Recurrent Neural Filters: Learning Independent Bayesian Filtering Steps for Time Series Prediction

Bryan Lim, Stefan Zohren and Stephen Roberts
Oxford-Man Institute of Quantitative Finance
Department of Engineering Science
University of Oxford
Oxford, UK
{blim,zohren,sjrob}@robots.ox.ac.uk

***Abstract*—Despite the recent popularity of deep generative state space models, few comparisons have been made between network architectures and the inference steps of the Bayesian filtering framework – with most models simultaneously approximating both state transition and update steps with a single recurrent neural network (RNN). In this paper, we introduce the Recurrent Neural Filter (RNF), a novel recurrent autoencoder architecture that learns distinct representations for each Bayesian filtering step, captured by a series of encoders and decoders. Testing this on three real-world time series datasets, we demonstrate that the decoupled representations learnt improve the accuracy of one-step-ahead forecasts while providing realistic uncertainty estimates, and also facilitate multistep prediction through the separation of encoder stages.**

***Index Terms*—recurrent neural networks, Bayesian filtering, variational autoencoders, multistep forecasting**

I. Introduction

Bayesian filtering [1] has been extensively used within the domain of time series prediction, with numerous applications across different fields – including target tracking [2], robotics [3], finance [4], and medicine [5]. Performing inference via a series of prediction and update steps [6], Bayesian filters recursively update the posterior distribution of predictions – or the belief state [7] – with the arrival of new data. For many filter models – such as the Kalman filter [8] and the unscented Kalman filter [9] – deterministic functions are used at each step to adjust the sufficient statistics of the belief state, guided by generative models of the data. Each function quantifies the impact of different sources of information on latent state estimates – specifically time evolution and exogenous inputs in the prediction step, and realised observations in the update step. On top of efficient inference and uncertainty estimation, this decomposition of inference steps enables Bayes filters to be deployed in use cases beyond basic one-step-ahead prediction – with simple extensions for multistep prediction [10] and prediction in the presence of missing observations [11].

With the increasing use of deep neural networks for time series prediction, applications of recurrent variational autoencoder (RVAE) architectures have been investigated for forecasting non-linear state space models [12]–[15]. Learning dynamics directly from data, they avoid the need for explicit model specification – overcoming a key limitation in standard Bayes filters. However, these RVAEs focus predominantly on encapsulating the generative form of the state space model – implicitly condensing both state transition and update steps into a single representation learnt by the RVAE decoder – and make it impossible to decouple the Bayes filter steps.

Recent works in deep generative modelling have focused on the use of neural networks to learn independent factors of variation in static datasets – through the encouragement of disentangled representations [16], [17] or by learning causal mechanisms [18], [19]. While a wide range of training procedures and loss functions have been proposed [20], methods in general use dedicated network components to learn distinct interpretable relationships – ranging from orthogonalising latent representations in variational autoencoders [21] to learning independent modules for different causal pathways [18]. By understanding the relationships encapsulated by each component, we can subsequently decouple them for use in related tasks – allowing the learnt mechanisms to generalise to novel domains [18], [22] or to provide building blocks for transfer learning [21].

In this paper, we introduce the Recurrent Neural Filter (RNF) – a novel recurrent autoencoder architecture which aligns network modules (encoders and decoders) with the inference steps of the Bayes filter – making several contributions over standard approaches. Firstly, we propose a new training procedure to encourage independent representations within each module, by directly training intermediate encoders with a common emission decoder. In doing so, we augment the loss function with additional regularisation terms (see Section V), and directly encourage each encoder to learn functions to update the filter's belief state given available information. Furthermore, to encourage the decoupling of encoder stages, we randomly drop out the input dynamics and error correction encoders during training – which can be viewed as artificially introducing missingness to the inputs and observations respectively. Finally, we highlight performance gains for one-step-ahead predictions through experiments on 3 real-world time series datasets, and investigate multistep predictions as a use case for generalising the RNF's decoupled representations to other tasks – demonstrating performance improvements from the recursive application of the state transition encoders alone.

II. Related Work

RVAEs for State Space Modelling: The work of [12] identifies close parallels between RNNs and latent state space models, both consisting of an internal hidden state that drives output forecasts and observations. Using an RVAE architecture described as a variational RNN (VRNN), they build their recognition network (encoder) with RNNs and produce samples for the stochastic hidden state at each time point. Deep Kalman filters (DKFs) [13], [14] take this a step further by allowing for exogenous inputs in their network and incorporating a special KL loss term to penalise state transitions between time steps. Deep Variational Bayes Filters (DVBFs) [15] enhance the interpretability of DKFs by modelling state transitions with parametric – e.g. linear – models, which take in stochastic samples from the recognition model as inputs. In general, while the above models capture the generative modelling aspects of the state space framework, their inference procedure blends both state transition and error correction steps, obliging the recognition model to learn representations for both simultaneously. In contrast, the RNF uses separate neural network components to directly model the Bayes filter steps – leading to improvements in representation learning and enhanced predictive performance in time series applications.

Hybrid Approaches: In [23], the authors take a hybrid approach with the structured variational autoencoder (SVAE), proposing an efficient general inference framework that combines probabilistic graphical models for the latent state with neural network observation models. This is similar in spirit to the Kernel Kalman Filter [24], allowing for predictions to be made on complex observational datasets – such as raw images – by encoding high dimensional outputs onto a lower dimensional latent representation modelled with a dynamical systems model. Although SVAEs provide a degree of interpretability to temporal dynamics, they also require a parametric model to be defined for the latent states which may be challenging for arbitrary time series datasets. The RNF, in comparison, can learn the relationships directly from data, without the need for explicit model specification. The Kalman variational autoencoder (KVAE) [25] extends ideas from the SVAE, modelling latent state using a linear Gaussian state space model (LGSSM). To allow for non-linear dynamics, the KVAE uses a recognition model to produce time-varying parameters for the LGSSM, weighting a set of K constant parameters using weights generated by a neural network. Deep State Space Models (DSSM) [26] investigate a similar approach within the context of time series prediction, using an RNN to generate parameters of the LGSSM at each time step. While the LGSSM components do allow for the application of the Kalman filter, we note that updates to the time-varying weights from the RNN once again blend the prediction and update steps – making the separation of Bayes filter steps and generalisation to other tasks non-trivial. On the other hand, the RNF naturally supports simple extensions (e.g. multistep prediction) similarly to other Bayes filter – due to the close alignment of the RNF architecture with the Bayes filter steps and the use of decoupled representations across encoders and decoders.

Autoregressive Architectures: An alternative approach to deep generative modelling focuses on the autoregressive factorisation of the joint distribution of observations (i.e. $p(\boldsymbol{y}_{1:T}) = \prod_t p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t})$), directly generating the conditional distribution at each step. For instance, WaveNet [27] and Transformer [28], [29] networks use dilated CNNs and attention-based models to build predictive distributions. While successful in speech generation and language applications, these models suffer from several limitations in the context of time series prediction. Firstly, the CNN and attention models require the pre-specification of the amount of relevant history to use in predictions – with the size of the look-back window controlled by the length of the receptive field or extended context – which may be difficult when the data generating process is unknown. Furthermore, they also rely on a discretisation of the output, generating probabilities of occurrence within each discrete interval using a softmax layer. This can create generalisation issues for time series where outputs are unbounded. In contrast, the LSTM cells used in the RNF recognition model remove the need to define a look-back window, and the parametric distributions used for outputs are compatible with unbounded continuous observations.

In other works, the use of RNNs in autoregressive architectures for time series prediction have been explored in DeepAR models [30], where LSTM networks output Gaussian mean and standard deviation parameters of predictive distributions at each step. We include this as a benchmark in our tests, noting the improvements observed with the RNF through its alignment with the Bayesian filtering paradigm.

Predictive State Representations: Predictive state RNNs (PSRNN) [31]–[33] use an alternative formulation of the Bayes filter, utilising a state representation that corresponds to the statistics of the predictive distribution of future observations. Predictions are made using a two-stage regression approach modelled by their proposed architectures. Compared to alternative approaches, PSRNNs only produce point estimates for their forecasts – lacking the uncertainty bounds from predictive distributions produced by the RNF.

Non-Parametric State Space Models: Gaussian Process state space models (GP-SSMs) [34], [35] and variational approximations [36], provide an alternative non-parametric approach to forecasting non-linear state space models – modelling hidden states and observation dynamics using GPs. While they have similar benefits to Bayes filters (i.e. predictive uncertainties, natural multistep prediction etc.), inference at each time step has at least an $O(T)$ complexity in the number of past observations – either via sparse GP approximations or Kalman filter formulations [37]. In contrast, the RNF updates its belief state at each time point only with the latest observations and input, making it suitable for real-time prediction on high-frequency datasets.

RNNs for Multistep Prediction: Customised sequence-to-sequence architectures have been explored in [38], [39] for multistep time series prediction, typically predefining the forecast horizon, and using computationally expensive

customised training procedures to improve performance. In contrast, the RNF does not require the use of a separate training procedure for multistep predictions – hence reducing the computational overhead – and does not require the specification of a fixed forecast horizon.

III. Problem Definition

Let $\boldsymbol{y}_t = [y_t(1), \ldots, y_t(O)]^T$ be a vector of observations, driven by a set of stochastic hidden states $\boldsymbol{x}_t = [x_t(1), \ldots, x_t(J)]^T$ and exogenous inputs $\boldsymbol{u}_t = [u_t(1), \ldots, u_t(I)]^T$. We consider non-linear state space models of the following form:

$$\boldsymbol{y}_t \sim \Pi\big(\ f(\boldsymbol{x}_t\ )\big) \tag{1}$$

$$\boldsymbol{x}_t \sim N\big(\ \mu(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t),\ \Sigma(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t)\ \big) \tag{2}$$

where Π is an arbitrary distribution parametrised by a non-linear function $f(\boldsymbol{x}_t)$, with $\mu(\cdot)$ and $\Sigma(\cdot)$ being mean and covariance functions respectively.

Bayes filters allow for efficient inference through the use of a belief state, i.e. a posterior distribution of hidden states given past observations $\boldsymbol{y}_{1:t} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t\}$ and inputs $\boldsymbol{u}_{1:t} = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_t\}$. This is achieved through the maintenance of a set sufficient statistics $\boldsymbol{\theta}_t$ – e.g. means and covariances $\boldsymbol{\theta}_t \in \{\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\}$ – which compactly summarise the historical data:

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}, \boldsymbol{u}_{1:t}) = bel(\boldsymbol{x}_t; \boldsymbol{\theta}_t) \tag{3}$$

$$= N\left(\boldsymbol{x}_t;\ \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\right) \tag{4}$$

where $bel(.)$ is a probability distribution function for the belief state.

For filters such as the Kalman filter – and non-linear variants like the unscented Kalman filter [9] – $\boldsymbol{\theta}_t$ is recursively updated through a series of prediction and update steps which take the general form:

Prediction (State Transition):

$$\tilde{\boldsymbol{\theta}}_t = \phi_u\left(\boldsymbol{\theta}_{t-1}, \boldsymbol{u}_t\right) \tag{5}$$

Update (Error Correction):

$$\boldsymbol{\theta}_t = \phi_y\left(\tilde{\boldsymbol{\theta}}_t, \boldsymbol{y}_t\right) \tag{6}$$

where $\phi_u(\cdot)$ and $\phi_y(\cdot)$ are non-linear deterministic functions. Forecasts can then be computed using one-step ahead predictive distributions:

$$p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t}) = \int p(\boldsymbol{y}_t|\boldsymbol{x}_t)\ bel\left(\boldsymbol{x}_t; \tilde{\boldsymbol{\theta}}_t\right)\ d\boldsymbol{x}_t. \tag{7}$$

In certain cases – e.g. with the Kalman filter – the predictive distribution can also be directly parameterised using analytical functions $g(.)$ for belief state statistics :

$$p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t}) = p\left(\boldsymbol{y}_t|\ g(\tilde{\boldsymbol{\theta}}_t)\right). \tag{8}$$

When observations are continuous, such as in standard linear Gaussian state space models, $\boldsymbol{y}_t$ can be modelled using a Normal distribution – i.e. $\boldsymbol{y}_t \sim N\left(g_\mu(\tilde{\boldsymbol{\theta}}_t),\ g_\Sigma(\tilde{\boldsymbol{\theta}}_t)\right)$.

IV. Recurrent Neural Filter

Recurrent Neural Filters use a series of encoders and decoders to learn independent representations for the Bayesian filtering steps. We investigate two RNF variants as described below, based on Equations (7) and (8) respectively.

Variational Autoencoder Form (VRNF) Firstly, we capture the belief state of Equation (4) using a recurrent VAE-based architecture. At run time, samples of $\boldsymbol{x}_t$ are generated from the encoder – approximating the integral of Equation (7) to compute the predictive distribution of $\boldsymbol{y}_t$.

Standard Autoencoder Form (RNF)[1] Much recent work has demonstrated the sensitivity of VAE performance to the choice of prior distribution, with suboptimal priors either having an "over-regularising" effect on the loss function during training [40]–[42], or leading to posterior collapse [43]. As such, we also implement an autoregressive version of the RNF based on Equation (8) – directly feeding encoder latent states into the common emission decoder.

A general architecture diagram for both forms is shown in Figure 1, with the main differences encapsulated within $z(s)$ (see Section IV-A).

A. Network Architecture

First, let $\boldsymbol{s}_t$ be a latent state that maps to sufficient statistics $\boldsymbol{\theta}_t$, which are obtained as outputs from our recognition model. Per Equations (5) and (6), inference at run-time is controlled through the recursive update of $\boldsymbol{s}_t$, using a series of Long Short-Term Memory (LSTM) [44] encoders with exponential linear unit (ELU) activations [45].

Encoder To directly estimate the impact of exogenous inputs on the belief state, the prediction step, Equation (5), is divided into two parts with separate LSTM units $\phi_x(\cdot)$ and $\phi_u(\cdot)$. We use $\boldsymbol{h}_t$ to represent all required memory components – i.e. both output vector and cell state for the standard LSTM – with $\boldsymbol{s}_t$ being the output of the cell. A third LSTM cell $\phi_y(\cdot)$ is then used for the update step, Equation (6), with the full set of equations below.

Prediction:

$$\textit{Propagation} \qquad \left[\tilde{\boldsymbol{s}}_t', \tilde{\boldsymbol{h}}_t'\right] = \phi_x(\boldsymbol{h}_{t-1}) \tag{9}$$

$$\textit{Input Dynamics} \qquad \left[\tilde{\boldsymbol{s}}_t, \tilde{\boldsymbol{h}}_t\right] = \phi_u(\tilde{\boldsymbol{h}}_t', \boldsymbol{u}_t) \tag{10}$$

Update:

$$\textit{Error Correction} \qquad [\boldsymbol{s}_t, \boldsymbol{h}_t] = \phi_y(\tilde{\boldsymbol{h}}_t, \boldsymbol{y}_t) \tag{11}$$

For the variational RNF, hidden state variable $\boldsymbol{x}_t$ is modelled as multivariate Gaussian, given by:

$$\boldsymbol{x}_t \sim N(m(\tilde{\boldsymbol{s}}_t), V(\tilde{\boldsymbol{s}}_t)) \tag{12}$$

$$m(\tilde{\boldsymbol{s}}_t) = \boldsymbol{W}_m \tilde{\boldsymbol{s}}_t + \boldsymbol{b}_m \tag{13}$$

$$V(\tilde{\boldsymbol{s}}_t) = \text{diag}(\sigma(\tilde{\boldsymbol{s}}_t) \odot \sigma(\tilde{\boldsymbol{s}}_t)) \tag{14}$$

$$\sigma(\tilde{\boldsymbol{s}}_t) = \text{Softplus}(\boldsymbol{W}_\sigma \tilde{\boldsymbol{s}}_t + \boldsymbol{b}_\sigma), \tag{15}$$

[1] An open-source implementation of the standard RNF can be found at: https://github.com/sjblim/rnf-ijcnn-2020

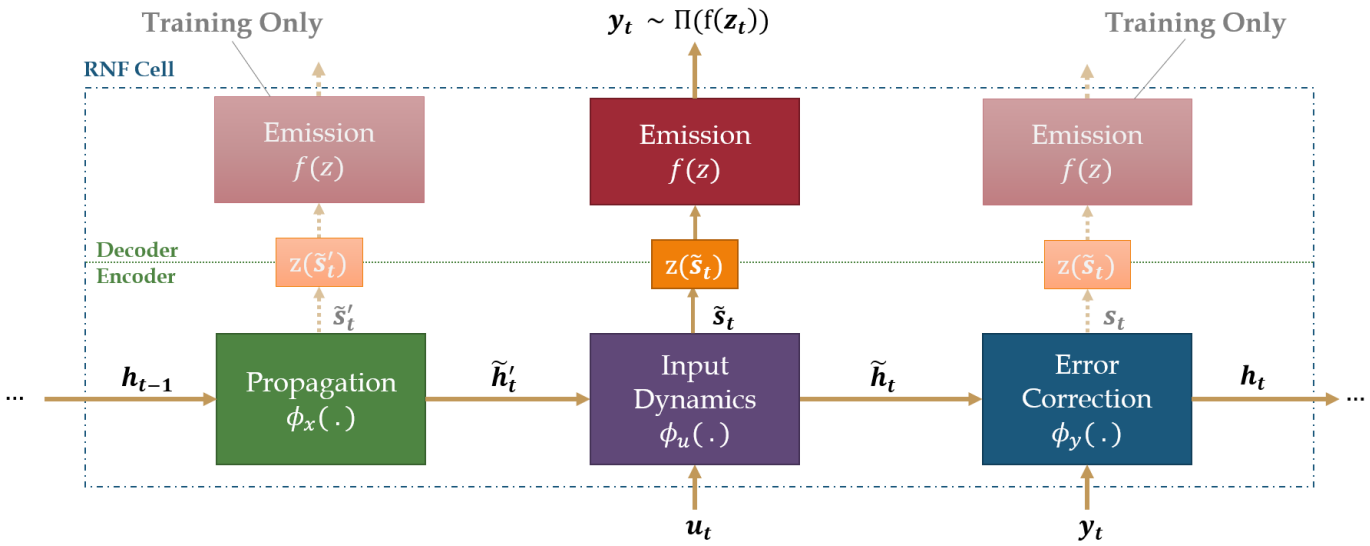

Fig. 1. RNF Network Architecture

where $\boldsymbol{W}_{(\cdot)}, \boldsymbol{b}_{(\cdot)}$ are the weights/biases of each layer, and $\odot$ is an element-wise (Hadamard) product.

For the standard RNF, the encoder state $\tilde{\boldsymbol{s}}_t$ is directly fed into the emission decoder leading to the following forms for $\tilde{\boldsymbol{z}}_t = z(\tilde{\boldsymbol{s}}_t)$:

$$z_{\text{VRNF}}(\tilde{\boldsymbol{s}}_t) = \boldsymbol{x}_t, \qquad z_{\text{RNF}}(\tilde{\boldsymbol{s}}_t) = \tilde{\boldsymbol{s}}_t. \tag{16}$$

While the connection to non-linear state-space models facilitates our interpretation of $z_{\text{RNF}}(\tilde{\boldsymbol{s}}_t)$, we note that the standard RNF no longer relies on an explicit generative model for the latent state $\boldsymbol{x}_t$. This potentially allows the standard RNF to learn more complex update rules for non-Gaussian latent states.

Decoder Given an encoder output $\tilde{\boldsymbol{z}}_t$, we use a multi-layer perceptron to model the emission function $f(\cdot)$:

$$f(\tilde{\boldsymbol{z}}_t) = \boldsymbol{W}_{z_2}\ \text{ELU}(\boldsymbol{W}_{z_1}\ \tilde{\boldsymbol{z}}_t + \boldsymbol{b}_{z_1}) + \boldsymbol{b}_{z_2}. \tag{17}$$

This allows us to handle both continuous or binary observations using the output models below:

$$\boldsymbol{y}_t^{\text{continuous}} \sim N\Big(f_\mu(\tilde{\boldsymbol{z}}_t)\ , \Gamma(\tilde{\boldsymbol{z}}_t)\Big), \tag{18}$$

$$\boldsymbol{y}_t^{\text{binary}} \sim \text{Bernoulli}\Big(\text{Sigmoid}(f(\tilde{\boldsymbol{z}}_t))\Big). \tag{19}$$

where $\Gamma(\tilde{\boldsymbol{z}}_t) = diag\,(g_\sigma(\tilde{\boldsymbol{z}}_t))$ is a time-dependent diagonal covariance matrix, and $g_\sigma(\tilde{\boldsymbol{z}}_t) = \text{Softplus}(f_\sigma(\tilde{\boldsymbol{z}}_t))$.

For $\boldsymbol{y}_t^{\text{continuous}}$, the weights W_{z_1}, b_{z_1} are shared between $f_\mu(\cdot)$ and $f_\sigma(\cdot)$ – i.e. both observation means and covariances are generated from the same encoder hidden layer.

B. Handling Missing Data and Multistep Prediction

From the above, we can see that each encoder learns how specific inputs (i.e. time evolution, exogenous inputs and the target) modify the belief state. As such, in a similar fashion to Bayes filters, we decouple the RNF stages at run-time based on the availability of inputs for prediction – allowing it to handle applications involving missing data or multistep forecasting.

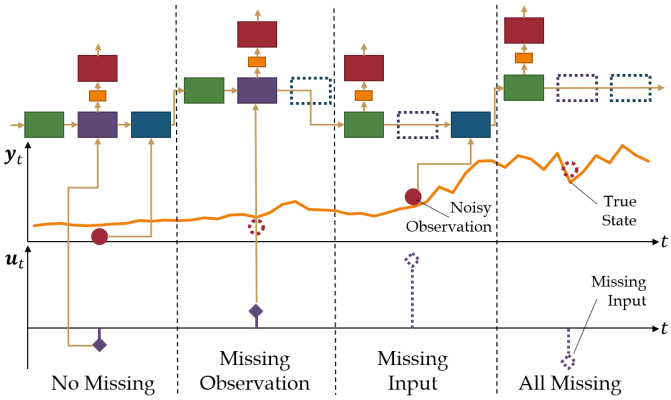

Fig. 2. RNF Configuration with Missing Data

Figure 2 demonstrates how the RNF stages can be combined to accommodate missing data, noting that the colour scheme of the encoders/decoders shown matches that of Figure 1. From the schematic, the propagation encoder – which is responsible for changes to the belief state due to time evolution – is always applied, with the input dynamics and error correction encoders only used when inputs or observations are observed respectively. Where inputs are available, the emission decoder is applied to the input dynamics encoder to generate predictions at each step. Failing that, the decoder is applied to the propagation encoder alone.Multistep forecasts can also be treated as predictions in the absence of inputs or observations, with the encoders used to project the belief state in a similar fashion to missing data.

V. Training Methodology

Considering the joint probability for a trajectory of length T, we train the standard RNF by minimising the negative log-

likelihood of the observations. For continuous observations, this involves Gaussian likelihoods from Equation (18):

$$\mathcal{L}_{\text{RNF}}(\boldsymbol{\omega}, \tilde{\boldsymbol{s}}_{1:T}) = -\sum_{t=1}^{T} \log p(\boldsymbol{y}_t|\tilde{\boldsymbol{s}}_t), \tag{20}$$

$$\log p(\boldsymbol{y}_t|\tilde{\boldsymbol{s}}_t) = -\frac{1}{2}\sum_{j=1}^{J}\Bigg\{\log(2\pi g_\sigma(j,\tilde{\boldsymbol{s}}_t)^2) + \left\|\frac{\boldsymbol{y}_t(j) - f_\mu(j,\tilde{\boldsymbol{s}}_t)}{g_\sigma(j,\tilde{\boldsymbol{s}}_t)}\right\|^2\Bigg\}, \tag{21}$$

where $\boldsymbol{\omega}$ are the weights of the deep neural network, $f_\mu(j,\tilde{\boldsymbol{z}}_t)$ is the j-th element of $f_\mu(\tilde{\boldsymbol{z}}_t)$, and $g_\sigma(j,\tilde{\boldsymbol{z}}_t)$ the j-th element of $g_\sigma(\tilde{\boldsymbol{z}}_t)$.

For the VRNF, we adopt the Stochastic Gradient Variational Bayes (SGVB) estimator of [46] for our VAE evidence lower bound, expressing our loss function as:

$$\mathcal{L}_{\text{VRNF}}(\boldsymbol{\omega}, \tilde{\boldsymbol{s}}_{1:T}) = \sum_{t=1}^{T}\Bigg\{\frac{1}{L}\sum_{i=1}^{L}\log p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)}(\tilde{\boldsymbol{s}}_t))\Bigg\} - KL\big(q(\boldsymbol{x}_{1:T}) \;||\; p(\boldsymbol{x}_{1:T})\big), \tag{22}$$

where L is the number of samples used for calibration, $\boldsymbol{x}_k^{(i)}(\tilde{\boldsymbol{s}}_k)$ is the i-th sample given the latent state $\tilde{\boldsymbol{s}}_k$, and $KL(\cdot)$ is the KL divergence term defined based on the priors in Section V-A.

A. VAE Priors for VRNF

Using the generative model for $\boldsymbol{x}_t$ in Equation (2), we consider the definition of two priors for the VRNF, as described briefly below. A full definition can be found in Appendix[2] A, which also includes derivations for the KL term used in $\mathcal{L}_{\text{VRNF}}(\boldsymbol{\omega}, \tilde{\boldsymbol{s}}_{1:T})$.

Kalman Filter Prior (VRNF-KF) Considering a linear Gaussian state space form for Equations (1) and (2), we can apply the Kalman filtering equations to obtained distributions for $\boldsymbol{x}_t$ at each time step (e.g. $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}, \boldsymbol{u}_{1:t})$). This also lets us analytically define how the means and covariances of the belief state change with different sets of information – aligning the VRNF's encoder stages with the filtering equations.

Neural Network Prior (VRNF-NN) In the spirit of the DKF [13], the analytical equations from the Kalman filter prior above can also be approximated using simple multilayer perceptrons. This would also allow belief state updates to accommodate non-linear states space dynamics, making it fa less restrictive prior model.

B. Encouraging Decoupled Representations

Combined Encoder Training To improve representation learning, the RNF is trained in a "multi-task" fashion – with each intermediate stage trained to encode latent states for output distributions. This is achieved by applying the same emissions decoder to all encoders during training as indicated in Figure 1, with each encoder/decoder aligned with the Bayesian filtering steps described in Section IV-A. Encoders are then trained jointly using the combined loss function below:

$$\mathcal{L}_{\text{combined}}(\boldsymbol{\omega}, \boldsymbol{y}_{1:T}, \boldsymbol{u}_{1:T}) = \underbrace{\mathcal{L}(\boldsymbol{\omega}, \tilde{\boldsymbol{s}}_{1:T})}_{\text{Input Dynamics}} + \overbrace{\underbrace{\alpha_x \mathcal{L}(\boldsymbol{\omega}, \tilde{\boldsymbol{s}}'_{1:T})}_{\text{Propagation}} + \underbrace{\alpha_y \mathcal{L}(\boldsymbol{\omega}, \boldsymbol{s}_{1:T})}_{\text{Error Correction}}}^{\text{Additional Regularisation Terms}}. \tag{23}$$

As such, the additional stages can be interpreted as regularisation terms for the VRNF or RNF loss functions – which we weight by constants α_x and α_y to control the relative importance of the intermediate encoder representations. For our main experiments, we place equal importance on all encoders, i.e. $\alpha_x = \alpha_y = 1$, to facilitate the subsequent separation of stages for multistep prediction – with a full ablation analysis performed to assess the impact of various α settings during training.

Furthermore, the error correction component $\phi_y(\cdot)$ can also be interpreted as a pure auto-encoding step for the latest observation, recovering distributions $p(\boldsymbol{y}_t|\boldsymbol{x}_t)$ based on filtered distributions of $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}, \boldsymbol{u}_{1:t})$. Given that all stages share the same emissions decoder, this obliges the network to learn representations for $\boldsymbol{s}_t$ that are able to reconstruct the current observation when it is available.

Introducing Artificial Missingness Next, to encourage the clean separation of encoder stages for generalisation to other tasks, we break dependencies between the encoders by introducing artificial missingness into the dataset – randomly dropping out inputs and observations with a missingness rate r. As encoders are only applied where data is present (see Figure 2), input dynamics and error correction encoders are hence randomly skipped over during training – encouraging the encoder to perform regardless of which encoder stage preceded it. This also bears a resemblance to input dropout during training, which we apply to competing benchmarks to ensure comparability.

VI. Performance Evaluation

A. Time Series Datasets

We conduct a series of tests on 3 real-world time series datasets to evaluate performance:

1) **Electricity:** The public UCI Individual Household Electric Power Consumption Data [47]
2) **Volatility:** A 30-min realised variance [48] dataset for 30 different stock indices
3) **Quote:** A high-frequency market microstructure dataset containing Barclays Level-1 quote data from Thomson Reuters Tick History (TRTH)

Details on input/output features and preprocessing are fully documented in Appendix C for reference.

B. Conduct of Experiment

Benchmarks: We compare the VRNF-KF, VRNF-NN and standard RNF against a range of autoregressive and RVAE benchmarks – including the DeepAR Model [30], Deep State

[2]URL for full paper with appendix: https://arxiv.org/abs/1901.08096

TABLE I
NORMALISED MSES FOR ONE-STEP-AHEAD PREDICTIONS

	DeepAR	DSSM	VRNN	DKF	VRNF-KF	VRNF-NN	RNF
Electricity	0.908	1.000	2.002	0.867	0.861	0.852	**0.780***
Volatility	3.956	1.000	0.991	0.982	1.914	1.284	**0.976***
Quote	0.998	1.000	3.733	1.001	1.000	1.001	**0.997***

TABLE II
COVERAGE PROBABILITY OF ONE-STEP-AHEAD 90% PREDICTION INTERVAL

	DeepAR	DSSM	VRNN	DKF	VRNF-KF	VRNF-NN	RNF
Electricity	0.966	0.964	0.981	0.965	**0.320**	**0.271**	**0.961***
Volatility	**0.997***	0.999	1.000	1.000	1.000	1.000	1.000
Quote	0.997	0.991	**0.005**	0.998	**0.924 ***	0.992	0.997

TABLE III
NORMALISED MSES FOR MULTISTEP PREDICTIONS WITH BOTH UNKNOWN AND KNOWN INPUTS

Input Type	Dataset	$\tau =$	DeepAR	DSSM	VRNN	DKF	VRNF-KF	VRNF-NN	RNF
Unknown Inputs	**Electricity**	5	3.260	3.308	3.080	2.946	2.607	2.015	**1.996***
		10	4.559	4.771	4.533	4.419	5.467	**3.506***	3.587
		20	6.555	6.827	6.620	6.524	9.817	**5.449***	6.098
	Volatility	5	3.945	1.628	0.994	0.986	4.084	1.020	**0.967***
		10	3.960	1.639	0.994	0.985	4.140	1.017	**0.967***
		20	3.955	1.641	0.993	0.983	4.163	1.014	**0.966***
	Quote	5	1.000	1.000	1.001	1.000	1.002	0.999	**0.998***
		10	1.000	1.001	1.000	1.001	1.009	1.002	**1.000***
		20	1.000	1.001	1.003	1.001	1.488	1.003	**1.000***
Known Inputs	**Electricity**	5	3.260	3.199	3.045	1.073	1.112	0.877	**0.813***
		10	4.559	4.382	4.470	1.008	1.180	0.882	**0.831***
		20	6.555	6.174	6.514	0.989	1.209	0.884	**0.846***
	Volatility	5	3.988	1.615	0.994	0.986	2.645	1.009	**0.981***
		10	3.992	1.620	0.994	0.985	2.652	1.009	**0.981***
		20	3.991	1.627	0.993	0.984	2.652	1.008	**0.980***
	Quote	5	1.000	1.000	0.998	1.000	1.001	1.000	**0.997***
		10	1.000	1.000	0.999	1.000	1.003	1.000	**0.998***
		20	1.000	1.000	1.003	1.000	1.009	1.000	**0.999***

Space Model (DSSM) [26], Variational RNN (VRNN) [12], and Deep Kalman Filter (DKF) [13].

For multistep prediction, we consider two potential use cases for exogenous inputs: (i) when future inputs are unknown beforehand and imputed using their last observed values, and (ii) when inputs are known in advance and used as given. When models require observations of $\boldsymbol{y}_t$ as inputs, we recursively feed outputs from the network as inputs at the next time step. These tweaks allow the benchmarks to be used for multistep prediction without modifying network architectures. For the RNF, we consider the application of the propagation encoder alone for the former case, and a combination of the propagation and input dynamics encoder for the latter – as detailed in Section IV-B.

Metrics: To determine the accuracy of forecasts, we evaluate the mean-squared-error (MSE) for single-step and multistep predictions, normalising each using the MSE of the one-step-ahead forecast for the best autoregressive model (i.e. the DSSM). For multistep forecasts, we measure the average squared error up to the maximum prediction horizon (τ). As observations are 1D continuous variables for all our datasets, we evaluate uncertainty estimates using the prediction interval coverage probability (PICP) of a 90% prediction interval, defined as:

$$\text{PICP} = \frac{1}{T}\sum_{t=1}^{T} c_t, \tag{24}$$

$$c_t = \begin{cases} 1, & \text{if } \psi(0.05,t) < y_t < \psi(0.95,t) \\ 0, & \text{otherwise} \end{cases} \tag{25}$$

where $\psi(0.05,t)$ is the 5^{th} percentile of samples from $N\left(f(\boldsymbol{x}_t), \boldsymbol{\Gamma}\right)$.

Training Details: Please refer to Appendix B for full details of network calibration.

C. Results and Discussion

On the whole, the standard RNF demonstrates the best overall performance – improving MSEs in general for one-step-ahead

TABLE IV
NORMALISED MSES FOR ABLATION STUDIES

		Electricity				Volatility				Quote			
	$\tau =$	1	5	10	20	1	5	10	20	1	5	10	20
Unknown Inputs	**RNF**	-	**1.996***	**3.587***	**6.098***	-	**0.967***	**0.967***	**0.966***	-	**0.998***	**1.000***	**1.000***
	RNF-NS	-	2.801	**13.409**	45.625	-	1.006	1.006	1.005	-	1.137	1.294	1.260
	RNF-IO	-	14.047	14.803	15.414	-	1.377	1.458	1.494	-	1.029	1.042	1.045
Known Inputs	**RNF**	0.780	0.813	**0.831***	**0.846***	**0.976***	**0.981***	**0.981***	**0.980***	**0.997***	**0.997***	**0.998***	**0.999***
	RNF-NS	0.828	0.948	0.997	1.042	0.979	0.983	0.983	0.982	1.001	1.003	1.019	1.026
	RNF-IO	**0.770***	**0.809***	0.873	0.918	1.012	1.016	1.016	1.015	1.020	1.015	1.023	1.030

and multistep prediction. From the one-step-ahead MSEs in Table I, the RNF improves forecasting accuracy by 19.6% on average across all datasets and benchmarks. These results are also echoed for multistep predictions in Table III, with the RNF beating the majority of baselines for all horizons and datasets. The only exception is the slight out-performance of another RNF variant (the VRNF-NN) on the Electricity dataset with unknown inputs – possibly due to the adoption of a suitable prior for this specific dataset – with the standard RNF coming in a close second. The PICP results of Table II also show that performance is achieved without sacrificing the quality of uncertainty estimates, with the RNF outputting similar uncertainty intervals compared to other deep generative and autoregressive models. On the whole, this demonstrates the benefits of the proposed training approach for the RNF, which encourages decoupled representations using regularisation terms and skip training.

To measure the benefits of the skip-training approach and proposed regularisation terms, we also perform a simple ablation study and train the RNF without the proposed components. Table IV shows the normalised MSEs for the ablation studies, with one-step and multistep forecasts combined into the same table. Specifically, we test the RNF with no skip training in RNF-NS, and the RNF with only the input dynamics stage in RNF-IO (i.e. $\alpha_x = \alpha_y = 0$). As inputs are always known for one-step-ahead predictions, normalised MSEs for $\tau = 1$ are omitted for unknown inputs. In general, the inclusion of both skip training and regularisation terms improves forecasting performance, particularly in the case of longer-horizon predictions. We observe this from the MSE improvements for all but short-term ($\tau \in \{1, 5\}$) predictions for known inputs, where the RNF-IO. However, the importance of both skip-training and regularisation can be seen from the large multistep MSEs of both the RNF-NS and RNF-IO on the Electricity dataset with unknown inputs– which results from error propagation when the input dynamics encoder is removed.

As mentioned in Section IV, the challenges of prior selection for VAE-based methods can be seen from the PICPs in Table II – with small PICPs for VRNN models indicative of miscalibrated distributions in the Electricity data, and the poor MSEs and PICPs for the VRNN indicative of posterior collapse on the Quote data. However, this can also be beneficial when applied to appropriate datasets – as seen from the closeness of the VRNN-KF's PICP to the expected 90% on the Quote data. As such, the autoregressive form of standard RNF leads to more reliable performance from both a prediction accuracy and uncertainty perspective – doing away with the need to define a prior for $\boldsymbol{x}_t$.

VII. CONCLUSIONS

In this paper, we introduce a novel recurrent autoencoder architecture, which we call the Recurrent Neural Filter (RNF), to learn decoupled representations for the Bayesian filtering steps – consisting of separate encoders for state propagation, input and error correction dynamics, and a common decoder to model emission. Based on experiments with three real-world time series datasets, the direct benefits of the architecture can be seen from the improvements in one-step-ahead predictive performance, while maintaining comparable uncertainty estimates to benchmarks. Due to its modular structure and close alignment with Bayesian filtering steps, we also show the potential to generalise the RNF to similar predictive tasks – as seen from improvements in multistep prediction using extracted state transition encoders.

REFERENCES

[1] J. V. Candy, *Bayesian Signal Processing: Classical, Modern and Particle Filtering Methods*. New York, NY, USA: Wiley-Interscience, 2009.

[2] A. J. Haug, *Bayesian estimation and tracking: a practical guide*. Hoboken, NJ: John Wiley & Sons, 2012.

[3] T. D. Barfoot, *State Estimation for Robotics*. New York, NY, USA: Cambridge University Press, 2017.

[4] H. Ghosh, B. Gurung, and Prajneshu, "Kalman filter-based modelling and forecasting of stochastic volatility with threshold," *Journal of Applied Statistics*, vol. 42, no. 3, pp. 492–507, 2015.

[5] R. Sukkar, E. Katz, Y. Zhang, D. Raunig, and B. T. Wyman, "Disease progression modeling using hidden Markov models," in *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug 2012, pp. 2845–2848.

[6] S. Sarkka, *Bayesian Filtering and Smoothing*. New York, NY, USA: Cambridge University Press, 2013.

[7] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[8] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[9] S. J. Julier and J. K. Uhlmann, "New extension of the Kalman filter to nonlinear systems," vol. 3068, 1997.

[10] A. Harvey, *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, 1991.

[11] A. C. Harvey and R. G. Pierse, "Estimating missing observations in economic time series," *Journal of the American Statistical Association*, vol. 79, no. 385, pp. 125–131, 1984.

[12] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," in *Advances in Neural Information Processing Systems 28 (NIPS 2016)*, 2015.

[13] R. G. Krishnan, U. Shalit, and D. Sontag, "Deep Kalman Filters," *ArXiv e-prints*, 2015. [Online]. Available: http://arxiv.org/abs/1511.05121

[14] R. G. Krishnan, U. Shalit, and D. Sontag, "Structured inference networks for nonlinear state space models," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 2017.

[15] M. Karl, M. Soelch, J. Bayer, and P. van der Smagt, "Deep variational Bayes filters: unsupervised learning of state space models from raw data," in *International Conference on Learning Representations (ICLR 2017)*, 2017.

[16] S. Narayanaswamy, T. B. Paige, J.-W. van de Meent, A. Desmaison, N. Goodman, P. Kohli, F. Wood, and P. Torr, "Learning disentangled representations with semi-supervised deep generative models," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017, pp. 5925–5935.

[17] H. Kim and A. Mnih, "Disentangling by factorising," in *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

[18] G. Parascandolo, N. Kilbertus, M. Rojas-Carulla, and B. Schölkopf, "Learning independent causal mechanisms," in *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

[19] V. Thomas, E. Bengio, W. Fedus, J. Pondard, P. Beaudoin, H. Larochelle, J. Pineau, D. Precup, and Y. Bengio, "Disentangling the independently controllable factors of variation by interacting with the world," in *NIPS 2017 Workshop on Learning Disentangled Representations*, 2018.

[20] F. Locatello, S. Bauer, M. Lucic, S. Gelly, B. Schölkopf, and O. Bachem, "Challenging common assumptions in the unsupervised learning of disentangled representations," *CoRR*, vol. abs/1811.12359, 2018. [Online]. Available: http://arxiv.org/abs/1811.12359

[21] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-VAE: Learning basic visual concepts with a constrained variational framework," in *International Conference on Learning Representations (ICLR 2017)*, 2017.

[22] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behavioral and Brain Sciences*, vol. 40, p. e253, 2017.

[23] M. Johnson, D. K. Duvenaud, A. Wiltschko, R. P. Adams, and S. R. Datta, "Composing graphical models with neural networks for structured representations and fast inference," in *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, 2016.

[24] L. Ralaivola and F. d'Alche Buc, "Time series filtering, smoothing and learning using the kernel Kalman filter," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks*, vol. 3, July 2005, pp. 1449–1454.

[25] M. Fraccaro, S. Kamronn, U. Paquet, and O. Winther, "A disentangled recognition and nonlinear dynamics model for unsupervised learning," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[26] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski, "Deep state space models for time series forecasting," in *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, 2018.

[27] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[29] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," *CoRR*, vol. abs/1901.02860, 2019. [Online]. Available: http://arxiv.org/abs/1901.02860

[30] V. Flunkert, D. Salinas, and J. Gasthaus, "Deepar: Probabilistic forecasting with autoregressive recurrent networks," *CoRR*, vol. abs/1704.04110, 2017. [Online]. Available: http://arxiv.org/abs/1704.04110

[31] K. Choromanski, C. Downey, and B. Boots, "Initialization matters: Orthogonal predictive state recurrent neural networks," in *International Conference on Learning Representations (ICLR 2018)*, 2018.

[32] C. Downey, A. Hefny, B. Boots, G. J. Gordon, and B. Li, "Predictive state recurrent neural networks," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[33] A. Venkatraman, N. Rhinehart, W. Sun, L. Pinto, M. Hebert, B. Boots, K. Kitani, and J. Bagnell, "Predictive-state decoders: Encoding the future into recurrent networks," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[34] R. Turner, M. Deisenroth, and C. Rasmussen, "State-space inference and learning with Gaussian processes," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, 2010, pp. 868–875.

[35] H. Nickisch, A. Solin, and A. Grigorevskiy, "State space Gaussian processes with non-Gaussian likelihood," in *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018, pp. 3789–3798.

[36] A. Doerr, C. Daniel, M. Schiegg, N.-T. Duy, S. Schaal, M. Toussaint, and T. Sebastian, "Probabilistic recurrent state-space models," in *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

[37] S. Sarkka, A. Solin, and J. Hartikainen, "Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing: A look at Gaussian process regression through Kalman filtering," *IEEE Signal Processing Magazine*, vol. 30, no. 4, pp. 51–61, July 2013.

[38] B. Pérez Orozco, G. Abbati, and S. Roberts, "MOrdReD: Memory-based Ordinal Regression Deep Neural Networks for Time Series Forecasting," *CoRR*, vol. arXiv:1803.09704, 2018. [Online]. Available: http://arxiv.org/abs/1803.09704

[39] R. Wen and K. T. B. M. Narayanaswamy, "A multi-horizon quantile recurrent forecaster," in *NIPS 2017 Time Series Workshop*, 2017.

[40] H. Takahashi, T. Iwata, Y. Yamanaka, M. Yamada, and S. Yagi, "Variational autoencoder with implicit optimal priors," *CoRR*, vol. abs/1809.05284, 2018. [Online]. Available: https://arxiv.org/abs/1809.05284

[41] Tomczak and Welling, "VAE with a VampPrior," in *Proceedings of the 21st Internation Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

[42] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Józefowicz, and S. Bengio, "Generating sentences from a continuous space," *CoRR*, vol. abs/1511.06349, 2015. [Online]. Available: http://arxiv.org/abs/1511.06349

[43] A. van den Oord, O. Vinyals, and k. kavukcuoglu, "Neural discrete representation learning," in *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017, pp. 6306–6315.

[44] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[45] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," in *International Conference on Learning Representations (ICLR 2016)*, 2016.

[46] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *International Conference on Learning Representations (ICLR 2014)*, 2014.

[47] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository – individual household electric power consumption data set," 2017. [Online]. Available: https://archive.ics.uci.edu/ml/datasets

[48] T. Andersen, T. Bollerslev, F. Diebold, and P. Labys, "Modeling and forecasting realized volatility," *Econometrica*, vol. 71, no. 2, pp. 579–625, 2003.

[49] A. Yadav, P. Awasthi, N. Naik, and M. R. Ananthasayanam, "A constant gain kalman filter approach to track maneuvering targets," in *2013 IEEE International Conference on Control Applications (CCA)*, 2013.

[50] T. G. Andersen and T. Bollerslev, "Intraday periodicity and volatility persistence in financial markets," *Journal of Empirical Finance*, vol. 4, no. 2, pp. 115 – 158, 1997.

[51] A. Todd, R. Hayes, P. Beling, and W. Scherer, "Micro-price trading in an order-driven market," in *2014 IEEE Conference on Computational Intelligence for Financial Engineering Economics (CIFEr)*, March 2014, pp. 294–297.

[52] lvaro Cartea, R. Donnelly, and S. Jaimungal, "Enhancing trading strategies with order book signals," *Applied Mathematical Finance*, vol. 25, no. 1, pp. 1–35, 2018.

APPENDIX

A. VRNF Priors and Derivation of KL Term

Defining a prior distribution for the VRNF starts with the specification of a model for the distribution of hidden state $\boldsymbol{x}_t$, conditioned on the amount of available information at each encoder to achieve alignment with the VRNF stages. Per the generative model of Equation (2), we model $\boldsymbol{x}_t$ as a multivariate normal distribution with a mean and covariance that varies with time and the information present at each encoder, based on the notation below:

Propagation:

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t-1}) \sim N(\tilde{\boldsymbol{\beta}}_t^{'}, \tilde{\boldsymbol{\nu}}_t^{'}), \tag{26}$$

Input Dynamics:

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t}) \sim N(\tilde{\boldsymbol{\beta}}_t, \tilde{\boldsymbol{\nu}}_t), \tag{27}$$

Error Correction:

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}, \boldsymbol{u}_{1:t}) \sim N(\boldsymbol{\beta}_t, \boldsymbol{\nu}_t). \tag{28}$$

For the various priors defined in this section, we adopt the use of diagonal covariance matrices for the inputs, defined as:

$$\boldsymbol{\nu}_t = diag(\boldsymbol{\gamma}_t \odot \boldsymbol{\gamma}_t), \tag{29}$$

where $\boldsymbol{\gamma}_t \in \mathbb{R}^J$ is a vector of standard deviation parameters.

This approximation helps to reduce the computational complexity associated with the matrix multiplications using full covariance matrices, and the $O(J^2T)$ memory requirements from storing full covariances matrices for an RNN unrolled across T timesteps.

KL Divergence Term

Considering the application of the input dynamics encoder alone (i.e. $\alpha_x = \alpha_y = 0$), the KL divergence between independent conditional multivariate Gaussians at each time step can be hence expressed analytically as:

$$KL_{\text{Input}}\big(q(\boldsymbol{x}_{1:T}) \;||\; p(\boldsymbol{x}_{1:T})\big) \tag{30}$$

$$= \mathbb{E}_{q(\boldsymbol{x}_{1:T})}\Bigg[\log \frac{p(\boldsymbol{x}_1)}{q(\boldsymbol{x}_1|\tilde{\boldsymbol{s}}_1)} + \sum_{t=2}^{T} \log \frac{p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{u}_{1:t}, \boldsymbol{y}_{1:t-1})}{q(\boldsymbol{x}_t|\tilde{\boldsymbol{s}}_t)}\Bigg] \tag{31}$$

$$= \sum_{t=1}^{T}\sum_{j=1}^{J}\Bigg\{\log \frac{\tilde{\gamma}_t(j)}{\sigma(j, \tilde{\boldsymbol{s}}_t)} + \frac{\sigma(j, \tilde{\boldsymbol{s}}_t)^2 + (m(j, \tilde{\boldsymbol{s}}_t) - \tilde{\beta}_t(j))^2}{2\tilde{\gamma}_t(j)^2} - \frac{1}{2}\Bigg\}, \tag{32}$$

where $m(j, \tilde{\boldsymbol{s}}_t), \sigma(j, \tilde{\boldsymbol{s}}_t)$ are j-th elements of $m(\tilde{\boldsymbol{s}}_t), \sigma(\tilde{\boldsymbol{s}}_t)$ as defined in Equations (14) and (15) respectively.

The KL divergence terms are defined similarly for the propagation and error correction encoders, using the means and standard deviations defined above.

Kalman Prior (VRNF-KF)

The use Kalman filter relies on the definition of a linear Gaussian state space model, which we specify below:

$$\boldsymbol{y}_t = \boldsymbol{H}\boldsymbol{x}_t + \boldsymbol{e}_t, \tag{33}$$

$$\boldsymbol{x}_t = \boldsymbol{A}\boldsymbol{x}_{t-1} + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{\epsilon}_t, \tag{34}$$

where $\boldsymbol{H}, \boldsymbol{A}, \boldsymbol{B}$ are constant matrices, and $\boldsymbol{e}_t \sim N(0, \boldsymbol{R}), \boldsymbol{\epsilon}_t \sim N(0, \boldsymbol{Q})$ are noise terms with constant noise covariances $\boldsymbol{R}$ and $\boldsymbol{Q}$.

a) Propagation: Assuming that inputs $\boldsymbol{u}_t$ is unknown at time t, predictive distributions can still be computed for the hidden state if we have a model for $\boldsymbol{u}_t$. In the simplest case, this can be a standard normal distribution, i.e. $\boldsymbol{u}_t \sim N(\boldsymbol{c}, \boldsymbol{D})$ – where $\boldsymbol{c}$ is a constant mean vector and $\boldsymbol{D}$ a constant covariance matrix. Under this model, predictive distributions can be computed as below:

$$\begin{aligned}\tilde{\boldsymbol{\beta}}_t^{'} &= \boldsymbol{A}\boldsymbol{\beta}_{t-1} + \boldsymbol{B}\boldsymbol{c} \\ &= \boldsymbol{A}\boldsymbol{\beta}_{t-1} + \boldsymbol{c}^{'},\end{aligned} \tag{35}$$

$$\begin{aligned}\tilde{\boldsymbol{\nu}}_t^{'} &= \boldsymbol{A}\boldsymbol{\nu}_{t-1}\boldsymbol{A}^T + \boldsymbol{B}\boldsymbol{D}\boldsymbol{B}^T + \boldsymbol{Q} \\ &= \boldsymbol{A}\boldsymbol{\nu}_{t-1}\boldsymbol{A}^T + \boldsymbol{Q}^{'},\end{aligned} \tag{36}$$

with $\boldsymbol{c}^{'}, \boldsymbol{Q}^{'}$ collapsing constant terms together into a single parameters.

b) Input Dynamics: When inputs are known, the forecasting equations take on a similar form:

$$\tilde{\boldsymbol{\beta}}_t = \boldsymbol{A}\boldsymbol{\beta}_{t-1} + \boldsymbol{B}\boldsymbol{u}_t \tag{37}$$

$$\tilde{\boldsymbol{\nu}}_t = \boldsymbol{A}\boldsymbol{\nu}_{t-1}\boldsymbol{A}^T + \boldsymbol{Q} \tag{38}$$

Comparing this with the forecasting equations of the propagation step, we can express also the above as functions $\tilde{\boldsymbol{\beta}}_t^{'}$ and $\tilde{\boldsymbol{\nu}}_t^{'}$, i.e.:

$$\tilde{\boldsymbol{\beta}}_t = \tilde{\boldsymbol{\beta}}_t^{'} + \boldsymbol{B}\boldsymbol{u}_t - \boldsymbol{c}^{'}, \tag{39}$$

$$\tilde{\boldsymbol{\nu}}_t = \tilde{\boldsymbol{\nu}}_t^{'} - \boldsymbol{Q}^{'} + \boldsymbol{Q}. \tag{40}$$

c) Error Correction: Upon receipt of a new observation, the Kalman filter computes a Kalman Gain $\boldsymbol{K}_t$, using it to correct the belief state as below:

$$\boldsymbol{\beta}_t = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H})\,\tilde{\boldsymbol{\beta}}_t - \boldsymbol{K}_t\boldsymbol{H}\boldsymbol{y}_t, \tag{41}$$

$$\boldsymbol{\nu}_t = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H})\,\tilde{\boldsymbol{\nu}}_t, \tag{42}$$

where $\boldsymbol{I}$ is an identity matrix and $\boldsymbol{K}_t = \tilde{\boldsymbol{\nu}}_t\boldsymbol{H}^T\left(\boldsymbol{H}\tilde{\boldsymbol{\nu}}_t\boldsymbol{H}^T + \boldsymbol{R}\right)^{-1}$.

Approximations for Efficiency

To avoid the complex memory and space requirements associated with full matrix computations, we make the following approximations in our Kalman Filter equations.

d) Constant Kalman Gain: Firstly, as noted in [49], Kalman gain values in stable filters usually tend towards a steady state value after a initial transient period. We hence fix the Kalman gain at a constant value, and collapse constant coefficients in the error correction equations to give:

$$\boldsymbol{\beta}_t = \boldsymbol{K}^{'} \tilde{\boldsymbol{\beta}}_t - \boldsymbol{H}^{'} \boldsymbol{y}_t, \tag{43}$$

$$\boldsymbol{\nu}_t = \boldsymbol{K}^{'} \tilde{\boldsymbol{\nu}}_t, \tag{44}$$

Where $\boldsymbol{K}^{'} = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})$ and $\boldsymbol{H}^{'} = \boldsymbol{K}\boldsymbol{H}$.

e) Independent Hidden State Dimensions: Next, we assume that hidden state dimensions are independent of one another, which effectively diagonalising state related coefficients $\boldsymbol{A} = diag(\boldsymbol{a})$ and $\boldsymbol{Q} = diag(\boldsymbol{q})$.

f) Diagonalising $\boldsymbol{Q}^{'}, \boldsymbol{K}^{'}$: Finally, to allow us to diagonal covariance matrices throughout our equations, we also diagonalise $\boldsymbol{Q}^{'} = diag(\boldsymbol{q}^{'})$ and $\boldsymbol{K}^{'} = diag(\boldsymbol{k}^{'})$.

Prior Definition

Using the above definitions and approximations, the Kalman filter prior can hence be expressed in vector form using the equations below:

Propagation:

$$\tilde{\boldsymbol{\beta}}_t^{'} = \boldsymbol{a} \odot m(\boldsymbol{s}_{t-1}) + \boldsymbol{c}^{'}, \tag{45}$$

$$\tilde{\boldsymbol{\nu}}_t^{'} = \boldsymbol{a} \odot V(\boldsymbol{s}_{t-1}) \odot \boldsymbol{a} + \boldsymbol{q}^{'}. \tag{46}$$

Input Dynamics:

$$\tilde{\boldsymbol{\beta}}t = \boldsymbol{a} \odot m(\boldsymbol{s}_{t-1}) + \boldsymbol{B}\boldsymbol{u}_t, \tag{47}$$

$$\tilde{\boldsymbol{\nu}}_t = \boldsymbol{a} \odot V(\boldsymbol{s}_{t-1}) \odot \boldsymbol{a} + \boldsymbol{q}. \tag{48}$$

Error Correction:

$$\boldsymbol{\beta}_t = \boldsymbol{k}^{'} \odot m(\tilde{\boldsymbol{s}}_t) - \boldsymbol{H}^{'} \boldsymbol{y}_t, \tag{49}$$

$$\boldsymbol{\nu}_t = \boldsymbol{k}^{'} \odot V(\tilde{\boldsymbol{s}}_t). \tag{50}$$

All constant standard deviation are implemented as coefficients wrapped in a softmax layer (e.g. $\boldsymbol{a} = \text{softplus}(\phi)$)) to prevent the optimiser from converging on invalid negative numbers.

In addition, we note that the form input dynamics prior is not conditioned on the propagation encoder outputs, although we could in theory express it terms of its statistics (i.e. Equations (39) and (40)). This to avoid converging on negative values for variances, which can be obtained from the subtraction of positive constant $\boldsymbol{Q}^{'}$, although we revisit this in this form in the next section.

Neural Network Prior (VRNF-NN)

Despite the convenient tractable from of the Kalman filtering equations, this relies on the use of linear state space assumptions which might not be suitable for complex datasets. As such, we also consider the use of multilayer perceptrons ($\text{MLP}(\cdot)$) to approximate the equations described in the previous section, conditioning it on the previous active encoder stage, i.e.:

Propagation:

$$\tilde{\boldsymbol{\beta}}_t^{'} = \text{MLP}_{\tilde{\boldsymbol{\beta}}'}(m(\boldsymbol{s}_{t-1})) \tag{51}$$

$$\tilde{\boldsymbol{\nu}}_t^{'} = \text{MLP}_{\tilde{\boldsymbol{\nu}}'}(V(\boldsymbol{s}_{t-1})) \tag{52}$$

Input Dynamics:

$$\tilde{\boldsymbol{\beta}}_t = \text{MLP}_{\tilde{\boldsymbol{\beta}}}(m(\tilde{\boldsymbol{s}}_t), \boldsymbol{u}_t) \tag{53}$$

$$\tilde{\boldsymbol{\nu}}_t = \text{MLP}_{\tilde{\boldsymbol{\nu}}}(V(\tilde{\boldsymbol{s}}_t), \boldsymbol{u}_t) \tag{54}$$

Error Correction:

$$\boldsymbol{\beta}_t = \text{MLP}_{\boldsymbol{\beta}}(m(\boldsymbol{s}_t), \boldsymbol{y}_t) \tag{55}$$

$$\boldsymbol{\nu}_t = \text{MLP}_{\boldsymbol{\nu}}(V(\boldsymbol{s}_t), \boldsymbol{y}_t) \tag{56}$$

Similar to the state transition functions used in [13], this can be interpreted as using MLPs to approximate the true Kalman filter functions for linear datasets, while also permitting the learning of more sophisticated non-linear models. All MLPs defined here use an ELU activation function for their hidden layer, fixing the hidden state size to be J. Furthermore, we use linear output layers for $\boldsymbol{\beta}$ MLPs, while passing that of $\boldsymbol{\nu}$ MLPs through a softplus activation function to maintain positivity.

B. Training Procedure for RNF

a) Training Details: During network calibration, trajectories were partitioned into segments of 50 time steps each – which were randomly combined to form minibatches during training. Also, networks were trained for up to a maximum of 100 epochs or convergence. For the electricity and volatility datasets, 50 iterations of random search were performed, using the grid found in Table V. 20 iterations of random search were used for the quote dataset, as the significantly larger dataset led to longer training times for a given set of hyperparameters.

TABLE V
RANDOM SEARCH GRID FOR HYPERPARAMETER OPTIMISATION

	Hyperparameter Ranges
Dropout Rate	0.0, 0.1, 0.2, 0.3, 0.4, 0.5
State Size	5, 10, 25, 50, 100, 150
Minibatch Size	256, 512, 1024
Learning Rate	0.0001, 0.001, 0.01, 0.1, 1.0
Max Gradient Norm	0.0001, 0.001, 0.01, 0.1, 1.0, 10.0
Missing Rate	0.25, 0.5, 0.75

b) State Sizes: To ensure that consistency across all models used, we constrain both the memory state of the RNN and the latent variable modelled to have the same dimensionality – i.e. $J = \dim(\boldsymbol{s}_t) = \dim(\boldsymbol{x}_t)$ for the RNF. The exception is the DSSM, as the full covariance matrix of the Kalman filter would result in a prohibitive J^2 memory requirement if left unchecked. As such, we use the constraint where both the RNN and the Kalman filter to have the same memory capacity for the DSSM – i.e. $J = \dim(\boldsymbol{s}_t) = \dim(\boldsymbol{x}_t) + \dim(\boldsymbol{x}_t)^2$.

c) Dropout Application: Across all benchmarks, dropout was applied only onto the memory state of the RNNs ($\boldsymbol{h}_t$) in the standard fashion and not to latent states $\boldsymbol{x}_t$. For the LSTM, DeepAR Model and RNF, this corresponds to applying dropout masks to the outputs and internal states of the network. For the VRNN, DKF and DSSM, we apply dropout only to the inputs of the network – in line with [14] to maintain comparability to the encoder skipping in the VRNFs.

d) Artificial Missingness: Encoder skipping is restricted to only the VRNFs and standard RNF, controlled by the missing rates defined above.

e) Sample Generation: At prediction time, latent states for the VRNN, DKF and VRNFs are sampled as per the standard VAE case – using $L = 1$ during training, $L = 30$ for our validation error and $L = 100$ for at test time. Predictions from the DeepAR Model, DSSM and standard RNF, however, were obtained directly from the mean estimates, with that of the DSSM computed analytically using the Kalman filtering equations. While this differs slightly from the original paper [26], it also leads to improvements in the performance DSSM by avoiding sampling errors.

C. Description of Datasets

For the experiments in Section VI, we focus on the use of 3 real-world time series datasets, each containing over a million time steps per dataset. These use-cases help us evaluate performance for scenarios in which real-time predictions with RNNs are most beneficial – i.e. when the underlying dynamics is highly non-linear and trajectories are long.

Summary

Electricity: The UCI Individual Household Electric Power Consumption Dataset [47] is a time series of 7 different power consumption metrics measured at 1-min intervals for a single household between December 2006 and November 2010 – coming to a total of 2,075,259 time steps over 4 years. In our experiments, we treated active power as the main observation of interest, taking the remainder to be exogenous inputs into the RNNs.

Intraday Volatility: We compute 30-min realised variances [48] for a universe of 30 different stock indices – derived using 1-min index returns subsampled from Thomson Reuters Tick History Level 1 (TRTH L1) quote data. On the whole, the entire dataset contains 1,706,709 measurements across all indices, with each trajectory spanning 17 years on average. Given the strong evidence for the intraday periodicity of returns volatility [50], we also include the time-of-day as an additional exogenous input.

High-Frequency Stock Quotes: This dataset consists of extracted features from TRTH L1 stock quote data for Barclays (BARC.L) – specifically forecasting microprice returns [51] using volume imbalance as an input predictor – comprising a total of 29,321,946 time steps between 03 January 2017 to 29 December 2017. From [52], volume imbalance in the limit order book is a good predictor of the direction (sign) of the next liquidity taking order, and the price changes immediately after the arrival of a liquidity-taking order.

Electricity

a) Data Processing: The full trajectory was segmented into 3 portions, with the earliest 60% of measurements for training, the next 20% as a validation set, and the final 20% as an independent test set – with results reported in Section VI. All data sets were normalised to have zero mean and unit standard deviations, with normalising constants computed using the training set alone.

b) Summary Statistics: A list of summary statistics can be seen in Table VI.

TABLE VI
SUMMARY STATISTICS FOR ELECTRICITY DATASET

	Mean	S.D.	Min	Max
Active Power*	1.11	1.12	0.08	11.12
Reactive Power	0.12	0.11	0.00	1.39
Intensity	4.73	4.70	0.20	48.40
Voltage	240.32	3.33	223.49	252.72
Sub Metering 1	1.17	6.31	0.00	82.00
Sub Metering 2	1.42	6.26	0.00	78.00
Sub Metering 3	6.04	8.27	0.00	31.00

Intraday Volatility

c) Data Processing: From the 1-min index returns, realised variances were computed as:

$$r_k = \ln p_k - \ln p_{k-1}$$
$$y(t, 30) = \sum_{k=t-30}^{t} r_k^2, \tag{57}$$

where r_k is the 1-min index return at time k, $\ln p_k$ is the log price at k, and $y(t, 30)$ is the 30-min realised variance at time t.

Before computation, the data was cleaned by only considering prices during exchange hours to avoid spurious jumps. In addition, realised variances greater than 10 times the 200-step rolling standard deviation were removed and replaced by its previous value – so as to reduce the impact of outliers.

For the experiments in Section VI, data across all stock indices were grouped together for training and testing – using data prior to 2014 for training, data between 2014-2016 for validation and data from 2016 to 4 July 2018 for independent testing. Min-max normalisation was applied to the datasets, with time normalised by the maximum trading window of each exchange and realised variances by the max and min values of the training dataset.

d) Stock Index Identifiers (RICs):: AEX, AORD, BFX, BSESN, BVLG, BVSP, DJI, FCHI, FTMIB, FTSE, GDAXI, GSPTSE, HSI, IBEX, IXIC, KS11, KSE, MXX, N225, NSEI, OMXC20, OMXHPI, OMXSPI, OSEAX, RUT, SMSI, SPX, SSEC, SSMI, STOXX50E

e) Summary Statistics: A table of summary statistics can be found in Table VII and give an indication of the general ranges of trajectories.

TABLE VII
SUMMARY STATISTICS FOR VOLATILITY DATASET

	Mean	S. D.	Min	Max
Realised Variance*	0.0007	0.0017	0.0000	0.1013
Normalised Time	0.43	0.27	0.00	0.97

High-Frequency Stock Quotes

f) Input/Output Definitions: Microprice returns y_t are defined as:

$$p_t = \frac{V_a(t)p_b(t) + V_b(t)p_a(t)}{V_a(t) + V_b(t)}$$
$$y_t = \frac{p_t - p_{t-1}}{p_{t-1}}$$

Where $V_b(t)$ and $V_a(t)$ are the bid and ask volumes at time t respectively, $p_b(t)$ and $p_a(t)$ are the bid/ask prices, and p_t the microprice.

Volume imbalance I_t is then defined as:

$$I_t = \frac{V_b(t) - V_a(t)}{V_b(t) + V_a(t)}$$

g) Data Processing: From the raw Level 1 (best bid and ask prices and volumes) data from TRTH, we isolate measurements between 08.30 to 16.00 UK time, avoiding the effects of opening and closing auctions in our forecasts. Furthermore, microprice returns were also normalised using an exponentially weighting moving standard deviation with a half-life of 10,000 steps. We note that volume imbalance by definition is restricted to be $I_t \in [-1, 1]$, and hence does not require additional normalisation. Finally, the data was partitioned with training data from January to June, validation data from June to September, and the remainder for independent testing.

h) Summary Statistics: Basic statistics can be found in Table VIII, and give an indication of the range of different variables.

TABLE VIII
SUMMARY STATISTICS FOR QUOTE DATASET

	Mean	S. D.	Min	Max
Normalised Returns*	0.00	0.80	-117.72	117.13
Volume Imbalance	0.02	0.48	-1.00	1.00