

Learning state and proposal dynamics in state-space models using differentiable particle filters and neural networks

Benjamin Cox^a, Santiago Segarra^b, Víctor Elvira^a

^a*School of Mathematics, University of Edinburgh, James Clerk Maxwell Building, Edinburgh, EH9 3FD, Scotland, United Kingdom*

^b*Electrical and Computing Engineering, Rice University, 6100 Main Street MS 366, Houston, 77005, Texas, USA*

Abstract

State-space models are a popular statistical framework for analysing sequential data. Within this framework, particle filters are often used to perform inference on non-linear state-space models. We introduce a new method, StateMixNN, that uses a pair of neural networks to learn the proposal distribution and transition distribution of a particle filter. Both distributions are approximated using multivariate Gaussian mixtures. The component means and covariances of these mixtures are learnt as outputs of learned functions. Our method is trained targeting the log-likelihood, thereby requiring only the observation series, and combines the interpretability of state-space models with the flexibility and approximation power of artificial neural networks. The proposed method significantly improves recovery of the hidden state in comparison with the state-of-the-art, showing greater improvement in highly non-linear scenarios.

Keywords:

State-space models, importance sampling, particle filter proposal distribution, differentiable particle filter, neural networks.

1. Introduction

In many fields of science and engineering, it is common to process sequential data resulting from the observation of dynamical systems. These systems appear in fields such as target tracking, [1], finance [2], epidemiology [3], ecology [4], and meteorology [5]. We can describe these systems and their observations mathematically, utilising the state-space model (SSM) framework. SSMs represent a dynamical system via a latent state and a series of related, noisy, observations, encoding the system via a pair of distributions describing the latent state dynamics and its relationship to the observation. Within state-space modelling, it is common to compute the distribution of the state conditional on the distribution of the previous state and the current observation, a problem known as the filtering problem.

In the case of linear dynamics with Gaussian noise, the Kalman filter (KF) provides the optimal solution to the filtering problem via a sequence of closed form estimates. However, if the

B.C. acknowledges support from the *Natural Environment Research Council* of the UK through a SENSE CDT studentship (NE/T00939X/1). The work of V. E. is supported by the *Agence Nationale de la Recherche* of France under PISCES (ANR-17-CE40-0031-01), the Leverhulme Research Fellowship (RF-2021-593), and by ARL/ARO under grant W911NF-22-1-0235. In addition, we acknowledge support from the 2022 University of Edinburgh – Rice University Strategic Collaboration Award.

system is more complex, the KF can no longer be used directly. A number of extensions to the KF to non-linear systems have been proposed, such as the extended Kalman filter (EKF) [6] and the unscented Kalman filter (UKF) [7]. These methods approximate the state posterior via a Gaussian distribution, which is not always appropriate. The particle filter (PF), also known as sequential Monte Carlo (SMC), is an alternative method for the general SSM, which approximates the state posterior via a series of Monte Carlo samples [8, 9, 10].

In PFs, we must evaluate the likelihood of the distribution of the state given the previous state, known as the transition distribution, which is in general unknown. If the form of the transition distribution is known, with only the parameters unknown, then several methods exist to estimate the system parameters, and hence estimate the transition distribution [11, Chapter 12]. Furthermore, learning parameters in the PF has been greatly eased with the advances in differentiable particle filters (DPFs) [12, 13, 14, 15], which allow the use of gradient-based optimisation methods for parameter estimation.

Differentiable particle filters (DPFs) modify the resampling step of the PF to be differentiable, and therefore, when combined with differentiable transition and observation likelihoods, makes the overall particle filter differentiable. There exist a number of differentiable particle filters, which utilise many different methods for making the resampling step differentiable, such as soft resampling [16], stop-gradient operators [13], or optimal transport schemes [12].

If the form of the transition distribution is unknown, current methods require one to be assumed. In addition to the transition distribution, in the PF we must provide a proposal distribution to generate Monte Carlo samples. The diversity of the samples, which is critically important to the method’s ability to represent the system, is heavily dependent on the proposal distribution.

One approach to choosing the proposal distribution is to use the bootstrap particle filter (BPF) [8], which uses the state transition distribution as the proposal. However, the bootstrap proposal does not incorporate the observation, and therefore does not utilise all available information at each time step. Not using observation information can cause significant issues, for example, if the state transition is a very diffuse distribution but the observation is very informative, few, if any, particles will have high posterior probability, and therefore the effective sample size will be very low. Several methods incorporate the observation in their proposal distribution, such as the auxiliary particle filter (APF) [17, 18, 19, 20], which adjusts the particle weights based on both the transition dynamics and the likelihood of the new observation. Deep learning methods have also been used within the SMC and PF framework to learn the proposal distribution, such as neural adaptive SMC [21] and the variational SMC [22].

Contribution. In this paper, we propose StateMixNN, a method to approximate the transition distribution and proposal distribution of a particle filter using a pair of adaptive Gaussian mixtures.¹ StateMixNN improves upon methods that learn only the transition dynamics, as we also estimate the optimal proposal density, and therefore incorporate more information in our estimates than would otherwise be possible.

StateMixNN learns the mean and covariance parameters of the components of a multivariate Gaussian mixture as the output of a dense neural network. By estimating both the transition

¹A limited version of this work was presented by the authors in the conference paper [23], which presented a version of this work learning only the proposal distribution. We propose here a more advanced method that can learn both the proposal distribution and the transition distribution. We include here extensive methodological discussion, and provide practical guidance for practitioners. Finally, we include a large number of numerical experiments, which were not present in the prior work.

and proposal distributions, we can estimate the hidden state from an observation series given only the observation model, which cannot be estimated here for identifiability reasons. This approximation allows us to estimate distributions resulting from complex models, becoming more expressive as the number of mixture components increases.

In order to train the neural networks, we utilise the DPF framework of Ścibior and Wood [13], allowing gradient propagation through the resampling step of the particle filter, and therefore the use of gradient-based optimisation schemes. StateMixNN is semi-supervised since it trains targeting the parameter log-likelihood, which requires the sequence of observations, but does not require the underlying hidden states.

2. Background

2.1. State-space models (SSMs)

We are interested in state-space dynamical systems, which we can describe by

$$\begin{aligned}\mathbf{x}_t &\sim f(\mathbf{x}_t|\mathbf{x}_{t-1}; \boldsymbol{\theta}^{(f)}), \\ \mathbf{y}_t &\sim g(\mathbf{y}_t|\mathbf{x}_t; \boldsymbol{\theta}^{(g)}),\end{aligned}\tag{1}$$

where $t \in \{1, \dots, T\}$ denotes discrete time, $\mathbf{x}_t \in \mathbb{R}^{d_x}$ is the state of the system at time t , $\mathbf{y}_t \in \mathbb{R}^{d_y}$ is the observation associated with \mathbf{x}_t , $\boldsymbol{\theta}^{(g)}$ and $\boldsymbol{\theta}^{(f)}$ are sets of parameters relating to the observation and state dynamics respectively, and the distributions f and g encode the transition and observation model respectively.

In terms of probability densities, $f(\mathbf{x}_t|\mathbf{x}_{t-1}; \boldsymbol{\theta}^{(f)})$ is the conditional density of the state \mathbf{x}_t given \mathbf{x}_{t-1} , and $g(\mathbf{y}_t|\mathbf{x}_t; \boldsymbol{\theta}^{(g)})$ is the conditional density of the observation \mathbf{y}_t given the hidden state \mathbf{x}_t . The initial value of the state, \mathbf{x}_0 , is distributed as $\mathbf{x}_0 \sim p(\mathbf{x}_0|\boldsymbol{\theta}^{(p)})$. The hidden state sequence $\mathbf{x}_{0:T}$ is not observed, while the related sequence of observations $\mathbf{y}_{1:T}$ is observed. In many cases, we wish to estimate the sequence of hidden states given the observation sequence. If we estimate the state at time t , given by \mathbf{x}_t , conditional on observations collected up to and including time t , given by $\mathbf{y}_{1:t}$, then this we call this the filtering problem, with $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ being known as the filtering distribution.

2.2. Particle filtering

Filtering methods for SSMs aim to probabilistically recover the state of the systems described in Eq. (1) by estimating the posterior probability density function (pdf) of the state \mathbf{x}_t conditional on all prior observations $\mathbf{y}_{1:t}$, i.e., $p(\mathbf{x}_t|\mathbf{y}_{1:t}; \boldsymbol{\theta}^{(f)})$. PFs approximate this pdf utilising a set of K Monte Carlo samples and associated weights, $\{\mathbf{x}_t^{(k)}, \tilde{w}_t^{(k)}\}_{k=1}^K$. The filtering distribution can thus be approximated by

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}; \boldsymbol{\theta}^{(f)}) \approx \sum_{k=1}^K \tilde{w}_t^{(k)} \delta_{\mathbf{x}_t^{(k)}}.\tag{2}$$

A ubiquitous algorithm for particle filtering is the sequential importance resampling (SIR) algorithm, which we give in Alg. 1. In this method, at time step t , we compute a set of particles and associated importance weights via the following steps. First, we draw K particles $\{\mathbf{x}_t^{(k)}\}_{k=1}^K$ from the proposal distribution $\pi(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t; \boldsymbol{\theta}^{(p)})$ (line 4). Then, we compute the importance weights $\{w_t^{(k)}\}_{k=1}^K$ (line 5). We next normalise the weights to sum up to 1, obtaining $\{\tilde{w}_t^{(k)}\}_{k=1}^K$ and perform resampling (line 7), in which we sample the particle set K times with replacement, with the

probability of drawing particle $\mathbf{x}_t^{(k)}$ equal to its weight $\tilde{w}_t^{(k)}$. Resampling is important to avoid the degeneracy of the filter, as it helps to avoid the weights becoming concentrated in a diminutive subset of the sampled particles (at the expense of path degeneracy, see for instance [24]).

Algorithm 1 Sequential importance resampling (SIR)

- 1: Draw $\mathbf{x}_0^{(k)} \sim p(\mathbf{x}_0|\theta^{(p)})$, for $k \in 1, \dots, K$.
 - 2: Set $\tilde{w}_0^{(k)} = 1/K$, for $k \in 1, \dots, K$.
 - 3: **for** $t \in 1, \dots, T$ and $k \in 1, \dots, K$ **do**
 - 4: Draw $\mathbf{x}_t^{(k)} \sim \pi(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t; \theta^{(\pi)})$.
 - 5: Compute $w_t^{(k)} = \frac{g(\mathbf{y}_t|\mathbf{x}_t^{(k)}; \theta^{(g)})f(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{(k)}; \theta^{(f)})}{\pi(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t; \theta^{(\pi)})}$.
 - 6: Compute $\tilde{w}_t^{(k)} = \tilde{w}_{t-1}^{(k)} w_t^{(k)} / \sum_k \tilde{w}_{t-1}^{(k)} w_t^{(k)}$.
 - 7: Perform resampling with replacement over $\{\mathbf{x}_t^{(k)}\}_{k=1}^K$, sampling $\mathbf{x}_t^{(k)}$ with probability $\tilde{w}_t^{(k)}$.
 - 8: **end for**
-

Given the SIR algorithm, it is apparent that the choice of proposal distribution $\pi(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t; \theta^{(\pi)})$ is critical to the success of the estimation; the particles should be concentrated in regions of the state space with high probability mass. There exist several methods to choose the proposal distribution.

One such method is the BPF [8], where the proposal distribution is set equal to the transition distribution of the SSM, with $\pi(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t; \theta^{(\pi)}) = f(\mathbf{x}_t|\mathbf{x}_{t-1}; \theta^{(f)})$. Such an approach is simple, and results in less computation due to cancellation in line 5 of Alg. 1, resulting in $w_t^{(k)} = g(\mathbf{y}_t|\mathbf{x}_t^{(k)}; \theta^{(g)})$ for the bootstrap particle filter. However, the BPF does not incorporate the observation \mathbf{y}_t in the proposal distribution, and therefore omits potentially important information. For example, if the transition distribution is far more diffuse than the observation distribution, then the observation contains a lot of information relative to the previous state, which the BPF does not use.

The optimal proposal distribution incorporates the observation at the current time step t , i.e., $\pi(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t; \theta^{(\pi)}) = p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t)$. In general distribution is intractable or unknown. Methods such as the APF and its derivatives [17, 20, 18, 19] incorporate the observation information via a pre-weighting step.

Gama et al. [25, 26] propose several methods to learn the proposal distribution in a general setting, such as modelling it as a Gaussian distribution with mean and covariance output by a neural network, or utilising normalising flows. These methods learn a distinct network for each time step, and therefore do not generalise outside of a single series of observations.

2.3. Differentiable particle filters

The particle filter as given in Alg. 1 is not differentiable with respect to the weights, and hence is not differentiable with respect to the parameters $\theta^{(f)}$, $\theta^{(\pi)}$, or $\theta^{(g)}$, following from the two sampling steps on lines 4 and 7. The proposal sampling step on line 4 can often be rewritten to be differentiable using distribution specific reparameterisations, such as the reparameterisation trick for Gaussian distributions [27].

The resampling step on line 7 remains, which requires sampling a multinomial distribution. Resampling a multinomial distribution is non differentiable, as infinitesimal changes in the weights result in discrete changes in the sampled indices [28]. Therefore, any function of the weights is non-differentiable, with one such function being the likelihood of the observations, which is a common target function for parameter estimation in particle filtering. Note that

the particle trajectories are also functions of the weights, so functions of trajectories are also non-differentiable if using Alg. 1.

Differentiable particle filtering (DPF) modifies the resampling step to achieve differentiability. Therefore, when combined with a differentiable transition step, the overall particle filter is differentiable [12, 14, 13, 15]. There exist several differentiable particle filtering methods, each utilising a different method to make the resampling step differentiable. In this work, we use the stop-gradient differentiable particle filter of [13], which we present in Alg. 2. The stop-gradient DPF yields unbiased gradient estimates with minimal computational overhead [13].

Algorithm 2 Stop-gradient differentiable particle filter (DPF)

[13]

-
- 1: Draw $\mathbf{x}_0^{(k)} \sim p(\mathbf{x}_0|\boldsymbol{\theta}^{(p)})$, for $k \in 1, \dots, K$.
 - 2: Set $\tilde{w}_0^{(k)} = 1/K$, for $k \in 1, \dots, K$.
 - 3: **for** $t \in 1, \dots, T$ and $k \in 1, \dots, K$ **do**
 - 4: Sample $a_t^{(k)} \sim \text{Categorical}(\perp(\tilde{w}_{t-1}))$.
 - 5: Set $\tilde{w}_t^{(k)} = \frac{1}{K} \tilde{w}_t^{a_t^{(k)}} / \perp(\tilde{w}_t^{a_t^{(k)}})$.
 - 6: Sample $\mathbf{x}_t^{(k)} \sim \pi(\mathbf{x}_t|\mathbf{x}_{t-1}^{a_t^{(k)}}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)})$.
 - 7: Compute $w_t^{(k)} = \frac{g(\mathbf{y}_t|\mathbf{x}_t^{(k)}; \boldsymbol{\theta}^{(g)})f(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{a_t^{(k)}}; \boldsymbol{\theta}^{(f)})}{\pi(\mathbf{x}_t|\mathbf{x}_{t-1}^{a_t^{(k)}}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)})}$.
 - 8: Compute $\tilde{w}_t^{(k)} = \tilde{w}_{t-1}^{(k)} w_t^{(k)} / \sum_{k=1}^K \tilde{w}_{t-1}^{(k)} w_t^{(k)}$.
 - 9: **end for**
-

We can use differentiable particle filters to estimate the gradient of the parameter log-likelihood, and therefore apply gradient methods to estimate unknown parameters of our state-space model. However, as the log-likelihood is estimated, we will also have noise in gradients thereof, and therefore the parameter estimation scheme must be robust to noisy gradients. Noisy gradients are common in deep learning, as it is typical to compute the gradient of a stochastically selected subset of the training data, which results in gradient noise due to sampling. Several optimisation methods have been proposed for use in deep learning, with a common feature being robustness to stochastic gradients. We can utilise these methods when estimating parameters using differentiable particle filters, with schemes such as ADAM [29], RADAM [30], and Novograd [31, 32], all being applicable to this problem.

3. Proposed algorithm

We propose StateMixNN, a method to learn the transition and proposal distribution of a generic state-space model. We approximate the transition distribution and the proposal distribution by a pair of multivariate Gaussian mixtures parameterised by neural networks. By combining learnt estimates of both the transition distribution and the proposal distribution, we can learn a generic state-space model conditional only on knowledge of the observation model.

3.1. Parameterising the Gaussian mixture

StateMixNN learns the transition distribution conditional on only the previous state value, thus preserving the Markovianity of the SSM. We approximate $f(\mathbf{x}_t|\mathbf{x}_{t-1}; \boldsymbol{\theta}^{(f)})$ by an equally weighted multivariate Gaussian mixture of S_f components with diagonal covariances

$$f(\mathbf{x}_t|\mathbf{x}_{t-1}; \boldsymbol{\theta}^{(f)}) := (S^{(f)})^{-1} \sum_{s=1}^{S_f} f_s(\mathbf{x}_t|\mathbf{x}_{t-1}; \boldsymbol{\theta}^{(f)}), \quad (3)$$

with $s \in \{1, \dots, S_f\}$, where $\boldsymbol{\theta}^{(f)}$ are the parameters of the state neural network, and f_s is given by

$$f_s(\mathbf{x}_t|\mathbf{x}_{t-1}; \boldsymbol{\theta}^{(f)}) = \mathcal{N}\left(\mu_s^{(f)}(\mathbf{x}_{t-1}), \Sigma_s^{(f)}(\mathbf{x}_{t-1})\right). \quad (4)$$

StateMixNN learns the proposal distribution conditioned on the previous state value and the current observation, as is the case for the intractable optimal proposal distribution. We parameterise $\pi(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)})$ as an equally weighted mixture of S_π multivariate Gaussian distributions with diagonal covariances

$$\pi(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)}) := (S^{(\pi)})^{-1} \sum_{s=1}^{S_\pi} \pi_s(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)}), \quad (5)$$

with $s \in \{1, \dots, S_\pi\}$, where $\boldsymbol{\theta}^{(\pi)}$ are the parameters of the proposal neural network, and π_s is given by

$$\pi_s(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)}) = \mathcal{N}\left(\mu_s^{(\pi)}(\mathbf{x}_{t-1}, \mathbf{y}_t), \Sigma_s^{(\pi)}(\mathbf{x}_{t-1}, \mathbf{y}_t)\right). \quad (6)$$

Mixtures of multivariate Gaussians can represent a wide range of possible distributions, offering flexibility and expressiveness that is beneficial for modelling complex systems [33].

3.2. Network architecture and learning

Our method learns the mean functions $\mu^{(f)}(\cdot)$, $\mu^{(\pi)}(\cdot)$ and the covariance functions $\Sigma^{(f)}(\cdot)$, $\Sigma^{(\pi)}(\cdot)$ for the transition and proposal distributions using a pair of dense neural networks. We denote these networks by $\text{NN}^{(f)}(\mathbf{x}_{t-1}; \boldsymbol{\theta}^{(f)})$ for the network associated with the transition distribution and by $\text{NN}^{(\pi)}(\mathbf{x}_{t-1}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)})$ for the network associated with the proposal distribution.

The transition network takes as input only the previous particle value, therefore preserving the Markovianity assumption of the SSM, which is often violated when using neural networks [34, 35, 36, 37].

The network $\text{NN}(\cdot; \boldsymbol{\theta})$ is comprised of L layers, with

$$\text{NN}(\cdot; \boldsymbol{\theta}) = \mathbf{z}_L, \quad \mathbf{z}_l = \rho_l(\mathbf{A}_l \mathbf{z}_{l-1} + \mathbf{b}_l), \quad (7)$$

for $l \in 1, \dots, L$. Therefore, $\boldsymbol{\theta} = \{\mathbf{A}_1, \mathbf{b}_1, \dots, \mathbf{A}_L, \mathbf{b}_L\}$ are the learned parameters for a network. For the proposal network the initial value $\mathbf{z}_0^{(\pi)} = [\mathbf{x}_{t-1}^\top, \mathbf{y}_t^\top]^\top$ is the concatenation of the previous state \mathbf{x}_{t-1} and the current observation \mathbf{y}_t . Thus, $\mathbf{z}_0^{(\pi)}$ has dimension $d_0^{(\pi)} = d_x + d_y$. For the state network the initial value $\mathbf{z}_0^{(f)} = \mathbf{x}_{t-1}$ is the previous state \mathbf{x}_{t-1} . Thus, $\mathbf{z}_0^{(f)}$ has dimension $d_0^{(f)} = d_x$.

For both networks, the dimension of the output

$$\mathbf{z}_L = [\boldsymbol{\mu}^{(1)\top}, \mathbf{c}^{(1)\top}, \dots, \boldsymbol{\mu}^{(S)\top}, \mathbf{c}^{(S)\top}]^\top \quad (8)$$

is $d_L := 2Sd_x$, as for each of the mixture components of both distributions, we require a d_x -dimensional mean vector $\boldsymbol{\mu}^{(n)}$ and a d_x -dimensional covariance scale vector $\mathbf{c}^{(n)}$.

In both cases, we construct a Gaussian mixture distribution from a neural network $\text{NN}(\cdot; \boldsymbol{\theta})$ by

$$\mathcal{GM}(\text{NN}(\cdot; \boldsymbol{\theta})) = \sum_{s=1}^S S^{-1} \mathcal{N}(\boldsymbol{\mu}^{(s)}, \mathbf{C}^{(s)}), \quad (9)$$

where S is the number of mixture components, with $\boldsymbol{\mu}^{(s)}, \mathbf{c}^{(s)}, s \in 1, \dots, S$ extracted from \mathbf{z}_L by indexing per Eq. (8), where we discard the components of $\boldsymbol{\theta}$ that are not relevant to NN, and $\mathbf{C}^{(s)} = \text{diag}(\mathbf{c}^{(s)})^2$. For example, when constructing f , we use $\mathcal{GM}(\text{NN}^{(f)}(\cdot; [\boldsymbol{\theta}^{(f)}, \boldsymbol{\theta}^{(\pi)}]))$, discarding $\boldsymbol{\theta}^{(\pi)}$ if it is not used in $\text{NN}^{(f)}$. We make use of this property in Alg. 5. The learned parameters for a neural network are the weights \mathbf{A}_l and the biases \mathbf{b}_l , with $l \in 1, \dots, L$. The number of layers L , the dimension of each layer $d_l, l \in 1, \dots, L$, and the activation functions $\rho_l, l \in 1, \dots, L$ are fixed as part of the network architecture.

We learn separate networks for the transition and proposal distributions, denoted by $\text{NN}^{(f)}$ and $\text{NN}^{(\pi)}$ respectively. Therefore, when learning the transition distribution we estimate $\boldsymbol{\theta}^{(f)} := \{\mathbf{A}_1^{(f)}, \mathbf{b}_1^{(f)}, \dots, \mathbf{A}_{L^{(f)}}^{(f)}, \mathbf{b}_{L^{(f)}}^{(f)}\}$, and estimate $\boldsymbol{\theta}^{(\pi)} := \{\mathbf{A}_1^{(\pi)}, \mathbf{b}_1^{(\pi)}, \dots, \mathbf{A}_{L^{(\pi)}}^{(\pi)}, \mathbf{b}_{L^{(\pi)}}^{(\pi)}\}$ when learning the proposal distribution.

To train the networks we maximise the log-likelihood, given by

$$\ell(\boldsymbol{\theta}|\mathbf{y}_{1:T}) \propto \sum_{t=1}^T \left(\sum_{k=1}^K \log(w_t^{(k)} \cdot \tilde{w}_{t-1}^{(k)}) \right), \quad (10)$$

where $w_t^{(k)}$ and $\tilde{w}_{t-1}^{(k)}$ are the unnormalised and normalised weights of the PF in Alg. 1 [11, Chapter 12]. In the case that we perform resampling at every step, Eq. (10) reduces to

$$\ell(\boldsymbol{\theta}|\mathbf{y}_{1:T}) \propto \sum_{t=1}^T \left(\sum_{k=1}^K \log(w_t^{(k)}) \right), \quad (11)$$

as the previous step normalised weights are identically equal to $1/K$. Note that the weights are dependent on the parameters $\boldsymbol{\theta}^{(f)}, \boldsymbol{\theta}^{(g)}$, and $\boldsymbol{\theta}^{(\pi)}$, as the weights are computed using densities dependent on these parameters. In our case, we assume that the observation model g is known, and hence we omit the dependence on $\boldsymbol{\theta}^{(g)}$. The log-likelihood is maximised when all weights are equal, a desirable property as it reduces weight degeneracy over time [11]. Furthermore, maximising the log-likelihood does not require knowledge of the true value of the hidden state, which is often unavailable, requiring only the observation series.

3.3. StateMixNN algorithm

Algorithm 3 StateMixNN algorithm	StateMixNN($B, J, A, \mathbf{y}, \text{NN}^{(f)}, \text{NN}^{(\pi)}$)
1: Input: Number of batches B , steps per batch J , number of iterations A , observations \mathbf{y} , transition network $\text{NN}^{(f)}$, proposal network $\text{NN}^{(\pi)}$.	
2: Initialise $\boldsymbol{\theta}_0^{(\pi)} := \{\mathbf{A}_1^{(\pi)}, \mathbf{b}_1^{(\pi)}, \dots, \mathbf{A}_{L^{(\pi)}}^{(\pi)}, \mathbf{b}_{L^{(\pi)}}^{(\pi)}\}$.	
3: Initialise $\boldsymbol{\theta}_{-1}^{(f)} := \{\mathbf{A}_1^{(f)}, \mathbf{b}_1^{(f)}, \dots, \mathbf{A}_{L^{(f)}}^{(f)}, \mathbf{b}_{L^{(f)}}^{(f)}\}$.	
4: Initialise $f_{-1}(\mathbf{x}_t \mathbf{x}_{t-1}) := \mathcal{GM}(\text{NN}^{(f)}(\mathbf{x}_{t-1}; [\boldsymbol{\theta}^{(f)}, \boldsymbol{\theta}^{(\pi)}]))$.	
5: for $a \in 1, \dots, A$ do	
6: Set $\boldsymbol{\theta}_a^{(\pi)} := \text{ConditionalUpdate}(B, J, \boldsymbol{\theta}_{a-1}^{(\pi)}, \boldsymbol{\theta}_{a-1}^{(f)}, \mathbf{y})$ using Alg. 4.	
7: Set $\boldsymbol{\theta}_a^{(f)} := \text{ConditionalUpdate}(B, J, \boldsymbol{\theta}_{a-1}^{(f)}, \boldsymbol{\theta}_a^{(\pi)}, \mathbf{y})$ using Alg. 4.	
8: end for	
9: return $\boldsymbol{\theta}_A^{(f)}, \boldsymbol{\theta}_A^{(\pi)}$.	

StateMixNN is described in Alg. 3. We first describe the overall algorithm, given in Alg. 3, and proceed to describe the subordinate algorithms, Alg. 4 and Alg. 5 in turn.

In Alg. 3, we begin by initialising the network parameters (lines 2-3 of Alg. 3). It is not crucial how these parameters are initialised, but we note that our software implementation utilises element wise random uniform initialisations for \mathbf{A}_l and \mathbf{b}_l , with values in the range $\pm \sqrt{d_l}$, following [38]. We then learn an initial value for the transition distribution (line 4 of Alg. 3). We do this by optimising the network parameters of the transition distribution in a bootstrap particle filter, wherein the transition and proposal distributions are the same.

We then perform A alternating conditional updates. We optimise each of $\theta^{(f)}$ and $\theta^{(\pi)}$ conditional on the other, similar to a coordinate descent method. At iteration a , we first optimise the proposal distribution parameters $\theta_a^{(\pi)}$, conditional on the value of the transition distribution parameters $\theta_{a-1}^{(f)}$. We then optimise the transition distribution parameters $\theta_a^{(f)}$ conditional on the value of the proposal distribution parameters $\theta_a^{(\pi)}$.

Algorithm 4 Conditional update algorithm	ConditionalUpdate($B, J, \theta_0, \theta_{\text{static}}, \mathbf{y}$)
1: Input: Number of batches B , steps per batch J , initial parameters θ_0 , static parameters θ_{static} , observations \mathbf{y} .	
2: Initialise $\theta_{0,J} := \theta_0$.	
3: for $b \in 1, \dots, B$ do	
4: Set $\mathbf{y}^{(b)} := \mathbf{y}_{1:[bT/B]}$.	
5: Set $\theta_{b,0} := \theta_{b-1,J}$.	
6: for $j \in 1, \dots, J$ do	
7: Set $\theta_{b,j} := \text{UpdateStep}(\theta_{b,j-1}, \theta_{\text{static}}, \mathbf{y}^{(b)})$ using Alg. 5.	
8: end for	
9: end for	
10: return $\theta_{B,J}^{(p)}$.	

We now describe the conditional update, given in Alg. 4. The conditional update algorithm takes an initial value of the parameters of the distribution of interest θ_0 , and the parameters of the distribution we are conditioning on θ_{static} . We split the observation series $\mathbf{y}_{1:T}$ into B batches, with the b -th batch given by $\mathbf{y}^{(b)} = \mathbf{y}_{1:[bT/B]}$. Note that this construction implies $\mathbf{y}^{(1)} \subseteq \mathbf{y}^{(2)} \subseteq \dots \subseteq \mathbf{y}^{(B-1)} \subseteq \mathbf{y}^{(B)}$. We construct these telescoping batches of observations to avoid likelihood issues stemming from unadapted parameters, since the first sampled trajectories often have an extremely small log-likelihood, which causes numerical errors when computing the weights in Alg. 1, leaving to the gradients of the log-likelihood exploding [10, 13].

We iterate over each of the b batches, and for each batch perform J optimisation steps. For the j -th optimisation step of batch b , we run Alg. 5 with $\theta_{\text{learn}} = \theta_{b,j-1}$, $\theta_{\text{static}} = \theta_{\text{static}}$ and observations $\mathbf{y} = \mathbf{y}^{(b)} := \mathbf{y}_{1:[bT/B]}$. Note that Alg. 4 runs the particle filter a total of JB times, once for each of the J steps taken in each of the B batches. As we perform 2 runs of Alg. 4 in each of the A iterative steps of Alg. 3, we run the particle filter a total of $2ABJ$ times in total.

Finally, we describe a single step of the inner update algorithm, given by Alg. 5. In each step, we construct the transition distribution (line 2 of Alg. 5) and proposal distribution (line 3 of Alg. 5) corresponding to the parameters at that step. We construct each distribution using Eq. (9), and thereby discard the parameters not relevant to a given network. In particular, $\text{NN}^{(\cdot)}(\cdot; [\theta_{\text{learn}}, \theta_{\text{static}}])$ takes as arguments both the static parameter, and the parameter we are learning when constructing both distributions. However, each distribution takes either the static parameter, or the parameter we are learning, and we discard the unused parameter for the purposes of constructing that distribution. For example, when learning the the transition distribution

-
- 1: **Input:** Parameters to learn θ_{learn} , static parameters θ_{static} observations \mathbf{y} .
 - 2: Set $f(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{GM}(\text{NN}^{(f)}(\mathbf{x}_{t-1}; [\theta_{\text{learn}}, \theta_{\text{static}}]))$.
 - 3: $\pi(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t) := \mathcal{GM}(\text{NN}^{(\pi)}(\mathbf{x}_{t-1}, \mathbf{y}_t; [\theta_{\text{learn}}, \theta_{\text{static}}]))$.
 - 4: Run a particle filter (Alg. 2) with transition distribution f , proposal distribution π , and observations \mathbf{y} .
 - 5: Obtain $\ell(\theta_{\text{learn}})$ and $\nabla\ell(\theta_{\text{learn}})$ from the particle filter via Eq. (10) and automatic differentiation.
 - 6: Obtain θ_{out} by applying a gradient based update to θ_{learn} with gradients $\nabla\ell(\theta_{\text{learn}})$.
 - 7: **return** θ_{out} .
-

f , $\text{NN}^{(f)}$ takes both θ_{learn} , corresponding to $\theta^{(f)}$ and θ_{static} , corresponding to $\theta^{(\pi)}$, in line 2 of Alg. 5, but discards θ_{static} as θ_{static} parameterises the proposal distribution π when learning f .

After constructing the transition distribution f and the proposal distribution π , we run a particle filter with these distributions (line 4 of Alg. 5), using the observations \mathbf{y} , which may be a subset of the overall series of observations. From the particle filter we obtain an estimate of the log-likelihood of the parameter θ_{learn} , denoted by $\ell(\theta_{\text{learn}})$, and an estimate of the gradient of the log-likelihood with respect to θ_{learn} , denoted by $\nabla\ell(\theta_{\text{learn}})$ (line 5). We then utilise a gradient update scheme, such as ADAM [29] or Novograd [31, 32], to apply a gradient update to the parameter θ_{learn} , and output the result of this update to be used within line 7 of Alg. 4.

3.4. Discussion

Our method approximates the transition and proposal distributions by multivariate Gaussian mixture distributions. These mixtures are capable of representing complex unknown distributions, and are in many situations both more interpretable and more reliable than methods common in machine learning literature such as normalising flows [25, 26, 39, 40]. Furthermore, in the context of particle filters, normalising flows have been observed to be susceptible to overfitting, in addition to being less intuitive and harder to train [26, 25].

We note that the particle weights w_t depend on the samples $\mathbf{x}_{0:t}$, which are drawn from the proposal distribution that we are learning. Furthermore, the computation of the weights requires evaluating the density of the transition distribution. Therefore, we require a way to propagate gradients through particle resampling and sampling the proposal mixture distribution. We use the stop-gradient differentiable particle filter of [13], given in Alg. 2, to propagate gradients through the resampling step of the particle filter. By including resampling in the training using this method, we remove bias in our gradients, [13], improving the convergence characteristics of StateMixNN, and allow our method to be utilised to learn complex systems, as weight degeneracy is addressed in training via the resampling step [16, 14].

In order to sample the multivariate Gaussian mixture distributions f and π , we draw the component from a categorical distribution, and then sample the associated multivariate Gaussian distribution. We reparameterise the categorical distribution using the Gumbel-Softmax reparameterisation [41], thereby allowing gradient propagation through the categorical sampling. However, it is equally valid to use a stop-gradient sampling for the categorical distribution, which is computationally more efficient. We can propagate gradients through sampling a multivariate Gaussian using the reparameterisation trick [27], where we externalise the randomness to an input of an affine transformation, which, crucially, is independent $\theta^{(f)}$ and $\theta^{(\pi)}$, and therefore has zero

gradient with respect to our learned parameters. The above, in combination with the differentiable particle filter method of [13], allow us to compute the gradient of Eq. (10) with respect to our parameters $\theta^{(p)} = \{\mathbf{A}_1^{(p)}, \mathbf{b}_1^{(p)}, \dots, \mathbf{A}_{L^{(p)}}^{(p)}, \mathbf{b}_{L^{(p)}}^{(p)}\}$, $p \in \{\pi, f\}$, and therefore train the proposal and transition networks using a gradient scheme, such as [29, 30].

4. Discussion about the StateMixNN framework

We now discuss some characteristics of Alg. 3, that will help explain the choices we have made. First, we discuss the reasoning for using the multivariate Gaussian mixture distribution for our approximating distributions. Then, we justify the restriction to diagonal covariance and equal mixture weights. Third, we justify the alternating estimation scheme of Alg.3. Finally, we discuss how we combat the phenomena of likelihood degeneracy and concentration in our method.

4.1. Choice of a Gaussian mixture distribution for π and f , and extensions

In this work, we propose in Eq. (9) to use a multivariate Gaussian mixture distribution to approximate the optimal proposal distribution and the state transition distribution. The Gaussian mixture is able to approximate a wide range of distributions, whilst retaining computational speed. Furthermore, Gaussian mixtures are interpretable, and are easier to understand and infer from than approximating distributions such as normalising flows [42].

Most deep learning frameworks have fast, differentiable implementations of the Gaussian distribution, and typically will have the capability to combine distributions resulting in a mixture distribution. Using standard methods such as these allows for rapid, optimised implementation of our method in a given deep learning framework, for example PyTorch [43], JAX [44], and Tensorflow [45] are popular frameworks, which all have standard constructs which can be used to construct efficient multivariate Gaussian mixtures.

We note that our method can be trivially extended to mixture distributions for which the components can be parameterised by their location and scale parameters, with an example of such an extension being a mixture of multivariate t distributions with a fixed degree of freedom ν . To do so, we would replace the $\mathcal{N}(\boldsymbol{\mu}^{(s)}, \mathbf{C}^{(s)})$ mixands in Eq. (9) with $t_\nu(\boldsymbol{\mu}^{(s)}, \mathbf{C}^{(s)})$, noting that ν is fixed. Using the t distribution may capture tail behaviour better; however, the t distribution is more expensive both to sample from and to evaluate the density of. For example, the density of the multivariate t distribution requires evaluating the Γ function, whereas the density for the Gaussian distribution requires only exponentiation and standard linear algebra operations.

4.2. Restriction to diagonal covariance and equal weights in the approximating mixture

In Section 3.2 we present our method, restricting the covariance of mixture components, $\mathbf{C}^{(s)}$, to a diagonal form, and the mixture weights to be uniformly equal to S^{-1} in Eq. (9). We present our reasoning for these restrictions below.

Covariances As we estimate both the state and proposal distributions f and π , we infer complex dependencies between state dimensions in the interaction between the distributions, and therefore do not need to estimate full covariance matrices. We assume the form of the covariance matrices in order to reduce the number of parameters required; we can parameterise a N dimensional distribution by $2N$ parameters, whereas estimating the full covariance matrix would require $N + N(N + 1)/2$ parameters, N for the mean $\boldsymbol{\mu}^{(s)}$, and $N(N + 1)/2$ for the covariance $\mathbf{C}^{(s)}$,

as covariance matrices are positive semi-definite and are therefore symmetric. Furthermore, increasing the number of estimated parameters also increases the required dimension of the output of each network, \mathbf{z}_L , and therefore increases the dimension of the hidden layers parameters, \mathbf{A}_l and \mathbf{b}_l , $l \in 1, \dots, L$, required to obtain a comparable estimation power, drastically increasing the number of network parameters that must be learnt at each step.

A diagonal covariance also allows for an efficient implementation of the multivariate Gaussian distribution. For example, we do not need to compute a full matrix inverse $\mathbf{C}^{(s)-1}$ when computing the density of a multivariate Gaussian distribution; if we know that the covariance $\mathbf{C}^{(s)}$ is diagonal, we can simply invert the diagonal element-wise, as $\mathbf{C}^{(s)-1} = \text{diag}(1/\mathbf{c}^{(s)})$ for diagonal $\mathbf{C}^{(s)}$.

Mixture weights In Eq. (9) we impose that the mixture weights are uniformly equal to S^{-1} , where S is the number of mixture components. We can loosen this restriction and learn the mixture weights, but this is a difficult task. Much of this difficulty comes from the fact that changes in the log-likelihood can now be brought about by changing either the component parameters $\boldsymbol{\mu}^{(s)}$ and $\mathbf{C}^{(s)}$, or the mixture weights, which we denote here by $m^{(s)}$.

For each mixture component, the mean parameter $\boldsymbol{\mu}^{(s)}$ and the covariance parameter $\mathbf{C}^{(s)}$ are vector valued and matrix valued, although we restrict the covariance to be diagonal as Eq. (9). The mean and covariance of each component are thus determined by $2d_x$ values; however, the mixture weight $m^{(s)}$ is a single value, which has a far larger impact on the likelihood than any single element of the mean or covariance. Therefore, the gradient of the log-likelihood, computed via Eq. (10) with backpropagation, with respect to the network parameters $\boldsymbol{\theta}^{(\pi)}$ and $\boldsymbol{\theta}^{(f)}$, will be primarily attributable to the effect these parameters have on the mixture weight, which will result in convergence issues as only the mixture weights will meaningfully change between iterations. Identifiability would also be a problem, as changes in the likelihood could be attributed to changes in either the mean/covariance, or changes to the mixture weights.

The parameters would thus be significantly more difficult to train, and the method given in Alg. 3 would not be sufficient. A potential solution would be to introduce two more networks, each outputting only the mixture weights for the transition and proposal distributions respectively, and then to train these networks as part of the iterative step of Alg. 4, thereby training 4 networks in an alternating scheme. For the sake of brevity, and for the clarity of this paper, we utilise the equal weighted mixture, which we note can well approximate arbitrarily weighted mixtures with a sufficiently large number of components.

4.3. Use of an alternating scheme when estimating network parameters

In order to learn the network parameters $\boldsymbol{\theta}^{(f)}$ and $\boldsymbol{\theta}^{(\pi)}$, Alg. 3 uses an alternating scheme to learn each parameter conditional on the value of the other. Learning one parameter conditional on the other stabilises inference, as the parameters heavily influence each other given that the proposal and transition interact in the weighting step of Alg. 2. Learning in such a manner may lead to identifiability issues, as any change in the weights or particles, and hence in the log-likelihood, can be attributed to either a change in the transition distribution, or a change in the proposal distribution. Furthermore, changes in one distribution should be reflected by changes in the other distribution, as both are tightly linked in interpretation and usage within the filter.

However, if we were to update networks $\text{NN}^{(f)}$ and $\text{NN}^{(\pi)}$ simultaneously, i.e., we replace Alg. 4 with a gradient update of both $\boldsymbol{\theta}^{(f)}$ and $\boldsymbol{\theta}^{(\pi)}$ at the same time, we cannot attribute changes in the likelihood to a specific network, nor can we update each network conditional on the changes made to the other network. In addition, learning both networks at the same time can lead to

divergence, as each network may change in such a way that they yield numerically incompatible distributions; in such distributions the transition distribution has near zero probability mass where the proposal has large probability mass, thereby leading to numerically zero weights when computed in line 7 of Alg. 2.

In Alg. 3 we learn each distribution conditional on the other, and therefore do not suffer from an identifiability problem, as any changes are attributable only to the learned distribution. Further, we learn the effects the changes made to each distribution have on the other, hence stabilising the learning, as the distributions do not adapt at the same time. Alternating between learning and conditioning on each distribution allows the distribution to adapt to each other, and mitigates the potential problem of distributions diverging in density.

4.4. Combating likelihood degeneracy when estimating parameters using the particle filter

Our method learns $\theta^{(f)}$ and $\theta^{(\pi)}$ by maximising the estimated log likelihood, $\ell([\theta^{(f)}, \theta^{(\pi)}])$, using an alternating scheme. We estimate the log-likelihood using Eq. (10), which, in turn, uses the particle weights $w_t^{(k)}$, $t = 1, \dots, T$, $k = 1, \dots, K$, from all iterations of the filter. Therefore, the success of our method hinges on the accurate and stable computation of these weights, which is a challenge in particle filtering, as the weight $w_t^{(k)}$ of a given particle $\mathbf{x}_t^{(k)}$ can be very close to zero. One way to numerically stabilise weight computations is to use log weights [11]. Implementing log weights is a simple change, as we need only rewrite the weight calculation in line 7 of Alg. 2 to use log likelihoods, and rewrite the normalisation to remain on the log scale.

We define $\text{logsumexp}([x_1, x_2, \dots, x_N]) := \log(\sum_{n=1}^N \exp(x_n))$.² Note that

$$\begin{aligned} \log(w_t^{(k)}) &= \log\left(\frac{g(\mathbf{y}_t|\mathbf{x}_t^{(k)}; \theta^{(g)})f(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{(k)}; \theta^{(f)})}{\pi(\mathbf{x}_t|\mathbf{x}_{t-1}^{(k)}, \mathbf{y}_t; \theta^{(\pi)})}\right), \\ &= \log(g(\mathbf{y}_t|\mathbf{x}_t^{(k)}; \theta^{(g)})) + \log(f(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{(k)}; \theta^{(f)})) - \log(\pi(\mathbf{x}_t|\mathbf{x}_{t-1}^{(k)}, \mathbf{y}_t; \theta^{(\pi)})). \end{aligned} \quad (12)$$

By writing $p_t^{(k)} = \tilde{w}_{t-1}^{(k)} w_t^{(k)}$, and noting that $\log(p_t^{(k)}) = \log(\tilde{w}_{t-1}^{(k)}) + \log(w_t^{(k)})$, we have

$$\begin{aligned} \tilde{w}_t^{(k)} &= \frac{p_t^{(k)}}{\sum_{k=1}^K p_t^{(k)}} = \frac{\exp(\log(p_t^{(k)}))}{\sum_{k=1}^K \exp(\log(p_t^{(k)}))}, \\ \log(\tilde{w}_t^{(k)}) &= \log(p_t^{(k)}) - \log\left(\sum_{k=1}^K \exp(\log(p_t^{(k)}))\right), \\ &= \log(p_t^{(k)}) - \text{logsumexp}\left([\log(p_t^{(k)})]_{k=1}^K\right), \\ &= \log(\tilde{w}_{t-1}^{(k)}) + \log(w_t^{(k)}) - \text{logsumexp}\left([\log(\tilde{w}_{t-1}^{(k)}) + \log(w_t^{(k)})]_{k=1}^K\right), \end{aligned} \quad (13)$$

and can construct the log-likelihood estimator following Eq. (10) directly using the log weights. The use of log weights significantly improves numerical stability; nevertheless, it does not address the issue of likelihood concentration, nor does it mitigate the problem of learning parameters with initial value having very low likelihood.

²The logsumexp can be additionally stabilised by observing that $\text{logsumexp}([x_n]_{n=1}^N) = \max([x_n]_{n=1}^N) + \text{logsumexp}([x_n - \max([x_m]_{m=1}^N)]_{n=1}^N)$, which helps to avoid numerical overflow when evaluating logsumexp. Note that this modification is typically already present in pre-existing implementations of logsumexp such as those present in [43, 44, 45].

State-space models in general suffer from likelihood concentration, where $\ell(\theta|\mathbf{y}_{1:T})$, which we estimate by Eq. (10), becomes increasingly concentrated around a single value of θ as T increases. Likelihood concentration thereby results in vanishingly small likelihoods for parameter values that are not well adapted to the system, leading to the gradient of the parameter values being numerically zero, and the parameter is hence unable to be learnt, without careful initialisation near a value with high likelihood. The issue is therefore that we must choose an initial parameter that has high likelihood, but we assume we do not know the parameter beforehand; these statements contradict each other.

We address likelihood concentration, and hence the problem of learning parameters under random initialisation, using observation batching. We infer our parameters, $\theta^{(f)}$ and $\theta^{(\pi)}$, in Alg. 5 using B increasingly large batches $\mathbf{y}^{(b)}$, $b \in \{1, \dots, B\}$, of the observation series, where $\mathbf{y}^{(1)} \subseteq \mathbf{y}^{(2)} \subseteq \dots \subseteq \mathbf{y}^{(B-1)} \subseteq \mathbf{y}^{(B)}$, choosing $\mathbf{y}^{(B)} := \mathbf{y}$. Warming up the $\theta^{(f)}$ and $\theta^{(\pi)}$ parameters before learning on the entire series mitigates the effect of likelihood concentration for unadapted parameters, and, when combined with log weights, allows our method to be used on long observation series for complex problems.

We initially learn our parameters, $\theta^{(f)}$ and $\theta^{(\pi)}$, on a small subset, $\mathbf{y}^{(1)}$, of the observation series \mathbf{y} . Thereby, we obtain a relatively diffuse parameter likelihood function compared to that obtained with a longer observation series. Therefore, we can initialise the parameters of the neural networks, \mathbf{A}_l , \mathbf{b}_l , $l \in 1, \dots, L$, randomly, as the likelihood and its gradient will not be numerically zero for unadapted parameters, due to the relatively diffuse parameter likelihood function. Further, under the above method, our estimated f and π distributions incorporate information from the observation series \mathbf{y} in sequence, adapting to one batch of observations $\mathbf{y}^{(b)}$ before taking in more. Sequentially incorporating observation information into our estimation prevents behaviour where the start and end of the series are well represented by the learnt parameters, but the middle is not; this is a common problem when using neural networks to learn between-step dynamics in time series models [46, 47, 48].

5. Numerical Experiments

We will now illustrate the performance of our proposed method on two systems. First, we will test our method on a non-linear polynomial system: the Lorenz 96 chaotic oscillator. We will then apply our method to a non-linear non-polynomial system: the Kuramoto oscillator. In both instances we compare our method to the improved auxiliary particle filter (IAPF) [20] and the bootstrap particle filter (BPF) [8].

5.1. Lorenz 96 model

We consider a stochastic version of the Lorenz 96 model [49], a dynamical system known to exhibit chaotic behaviour. We insert additive noise terms to obtain a stochastic system to use for testing, and discretise using the Euler-Maruyama scheme, resulting in the system

$$\begin{aligned} x_{i,t+1} &= x_{i,t} + \Delta t(x_{i-1,t}(x_{i+1,t} - x_{i-2,t}) - x_{i,t} + F) + \sqrt{\Delta t} \cdot v_{i,t+1}, \\ y_{i,t+1} &= x_{i,t+1} + \sqrt{\Delta t} \cdot r_{i,t+1}, \end{aligned} \tag{14}$$

for $i \in \{1, \dots, d_x\}$ and $t \in \{0, \dots, T\}$, where we define $x_{-1} := x_{d_x-1}$, $x_0 := x_{d_x}$, and $x_{d_x+1} := x_1$, $v_t \sim \mathcal{N}(\mathbf{0}, \Sigma_v)$, $r_t \sim \mathcal{N}(\mathbf{0}, \Sigma_r)$, and F is a forcing constant; we use $F = 8$.

We set $\Delta t = 0.05$ in Eq. (14). Unless explicitly stated otherwise, we choose the dimension of the system as $d_x = d_y = 20$, and set $\Sigma_v = 0.25\mathbf{I}_{d_x}$ and $\Sigma_r = 0.1\mathbf{I}_{d_x}$. We initialise the hidden

state at \mathbf{x}_0 such that $x_{1,0} = 1$, with all other elements equal to 0. We assess the proposed method in terms of relative mean square error (MSE), showing the accuracy of the method as a fraction of the MSE obtained with the BPF [8]. We compare our method with the improved auxiliary particle filter (IAPF) [20], to illustrate the performance of a standard improved proposal. The MSE compares the weighted mean of the samples (the estimated state) with the true underlying hidden state. Note that the MSE is not targeted for optimisation. For both the IAPF and the BPF, we assume the state transition model is known. We compute the MSE for 200 independent runs of the filter, and plot the mean and symmetric 95% intervals.

We test the proposed method using a variable number of mixture components, with $S \in \{1, 6, 10\}$. All variants utilise the same network architecture, with 3 layers of output sizes $d_1 = 128, d_2 = 256, d_3 = 2Sd_x$ for both networks. For the activation function ρ_l in Eq. (7), we set $\rho_{1,2}(x) = \text{relu}(x) = \max(0, x)$, and $\rho_3(x) = x$, with this applying to both the proposal and transition networks. We train StateMixNN using the ADAM optimiser [29], using a fixed learning rate of $3 \cdot 10^{-3}$, and setting the parameters of Alg. 3 to $B = \lceil T/5 \rceil, J = 50, A = 20$. We train the method using a series of observations distinct from those on which we test StateMixNN; however, all series are instances of the Lorenz 96 system.

Variable number of particles. We test StateMixNN for a variable number of particles K , with $K \in \{30, 50, 100, 200\}$. We use a fixed series length $T = 100$. Fig. 1 shows that StateMixNN outperforms the BPF for all given values of K , obtaining at most 0.9 times the MSE of the BPF, and typically less than 0.75 the MSE. StateMixNN with $S = 10$ components suffers with few particles, performing worse than the other parameterisations of StateMixNN, as few samples are taken from each component, therefore the gradient estimates are more dispersed, making training less reliable. Our method outperforms the IAPF at all tested numbers of particles, by an increasingly large margin as the number of particles increases.

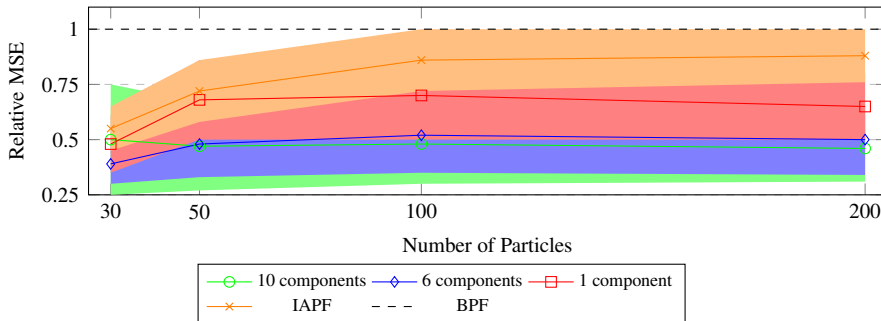


Figure 1: Comparison of StateMixNN with the BPF and IAPF over variable numbers of particles. Quantities are divided by the MSE of the corresponding BPF. The lines denote mean performance, with bands denoting symmetric 95% intervals.

Variable series length. Next, we test StateMixNN with a variable series length T , with $T \in \{30, 60, 100, 200, 500\}$. In this case we fix the number of particles $K = 100$. We show in Fig. 2 that the proposed method obtains lower values of MSE than the BPF and IAPF for all given values of T . The $S = 10$ component method slightly outperforms the $S = 6$ component method, which significantly outperforms the $S = 1$ component method.

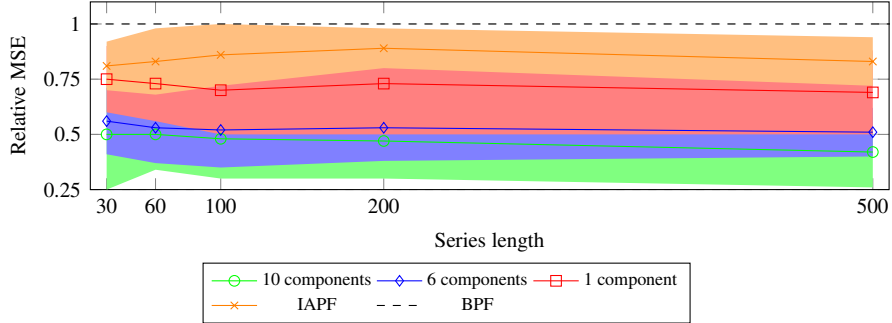


Figure 2: Comparison of StateMixNN with the BPF and IAPF over variable series length. Quantities are divided by the MSE of the corresponding BPF. The lines denote mean performance, with bands denoting symmetric 95% intervals.

Variable state noise. We now test StateMixNN for a variable state noise $\Sigma_v = \sigma_v^2 \mathbf{I}_{20}$, with $\sigma_v^2 \in \{0.05, 0.1, 0.25, 0.5, 1\}$. In this case, we fix the number of particles $K = 100$ and the series length $T = 100$. Fig. 3 shows that StateMixNN is superior to the BPF for all given values of σ_v^2 . The improvement in accuracy is lesser for small noise variances, as small perturbations give a concentrated distribution of the state values at the next time step. However, the performance improves for larger values of σ_v^2 . This is due to the chaotic behaviour of the system, which leads to multimodal distributions for the next state, as the state follows one of several diverging paths. The mixture in the proposed method captures this behaviour, with components representing different modes, thereby outperforming both the BPF and the IAPF.

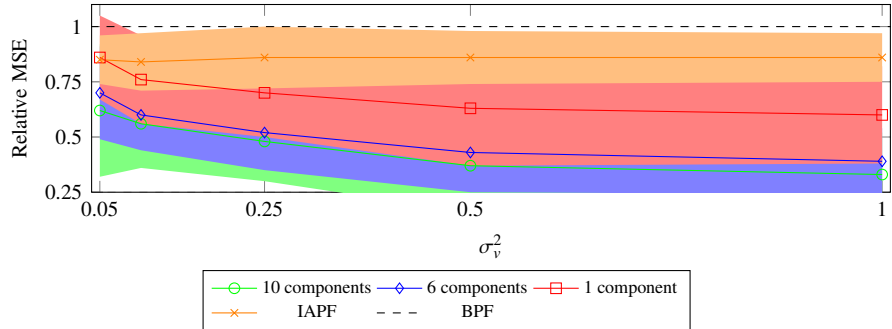


Figure 3: Comparison of StateMixNN with the BPF and IAPF over variable state noise magnitude. Quantities are divided by the MSE of the corresponding BPF. The lines denote mean performance, with bands denoting symmetric 95% intervals.

Variable system dimension.

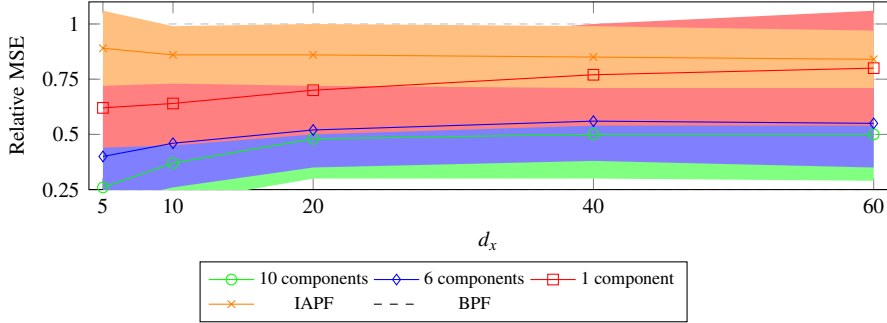


Figure 4: Comparison of StateMixNN with the BPF and IAPF over variable system dimension. Quantities are divided by the MSE of the corresponding BPF. The lines denote mean performance, with bands denoting symmetric 95% intervals.

Finally, we test StateMixNN over variable state and observation dimension $d_x = d_y$, with $d_x \in \{5, 10, 20, 40, 60\}$. We fix the number of particles $K = 100$, and the series length $T = 100$. Fig. 4 shows that StateMixNN is superior to the BPF for all given values of d_x . We observe that the margin of out-performance decreases as the state dimension increases, but seems to stabilise after $d_x = 40$. This is due to the static number of iterations used in training, as the state-space is easier to learn for smaller d_x , and therefore requires fewer iterations to achieve the same level of performance. We observe that StateMixNN performs well at all tested values of d_x , and does not rapidly deteriorate in performance when increasing d_x without changing the training regime. Furthermore, the filters with a larger number of components in the learned distributions outperform those with a smaller number of components at all times, displaying the mixture distributions ability to approximate complex systems.

5.2. Kuramoto oscillator

The Kuramoto oscillator [50] is a mathematical model that describes the behavior of a system of d_x phase-coupled oscillators. The model is described by

$$\frac{d\theta_i}{dt} = \omega_i + d_x^{-1} \sum_{n=1}^{d_x} K \sin(\theta_i - \theta_j), \quad (15)$$

where θ_i denotes the phase of the i th oscillator, and $K \in \mathbb{R}$ is the coupling constant between oscillators. This does not restrict θ however, which will, in general, diverge to an infinity as $t \rightarrow \infty$. To address this, we transform Eq. (15) by introducing derived parameters $R \in \mathbb{R}$ and $\phi \in \mathbb{R}$ such that

$$R \exp(\sqrt{-1}\phi) = d_x^{-1} \sum_{j=1}^{d_x} \exp(\sqrt{-1}\theta_j), \quad (16)$$

$$\frac{d\theta_i}{dt} = \omega_i + KR \sin(\phi - \theta_i),$$

which restricts $\theta \in [-\pi, \pi]^{d_x}$. We insert additive Gaussian noise to Eq. (16), and discretise using the Euler-Maruyama scheme, yielding the NLSSM

$$\begin{aligned}
 R \exp(\sqrt{-1}\phi) &= d_x^{-1} \sum_{j=1}^{d_x} \exp(\sqrt{-1}x_{j,t}), \\
 x_{i,t+1} &= x_{i,t} + \Delta t (\omega_i + KR \sin(\phi - x_i)) + \sqrt{\Delta t} \cdot v_{i,t+1}, \\
 y_{i,t+1} &= x_{i,t+1} + \sqrt{\Delta t} \cdot r_{i,t+1},
 \end{aligned} \tag{17}$$

for $i \in \{1, \dots, d_x\}$. We choose $d_x = 20$, and $K = 0.8$. We set $\Sigma_v = \sigma_v^2 \text{Id}_{20}$, $\Sigma_r = \sigma_r^2 \text{Id}_{20}$, with $\sigma_v = 1$, $\sigma_r = 0.05$ unless stated otherwise, which tests the performance of our method in the case that the observation is much more informative than the state, where the standard BPF is known to suffer. We discretise this model with a time step of $\Delta t = 0.05$. We sample $\omega_i \sim \mathcal{N}(0.5, 0.5^2)$, and $\mathbf{x}_{i,0} \sim U(-\pi, \pi)$. We run the system until $t = 10$, and then begin collecting observations.

Variable series length. We test StateMixNN with a variable series length T , with $T \in \{30, 60, 100, 200, 500\}$ on the Kuramoto oscillator. In this case we fix the number of particles $K = 100$. We show in Fig. 5 that the proposed method obtains lower values of MSE than the BPF and IAPF for all given values of T . The $S = 10$ component method outperforms the $S = 6$ component method, which in turn outperforms the $S = 1$ component method.

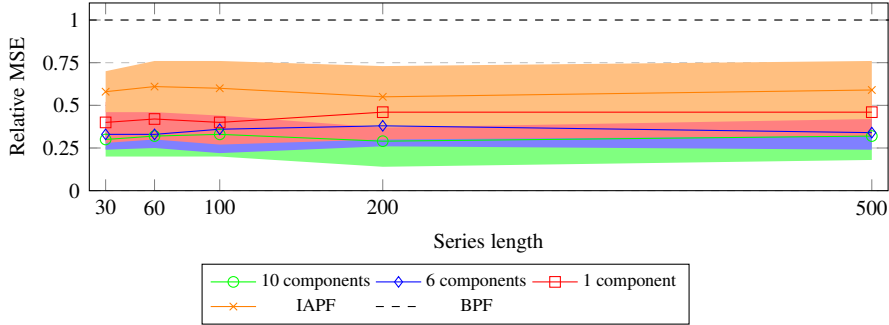


Figure 5: Comparison of StateMixNN with the BPF and IAPF over variable series length on the Kuramoto oscillator. Quantities are divided by the MSE of the corresponding BPF. The lines denote mean performance, with bands denoting symmetric 95% intervals.

Variable number of particles. We test StateMixNN for a variable number of particles K , with $K \in \{30, 50, 100, 200\}$. We use a fixed series length $T = 100$. Fig. 6 shows that StateMixNN outperforms the BPF and IAPF for all given values of K . StateMixNN with $S = 10$ components suffers with few particles, performing worse than the other parameterisations of StateMixNN, as few samples are taken from each component, therefore the gradient estimates are more dispersed, making training less reliable.

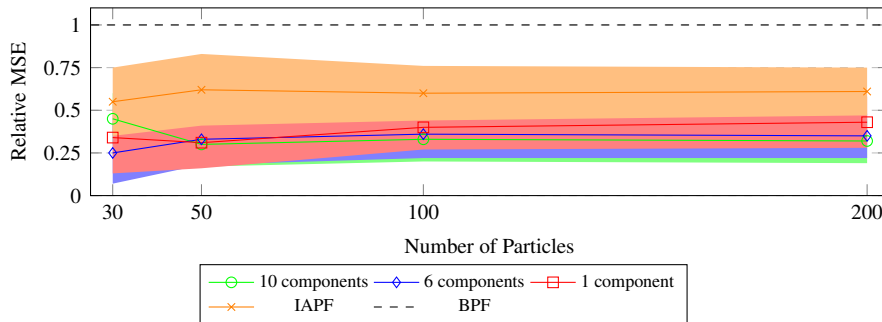


Figure 6: Comparison of StateMixNN with the BPF and IAPF over variable number of particles on the Kuramoto oscillator. Quantities are divided by the MSE of the corresponding BPF. The lines denote mean performance, with bands denoting symmetric 95% intervals.

6. Conclusion

This work proposes a novel method, called StateMixNN, which simultaneously learns the transition and proposal distributions of a particle filter. We utilise a pair of multivariate Gaussian mixture distributions to approximate the transition and proposal distributions, with the means and covariances of the mixands given by the output of a dense neural network. The proposed method does not require knowledge of the hidden state, as we optimise the observation likelihood, which requires only the observations to be known. We show some numerical results for a stochastic Lorenz 96 model, which has highly chaotic behaviour. We observe that our method outperforms the bootstrap particle filter, a standard method, as well as the improved auxiliary particle filter, a state-of-the-art method for improving the proposal distribution.

References

- [1] Xuedong Wang, Tiancheng Li, Shudong Sun, and Juan M Corchado. A survey of recent advances in particle filters and remaining challenges for multitarget tracking. *Sensors*, 17(12):2707, 2017.
- [2] Audronė Virbickaitė, Hedibert F Lopes, M Concepción Ausín, and Pedro Galeano. Particle learning for Bayesian semi-parametric stochastic volatility model. *Econometric Reviews*, 2019.
- [3] Toby A Patterson, Alison Parton, Roland Langrock, Paul G Blackwell, Len Thomas, and Ruth King. Statistical modelling of individual animal movement: an overview of key methods and a discussion of practical challenges. *AStA Advances in Statistical Analysis*, 101:399–438, 2017.
- [4] Ken Newman, Ruth King, Víctor Elvira, Perry de Valpine, Rachel S McCrea, and Byron JT Morgan. State-space models for ecological time-series data: Practical model-fitting. *Methods in Ecology and Evolution*, 14(1):26–42, 2023.
- [5] Adam M Clayton, Andrew C Lorenc, and Dale M Barker. Operational implementation of a hybrid ensemble/4d-Var global data assimilation system at the Met Office. *Quarterly Journal of the Royal Meteorological Society*, 139(675):1445–1461, 2013.
- [6] Maria Isabel Ribeiro. Kalman and extended Kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics*, 43(46):3736–3741, 2004.
- [7] Eric A Wan and Rudolph Van Der Merwe. The unscented Kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pages 153–158. IEEE, 2000.
- [8] Neil Gordon, David Salmond, and Adrian Frederick Melhuish Smith. Novel approach to nonlinear and non-Gaussian Bayesian state estimation. *IEE Proceedings-F Radar and Signal Processing*, 140:107–113, 1993.
- [9] Petar M Djuric, Jayesh H Kotecha, Jianqui Zhang, Yufei Huang, Tadesse Ghirmai, Mónica F Bugallo, and Joaquin Miguez. Particle filtering. *IEEE signal processing magazine*, 20(5):19–38, 2003.

- [10] Arnaud Doucet, Adam M Johansen, et al. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 12(656-704):3, 2009.
- [11] Simo Särkkä. *Bayesian filtering and smoothing*. Cambridge University Press, 1st edition, 2013.
- [12] Adrien Corenflos, James Thornton, George Deligiannidis, and Arnaud Doucet. Differentiable particle filtering via entropy-regularized optimal transport. In *International Conference on Machine Learning*, pages 2100–2111. PMLR, 2021.
- [13] Adam Ścibior and Frank Wood. Differentiable particle filtering without modifying the forward pass. *arXiv preprint arXiv:2106.10314*, 2021.
- [14] Xiongjie Chen and Yunpeng Li. An overview of differentiable particle filters for data-adaptive sequential Bayesian inference. *arXiv preprint arXiv:2302.09639*, 2023.
- [15] Wenhan Li, Xiongjie Chen, Wenwu Wang, Víctor Elvira, and Yunpeng Li. Differentiable bootstrap particle filters for regime-switching models. *arXiv preprint arXiv:2302.10319*, 2023.
- [16] Peter Karkus, David Hsu, and Wee Sun Lee. Particle filter networks with application to visual localization. In *Conference on robot learning*, pages 169–178. PMLR, 2018.
- [17] Michael K Pitt and Neil Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American statistical association*, 94(446):590–599, 1999.
- [18] Víctor Elvira, Luca Martino, Monica F Bugallo, and Petar M Djuric. Elucidating the auxiliary particle filter via multiple importance sampling [lecture notes]. *IEEE Signal Processing Magazine*, 36(6):145–152, 2019.
- [19] Nicola Branchini and Víctor Elvira. Optimized auxiliary particle filters: adapting mixture proposals via convex optimization. In *Uncertainty in Artificial Intelligence*, pages 1289–1299. PMLR, 2021.
- [20] Víctor Elvira, Luca Martino, Mónica F. Bugallo, and Petar M. Djurić. In search for improved auxiliary particle filters. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 1637–1641, 2018. doi: 10.23919/EUSIPCO.2018.8553361.
- [21] Shixiang Shane Gu, Zoubin Ghahramani, and Richard E Turner. Neural adaptive sequential Monte Carlo. *Advances in neural information processing systems*, 28, 2015.
- [22] Christian Naesseth, Scott Linderman, Rajesh Ranganath, and David Blei. Variational sequential Monte Carlo. In *International conference on artificial intelligence and statistics*, pages 968–977. PMLR, 2018.
- [23] Benjamin Cox, Sara Pérez-Vieites, Nicolas Zilberstein, Martin Sevilla, Santiago Segarra, and Víctor Elvira. End-to-end learning of gaussian mixture proposals using differentiable particle filters and neural networks. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 9701–9705. IEEE, 2024.
- [24] Víctor Elvira, Luca Martino, David Luengo, and Mónica F Bugallo. Improving population monte carlo: Alternative weighting and resampling schemes. *Signal Processing*, 131:77–91, 2017.
- [25] Fernando Gama, Nicolas Zilberstein, Richard G Baraniuk, and Santiago Segarra. Unrolling particles: Unsupervised learning of sampling distributions. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5498–5502. IEEE, 2022.
- [26] Fernando Gama, Nicolas Zilberstein, Martin Sevilla, Richard G. Baraniuk, and Santiago Segarra. Unsupervised learning of sampling distributions for particle filters. *IEEE Transactions on Signal Processing*, 71:3852–3866, 2023. doi: 10.1109/TSP.2023.3324221.
- [27] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [28] Michael Zhu, Kevin Murphy, and Rico Jonschkowski. Towards differentiable resampling. *arXiv preprint arXiv:2004.11938*, 2020.
- [29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL <http://arxiv.org/abs/1412.6980>.
- [30] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.
- [31] Boris Ginsburg, Patrice Castonguay, Oleksii Hrinchuk, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, Huyen Nguyen, Yang Zhang, and Jonathan M Cohen. Stochastic gradient methods with layer-wise adaptive moments for training of deep networks. *arXiv preprint arXiv:1905.11286*, 2019.
- [32] Jason Li, Vitaly Lavrukhin, Boris Ginsburg, Ryan Leary, Oleksii Kuchaiev, Jonathan M Cohen, Huyen Nguyen, and Ravi Teja Gadde. Jasper: An end-to-end convolutional neural acoustic model. *arXiv preprint arXiv:1904.03288*, 2019.
- [33] Monica F Bugallo, Víctor Elvira, Luca Martino, David Luengo, Joaquin Miguez, and Petar M Djuric. Adaptive importance sampling: The past, the present, and the future. *IEEE Signal Processing Magazine*, 34(4):60–79, 2017.
- [34] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.
- [35] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.
- [36] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent

- neural networks. *arXiv preprint arXiv:1801.01078*, 2017.
- [37] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021.
- [38] Patrick Kidger and Cristian Garcia. Equinox: neural networks in JAX via callable PyTrees and filtered transformations. *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.
- [39] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.
- [40] Brian L Trippe and Richard E Turner. Conditional density estimation with bayesian normalising flows. *arXiv preprint arXiv:1802.04908*, 2018.
- [41] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.
- [42] Esteban G Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.
- [43] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
- [44] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL <http://github.com/google/jax>.
- [45] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>. Software available from tensorflow.org.
- [46] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.
- [47] Hanshu Yan, Jiawei Du, Vincent YF Tan, and Jiashi Feng. On robustness of neural ordinary differential equations. *arXiv preprint arXiv:1910.05513*, 2019.
- [48] Chris Rackauckas, Mike Innes, Yingbo Ma, Jesse Bettencourt, Lyndon White, and Vaibhav Dixit. Diffeqflux.jl: a julia library for neural differential equations. *arXiv preprint arXiv:1902.02376*, 2019.
- [49] Edward N Lorenz. Predictability: A problem partly solved. In *Proc. Seminar on predictability*. ECMWF, 1996.
- [50] Yoshiki Kuramoto. *Chemical Oscillations, Waves, and Turbulence*. Springer, 1984. ISBN 978-3-642-69691-6. doi: 10.1007/978-3-642-69689-3.