.94±0.00	0.97±0.00
	`RED-SDS` (ours)	2	**0.97±0.00**	0.95±0.00
		4	**0.97±0.00**	**0.98±0.00**
NMI	`SNLDS`	2	**0.83±0.01**	0.63±0.12
		4	0.82±0.01	0.63±0.08
	`ED-SDS` (ours)	2	0.71±0.00	0.89±0.00
		4	0.70±0.01	0.88±0.01
	`RED-SDS` (ours)	2	0.80±0.00	0.82±0.01
		4	0.81±0.00	**0.91±0.01**
ARI	`SNLDS`	2	**0.90±0.01**	0.67±0.17
		4	0.89±0.01	0.67±0.11
	`ED-SDS` (ours)	2	0.81±0.01	0.93±0.00
		4	0.79±0.01	0.93±0.01
	`RED-SDS` (ours)	2	0.88±0.01	0.89±0.01
		4	0.88±0.00	**0.95±0.01**



(a) Bouncing ball



(b) 3 mode system



(c) Dancing bees

Figure 7: Duration models learned by ED-SDS (left) and RED-SDS (right) for the bouncing ball, 3 mode system, and dancing bees datasets.

C Details on forecasting experiments

C.1 Datasets

We evaluated RED-SDS on the following five publicly available datasets, commonly used for evaluation forecasting models:

- `exchange`: daily exchange rate between 8 currencies as used in [31];

- `solar`: hourly photo-voltaic production of 137 stations in Alabama State used in [31];
- `electricity`: hourly time series of the electricity consumption of 370 customers [15];
- `traffic`: hourly occupancy rate, between 0 and 1, of 963 San Francisco car lanes [15];
- `wiki`: daily page views of 2000 Wikipedia pages used in [20].

Similar to [30], the forecasts of different methods are evaluated by splitting each dataset in the following fashion: all data prior to a fixed *forecast start date* comprise the training set and the remainder is used as the test set.

In Table 6, we provide an overview of the different properties of these datasets.

Table 6: Overview of the datasets used to test the forecasting accuracy of the models.

dataset	domain	frequency	# of time series	# of timesteps	T (context length)	τ (forecast horizon)
`exchange`	$\mathbb{R}^+$	daily	8	6071	124	150
`solar`	$\mathbb{R}^+$	hourly	137	7009	336	168
`electricity`	$\mathbb{R}^+$	hourly	370	5790	336	168
`traffic`	$\mathbb{R}^+$	hourly	963	10413	336	168
`wiki`	$\mathbb{N}$	daily	2000	792	124	150

C.2 Forecasting metric: CRPS

The continuous ranked probability score (CRPS) [37] is a proper scoring rule [22] that measures the compatibility of a quantile function F^{-1} with an observation y. It has an intuitive definition as the pinball loss integrated over all quantile levels $\alpha \in [0, 1]$, i.e.,

$$\text{CRPS}(F^{-1}, y) = \int_0^1 2\Lambda_\alpha(F^{-1}(\alpha), y)\, d\alpha, \tag{55}$$

where the pinball loss $\Lambda_\alpha(q, y)$ is defined as

$$\Lambda_\alpha(q, y) = (\alpha - \mathbb{1}_{\{y<q\}})(y - q), \tag{56}$$

with q being the respective quantile of the probability distribution.

We used the CRPS implementation provided in GluonTS [4] that approximates the integral of (55) with a grid of selected quantiles.

C.3 Training and hyperparameter details

In this section we provide the training and hyperparameter details for the forecasting experiments.

Training parameters. We trained all the datasets with a fixed batch size of 50 and tuned the number of training steps per dataset. We used the Adam optimizer for the gradient updates with 10^{-5} weight-decay and clipped the gradient norm to 10. The learning rate was warmed-up linearly from 1×10^{-4} to a higher value in the range $[5 \times 10^{-4}, 1 \times 10^{-2}]$ (optimized per dataset) for the first 1000 steps after which a cosine decay follows for the remaining time steps with a decay rate of 0.99.

Network types. We experimented with linear and non-linear functions for the continuous transition f_x and emission f_y functions as previously described in Appendix B.3. For the inference embedding network g^y_{emb}, we used a single transformer layer with one attention head. The input time series was first mapped into a 4-dimensional embedding and then concatenated with the positional encoding before feeding into the transformer layer. For the control network (f_u) and the duration network (f_d) we used single hidden layer MLPs. Table 7 summarizes the network architectures of the different model components for the five forecasting datasets.

Control embedding. As described in subsection A.2, the raw control features are comprised of static features $\mathbf{u}^{\text{static}}$ and time features $\mathbf{u}^{\text{time}}$. In our experiments, the static features are the time series IDs. Time series ID are first fed into an embedding layer that outputs a p-dimensional embedding; this time series embedding is concatenated with the time features and passed to the control network (f_u) to output a c-dimensional control $\mathbf{u}_t$. Table 8 lists the values of p and c for the different datasets.

Table 7: Network architectures for the different components of RED-SDS for forecasting experiments. `Linear` denotes a linear layer without bias, `MLP` $[a_1, \ldots, a_l]$ denotes a l-hidden-layer MLP with hidden units $a_1, \ldots, a_l$ and ReLU non-linearity, `biGRU` $[b]$ denotes a single-layer bidirectional GRU with b hidden units, `RNN` $[c]$ denotes a single-layer RNN with c hidden units, and `Transformer` denotes a transformer layer with one attention head and the dimension of feedforward network equal to 16.

Network	Datasets				
	`exchange`	`solar`	`electricity`	`traffic`	`wiki`
Control Network (f_u)	`MLP` [32]	`MLP` [32]	`MLP` [32]	`MLP` [64]	`MLP` [32]
Duration Network (f_d)	`MLP` [64]	`MLP` [64]	`MLP` [64]	`MLP` [64]	`MLP` [64]
Discrete Transition (f_z)	`MLP` $[4 \times 2^2]$	`MLP` $[4 \times 4^2]$	`MLP` $[4 \times 3^2]$	`MLP` $[4 \times 4^2]$	`MLP` $[4 \times 5^2]$
Continuous Transition (f_x)	`MLP` [32]	`Linear`	`MLP` [32]	`MLP` [32]	`Linear`
Emission Network (f_y)	`MLP` [8, 32]	`Linear`	`MLP` [8, 32]	`MLP` [8, 32]	`Linear`
Inference Embedder (g^y_{emb})	`biGRU` [4]	`Transformer`	`Transformer`	`Transformer`	`Transformer`
Causal RNN (g_{rnn})	`RNN` [16]	`RNN` [16]	`RNN` [16]	`RNN` [16]	`RNN` [16]
Parameter Network (g_{fc})	`MLP` [32]	`MLP` [32]	`MLP` [32]	`MLP` [32]	`MLP` [32]

Table 8: Control embedding hyperparameters.

Hyperparameter	Datasets				
	`exchange`	`solar`	`electricity`	`traffic`	`wiki`
p	5	8	32	50	8
c	16	16	32	32	32

Duration. We set $(d_{\min}, d_{\max})$ to $(1, 20)$ for all datasets.

Normalization. In many datasets considered by us, the scale of the time series varies significantly, sometimes by several orders of magnitude. This makes it difficult to train models—particularly those involving neural networks—and the individual time series require normalization before training. Such *per time series* normalization is a standard practice for these datasets and has been employed in several previous works [44, 42, 30]. To this end, we experimented with two normalization methods: standardization and scaling. Given a univariate time series $y_{1:T}$, the normalized version $y^{\text{norm}}_{1:T}$ for each method is computed as

$$\text{Standardization:} \quad y^{\text{norm}}_t = \frac{y_t - \text{mean}(y_{1:T})}{\text{std}(y_{1:T})}, \tag{57}$$

$$\text{Scaling:} \quad y^{\text{norm}}_t = \frac{y_t}{\frac{1}{T}\sum_{i=1}^{T} |y_i|}, \tag{58}$$

where $\text{mean}(\cdot)$ and $\text{std}(\cdot)$ denote the mean and the standard deviation, respectively. The normalization method for each dataset was tuned as a hyperparameter.

Log-determinant of the Jacobian. Normalization of data is equivalent to applying a linear transformation to the raw input. More formally, consider a function f that transforms a variable $\mathbf{v} \in \mathbb{R}^D$ to the normalized variable $\mathbf{y} \in \mathbb{R}^D$ via a function f, i.e., $f(\mathbf{v}) = \mathbf{y}$. If $p_V(\mathbf{v})$ is the probability density of the random variable $\mathbf{v}$ and f is an invertible and differentiable transformation, then by the change of variables formula the probability density $p_Y(\mathbf{y})$ of the transformed random variable $\mathbf{y}$ is given by

$$p_Y(\mathbf{y}) = p_V(f^{-1}(\mathbf{y})) \left| \det \left(\frac{\partial f^{-1}(\mathbf{y})}{\partial \mathbf{y}} \right) \right|, \tag{59}$$

where $\frac{\partial f^{-1}(\mathbf{y})}{\partial \mathbf{y}} \in \mathbb{R}^{D \times D}$ is the Jacobian of f^{-1}. The determinant term accounts for the space distortion of the transformation, i.e., how the volume has changed locally due to the transformation.

Using the change of variables formula, this transformation corresponds naturally to a $\log\det$ term in the log-likelihood which is equal to $J = -\log s$, where $s = \text{std}(y_{1:T})$ in the case of standardization and $s = \sum_{i=1}^{T} |y_i|$ in the case of scaling (the Jacobian in both cases is a diagonal matrix). In the forecasting experiments the $\log\det$ term is included in the objective during training. Note that this

holds only if we treat s as a scaling factor that is independent of the time series $y_{1:T}$; although this is not true, in practice this training heuristic slightly improves the quantitative forecasting performance.

Number of switches. In the case of segmentation, the ground truth number of switches K is known (at least in the context of our experiments). For forecasting, we consider a range of $K \in \{2, \ldots, 5\}$ and we tuned it for each dataset.

Dimensionality of state $\mathbf{x}_t$. We tuned the dimensionality of the state variables $\mathbf{x}_t$ from the set $\{2, 4, 8, 16\}$ for each dataset.

C.4 Baseline models

In this section, we provide additional details on the baseline forecasting models. The CRPS results for these baseline models have been taken from [30].

ARSGLS [30] is an extension of the vanilla SLDS that uses continuous (Gaussian) switch variables. It incorporates conditionally linear state-to-switch recurrence, keeping the conditional tractability of LDS, and augments the emission model by an auxiliary variable that allows for modelling multivariate and non-Gaussian observations with complex non-linear transformations. Inference in ARSGLS is performed via Rao-Blackwellised particle filters wherein the state conditional expectations are computed exactly using the available closed-form expressions whereas the switch and auxiliary variables expectations are approximated via Sequential Monte Carlo using a neural network.

The authors proposed two different instantiations of their model with differences in the underlying base LDSs. In the first instantiation, labelled as RSGLS-ISSM, the authors implemented the LDS as an innovation state space model (ISSM) with a constrained structure that models temporal patterns such as level, trend and seasonality where the transition and emission matrices are pre-defined and not learned. The second instantiation, labelled ARSGLS, uses an unconstrained LDS.

KVAE [19] uses a VAE to model auxiliary variables and a "mixture" of LDSs to model the dynamics. KVAE can be interpreted as an SLDS with deterministic switches, where the LDS parameters are predicted as a weighted average of a set of base matrices, with the weights given by an RNN. The RNN in the KVAE depends (autoregressively) on samples of the previous auxiliary variables. Variational inference is performed using a recognition network for the auxiliary variables and Kalman filtering/smoothing for the LDSs' latent variables.

The objective function in KVAE uses samples from the smoothing distribution. However, as noted in [30], the corresponding expectation can be computed in closed-form and the resulting objective function has lower variance. We report the results for both original KVAE (KVAE-MC) and the Rao-Blackwellised variant (KVAE-RB) proposed by Kurle et al. [30].

DeepState [42] parametrizes an LDS using an RNN which is conditioned on inputs (controls). The LDS parameters in DeepState have a fixed structure that model time series patterns such as level, trend and seasonality (same as in RSGLS-ISSM). The transition and emission matrices are fixed and the (diagonal) noise covariance matrices are predicted by the RNN directly. Given the LDS parameters from the RNN, DeepState uses the Kalman filter for inference and maximum likelihood for parameter learning.

DeepAR [44] is a strong *discriminative* baseline model for probabilistic forecasting that uses an autoregressive RNN conditioned on the history of the time series (lags) and other relevant features. DeepAR autoregressively outputs the parameters of the future distributions and is trained using maximum likelihood estimation.

D Computational details

We used a `p3.8xlarge` AWS EC2 instance for running our experiments. This instance comprises 4 Tesla V100 GPUs, 36 CPUs, and 244 GB of memory.