Modeling irregular time series with continuous recurrent units

Schirmer, M.; Eltayeb, M.; Lessmann, S.; Rudolph, M.

Publication date
2022

Document Version
Final published version

Published in
Proceedings of Machine Learning Research

License
Other

Link to publication

Citation for published version (APA):
Schirmer, M., Eltayeb, M., Lessmann, S., & Rudolph, M. (2022). Modeling irregular time series with continuous recurrent units. *Proceedings of Machine Learning Research*, *162*, 19388-19405. https://proceedings.mlr.press/v162/schirmer22a.html

Modeling Irregular Time Series with Continuous Recurrent Units

Mona Schirmer [1] **Mazin Eltayeb** [2] **Stefan Lessmann** [3] **Maja Rudolph** [4]

Abstract

Recurrent neural networks (RNNs) are a popular choice for modeling sequential data. Modern RNN architectures assume constant time-intervals between observations. However, in many datasets (e.g. medical records) observation times are irregular and can carry important information. To address this challenge, we propose continuous recurrent units (CRUs) – a neural architecture that can naturally handle irregular intervals between observations. The CRU assumes a hidden state, which evolves according to a linear stochastic differential equation and is integrated into an encoder-decoder framework. The recursive computations of the CRU can be derived using the continuous-discrete Kalman filter and are in closed form. The resulting recurrent architecture has temporal continuity between hidden states and a gating mechanism that can optimally integrate noisy observations. We derive an efficient parameterization scheme for the CRU that leads to a fast implementation f-CRU. We empirically study the CRU on a number of challenging datasets and find that it can interpolate irregular time series better than methods based on neural ordinary differential equations.

1. Introduction

Recurrent architectures, such as the long short-term memory network (LSTM) (Hochreiter & Schmidhuber, 1997) or gated recurrent unit (GRU) (Chung et al., 2014) have become a principal machine learning tool for modeling time series. Their modeling power comes from a hidden state, which is recursively updated to integrate new observations, and a gating mechanism to balance the importance of new information with history already encoded in the latent state.

Although continuous formulations were frequently considered in early work on recurrent neural networks (RNNs) (Pineda, 1987; Pearlmutter, 1989; 1995), modern RNNs typically assume regular sampling rates (Hochreiter & Schmidhuber, 1997; Chung et al., 2014). Many real world data sets, such as electronic health records or climate data, are irregularly sampled. Measurements of a patient's health status, for example, are only available when the patient sees a doctor. Hence, the time between observations also carries information about the underlying time series. A lab test not administered for many months could imply that the patient was doing well in the meantime, while frequent visits might indicate that the patient's health is deteriorating. Discrete RNNs face difficulties modeling such data as they do not reflect the continuity of the underlying temporal processes.

Recently, the work on neural ordinary differential equations (neural ODEs) (Chen et al., 2018) has established an elegant and practical way of modeling irregularly sampled time series. Recurrent architectures based on neural ODEs determine the hidden state between observations by an ordinary differential equation (ODE) and update its hidden state at observation times using standard RNN gating mechanisms (Rubanova et al. 2019; Brouwer et al. 2019; Lechner & Hasani 2020). These methods typically rely on some form of numerical ODE-solver, a network component that can prolong training time significantly (Rubanova et al. 2019; Shukla & Marlin 2020).

We propose continuous recurrent unit (CRU), a probabilistic recurrent architecture for modelling irregularly sampled time series. An encoder maps observations into a latent space, which is governed by a linear stochastic differential equation (SDE). The analytic solution for propagating the latent state between observations and the update equations for integrating new observations are given by the continuous-discrete formulation of the Kalman filter. Employing the linear SDE state space model and the Kalman filter has three advantages. First, a probabilistic state space provides an explicit notion of uncertainty for an uncertainty-driven gating mechanism and for confidence evaluation of predictions. Second, as the Kalman filter is the optimal solution for the linear filtering problem (Kalman, 1960), the gating

[1]Humboldt-Universität zu Berlin, Germany. Work done during an internship at Bosch Center for AI [2]Bosch Center for AI, Germany [3]Humboldt-Universität zu Berlin, Germany [4]Bosch Center for AI, USA. Correspondence to: Mona Schirmer <mona.schirmer@ensae.fr>, Maja Rudolph <maja.rudolph@us.bosch.com>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

mechanism is optimal in a locally linear state space. Third, the latent state at any point in time can be resolved analytically, therefore bypassing the need for numerical integration techniques or variational approximations. In summary, our contributions are as follows:

- In Sec. 3, we develop the CRU, a model that combines the power of neural networks for feature extraction with the advantages of probabilistic state-space models, specifically the continuous-discrete Kalman filter. As a result, the CRU is a powerful neural architecture that can naturally model data with irregular observation times. A PyTorch implementation is available on github.[1]
- In Sec. 3.4, we derive a novel parameterization of the latent state transition matrices via their eigendecomposition leading to a faster implementation we call fast CRU (f-CRU).
- In Sec. 4, we study the CRU on electronic health records, climate data and images. We find that (i) our method can better interpolate irregular time series than neural ODE-based methods, (ii) the CRU can handle uncertainty arising from both noisy and partially observed inputs, (iii) CRU outperforms both discrete RNN counterparts and neural ODE-based models on image data.

2. Related Work

Stochastic RNNs RNNs, such as LSTMs or GRUs, are powerful sequence models (Hochreiter & Schmidhuber, 1997; Chung et al., 2014), but due to the lack of stochasticity in their internal transitions, they may fail to capture the variability inherent in certain data (Chung et al. 2015). While there are various stochastic RNNs (e.g. Bayer & Osendorfer, 2014; Fraccaro et al., 2016; Goyal et al., 2017; Schmidt & Hofmann, 2018), our work is most closely related to deep probabilistic approaches based on Kalman filters (Kalman, 1960). Variations on deep Kalman filters (Krishnan et al. 2015; Karl et al. 2017; Fraccaro et al. 2017) typically require approximate inference, but Becker et al. (2019) employ a locally linear model in a high-dimensional factorized latent state for which the Kalman updates can be obtained in closed form. By extending this approach with a continuous latent state, the CRU can model sequences with irregular observation times.

RNNs for Irregular Time Series Applying discrete RNNs to irregularly sampled time series requires the discretization of the time line into uniform bins. This often reduces the number of observations, may result in a loss of information, and evokes the need for imputation and aggregation strategies. To avoid such preprocessing, Choi et al. (2018) and Mozer et al. (2017) propose to augment observations with timestamps. Lipton et al. (2016) suggest observation masks. However, such approaches have no notion of dynamics between observations. An alternative approach is to decay the hidden state exponentially between observations according to a trainable decay parameter (Che et al., 2018; Cao et al., 2018). These methods are limited to decaying dynamics, whereas the CRU is more expressive.

Continuous-Time RNNs Continuous-time RNNs have a long history, dating back to some of the original work on recurrent networks in the field. They are recurrent architectures whose internal units are governed by a system of ODEs with trainable weights (Pearlmutter, 1995). The theory for different gradient-based optimization schemes for their parameters have been developed by Pineda (1987), Pearlmutter (1989), and Sato (1990). Notably, LeCun et al. (1988)'s derivation using the adjoint method provides the theoretical foundation for modern implementations of neural ODEs (Chen et al., 2018).

Neural ODEs Neural ODEs model the continuous dynamics of a hidden state by an ODE specified by a neural network layer. Chen et al. (2018) propose latent ODE, a generative model whose latent state evolves according to a neural ODE. However, it has no update mechanism to incorporate incoming observations into the latent trajectory. Kidger et al. (2020) and Morrill et al. (2021) extend neural ODEs with concepts from rough analysis, which allow for online learning. ODE-RNN (Rubanova et al., 2019) and ODE-LSTM (Lechner & Hasani, 2020) use standard RNN gates to sequentially update the hidden state at observation times. GRU-ODE-B (Brouwer et al., 2019) and Neural Jump ODE (Herrera et al., 2021) couple ODE dynamics with an Bayesian update step. Neural ODE approaches typically rely on a numerical ODE solver, whereas the state evolution of a CRU is in closed form.

Neural SDEs As the stochastic analogue of neural ODEs, neural SDEs define a latent temporal process with a SDE parameterized by neural networks. Li et al. (2020) use the stochastic adjoint sensitive method to compute efficient gradients of their SDE-induced generative model. Jia & Benson (2019) allow for discontinuities in the latent state to mimic stochastic events. Deng et al. (2020) and Deng et al. (2021) use dynamic normalizing flows to map the latent state to a continuous-path observation sequence. Kidger et al. (2021) fit neural SDEs in a generator-discriminator framework. Like CRU, these methods accommodate noise in the latent process, but generally rely on variational approximations and numerical solvers. In contrast, CRU propagates the latent state in closed form and can be trained end-to-end.

[1]https://github.com/boschresearch/Continuous-Recurrent-Units

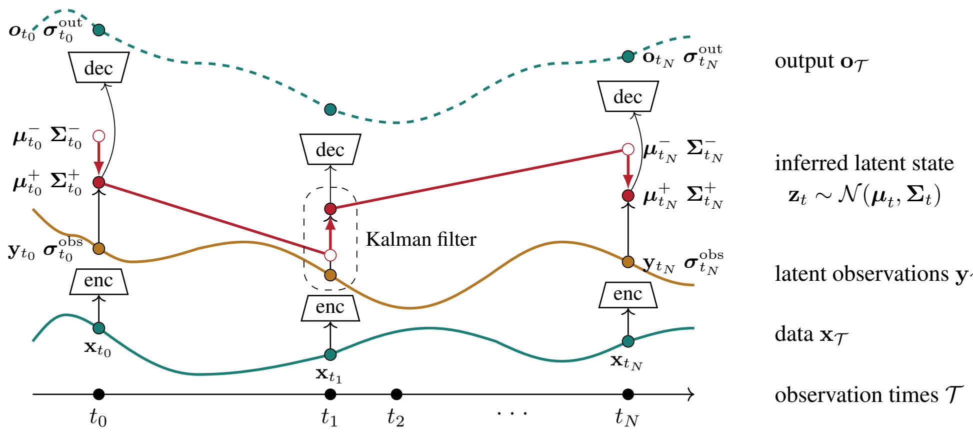
o_{t_0} $\sigma^{\text{out}}_{t_0}$
dec
$\mu^-_{t_0}$ $\Sigma^-_{t_0}$
$\mu^+_{t_0}$ $\Sigma^+_{t_0}$
y_{t_0} $\sigma^{\text{obs}}_{t_0}$
enc
x_{t_0}
Kalman filter
x_{t_1}
o_{t_N} $\sigma^{\text{out}}_{t_N}$
$\mu^-_{t_N}$ $\Sigma^-_{t_N}$
$\mu^+_{t_N}$ $\Sigma^+_{t_N}$
y_{t_N} $\sigma^{\text{obs}}_{t_N}$
x_{t_N}
t_0 t_1 t_2 $\cdots$ t_N
output $o_\mathcal{T}$
inferred latent state $z_t \sim \mathcal{N}(\mu_t, \Sigma_t)$
latent observations $y_\mathcal{T}$
data $x_\mathcal{T}$
observation times $\mathcal{T}$

Figure 1. CRU: An encoder maps the observation $\mathbf{x}_t$ to a latent observation $\mathbf{y}_t$ and elementwise uncertainties $\boldsymbol{\sigma}_t^{\text{obs}}$. Both are combined with the latent state prior $\mathcal{N}(\boldsymbol{\mu}_t^-, \boldsymbol{\Sigma}_t^-)$ to produce the posterior $\mathcal{N}(\boldsymbol{\mu}_t^+, \boldsymbol{\Sigma}_t^+)$ (red arrows). A decoder yields the output $\mathbf{o}_t$.

Transformers for Irregular Time Series Besides recurrent and differential equation-based architectures, recent work proposed attention-based methods (Vaswani et al., 2017) to model sequences with arbitrary timestamps. Zhang et al. (2019) combines time gap decay with attention mechanisms to weight for elapsed time. Horn et al. (2020) use set functions to compress irregular sequences to fixed-length representations. Multi-time attention network (mTAND) (Shukla & Marlin, 2021) feeds time embeddings to an attention mechanism. These models are typically quite large, whereas the CRU achieves high performance despite its small model size.

3. Method

The CRU is a RNN for processing sequential data with irregular observation times. It employs a nonlinear mapping (a neural network encoder and decoder) to relate individual observations with a latent state space. In this latent state space, it assumes a continuous latent state whose dynamics evolve according to a linear SDE. The recursive computations of the CRU can be derived using the continuous-discrete Kalman filter (Jazwinski, 1970) and are in closed form. As a result, the CRU has temporal continuity between hidden states and a gating mechanism that can optimally integrate noisy observations at arbitrary observation times.

We first specify the modeling assumptions for the continuous latent state of the CRU as well as the role of the encoder and the decoder in Sec. 3.1. In Sec. 3.2, we derive the recursive internal computations of the CRU with the resulting recurrent architecture summarized in Sec. 3.3. We then develop an efficient CRU parameterization scheme that affects modeling flexibility and run time (Sec. 3.4). Finally, in Sec. 3.5, we describe how to train a CRU.

3.1. Overview of Proposed Approach

The CRU addresses the challenge of modeling a time series $\mathbf{x}_\mathcal{T} = [\mathbf{x}_t | t \in \mathcal{T} = \{t_0, t_1, \cdots t_N\}]$ whose observation times $\mathcal{T} = \{t_0, t_1, \cdots t_N\}$ can occur at irregular intervals.

Modeling Assumptions for the Latent State Unlike the discrete hidden state formulation of standard RNNs, the latent state $\mathbf{z} \in \mathbb{R}^M$ of a CRU has continuous dynamics, which are governed by a linear SDE

$$d\mathbf{z} = \mathbf{A}\mathbf{z}dt + \mathbf{G}d\boldsymbol{\beta}, \tag{1}$$

with time-invariant transition matrix $\mathbf{A} \in \mathbb{R}^{M\times M}$ and diffusion coefficient $\mathbf{G} \in \mathbb{R}^{M\times B}$. The integration variable $\boldsymbol{\beta} \in \mathbb{R}^B$ is a Brownian motion process with diffusion matrix $\mathbf{Q} \in \mathbb{R}^{B\times B}$. The CRU assumes a Gaussian observation model $\mathbf{H} \in \mathbb{R}^{D\times M}$ that generates noisy latent observations

$$\mathbf{y}_t \sim \mathcal{N}(\mathbf{H}\mathbf{z}_t, (\boldsymbol{\sigma}_t^{\text{obs}})^2\mathbf{I}), \tag{2}$$

with observation noise $\boldsymbol{\sigma}_t^{\text{obs}}$.

Sequential Processing At each time point $t \in \mathcal{T}$, the latent observation $\mathbf{y}_t$ and its elementwise latent observation noise $\boldsymbol{\sigma}_t^{\text{obs}}$ are produced by a neural network encoder f_θ,

$$\text{encoder:} \qquad [\mathbf{y}_t, \boldsymbol{\sigma}_t^{\text{obs}}] = f_\theta(\mathbf{x}_t), \tag{3}$$

applied to the observation $\mathbf{x}_t$. At each observation time, we distinguish between a prior and a posterior distribution on $\mathbf{z}_t$.[2]

$$\text{prior:} \qquad p(\mathbf{z}_t|\mathbf{y}_{<t}) = \mathcal{N}(\boldsymbol{\mu}_t^-, \boldsymbol{\Sigma}_t^-) \tag{4}$$

$$\text{posterior:} \qquad p(\mathbf{z}_t|\mathbf{y}_{\le t}) = \mathcal{N}(\boldsymbol{\mu}_t^+, \boldsymbol{\Sigma}_t^+). \tag{5}$$

[2]We use the notation $\mathbf{y}_{<t} := \{\mathbf{y}_{t'} \text{ for } t' \in \mathcal{T} \text{ s.t. } t' < t\}$ for the set of all latent observations before t and $\mathbf{y}_{\le t} := \{\mathbf{y}_{t'} \text{ for } t' \in \mathcal{T} \text{ s.t. } t' \le t\}$ for this set including $\mathbf{y}_t$.

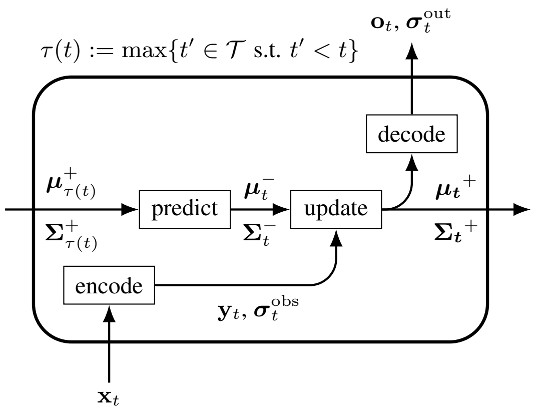
$\mathbf{o}_t, \boldsymbol{\sigma}_t^{\text{out}}$
$\tau(t) := \max\{t' \in \mathcal{T} \text{ s.t. } t' < t\}$
decode
$\boldsymbol{\mu}^+_{\tau(t)}$
$\boldsymbol{\Sigma}^+_{\tau(t)}$
predict
$\boldsymbol{\mu}_t^-$
$\boldsymbol{\Sigma}_t^-$
update
$\boldsymbol{\mu}_t^+$
$\boldsymbol{\Sigma}_t^+$
encode
$\mathbf{y}_t, \boldsymbol{\sigma}_t^{\text{obs}}$
$\mathbf{x}_t$

Figure 2. The internal hidden states of a CRU cell are the posterior mean and variance μ_t^+ and Σ_t^+ of the continuous state variable $\mathbf{z}$. They are computed recursively according to Algorithm 1.

The parameters of the prior, $\boldsymbol{\mu}_t^-$ and $\boldsymbol{\Sigma}_t^-$, are computed by propagating the latent state according to Eqn. (1) (we call this the "prediction step") while the parameters of the posterior, $\boldsymbol{\mu}_t^+$ and $\boldsymbol{\Sigma}_t^+$, are computed with a Bayesian update (which we call the "update step"). The optimal prediction and update step will be derived in closed form in Sec. 3.2.

Fig. 1 gives an overview of the CRU from a Kalman filtering perspective: observations (green) are mapped by the encoder into a latent observation space (orange). The mean and variance of the latent state is inferred using the predict and update step of the continuous-discrete Kalman filter (red). Finally, a decoder maps the posterior parameters to the desired output space along with elementwise uncertainties.

$$\text{decoder:} \qquad [\mathbf{o}_t, \boldsymbol{\sigma}_t^{\text{out}}] = g_\phi(\boldsymbol{\mu}_t^+, \boldsymbol{\Sigma}_t^+). \tag{6}$$

3.2. Continuous-Discrete Kalman Filter

The continuous-discrete Kalman filter (Jazwinski, 1970) is the optimal state estimator for a continuous state space model (Eqn. (1)) with a discrete-time Gaussian observation process (Eqn. (2)). This version of the Kalman filter allows modelling observations of a continuous process at potentially arbitrary but discrete observation times. Given the latent observations, the posterior distribution of the latent state (Eqn. (5)) is computed recursively, alternating between a predict and an update step. These steps are derived next.

3.2.1. Prediction Step

Between observation times, the prior density describes the evolution of $\mathbf{z}_t$. It is governed by the SDE in Eqn. (1), which has an analytical solution for linear, time-invariant systems as considered here. To compute the prior at time t, we assume that the posterior parameters $\boldsymbol{\mu}^+_{\tau(t)}, \boldsymbol{\Sigma}^+_{\tau(t)}$ at the

Algorithm 1 The CRU

```
**Input:** Datapoints and their timestamps $\{(\mathbf{x}_t, t)\}_{t\in\mathcal{T}}$
**Initialize:** $\boldsymbol{\mu}^+_{t_0} = \mathbf{0}, \boldsymbol{\Sigma}^+_{t_0} = 10 \cdot \mathbf{I}$
**for** observation times $t > t_0 \in \mathcal{T}$ **do**
  $\mathbf{y}_t, \boldsymbol{\sigma}_t^{\text{obs}} = f_\theta(\mathbf{x}_t)$
  $\boldsymbol{\mu}_t^-, \boldsymbol{\Sigma}_t^- = \text{predict}(\boldsymbol{\mu}^+_{\tau(t)}, \boldsymbol{\Sigma}^+_{\tau(t)}, t - \tau(t))$
  $\boldsymbol{\mu}_t^+, \boldsymbol{\Sigma}_t^+ = \text{update}(\boldsymbol{\mu}_t^-, \boldsymbol{\Sigma}_t^-, \mathbf{y}_t, \boldsymbol{\sigma}_t^{\text{obs}})$
  $\mathbf{o}_t, \boldsymbol{\sigma}_t^{\text{out}} = g_\phi(\boldsymbol{\mu}_t^+, \boldsymbol{\Sigma}_t^+)$
**end for**
**Return:** $\{\mathbf{o}_t, \boldsymbol{\sigma}_t^{\text{out}}\}_{t\in\mathcal{T}}$
```

last observation time,

$$\tau(t) := \max\{t' \in \mathcal{T} \text{ s.t. } t' < t\}, \tag{7}$$

have already been computed. The SDE solution at time t is

$$\mathbf{z}_t = \mathbf{exp}\big(\mathbf{A}(t-\tau(t))\big)\mathbf{z}_{\tau(t)} + \int_{\tau(t)}^{t} \mathbf{exp}\big(\mathbf{A}(t-s)\big)\mathbf{G}d\boldsymbol{\beta}_s,$$

which results in a prior mean and covariance of

$$\begin{aligned} \boldsymbol{\mu}_t^- &= \mathbf{exp}\big(\mathbf{A}(t-\tau(t))\big)\boldsymbol{\mu}^+_{\tau(t)} \\ \boldsymbol{\Sigma}_t^- &= \mathbf{exp}\big(\mathbf{A}(t-\tau(t))\big)\boldsymbol{\Sigma}^+_{\tau(t)}\mathbf{exp}\big(\mathbf{A}(t-\tau(t))\big)^T \\ &\quad + \int_{\tau(t)}^{t} \mathbf{exp}\big(\mathbf{A}(t-s)\big)\mathbf{G}\mathbf{Q}\mathbf{G}^T\mathbf{exp}\big(\mathbf{A}(t-s)\big)^T ds, \end{aligned} \tag{8}$$

where $\mathbf{exp}(\cdot)$ denotes the matrix exponential. The integral can be resolved analytically using matrix fraction decomposition and the computation is detailed in Appendix A.2.1. We summarize the prediction step (Eqn. (8)) for the parameters of the prior with

$$[\boldsymbol{\mu}_t^-, \boldsymbol{\Sigma}_t^-] = \text{predict}(\boldsymbol{\mu}^+_{\tau(t)}, \boldsymbol{\Sigma}^+_{\tau(t)}, t - \tau(t)). \tag{9}$$

3.2.2. Update Step

At the time of a new observation $\mathbf{y}_t$, the prior is updated using Bayes' theorem,

$$p(\mathbf{z}_t|\mathbf{y}_{\leq t}) \propto p(\mathbf{y}_t|\mathbf{z}_t)p(\mathbf{z}_t|\mathbf{y}_{<t}). \tag{10}$$

Due to the Gaussian assumption, the posterior is again Gaussian, and its mean and covariance are given by

$$\boldsymbol{\mu}_t^+ = \boldsymbol{\mu}_t^- + \mathbf{K}_t(\mathbf{y}_t - \mathbf{H}\boldsymbol{\mu}_t^-) \tag{11}$$

$$\boldsymbol{\Sigma}_t^+ = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\boldsymbol{\Sigma}_t^-. \tag{12}$$

The updates can be seen as weighted averages, where the Kalman gain $\mathbf{K}_t$ acts as a gate between prior and observation. It contrasts observation noise with prior uncertainty and is high when observations have a low noise level,

$$\mathbf{K}_t = \boldsymbol{\Sigma}_t^-\mathbf{H}^T(\mathbf{H}\boldsymbol{\Sigma}_t^-\mathbf{H}^T + \boldsymbol{\Sigma}_t^{\text{obs}})^{-1}. \tag{13}$$

We summarize the update step as

$$[\boldsymbol{\mu}_t^+, \boldsymbol{\Sigma}_t^+] = \text{update}(\boldsymbol{\mu}_t^-, \boldsymbol{\Sigma}_t^-, \mathbf{y}_t, \boldsymbol{\sigma}_t^{\text{obs}}). \tag{14}$$

3.3. Continuous Recurrent Units

A CRU is a recurrent neural architecture that uses the predict and update step of a continuous-discrete Kalman filter (Eqns. (9) and (14)) in an encoder-decoder framework (Eqns. (3) and (6)) to sequentially process irregularly sampled time series. An overview of a CRU cell is given in Fig. 2. Algorithm 1 summarizes the recursion, which is used by the CRU cell to update its internal parameters based on sequential inputs and to produce the output sequence.

Even though the derivation assumes a probabilistic latent state $\mathbf{z}$, the internal computations in the CRU cell are deterministic, in closed form, and amenable to back-propagation. This means that the CRU can be used (and trained end-to-end) like other recurrent architectures, such as LSTMs or GRUs on various sequence modeling tasks. Its advantage compared to these architectures is that the CRU handles irregular observation times in a principled manner.

We next describe parameterization choices for CRU which lead to more expressive modeling capacity (Sec. 3.4.1) and faster computation (Sec. 3.4.2) of the state equations. Finally, we present in Sec. 3.5 how to train a CRU.

3.4. Flexible and Efficient Parameterization of the CRU

The linearity assumption of the continuous-discrete Kalman filter is advantageous, as it leads to optimal closed-form computations. However, it also limits the expressiveness of the model. The idea of CRU is that the modelling flexibility of the encoder and decoder mitigates this limitation and that the dimensionality of the state space is large enough for a linearly evolving latent state to lead to expressive relationships between input and output sequences.

On the other hand, the dimensionality of the latent state cannot be too large in practice as it affects the runtime of the matrix inversion in Eqn. (13) and the matrix exponential in Eqn. (8). To address this trade-off between modeling flexibility and efficient computation, we make certain parameterization choices for the CRU. In Sec. 3.4.1, we describe a locally linear transition model, which maintains the closed form updates of Sec. 3.2 while making the model more flexible. In Sec. 3.4.2, we develop f-CRU, a version of the CRU with a novel parameterization of the transition matrices via their eigendecompositions. The resulting model has less modeling flexibility than the CRU but is significantly faster to train and amenable to larger state spaces.

3.4.1. Locally Linear State Transitions

A locally linear transition model increases the modeling flexibility of CRU while maintaining the closed form computation of the predict and update steps in Sec. 3.2. Similar approaches have been used in deep Kalman architectures (Karl et al., 2017; Fraccaro et al., 2017). We employ the parameterization strategy of Becker et al. (2019) and design the transition matrix $\mathbf{A}_t$ at time t as a weighted average of K parameterized basis matrices. The weighting coefficients $\alpha_t^{(k)}$ for $k \in \{1...K\}$ are obtained from the current posterior mean $\boldsymbol{\mu}_t^+$ by a neural network w_ψ with softmax output,

$$\mathbf{A}_t = \sum_{k=1}^{K} \alpha_t^{(k)} \mathbf{A}^{(k)}, \qquad \text{with } \boldsymbol{\alpha}_t = w_\psi(\boldsymbol{\mu}_t^+). \tag{15}$$

To reduce the number of parameters, each basis matrix consists of four banded matrices with bandwidth b. In addition, we assume a diagonal diffusion matrix $\mathbf{Q}$ whose vector of diagonal entries $\mathbf{q}$ is a time-invariant learnable parameter. The diffusion coefficient $\mathbf{G}$ of the SDE in Eqn. (1) is fixed at the identity matrix, i.e. $\mathbf{G} = \mathbf{I}$. This is not restrictive as $\mathbf{G}$ only occurs in combination with the learnable parameter vector $\mathbf{q}$ (Eqn. (8)), which is unconstrained.

3.4.2. Efficient Implementation

The runtime of the CRU is dominated by two operations: the matrix inversion in the computation of the Kalman gain (Eqn. (13)) and the matrix exponential in the prediction step (Eqn. (8)). As in Becker et al. (2019), there is a trade-off between modeling flexibility and runtime when choosing how to parametrize the model. Becker et al. (2019) use certain factorization assumptions on the state covariance $\boldsymbol{\Sigma}_t$ and the observation model $\mathbf{H}$ that increase speed and stability by simplifying the matrix inversion. CRU also benefits from these assumptions, which are detailed in Appendix B. However, the CRU has an additional computational bottleneck, namely the matrix exponential in Eqn. (8).

In this section, we develop fast CRU (f-CRU), a model variant that benefits from an efficient implementation of the prediction step. f-CRU bypasses the computation of the matrix exponential by allowing only commutative and symmetric base matrices $\mathbf{A}^{(k)}$ with related eigenspaces. While this limits the modeling flexibility, it reduces the runtime of the matrix exponential from complexity $O(n^3)$ to matrix multiplication and elementwise operations.

To avoid having to compute an eigenvalue decomposition, we directly parameterize the basis matrices $\mathbf{A}^{(k)}$ in terms of their eigenvalues and eigenvectors, which enable a change of basis. In the projected space, the state transitions are diagonal and the matrix exponential simplifies to the elementwise exponential function. By allowing only commutative, symmetric basis matrices, we can ensure that the matrix exponential in the projected space is invariant to the order in which the $\mathbf{A}^{(k)}$ are summed (Eqn. (15)).

In detail, we assume diagonalizable basis matrices $\{\mathbf{A}^{(k)}\}_{k=1...K}$ that share the same orthogonal eigenvectors. That is to say, for all $k \in \{1...K\}$ we have $\mathbf{A}^{(k)} = \mathbf{E}\mathbf{D}^{(k)}\mathbf{E}^T$ where $\mathbf{D}^{(k)}$ is a diagonal matrix whose i-th di-

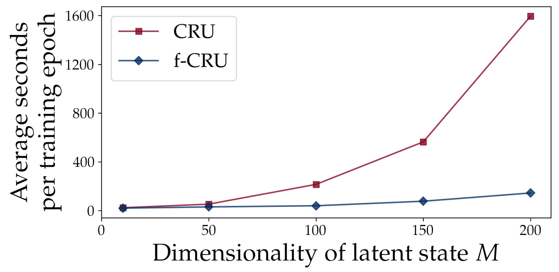

Figure 3. The parameterization of the f-CRU makes it faster than the CRU, especially as the latent dimensionality M increases.

agonal entry is the eigenvalue of $\mathbf{A}^{(k)}$ corresponding to the eigenvector in the i-th column of $\mathbf{E}$. When using $\mathbf{E}$ to perform a change of basis on the latent state $\mathbf{z}_t$, the SDE of the transformed state vector $\mathbf{w}$ has diagonal transitions $\mathbf{D}$. The prior mean at time t simplifies to,

$$\boldsymbol{\mu}_t^- = \mathbf{E} \exp\left((t-\tau(t)) \sum_{k=1}^{K} \alpha^{(k)} \mathbf{D}^{(k)}\right) \mathbf{E}^T \boldsymbol{\mu}_{\tau(t)}^+, \quad (16)$$

where $\exp(\cdot)$ denotes the elementwise exponential function. We follow Rome (1969) to efficiently compute the covariance of the projected state space $\boldsymbol{\Sigma}_t^{\mathbf{w}}$ at time t, which is mapped back to the original basis of $\mathbf{z}$ to yield the prior covariance at time t

$$\boldsymbol{\Sigma}_t^- = \mathbf{E} \boldsymbol{\Sigma}_t^{\mathbf{w}} \mathbf{E}^T. \quad (17)$$

We provide the thorough definitions and computations of the f-CRU prior computation in Appendix A.1.2. Fig. 3 illustrates the computational gain realized by the parameterization scheme of f-CRU. In Sec. 4, we further study the speed and accuracy trade-off between CRU and f-CRU.

3.5. Training

The trainable parameters of the CRU are the neural network parameters of the encoder and decoder (θ and ϕ), and parameters associated with the locally linear SDE, namely the diagonal $\mathbf{q}$ of the diffusion matrix, the network parameters ψ for producing the weighting coefficients of the locally linear model, and the transition matrices ($\{\mathbf{A}^{(k)}\}_{i=1...K}$ for the CRU and $\mathbf{E}$, $\{\mathbf{D}^{(k)}\}_{i=1...K}$ for the f-CRU).

To train these parameters, we assume a dataset of sequences where each input sequence $\mathbf{x}_\mathcal{T}$ is associated with a target output sequence $\mathbf{s}_\mathcal{T}$. For real valued outputs, the objective function is the Gaussian negative log-likelihood of the ground truth $\mathbf{s}_\mathcal{T}$ and is given for a single sample by

$$\mathcal{L}(\mathbf{s}_\mathcal{T}) = -\frac{1}{N} \sum_{t \in \mathcal{T}} \log \mathcal{N}(\mathbf{s}_t | \mathbf{o}_t, (\boldsymbol{\sigma}_t^{\text{out}})^2), \quad (18)$$

where $\mathbf{o}_\mathcal{T}$ and the elementwise uncertainty estimate $\boldsymbol{\sigma}_\mathcal{T}^{\text{out}}$ is the output computed by the CRU. For binary outputs, the model is trained on the Bernoulli negative log-likelihood

$$\mathcal{L}(\mathbf{s}_\mathcal{T}) = -\frac{1}{N} \sum_{t \in \mathcal{T}} \sum_{i=1}^{D_0} s_t^{(i)} \log(o_t^{(i)}) + (1-s_t^{(i)}) \log(1-o_t^{(i)}). \quad (19)$$

We also use this loss for imputing gray scale images, where the outputs $\mathbf{s}_t$ are D_0 pixel values in the range $[0, 1]$. To maintain the orthogonality constraint on $\mathbf{E}$ during training of f-CRU, we use the tools by Lezcano-Casado (2019).

4. Empirical Study

We study the CRU on three different tasks (interpolation, regression, and extrapolation) on challenging datasets from meteorology and health care. In sequence interpolation (Sec. 4.3), the task is to learn the underlying dynamics of the data based on a subset of the observations. In Sec. 4.4, we study a regression task consisting of predicting the angle of a pendulum from noisy images of the system. Finally, in Sec. 4.5, models learn the dynamics of a sequence from the first half of the observations to predict the remaining sequence. This extrapolation is challenging because the models need to capture long-term temporal interactions.

We study CRU in comparison to other sequence models in terms of both accuracy and runtime. We find that (i) CRU trains consistently faster than neural ODE-based models, (ii) our methods outperforms most baselines on interpolation and regression, (iii) the uncertainty driven gating mechanisms handles noisy and partially observed inputs systematically by attributing less weight to them in the latent state update.

4.1. Datasets

Pendulum Images We used the pendulum simulation of Becker et al. (2019) to generate 4 000 synthetic image sequences. The 24x24 pixel images show a pendulum at 50 irregular time steps $\mathcal{T}$. For the interpolation task, half of the frames of each sequence are removed at random, resulting in an input sequence $\mathbf{x}_\mathcal{S}$ with a reduced set of indices $\mathcal{S} \subset \mathcal{T}$. The target output is the full image sequence $\mathbf{s}_\mathcal{T} = \mathbf{x}_\mathcal{T}$.

We also use the pendulum image sequences to study CRU on a regression task. Each frame $\mathbf{x}_t$ is associated with a target label $\mathbf{s}_t = (\sin(s_t), \cos(s_t))$, which is the angle of the pendulum. As in Becker et al. (2019), we corrupt the images with a correlated noise process. We used 2 000 sequences for training and 1 000 for validation and testing each.

Climate Data (USHCN) The United States Historical Climatology Network (USHCN) dataset (Menne et al., 2015) contains daily measurements from 1 218 weather stations across the US for five variables: precipitation, snowfall, snow depth, minimum and maximum temperature. We used the cleaning procedure by Brouwer et al. (2019) to select a subset of 1168 meteorological stations over a range of four years (1990 - 1993). Though collected in regular time intervals, climatic data is often sparse due to, e.g., sensor failures. To further increase the sporadicity of the data across time and dimensions, we first subsample 50% of the time points and then randomly remove 20% of the measurements. We test models on a 20% hold-out set and trained on 80% of which we used 25% for validation.

Electronic Health Records (Physionet) Finally, we also benchmark the models on the data set of the Physionet Computing in Cardiology Challenge 2012 (Silva et al., 2012). The data reports 41 measurements of the first 48 hours of 8000 ICU patients. We follow the preprocessing of Rubanova et al. (2019) and round observation times to 6 minutes steps resulting in 72 different time points per patient on average. At a single time point, an observation contains on average only 16% of the features with measurements entirely missing for some patients. We split the data into 20% test and 80% train set of which we used 25% for validation.

4.2. Baselines

We study the CRU and f-CRU in comparison to various baselines including RNN architectures known to be powerful sequence models (but for regular observation times) as well as ODE and attention-based models, which have been developed specifically for observations from continuous processes.

Recurrent Neural Networks We compare our method against two RNN architectures with discrete hidden state assumption: GRUs (Chung et al., 2014) and recurrent Kalman networks (RKNs) (Becker et al., 2019). To aid these models with irregular time intervals, we run a version where we feed the time gap $\Delta_t = t - \tau(t)$ as an additional input to the model (denoted as RKN-Δ_t and GRU-Δ_t). Another baseline is GRU-D (Che et al., 2018), which uses trainable hidden state decay between observations to handle irregular inputs on a continuous time scale.

ODE-based Models We also test CRU against three ODE-based models that can naturally deal with irregularly sampled time series: (1) ODE-RNN (Rubanova et al., 2019) alternates between continuous hidden state dynamics defined by an ODE and classical RNN updates at observation times. (2) Latent ODE (Chen et al. (2018), Rubanova et al. (2019)) is a generative model that uses ODE-RNN as recognition network to infer the initial value of its latent state and models the state evolution with an ODE. (3) GRU-ODE-B (Brouwer et al., 2019) combines a continuous-time version of GRUs with a discrete update network.

Attention-based Model We also compare CRU against a transformer network: mTAND (Shukla & Marlin, 2021) is a generative model that employs multi-time attention modules in the encoder and decoder.

Implementation Details For a fair comparison, we use the same latent state dimension for all approaches ($M = 30$ for pendulum, $M = 20$ for Physionet and $M = 10$ for USHCN), except for GRU, where we increase the latent state size such that the number of parameters is comparable to CRU. For the Physionet and USHCN experiments, the encoder and decoder architecture of the CRU mimic that of the latent ODE in Rubanova et al. (2019)'s Physionet set-up. (Details can be found in Appendix D.)

For processing pendulum images, the CRU encoder and decoder are a convolutional architecture (see Appendix D.8). To ensure a fair comparison, we follow the set-up of Becker et al. (2019) and use the same encoder and decoder architecture as for RKN, CRU and f-CRU to give the same feature extraction capabilities to the other baseline models: the baseline models are applied to the latent observations that the encoder produces from the inputs and the decoder maps the baseline outputs to the target output. In this augmented framework, the encoder, the baseline model, and the decoder are trained jointly. More information and other implementation details can be found in Appendix D.

4.3. Results on Sequence Interpolation

We first examine the efficacy of CRU in sequence interpolation on pendulum images, USHCN and Physionet. The task consists of inferring the entire sequence $\mathbf{s}_\mathcal{T} = \mathbf{x}_\mathcal{T}$ based on a subset of observations $\mathbf{x}_\mathcal{S}$ where $\mathcal{S} \subseteq \mathcal{T}$. In the pendulum interpolation task, $\mathcal{S}$ contains half of the total time points sampled at random. For USHCN and Physionet, we follow the Physionet set-up in Rubanova et al. (2019), where reconstruction is based on the entire time series, i.e. $\mathcal{S} = \mathcal{T}$.

Tab. 1 reports mean squared error (MSE) on full test sequences and average runtime per epoch for USHCN and Physionet. Tab. 2 summarizes results on the pendulum image imputation task. All reported results are averages over 5 runs. The runtime measures a training pass through the entire dataset while keeping settings such as batch size and number of parallel threads comparable across architectures. Both CRU and f-CRU outperform baseline models on the interpolation task on most datasets. The efficient implementation variant, f-CRU, is consistently faster than neural ODE-based architectures. f-CRU produces results comparable to the CRU baseline, while reducing training time by up to 50% even in low-dimensional state spaces. Though discrete RNNs are still faster, f-CRU takes only a fraction of time to train as neural ODE-based methods.

Three noteworthy factors influence training and inference

Table 1. Test MSE (mean ± std) and runtime (average seconds/epoch) for interpolation and extrapolation on USHCN and Physionet.

	Interpolation MSE ($\times 10^{-2}$)		Extrapolation MSE ($\times 10^{-2}$)		Runtime (sec./epoch)	
Model	USHCN	Physionet	USHCN	Physionet	USHCN	Physionet
mTAND	1.766 ± 0.009	0.208 ± 0.025	2.360 ± 0.038	**0.340 ± 0.020**	7	10
RKN	0.021 ± 0.015	0.188 ± 0.088	1.478 ± 0.283	0.704 ± 0.038	94	39
RKN-Δ_t	**0.009 ± 0.002**	0.186 ± 0.030	1.491 ± 0.272	0.703 ± 0.050	94	39
GRU	0.184 ± 0.183	0.364 ± 0.088	2.071 ± 0.015	0.880 ± 0.140	3	5
GRU-Δ_t	0.090 ± 0.059	0.271 ± 0.057	2.081 ± 0.054	0.870 ± 0.077	3	5
GRU-D	0.944 ± 0.011	0.338* ± 0.027	1.718 ± 0.015	0.873 ± 0.071	292	5736
Latent ODE	1.798 ± 0.009	0.212* ± 0.027	2.034 ± 0.005	0.725 ± 0.072	110	791
ODE-RNN	0.831 ± 0.008	0.236* ± 0.009	1.955 ± 0.466	0.467 ± 0.006	81	299
GRU-ODE-B	0.841 ± 0.142	0.521 ± 0.038	5.437 ± 1.020	0.798 ± 0.071	389	90
f-CRU (ours)	0.013 ± 0.004	0.194 ± 0.007	1.569 ± 0.321	0.714 ± 0.036	61	62
CRU (ours)	0.016 ± 0.006	**0.182 ± 0.091**	**1.273 ± 0.066**	0.629 ± 0.093	122	114

* Results from Rubanova et al. (2019).

time: (1) Unlike ODE-based methods, CRU has closed-from computation, whereas ODE-based methods still need to call an ODE solver during inference. (2) Our method can make efficient use of batching and unlike neural ODE-based architectures, CRU does not require solving the state propagation in sync for the union of all timestamps in a minibatch. Thus, the complexity of CRU does not scale with the heterogeneity of timestamps in a minibatch. On data where timestamps vary widely across sequences (e.g. Physionet), the number of required update steps can be up to B-times more for recurrent neural ODE-architectures than for CRU, where B denotes batch size. A batch size of 1 foregoes this advantage of CRU. (3) The operations of the encoder and decoder can counteract the speedup gained from closed-form latent state propagation. We found the speedup to be less significant on the pendulum data, where the encoder and decoder have a stronger influence on runtime (Tab. 2).

4.4. Results on Pendulum Angle Prediction from Images

Next, we consider a regression task on irregularly sampled pendulum image sequences. Here, each observed image $\mathbf{x}_t$ is mapped to a continuous target variable $\mathbf{s}_t$ representing the pendulum angle at each time step. To assess the noise robustness of the methods, the image sequences are corrupted by a correlated noise process as in Becker et al. (2019).

Figure 4 shows how the gating mechanism of CRUs works under varying degrees of observation noise. The norm of the Kalman gain mirrors the noise process of the sample sequence. At times of high observation noise, the norm of the Kalman gain is small and consequently the state update is dominated by the history encoded in the latent state prior. In contrast, when the pendulum is clearly observed, the Kalman gain attributes high weight to the new observation.

This principled handling of noise is one of the factors that can help explain the success of CRU in pendulum angle prediction. Results in terms of MSE are shown in Tab. 2. CRU outperforms existing baselines. In Appendix C, we also report log-likelihood results for this task. CRU and f-CRU yield the best performance.

Table 2. Test MSE $\times 10^{-3}$ (mean ± std) and runtime (average sec/epoch) on pendulum interpolation and regression.

Model	Interpolation	R.time	Regression
mTAND	15.400 ± 0.071	3	65.640 ± 4.050
RKN	5.200 ± 0.051	20	8.433 ± 0.610
RKN-Δ_t	1.903 ± 0.137	20	5.092 ± 0.395
GRU	5.086 ± 0.028	12	9.435 ± 0.998
GRU-Δ_t	2.073 ± 0.185	12	5.439 ± 0.988
Latent ODE	15.060 ± 0.138	52	15.700 ± 2.848
ODE-RNN	2.830 ± 0.137	37	7.256 ± 0.406
GRU-ODE-B	9.144 ± 0.020	60	9.783 ± 3.403
f-CRU	1.386 ± 0.162	29	6.155 ± 0.881
CRU	**0.996 ± 0.052**	36	**4.626 ± 1.072**

4.5. Results on Sequence Extrapolation

Finally, we study the performance of CRU on extrapolating sequences far beyond the observable time frame. We split the timeline into two halves $\mathcal{T}_1 = \{t_0, ...t_k\}$ and $\mathcal{T}_2 = \{t_{k+1}, ...t_N\}$. Models are tasked to predict all time points of the sequence $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$ based on time points in $\mathcal{T}_1$ only. In the Physionet experiment, the input consists of the first 24 hours and the target output of the first 48 hours of patient measurements. For USHCN we split the timeline into two parts of equal length $t_k = N/2$. During training, the entire observation sequence is used as target to guide the training process, except for GRU-ODE-B, where we used the training strategy proposed by the authors. During evaluation, performance is assessed only on the extrapolated

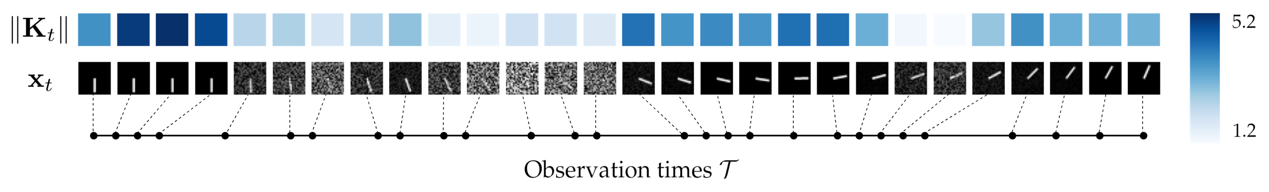

Figure 4. Pendulum angle prediction: The figures show noise corrupted observations and the corresponding Kalman gain norm for a pendulum trajectory sampled at irregular time intervals. The Kalman gain reflects the underlying noise process of the data.

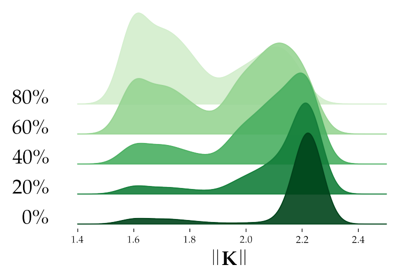

Figure 5. USHCN extrapolation: Distribution of the Kalman gain norm for different percentages of sparseness. For highly sparse observation vectors (e.g. 80% sparse), the distribution is shifted towards low norm values.

part of the test sequences.

Tab. 1 reports test errors on the extrapolated test sequences, $\mathbf{s}_{\mathcal{T}_2}$. On the Physionet data, mTAND achieves the lowest errors. On climate data, CRU reaches the highest performance.

Partial Observability To study the CRU gating mechanism in the presence of partially observed inputs, we also run the extrapolation task on the five-dimensional USHCN data while, this time, explicitly controlling the degree of partial observability. For each time point, we sample one to four features each with probability 0.1. The remaining 60% of observations are fully observed to ensure stable learning. Fig. 5 shows how the Kalman gain reacts to the sparseness of an observation. The distribution of fully observed feature vectors (0% sparseness) is centered around comparably high values of the Kalman gain norm, whereas sparse observations are associated with lower Kalman gain norm values. Thus, sparse observations are given less weight in the latent state update than fully observed feature vectors.

5. Conclusion

We have developed CRU, a RNN that models temporal data with non-uniform time intervals in a principled manner. It incorporates a continuous-discrete Kalman filter into an encoder-decoder structure thereby introducing temporal continuity into the hidden state and a notion of uncertainty into the network's gating mechanism. Our empirical study finds that the gating mechanism of the CRU weights noisy and partially observed input data accurately. Our method outperforms established recurrent sequence models such as GRU on irregularly sampled data and achieves better interpolation accuracy than neural ODE-based models on challenging datasets from various domains.

Acknowledgements

We thank Philipp Becker for providing the RKN code, Klaus Obermayer for helpful discussion, and Stefan Kurz for valuable feedback on the draft. The Bosch Group is carbon neutral. Administration, manufacturing and research activities do no longer leave a carbon footprint. This also includes GPU clusters on which the experiments have been performed.

References

Axelsson, P. and Gustafsson, F. Discrete-time solutions to the continuous-time differential lyapunov equation with applications to kalman filtering. *IEEE Transactions on Automatic Control*, 60(3):632–643, 2014.

Bayer, J. and Osendorfer, C. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.

Becker, P., Pandya, H., Gebhardt, G., Zhao, C., Taylor, C. J., and Neumann, G. Recurrent kalman networks: Factorized inference in high-dimensional deep feature spaces. In *International Conference on Machine Learning*, pp. 544–552. PMLR, 2019.

Brouwer, E. D., Simm, J., Arany, A., and Moreau, Y. Gru-ode-bayes: continuous modeling of sporadically-observed time series. In *Advances in Neural Information Processing Systems*, pp. 7379–7390, 2019.

Cao, W., Wang, D., Li, J., Zhou, H., Li, L., and Li, Y. Brits: Bidirectional recurrent imputation for time series. In *Advances in Neural Information Processing Systems*, pp. 6775–6785, 2018.

Che, Z., Purushotham, S., Cho, K., Sontag, D., and Liu, Y. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):1–12, 2018.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pp. 6572–6583, 2018.

Choi, E., Xiao, C., Stewart, W. F., and Sun, J. Mime: Multilevel medical embedding of electronic health records for predictive healthcare. *arXiv preprint arXiv:1810.09593*, 2018.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., and Bengio, Y. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems*, pp. 2980–2988, 2015.

Deng, R., Chang, B., Brubaker, M. A., Mori, G., and Lehrmann, A. Modeling continuous stochastic processes with dynamic normalizing flows. In *Advances in Neural Information Processing Systems*, pp. 7805–7815, 2020.

Deng, R., Brubaker, M. A., Mori, G., and Lehrmann, A. M. Continuous latent process flows. In *Advances in Neural Information Processing Systems*, pp. 5162–5173, 2021.

Fraccaro, M., Sønderby, S. K., Paquet, U., and Winther, O. Sequential neural models with stochastic layers. In *Advances in Neural Information Processing Systems*, pp. 2199–2207, 2016.

Fraccaro, M., Kamronn, S. D., Paquet, U., and Winther, O. A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *Advances in Neural Information Processing Systems*, pp. 3601–3610, 2017.

Goyal, A., Sordoni, A., Côté, M., Ke, N. R., and Bengio, Y. Z-forcing: Training stochastic recurrent networks. In *Advances in Neural Information Processing Systems*, pp. 6713–6723, 2017.

Herrera, C., Krach, F., and Teichmann, J. Neural jump ordinary differential equations: Consistent continuous-time prediction and filtering. In *9th International Conference on Learning Representations*, 2021.

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

Horn, M., Moor, M., Bock, C., Rieck, B., and Borgwardt, K. Set functions for time series. In *International Conference on Machine Learning*, pp. 4353–4363. PMLR, 2020.

Jazwinski, A. H. *Stochastic Processes and Filtering Theory*. Academic Press, 1970.

Jia, J. and Benson, A. R. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, pp. 9843–9854, 2019.

Kalman, R. E. A new approach to linear filtering and prediction problems. 1960.

Karl, M., Soelch, M., Bayer, J., and van der Smagt, P. Deep variational bayes filters: Unsupervised learning of state space models from raw data. In *International Conference on Learning Representations*, 2017.

Kidger, P., Morrill, J., Foster, J., and Lyons, T. J. Neural controlled differential equations for irregular time series. In *Advances in Neural Information Processing Systems*, 2020.

Kidger, P., Foster, J., Li, X., and Lyons, T. J. Neural sdes as infinite-dimensional gans. In *International Conference on Machine Learning*, pp. 5453–5463. PMLR, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Krishnan, R. G., Shalit, U., and Sontag, D. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

Lechner, M. and Hasani, R. Learning long-term dependencies in irregularly-sampled time series. *arXiv preprint arXiv:2006.04418*, 2020.

LeCun, Y., Touresky, D., Hinton, G., and Sejnowski, T. A theoretical framework for back-propagation. In *Proceedings of the 1988 Connectionist Models Summer School*, pp. 21–28, 1988.

Lezcano-Casado, M. Trivializations for gradient-based optimization on manifolds. In *Advances in Neural Information Processing Systems*, pp. 9154–9164, 2019.

Li, X., Wong, T.-K. L., Chen, R. T., and Duvenaud, D. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*, pp. 3870–3882. PMLR, 2020.

Lipton, Z. C., Kale, D., and Wetzel, R. Directly modeling missing data in sequences with rnns: Improved classification of clinical time series. In *Machine Learning for Healthcare Conference*, pp. 253–270. PMLR, 2016.

Menne, M., Williams Jr, C., and Vose, R. Long-term daily climate records from stations across the contiguous united states, 2015.

Morrill, J., Salvi, C., Kidger, P., and Foster, J. Neural rough differential equations for long time series. In *International Conference on Machine Learning*, pp. 7829–7838. PMLR, 2021.

Mozer, M. C., Kazakov, D., and Lindsey, R. V. Discrete event, continuous time rnns. *arXiv preprint arXiv:1710.04110*, 2017.

Pearlmutter, B. A. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

Pearlmutter, B. A. Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural networks*, 6(5):1212–1228, 1995.

Pineda, F. Generalization of back propagation to recurrent and higher order neural networks. In *Advances in Neural Information Processing Systems*, pp. 602–611, 1987.

Rome, H. A direct solution to the linear variance equation of a time-invariant linear system. *IEEE Transactions on Automatic Control*, 14(5):592–593, 1969.

Rubanova, Y., Chen, R. T., and Duvenaud, D. Latent odes for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pp. 5320–5330, 2019.

Sato, M.-a. A learning algorithm to teach spatiotemporal patterns to recurrent neural networks. *Biological Cybernetics*, 62(3):259–263, 1990.

Schmidt, F. and Hofmann, T. Deep state space models for unconditional word generation. In *Advances in Neural Information Processing Systems*, pp. 6161–6171, 2018.

Shukla, S. N. and Marlin, B. M. A survey on principles, models and methods for learning from irregularly sampled time series. *arXiv preprint arXiv:2012.00168*, 2020.

Shukla, S. N. and Marlin, B. M. Multi-time attention networks for irregularly sampled time series. In *International Conference on Learning Representations*, 2021.

Silva, I., Moody, G., Scott, D. J., Celi, L. A., and Mark, R. G. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. In *Computing in Cardiology*, pp. 245–248. IEEE, 2012.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Zhang, Y., Yang, X., Ivy, J. S., and Chi, M. ATTAIN: attention-based time-aware LSTM networks for disease progression modeling. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pp. 4369–4375, 2019.

A. Prior Computation

A.1. f-CRU

For an efficient implementation, we assume locally linear transitions with symmetric basis matrices $\{\mathbf{A}^{(k)}\}_{k=1...K}$ that share the same eigenvectors,

$$\mathbf{A}_t = \sum_{k=1}^{K} \alpha_t^{(k)} \mathbf{A}^{(k)}, \qquad \text{with} \qquad \mathbf{A}^{(k)} = \mathbf{E}\mathbf{D}^{(k)}\mathbf{E}^T. \tag{20}$$

A.1.1. Prior Mean

We can simplify the matrix exponential in Eqn. (8) to the elementwise exponential function:

$$\begin{aligned} \boldsymbol{\mu}_t^- &= \mathbf{exp}\Big(\mathbf{A}_t(t-\tau(t))\Big)\boldsymbol{\mu}_{\tau(t)}^+ \\ &= \mathbf{exp}\Big(\sum_{k=1}^{K} \alpha^{(k)}\mathbf{A}^{(k)}(t-\tau(t))\Big)\boldsymbol{\mu}_{\tau(t)}^+ \\ &= \mathbf{exp}\Big(\sum_{k=1}^{K} \alpha^{(k)}\mathbf{E}\mathbf{D}^{(k)}\mathbf{E}^T(t-\tau(t))\Big)\boldsymbol{\mu}_{\tau(t)}^+ \\ &= \mathbf{E}\exp\Big(\sum_{i=1}^{K} \alpha^{(k)}\mathbf{D}^{(k)}(t-\tau(t))\Big)\mathbf{E}^T\boldsymbol{\mu}_{\tau(t)}^+ \end{aligned} \tag{21}$$

In the last step, we exploit a property of the matrix exponential. The exponential of diagonalizable matrices can be obtained by exponentiating each entry on the main diagonal of the matrix of eigenvalues.

A.1.2. Prior Covariance

For diagonalizable transition matrices of a linear time-invariant system Rome (1969) proposes an analytical solution for the computation of the prior covariance based on the eigendecomposition of the transition matrix. Precisely, we can define a new state vector $\mathbf{w}$ with covariance $\boldsymbol{\Sigma}_{\tau(t)}^{\mathbf{w}}$

$$\mathbf{w}_{\tau(t)} = \mathbf{E}^T\mathbf{z}_{\tau(t)} \qquad \boldsymbol{\Sigma}_{\tau(t)}^{\mathbf{w}} = \mathbf{E}^T\boldsymbol{\Sigma}_{\tau(t)}^+\mathbf{E} \tag{22}$$

We further define $\tilde{\mathbf{D}}$ as the matrix whose ij-th element is the sum of the i-th and j-th diagonal entry of $\sum_{k=1}^{K}\mathbf{D}^{(k)}$,

$$\tilde{\mathbf{D}}_{ij} = \sum_{k=1}^{K} \mathbf{D}_{ii}^{(k)} + \mathbf{D}_{jj}^{(k)} \tag{23}$$

Let $\mathbf{S}$ be the transformed noise component,

$$\mathbf{S} = \mathbf{E}^T\mathbf{G}\mathbf{Q}\mathbf{G}^T\mathbf{E}, \tag{24}$$

then we can compute the covariance of $\mathbf{w}$ at time t with

$$\boldsymbol{\Sigma}_t^{\mathbf{w}} = (\mathbf{S}\odot\exp(\tilde{\mathbf{D}}(t-\tau(t))) - \mathbf{S}) \oslash \tilde{\mathbf{D}} + \boldsymbol{\Sigma}_{\tau(t)}^{\mathbf{w}} \odot \exp(\tilde{\mathbf{D}}(t-\tau(t))) \tag{25}$$

Mapping back to the space of $\mathbf{z}$, we obtain for the prior covariance of $\mathbf{z}$ at time t

$$\boldsymbol{\Sigma}_t^- = \mathbf{E}\boldsymbol{\Sigma}_t^{\mathbf{w}}\mathbf{E}^T \tag{26}$$

A.2. CRU

A.2.1. Prior Covariance

The integral in the prior covariance (Eqn. (8)) can be resolved analytically using matrix fraction decomposition (Axelsson & Gustafsson, 2014). The idea of the method consists in computing the integral by solving a matrix valued ODE. We compute the matrix exponential of the matrix $\mathbf{B}$

$$\mathbf{B} = \begin{pmatrix} \mathbf{A} & \mathbf{G}\mathbf{Q}\mathbf{G}^T \\ \mathbf{0} & -\mathbf{A}^T \end{pmatrix} \qquad \mathbf{exp}(\mathbf{B}(t-\tau(t))) = \begin{pmatrix} \mathbf{M}_1 & \mathbf{M}_2 \\ \mathbf{0} & \mathbf{M}_3 \end{pmatrix} \tag{27}$$

where $\mathbf{M}_1$, $\mathbf{M}_2$, $\mathbf{M}_3$ and $\mathbf{0}$ are of dimension $M \times M$. The prior covariance matrix at time t is then given by

$$\begin{aligned} \mathbf{\Sigma}_t^- =& \mathbf{exp}\big(\mathbf{A}(t-\tau(t))\big)\mathbf{\Sigma}_{\tau(t)}^+ \mathbf{exp}\big(\mathbf{A}(t-\tau(t))\big)^T \\ &+ \mathbf{M}_2 \mathbf{exp}\big(\mathbf{A}(t-\tau(t))\big)^T \end{aligned} \tag{28}$$

B. Approximations

In their work on RKNs, Becker et al. (2019) exploit assumptions on the structure of components of the Kalman filter to simplify computation. In particular, they reduce the matrix inversion in the Kalman gain to elementwise operations. We also exploit these approximations for the CRU. The following summarizes the approximations by Becker et al. (2019) and the resulting simplified update equations. We refer to Becker et al. (2019) for detailed derivations.

Observation Model The dimension of the latent state M is twice of the dimension of the latent observation space D, i.e. $M = 2D$. The observation model links both spaces and is fixed at $\mathbf{H} = [\mathbf{I}_D, \mathbf{0}_D]$ where $\mathbf{I}_D$ denotes the D-dimensional identity matrix and $\mathbf{0}_D$ the D-dimensional zero matrix. The idea is to split the latent state into an *observed part*, which extracts information directly from the observations and a *memory part*, which encodes features inferred over time.

State Covariance The covariance matrix of the latent state $\mathbf{\Sigma}_t$ is built of diagonal blocks $\mathbf{\Sigma}_t^u$, $\mathbf{\Sigma}_t^l$, $\mathbf{\Sigma}_t^s$, whose vectors of diagonal entries are denoted by $\boldsymbol{\sigma}_t^u, \boldsymbol{\sigma}_t^l, \boldsymbol{\sigma}_t^s$, respectively.

$$\mathbf{\Sigma}_t = \begin{pmatrix} \mathbf{\Sigma}_t^u & \mathbf{\Sigma}_t^s \\ \mathbf{\Sigma}_t^s & \mathbf{\Sigma}_t^l \end{pmatrix} \tag{29}$$

The observation part of the latent state is thus only correlated with the corresponding memory part. The argument behind this strong assumption is that the free parameters in the neural encoder and decoder suffice to find a representation where the above limitations hold.

Simplified Update Step The Kalman gain simplifies to a structure with two diagonal blocks of size $D \times D$, i.e. $\mathbf{K}_t = \begin{bmatrix}\mathbf{K}_t^u & \mathbf{K}_t^l\end{bmatrix}^T$. The vector of diagonal entries $\mathbf{k}_t^u$, $\mathbf{k}_t^l$ can be computed with element wise division ($\oslash$)

$$\mathbf{k}_t^u = \boldsymbol{\sigma}_t^{u,-} \oslash (\boldsymbol{\sigma}_t^{u,-} + \boldsymbol{\sigma}_t^{\text{obs}}) \qquad \mathbf{k}_t^l = \boldsymbol{\sigma}_t^{s,-} \oslash (\boldsymbol{\sigma}_t^{u,-} + \boldsymbol{\sigma}_t^{\text{obs}}) \tag{30}$$

The posterior mean update then simplifies to

$$\boldsymbol{\mu}_t^+ = \boldsymbol{\mu}_t^- + \begin{bmatrix}\mathbf{k}_t^u \\ \mathbf{k}_t^l\end{bmatrix} \odot \begin{bmatrix}\mathbf{y}_t - \boldsymbol{\mu}_t^{u,-} \\ \mathbf{y}_t - \boldsymbol{\mu}_t^{u,-}\end{bmatrix} \tag{31}$$

where $\boldsymbol{\mu}_t^u$ and $\boldsymbol{\mu}_t^l$ denote the upper and lower part of the prior mean respectively, i.e. $\boldsymbol{\mu}_t^- = \begin{bmatrix}\boldsymbol{\mu}_t^{u,-} & \boldsymbol{\mu}_t^{l,-}\end{bmatrix}^T$. The update of the posterior covariance reduces to

$$\boldsymbol{\sigma}_t^{u,+} = (\mathbf{1}_m - \mathbf{k}_t^u) \odot \boldsymbol{\sigma}_t^{u,-} \tag{32}$$

$$\boldsymbol{\sigma}_t^{s,+} = (\mathbf{1}_m - \mathbf{k}_t^u) \odot \boldsymbol{\sigma}_t^{s,-} \tag{33}$$

$$\boldsymbol{\sigma}_t^{l,+} = \boldsymbol{\sigma}_t^{l,-} - \mathbf{k}_t^l \odot \boldsymbol{\sigma}_t^{s,-} \tag{34}$$

where $\odot$ denotes elementwise multiplication.

C. Log-likelihood Results

To assess uncertainty computation in the presence of high observation noise, we report negative log-likelihood on pendulum regression. Tab. 3 shows Gaussian negative loglikelihood (NLL) on test data for methods providing uncertainty estimates. As for MSE results, CRU outperforms baseline models.

Table 3. Test Gaussian NLL (mean $\pm$ std) on pendulum regression.

Model	Regression
GRU	-4.78 $\pm$ 0.48
GRU-Δ_t	-5.45 $\pm$ 0.09
RKN	-4.38 $\pm$ 0.82
RKN-Δ_t	-5.26 $\pm$ 0.24
f-CRU	-5.46 $\pm$ 0.11
CRU	**-5.49** $\pm$ **0.05**

D. Implementation Details

Training In all experiments, we train each model for 100 epochs using the Adam optimizer (Kingma & Ba, 2015). Reported MSE and Gaussian NLL results are averages over 5 runs. We used a batch size of 50 for the pendulum and USHCN data and a batch size of 100 for Physionet. For USHCN and Physionet, we split the data into 80% train and 20% test and used 25% of the train set for validation. For the pendulum experiments, we generated 2 000 training sequences, 1 000 validation sequences and report results on a hold-out set of 1 000 sequences. The folds are reused for each compared model.

Hyperparameters and Architecture Here, we summarize hyperparameter choices made in the empirical study with details provided in Appendices D.1 to D.8. We used the following procedure to select latent state size, number of layers, number of hidden units, and training parameters: For GRU-D, latent ODE and ODE-RNN, we use the choices optimized by Rubanova et al. (2019) in their Physionet setup. For mTAND, we keep the hyperparameters chosen by the authors on Physionet and adjust the number of hidden units to control for model size. We then designed the encoder and decoder of the CRU such that the CRU has roughly the same number of parameters as the latent ODE model. We proceed analogously for hyperparameters of GRU-ODE-B: We employ the hyperparameter settings optimized by the authors on their USHCN experiment. We keep the baseline architectures fixed across experiments, except for the latent state size, which we chose separately for each experiment according to the dimensionality of the input and previous work. See Appendix D.8 for the latent state size of each experiment, which is shared across architectures. For the pendulum image set, encoder, decoder and latent space sizes of CRU follow the RKN (Becker et al., 2019) baseline on this set. To scale the methods based on neural ODEs to images, we also embedded ODE-RNN, latent ODE and GRU-ODE-B into the same encoder-decoder structure as CRU. Similarly, the GRU baseline is augmented with the same encoder and decoder. This was also done in the pendulum experiments of Becker et al. (2019) and more details and justification can be found there. The transformer method, mTAND, scales to high-dimensional input and we thus apply it directly on the raw images. We found GRU-ODE-B unstable when trained jointly with an encoder and decoder on the image interpolation task and therefore trained it directly on the raw images.

D.1. mTAND

We use a hidden state size of $M = 30$ for pendulum, $M = 20$ for Physionet, and $M = 10$ for USHCN. We use a learning rate of 0.001 and 64 reference time points as in Shukla & Marlin (2021). For interpolation and extrapolation, we train mTAND-Full with 10 hidden units in encoder and decoder resulting in a model that has still three times as many parameters as CRU. For the per-time-point regression task, we use mTAND-Enc with 10 hidden units.

D.2. RKN

We use a hidden state size of $M = 30$ for pendulum, $M = 20$ for Physionet, and $M = 10$ for USHCN. For the pendulum experiment, we keep the architecture and hyperparameter choices from Becker et al. (2019). We employ the same choices for CRU across all experiments, which are detailed in Appendix D.8. For the time-aware variant, RKN-Δ_t, we feed the time gaps as additional input to the transition net that learns weights $\alpha_{\tau(t)}$ for the basis matrices (Eqn. (16)), i.e. $\alpha_{\tau(t)} = w_\psi([t - \tau(t), \boldsymbol{\mu}^+_{\tau(t)}])$.

D.3. GRU

To make parameter sizes comparable, we use a hidden state size of 75 for GRU and GRU-Δ_t on all datasets. To test GRU on image data, we embed a GRU cell in the encoder-decoder architecture employed for RKN and CRU as in Becker et al. (2019). For GRU-Δ_t, the time gap between observations is concatenated to the input of the GRU cell.

D.4. GRU-D

We use a hidden state size of $M = 30$ for pendulum, $M = 20$ for Physionet, and $M = 10$ for USHCN. The other parameters are fixed across experiments. As described previously, we use hyperparameter and architecture choices by Rubanova et al. (2019), which result in 100 hidden units, a learning rate of 0.01 with a decay factor of 0.999.

D.5. Latent ODE

We use a latent state size of $M = 30$ for pendulum, $M = 20$ for Physionet, and $M = 10$ for USHCN. The other parameters are fixed across experiments. We used the latent ODE with an ODE-RNN recognition model. The recognition model has a hidden state of 40 dimensions, an ODE function with 3 layers and 50 hidden units and a GRU update with 50 hidden units. The ODE function of the generative model has 3 layers with 50 hidden units. We train the method with a learning rate of 0.01 and a decay rate of 0.999.

D.6. ODE-RNN

We use a hidden state size of $M = 30$ for pendulum, $M = 20$ for Physionet and $M = 10$ for USHCN. The other parameters are fixed across experiments. We use the hyperparameter choices proposed by Rubanova et al. (2019). Notably, we use 100 hidden units and a learning rate of 0.01 with a decay rate of 0.999.

D.7. GRU-ODE-B

We use a latent state size of $M = 30$ for pendulum, $M = 20$ for Physionet and $M = 10$ for USHCN. The other parameters are fixed across experiments. As outlined above, we keep the hyperparameters determined by the authors in their USHCN experiment. That is to say, we use 50 hidden units, a learning rate of 0.001, weight decay of 0.0001, and a dropout rate of 0.2. The f_{prep} function of the GRU-Bayes component has 10 hidden units, the f_{obs} mapping has 25 hidden units.

D.8. CRU and f-CRU

We trained CRU with an Adam optimizer (Kingma & Ba, 2015) on the Gaussian negative log-likelihood (Eqn. (18)) for Physionet, USHCN and pendulum angle prediction and on the loss of Eqn. (19) for the pendulum interpolation task. On the validation set in the Pendulum interpolation experiment, we found a learning rate of 0.001 to work best for CRU and a slightly higher learning rate of 0.005 for f-CRU. We kept this choice for all datasets throughout all other experiments. (Through hyperparameter optimization on each dataset separately, the experimental results might improve further.) Gradient clipping was used on USHCN and Physionet.

The initial conditions for the latent state are set to $\boldsymbol{\mu}^-_{t_0} = \mathbf{0}$ and $\boldsymbol{\Sigma}^-_{t_0} = 10 \cdot \mathbf{I}$. We found an initialization of the transitions such that the prior mean is close to the posterior mean of the previous time step crucial for performance and stability. Thus, we initialized the basis matrices $\{\mathbf{A}^{(k)}\}_{k=1...K}$ filled with zeros to start off with a prediction step of $\boldsymbol{\mu}^-_{t'} = \mathbf{I}\boldsymbol{\mu}^+_t$. Equivalently, we chose $\mathbf{E} = \mathbf{I}$ and $\mathbf{D}^{(k)} = 1e^{-5} \cdot \mathbf{I}, \forall k = 1...K$ for the f-CRU initialization. Missing features are zero-encoded and masked out in the NLL and MSE computation. For all experiments, we used a transition net w_ψ with one linear layer and softmax output.

The CRU architecture used in each experiment is explicitly summarized next.

D.8.1. Pendulum Interpolation

Continuous-discrete Kalman filter

- Latent observation dimension: 15
- Latent state dimension: 30
- Number of basis matrices: 15
- Bandwidth (for CRU): 3

Encoder: 2 convolution, 1 fully connected, linear output

- Convolution, 12 channels, 5×5 kernel, padding 2, ReLU, max pooling with 2×2 kernel and 2×2 stride
- Convolution, 12 channels, 3×3 kernel, padding 1, 2×2 stride, ReLU, max pooling with 2×2 kernel and 2×2 stride
- Fully-connected, 30 neurons, ReLU
- Linear output for latent observation; linear output, elu+1 activation for latent observation variance

Decoder output sequence $\mathbf{o}_{\mathcal{T}}$: 1 fully-connected, 3 Transposed convolution

- Fully connected, 144 neurons, ReLU
- Transposed convolution, 16 channels, 5×5 kernel, padding 2, 4×4 stride, ReLU
- Transposed convolution, 12 channels, 3×3 kernel, padding 1, 2×2 stride, ReLU
- Transposed convolution, 1 channel, 2×2 kernel, padding 5, 2×2 stride, sigmoid activation

D.8.2. Pendulum Regression

We used the same encoder and Kalman filter architecture as in the pendulum interpolation task.

Decoder output sequence $\mathbf{o}_{\mathcal{T}}$: 1 fully-connected, linear output

- Fully connected, 30 neurons, Tanh
- Linear output

Decoder output variance $\boldsymbol{\sigma}_{\mathcal{T}}^{out}$: 1 fully-connected, linear output

- Fully connected, 30 neurons, Tanh
- Linear output, elu+1 activation

D.8.3. USHCN

Continuous-discrete Kalman filter

- Latent observation dimension: 5
- Latent state dimension: 10
- Number of basis matrices: 15
- Bandwidth (for CRU): 3

Encoder: 3 fully connected, linear output

- Fully connected, 50 neurons, ReLU, layer normalization
- Fully connected, 50 neurons, ReLU, layer normalization
- Fully connected, 50 neurons, ReLU, layer normalization
- Linear output for latent observation; linear output, square activation for latent observation variance

Decoder output sequence $\mathbf{o}_{\mathcal{T}}$: 3 fully connected, linear output

- Fully connected, 50 neurons, ReLU, layer normalization
- Fully connected, 50 neurons, ReLU, layer normalization
- Fully connected, 50 neurons, ReLU, layer normalization
- Linear output for latent observation; linear output, square activation for latent observation variance

Decoder output variance $\boldsymbol{\sigma}_{\mathcal{T}}^{out}$: 1 fully-connected, linear output

- Fully connected, 50 neurons, ReLU, layer normalization
- Linear output, square activation

D.8.4. PHYSIONET

We used the same encoder and decoder architecture as in the USHCN experiment.

Continuous-discrete Kalman filter

- Latent observation dimension: 10
- Latent state dimension: 20
- Number of basis matrices: 20
- Bandwidth (for CRU): 10

E. Data Preprocessing

E.1. USHCN

Daily weather records can be downloaded at `https://cdiac.ess-dive.lbl.gov/ftp/ushcn_daily/`. We remove observations with a bad quality flag as in Brouwer et al. (2019). However, unlike Brouwer et al. (2019) we are interested in long term extrapolation and thus, select a different time window of four years from 1990 to 1993. We keep only centers that start reporting before 1990 and end reporting after 1993. We split the remaining 1168 centers into 60% train 20% validation and 20% test set. For each set, we remove measurements that are more than four standard deviations away from the set mean and normalize each feature to be in the [0,1] interval individually per set. For the baselines, we apply the time scaling strategies of previous work: we scale timestamps to be in the [0,1] for ODE-RNN, latent ODE and GRU-D as in Rubanova et al. (2019) and feed time points unprocessed to GRU-ODE-B as in Brouwer et al. (2019). For CRU, we scale the timestamps, which unit is days, by a factor of 0.3.

E.2. Physionet

The data is publicly available for download at `https://physionet.org/content/challenge-2012/1.0.0/`. We preprocess the data as in Rubanova et al. (2019): We discard four general descriptors reported once at admission (age, gender, height, ICU-type) and keep only the remaining set of 37 time-variant features. Time points are rounded to 6 minutes steps. We split the patients into 60% train, 20% validation and 20% test set. Lastly, we normalize each feature to be in the [0,1] interval separately per set. To mimic the training routine proposed by the authors, we scale the timestamps to the [0,1] interval for GRU-D, ODE-RNN and latent ODE, leave the timescale unchanged for GRU-ODE-B, and multiply timestamps by 0.2 for CRU.

F. Computing Infrastructure

Models were trained on one Nivida TU102GL Quatro RTX 6000/8000 with 40 physical Intel Xeon Gold 6242R CPU.

G. Source Code

Our code is available at `https://github.com/boschresearch/Continuous-Recurrent-Units`.

H. Sample Trajectory

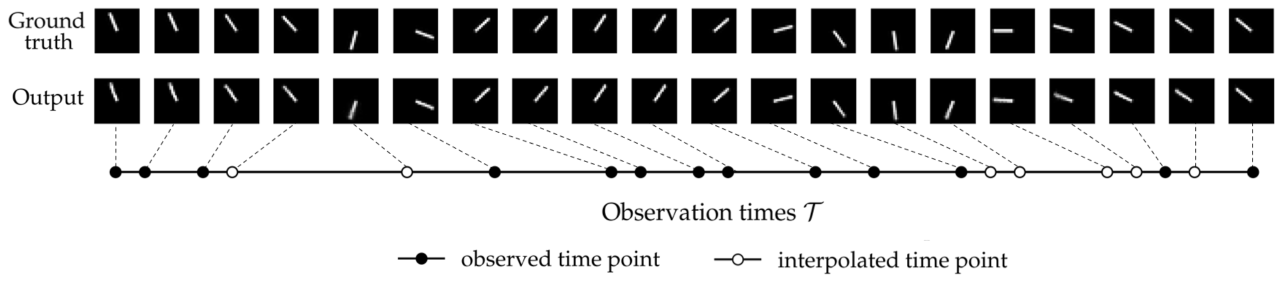

Figure 6. f-CRU

Figure 7. Test trajectory for the pendulum interpolation task: The f-CRU predicts images precisely despite irregular intervals between image frames.