A Theoretical Motivation

A.1 Remarks on general linear ODE

Note that by famed Picard-Lindelöf theorem, solutions to (1) exist on an interval $[t_0 - \varepsilon, t_0 + \varepsilon]$ and are unique, given that f is continuous in t and Lipschitz-continuous in x. Even then, to get *global existence*, which is a desirable property for a forecasting model, is difficult to achieve for a general vector field. The following example shows that even very well behaved functions can lead to a finite time blow-up.

Example 1 (Finite time blow up of an ODE)**.** The ODE $\dot{x}(t) = x(t)^2$ with initial condition $x(0) = x_0$ has the solution $x(t) = \frac{x_0}{1 - t \cdot x_0}$, which blows up in finite time $t = \frac{1}{x_0}$.

On the contrary, *linear* ODEs enjoy very generous global existence and uniqueness properties.

Theorem 1 (Existence & Uniqueness of Linear ODEs)**.** *If A and b are continuous in t, then the linear ODE* (6) *has a unique global solution. Teschl (2012)*

General linear ODEs are not trivial to work with, hence we will restrict ourselves to linear ODEs when the system matrix A is constant. In this case, the solution can be expressed as

Lemma 5. *Given a linear ODE* (2) *with A constant, the solution can be expressed as*

$$x(t + \Delta t) = e^{A\Delta t} \cdot \left(x(t) + \int_0^{\Delta t} e^{-A\Delta s} x(t + \Delta s) \, \mathrm{d}s \right) \tag{12}$$

In the case when b is a polynomial function of t, one can get completely rid of the integral using the so called φ-functions, related to the *exponential integral*, which are given recursively as Al-Mohy & Higham (2011)

$$\varphi_k(x) = x \cdot \varphi_{k+1}(x) + \frac{1}{k!} \qquad \varphi_0(x) = e^x \tag{13}$$

$$x(t + \Delta t) = e^{A\Delta t} \cdot x(t) + \sum_{k=1}^{p} \varphi_k(A\Delta t) u_k \Delta t^k \qquad u_k = \left. \frac{\mathrm{d}^{k-1} b(s)}{\mathrm{d}s^{k-1}} \right|_{s=t} \tag{14}$$

Secondly, note that using *Saad's trick*, and Al-Mohy and Higham's extension, the φ-functions can by computed through the matrix exponential of a larger block matrix, for instance:

$$\exp\left(\begin{bmatrix} A & b \\ 0 & 0 \end{bmatrix} \right) = \begin{bmatrix} e^A & \varphi_1(A)b \end{bmatrix} \tag{15}$$

And similarly (14) can be computed through the matrix exponential of a $(n+p) \times (n+p)$ matrix. This result further encourages us to only consider the most simplest form of linear differential equation, the homogeneous case with constant coefficient $\dot{x}(t) = A \cdot x(t)$, since the polynomial case is covered by simply embedding into a larger dimensional latent space.

B Proofs and Additional Model Properties

Proof of Lemma 2. First, note that if X is skew-symmetric, then e^{Xt} is orthogonal. This is because in this case X commutes with $Y = X^\top$, and so $e^{X+Y} = e^X e^Y$ holds. But then $(e^X)^\top e^X = e^{X^\top} e^X = e^{X + X^\top} = e^{-X^\top + X^\top} = e^{\mathbf{0}} = \mathbb{I}$. So let $Q = e^{Ct}$ be orthogonal. Assuming $\mathbf{E}[x] = \mathbf{0}$ and $\mathrm{Cov}[x] = \mathbb{I}$ we have

$$\begin{aligned} \mathbf{E}[Qx] &= Q \cdot \mathbf{E}[x] = Q \cdot \mathbf{0} = \mathbf{0} = \mathbf{E}[x] \\ \mathrm{Cov}[Qx] &= Q \cdot \mathrm{Cov}[x] \cdot Q^\top = Q \cdot \mathbb{I} \cdot Q^\top = Q \cdot Q^\top = \mathbb{I} = \mathrm{Cov}[x] \end{aligned}$$

□

Conjecture 1 (Self stabilizing property of large matrix exponential). We made experimentally the following observation. Assume A is random $n \times n$ matrix with component distributed iid. as $A_{ij} \sim \mathcal{N}(0, \sigma^2)$. Then

$$\forall i \neq j : \lim_{n \to \infty} \mathrm{Var}[(e^{A - A^\top})_{ij}] = \tfrac{1}{\sqrt{n}}$$

Proof of Self-consistency (1 and Corollary 1)

First, note that by definition the system component represents a *dynamical system* if and only if

$$\text{System}(0,\, z_t) = z_t \quad \text{and} \quad \text{System}(\Delta t' + \Delta t,\, z_t) = \text{System}(\Delta t',\, \text{System}(\Delta t,\, z_t)) \tag{16}$$

Now, assume we have a forecasting model $\hat{y}(t \mid D)$ and we add the prediction of the model $\hat{y}^* \coloneqq \hat{y}(t^* \mid D)$ at time t^* to the dataset: $\hat{D} = D \cup \{(t^*,\, \hat{y}^*)\}$.

Let $t^- = \max\{t_i \mid t_i \leq t^*\}$ such that $t^- \leq t^*$ with no other datapoint in-between. Given $\hat{y}^- = \hat{y}(t^- \mid D)$, it suffices to show that $\hat{y}(t \mid \{(t^-, y^-), (t^*, y^*)\}) = \hat{y}(t \mid \{(t^-, y^-)\}$ for all $t \geq t^*$.

But since the filter is idempotent, the state estimate does not change when observing (t^*, y^*), and if the encoder is left-inverse to the decoder, also the latent state estimate $\hat{z}$ does not change. Finally, the dynamical system assumption ensures that simply propagating from t^- to t yields the same result as propagating from t^- to t^* and then to t.

B.1 KALMAN FILTER

In the presence of missing values, i.e. when only a subset of the components of y are measured, the same formula holds after substituting $y_t \leftarrow S_t \cdot y_t$, $H_t \leftarrow S_t \cdot H_t$ and $R_t \leftarrow S_t \cdot R_t \cdot S_t^\top$ where S_t is the $m_t \times m$ projection matrix of the missing values. In this case the equation can be simplifies towards

B.2 RELATION OF THE FILTER UPDATE TO GRADIENT DESCENT

Proof of Lemma 3 - KalmanCell initialization. We assume the autoregessive case

$$\hat{x}'_t \leftarrow \hat{x}_t - \alpha \widetilde{B} \Pi_t \widetilde{A} (H \hat{x}_t - x_t^{\text{obs}})$$

Since ε_A and ε_N are initialized with zero, the linear KalmanCell is at the beginning of training equivalent to

$$\hat{x}' \leftarrow \hat{x} - \alpha \Pi \Pi (\hat{x} - x^{\text{obs}}) = (1 - \alpha \Pi)\, \hat{x} + \alpha \Pi x^{\text{obs}} \rightsquigarrow \hat{x}'_i = \begin{cases} \hat{x}: & \alpha = 0 \\ x^{\text{obs}}: & \alpha = 1 \wedge x^{\text{obs}} \neq \texttt{NaN} \end{cases}$$

For the case $\alpha = 1/2$, consider again the classical Kalman Filter:

$$\hat{x}' = \hat{x} - \Sigma H^\top \Pi^\top (H \Sigma H^\top + R)^{-1} \Pi (H\, \hat{x} - x^{\text{obs}})$$

Then, in the special case when the problem is autoregressive ($H = \mathbb{I}$), without missing values ($\Pi = \mathbb{I}$) and $\Sigma = R$, the formula reduces to:

$$\hat{x}' = \hat{x} - \Sigma \Pi^\top (\Sigma + R)^{-1} \Pi (\hat{x} - x^{\text{obs}}) = \hat{x} - \tfrac{1}{2} \Sigma \Pi^\top (\Sigma)^{-1} \Pi (\hat{x} - x^{\text{obs}}) = \hat{x} - \tfrac{1}{2} (\hat{x} - x^{\text{obs}})$$

□

Remark 1 (Filter is a Gradient update). Recall that the Kalman filter update, ignoring missing values, looks like

$$x'_t \leftarrow x_t - \eta f (x_t - x_t^{\text{obs}}) \tag{17}$$

Which is surprisingly similar to a gradient descent update. In fact, we can interpret it as such! Recall the standard Kalman Filter update

$$x' = x - P H^\top (H P H^\top + R)^{-1} (H x - y) \tag{18}$$

$$= x - P \frac{\mathrm{d}}{\mathrm{d}x} \frac{1}{2} \left\| (H P H^\top + R)^{-\frac{1}{2}} (H x - y) \right\|^2 \tag{19}$$

This was noted for example by Baltieri & Isomura (2021); Ollivier (2019)

B.3 Row- and Column correlation

As an alternative to the effective rank measure, we found that the average row-/column-correlation allows for similar observations.

Definition 6 (Row and Column correlation)**.** Let X be a real $m \times n$ matrix. We define the *average row/column correlation coefficient as*

$$\text{col-corr}(X) \coloneqq \frac{1}{n(n-1)} \sum_{ij} \left| \delta_{ij} - \frac{\langle X_{i,:} | X_{j,:} \rangle}{\|X_{i,:}\| \cdot \|X_{j,:}\|} \right| = \frac{1}{n(n-1)} \left\| \mathbb{I}_n - \frac{X^\top X}{\text{diag}(X^\top X) \otimes \text{diag}(X^\top X)} \right\|_{1,1}$$

$$\text{row-corr}(X) \coloneqq \frac{1}{m(m-1)} \sum_{ij} \left| \delta_{ij} - \frac{\langle X_{:,i} | X_{:,j} \rangle}{\|X_{:,i}\| \cdot \|X_{:,j}\|} \right| = \frac{1}{m(m-1)} \left\| \mathbb{I}_m - \frac{XX^\top}{\text{diag}(XX^\top) \otimes \text{diag}(XX^\top)} \right\|_{1,1}$$

Lemma 6. *Let ρ denote the row-/column-correlation. Then $0 \leq \text{col-corr}(X) \leq 1$, with $\rho(X) = 0$ if and only if $X^\top X = \mathbb{I}_n$ ($XX^\top = \mathbb{I}_m$) and $\rho(X) = 1$ if and only if all rows/columns of X are linearly dependent.*

C Implementation Details

C.1 Encoder Component

The encoding mechanism is responsible for transforming the input in a non-linear way from the observation space of size $\dim(x)$ into the latent space of size $\dim(z)$. the dimensionality from the input size $\dim(x)$ to the latent size $\dim(z)$. We will assume that $\dim(z) \geq \dim(x)$ and split the component into two parts:

$$\text{Encoder:} \quad x \mapsto \phi(\text{embedding}(x)) \qquad \text{Embedding:} \quad x \mapsto \text{concatenate}(x,\, b) \tag{20a}$$

$$\text{Decoder:} \quad z \mapsto \text{projection}(\pi(z)) \qquad \text{Projection:} \quad z \mapsto [\mathbb{I} \quad 0]\, z \tag{20b}$$

Where ϕ and π are neural networks mapping $\mathbb{R}^{\dim(z)} \to \mathbb{R}^{\dim(z)}$. One option is to use an invertible architecture such as the invertible Residual Network (iResNet) by Behrmann et al. (2019) or the invertible Attention layer by Zha et al. (2021). This would allow the encoder to be left-inverse to the decoder with in the case $\dim(z) \geq \dim(x)$. However we found that in our setup it was sufficient to use two independent ReZero-ResNets.

For the embedding, we propose to simply concatenate a learnable vector to the input. An alternative that also works when $\dim(z) < \dim(x)$ is to multiply by a weight matrix A in the embedding, and then by the pseudoinverse A^+ in the projection.

C.2 Pseudo-Code

The following is a close to source pseudo-code of the KalmanCell that works even when x^{obs} contains `NaN` values. The trick is to, instead of multiplying my mask matrices Π_t, to use $\text{where}(\text{mask}, \text{value}, \text{other})$ operators that select entries from one tensor or the other based on a condition.

Algorithm 4 Linear KalmanCell

Input: Current State estimate $\hat{x}_t \in \mathbb{R}^n$, observed datapoint $x_t^{\text{obs}} \in (\mathbb{R} \cup \{\texttt{NaN}\})^m$
Parameters: Learnable matrices A, B, H, zero-initialized scalars $\varepsilon_A, \varepsilon_B$, scalar α, parametrizations ψ_A, ψ_B.
Options: If autoregressive, $m = n$ and $H = \mathbb{I}_n$.

$m_t \leftarrow \text{not-missing}(x_t^{\text{obs}})$
$\widetilde{A} \leftarrow \mathbb{I}_n + \varepsilon_A \psi_A(A)$
$\widetilde{B} \leftarrow \mathbb{I}_m + \varepsilon_B \psi_B(A)$
$r_t \leftarrow \widetilde{A} \cdot \text{where}(m_t, H\hat{x}_t - x_t^{\text{obs}}, \mathbf{0})$
$\Delta x_t \leftarrow \widetilde{B} H^\top \cdot \text{where}(m_t, r_t, \mathbf{0})$
$\hat{x}'_t \leftarrow \hat{x}_t - \alpha \Delta x_t$
return Updated state estimate $\hat{x}'_t$.

Algorithm 5 Non-Linear KalmanCell

Input: Current State estimate $\hat{x}_t \in \mathbb{R}^n$, observed datapoint $x_t^{\text{obs}} \in (\mathbb{R} \cup \{\texttt{NaN}\})^m$
Parameters: Learnable matrices A, B, H zero-initialized scalar ε, neural network ϕ.
Options: If autoregressive, $m = n$ and $H = \mathbb{I}_n$.

$m_t \leftarrow \text{not-missing}(x_t^{\text{obs}})$
$r_t \leftarrow A \cdot \text{where}(m_t, H\hat{x}_t - x_t^{\text{obs}}, \mathbf{0})$
$z_t \leftarrow BH^\top \cdot \text{where}(m_t, r_t, \mathbf{0})$
$\Delta x_t \leftarrow \phi(z_t)$
$\hat{x}_t' \leftarrow \hat{x}_t - \varepsilon \Delta x_t$
Return: Updated state estimate $\hat{x}_t'$.

Algorithm 6 Encoder

Input: state estimate $\hat{x}_t$
Parameters: zero-initialized scalars ε_k, neural network blocks F_k

$u \leftarrow [\hat{x}_t, c]$
for $k = 1 \dots K$ **do**
    $u \leftarrow u + \varepsilon_k \cdot F_k(u)$
end for
Return: $\hat{z}_t \coloneqq u_K$

Algorithm 7 Decoder

Input: latent state estimate $\hat{z}_t$
Parameters: zero-initialized scalars ε_k, neural network blocks F_k

$u \leftarrow \hat{z}_t$
for $k = 1 \dots K$ **do**
    $u \leftarrow u + \varepsilon_k \cdot F_k(u)$
end for
$\hat{x}_t' \leftarrow [\mathbb{I}, \mathbf{0}]u$
Return: $\hat{x}_t'$

Algorithm 8 Stacked Filter

Input: Current State estimate $\hat{x}_t \in \mathbb{R}^n$, observed datapoint $x_t^{\text{obs}} \in \mathbb{R}^m$.
Parameters: Filters f_i, $i = 1 \dots m$

for $i = 1 \dots m$ **do**
    $\hat{x}_t \leftarrow f_i(x_t^{\text{obs}}, \hat{x}_t)$
end for
Return: Updated state estimate $\hat{x}_t$.

D Further Related Work

D.1 RNN based models

Recurrent Neural Networks (Rumelhart et al., 1986) are one of the oldest architectures used for sequential data. While the original formulation notoriously suffer from error propagation and instability, variants such as Long-Short-Term-Memory (LSTM; (Hochreiter & Schmidhuber, 1997)) and the simplified Gated Recurrent Units (GRU; Cho et al. (2014)) were developed in order to facilitate long-term forecasting. However, nowadays these methods often cannot compete in forecasting tasks with attention-based methods. In this domain, in particular a GRU variant with decay parameters

(GRU-D; Che et al. (2018)) laid an important foundation for applying RNNs to irregularly sampled data. This model can be considered as a special case of a neural ODE (cf. Table 1).

RNNs have been extended to probabilistic forecasting, e.g., Deep State Space Models (DeepState; Rangapuram et al. (2018)), and exploit autoregressive characteristics of time series data, e.g., Deep Autoregressive Recurrent Neural Networks (DeepAR; Salinas et al. (2020)).

On the other hand RNNs, still play an important role in time-series imputation, especially for irregularly sampled time series. This task consists of reconstructing missing values inside an interval of observed data. For this task, particularly multidirectional recurrent methods have been successful such as Multidirectional RNN (M-RNN; Yoon et al. (2019)) Bidirectional Recurrent Imputation (BRITS; Cao et al. (2018)) and Interpolation-Prediction Networks (IP-Nets; Shukla & Marlin (2018)).

D.2 Attention based models

The attention mechanism (Vaswani et al., 2017), first introduced in the context of natural language processing, has been successfully applied in the time series domain. One of the main advantages of attention is the short path-length for interactions, whereas one of the drawbacks is scalability with respect to the sequence length, which makes regular transformers not usable for time-series tasks with large observational horizon.

To counteract that different solutions have been proposed such as using sparse attention variants such as implemented in the Log-Sparse Transformer (LogTrans; Li et al. (2019)), Adversarial Sparse Transfomer (AST; Wu et al. (2020)) and Informer (Zhou et al., 2021) models.

Meanwhile, transformer based models are well established for regular time series tasks. Temporal Pattern Attention (TPA; Shih et al. (2019)) uses 1-D convolutions in order to embed the observations into a latent state on which feature-wise attention extracts inter-channel dependencies and updates the latent state accordingly. An LSTM model is used to produce the forecast. Multi-Horizon Temporal Attention (MHTA; Fan et al. (2019)) combines an an attention based encoder with a bidirectional LSTM decoder in order to facilitate probabilistic forecasting on regular time-series data. Topological Attention (TopAttn; Zeng et al. (2021)) uses methods from topological data analysis in order to extract feature embeddings used as input for a transformer model. Temporal Fusion Transformer (TFT; Lim et al. (2021)) combines a variable selection model and a LSTM encoder-decoder architecture with a latent Multi-Head Attention Mechanism in order to produce Multi-Horizon Quantile forecasts. Multi-Time Attention (mTAN; Shukla & Marlin (2021)) is one of the few transformer models specifically applied to irregularly sampled time-series, albeit being only used for interpolation and classification. The model makes use sinusoidal time embeddings, which are used via an attention mechanism in order to realize a kernel-smoother.

However, to the best of our knowledge none of the attention based models have been used for irregularly sampled time series forecasting.

D.3 Convolution based models

Time series data, in particular regularly sampled time series data, naturally lends itself the application of convolutions. Temporal Convolutional Networks (TCN; Bai et al. (2018a)) apply channel wise convolutions in order to obtain temporal embeddings.

For classification tasks, Trellis Networks (Bai et al., 2018b) use stacked, weight-shared temporal convolutions and achied good results on natural language tasks.

More recently, Continuous Kernel Convolutions (CKConv; Romero et al. (2021)) use an MLP in order to parametrize a continuous convolutional kernel, which allows the convolution to be used with irregularly sampled data. However, the model is only usable for classification and regression tasks.

D.4 VAE and GAN based models

Partial Bidirectional GAN (P-BiGAN; Li & Marlin (2020)) is an encoder-decoder architecture using continuous convolutions in the encoder and a kernel smoother as the decoder which are trained in a bidirectional GAN. They evaluate the model on image reconstruction and time series classification tasks. STRIPE (LE GUEN & THOME, 2020) is a probabilistic time series forecasting model based on

a sequence to sequence architecture that utilizes temporal point processes and variants of dynamical time warping.

D.5 Other

Autoregressive Integrated Moving Average (ARIMA; Box et al. (1974), Box et al. (1974) Box et al. (1974)) is one of the oldest time-series forecasting models. Neural Basis Expansion Analysis for Time Series (N-BEATS; Oreshkin et al. (2019)) and its successor Neural Hierarchical Interpolation (N-HITS; Challu et al. (2022)) are two time series forecasting models based on the idea of combining pre-defined or learned basis functions with a residual architecture. Set Functions for Time Series (SetTS; Horn et al. (2020)) represent a time series as a set of triplets consisting of observation time, channel indicator and observed value. This naturally removes any missing and allows them to directly apply DeepSet-type model (Zaheer et al., 2017). The result is a scalable model for time-series regression and classification tasks.

Neural Flows (NFLOWs; Biloš et al. (2021)) try to directly learn a model representing the solution of an ODE instead of the modeling the ODE itself, which completely removes the need for an ODE integrator.

E Experiment Details

E.1 Dataset Statistics

Table 4: Dataset Statistics

	USHCN	MIMIC-III	MIMIC-IV
observation horizon	36 hours	36 hours	36 hours
forecasting horizon	3 steps	3 steps	3 steps
data-split (train/val/test)%	72/18/10	72/18/10	68/17/15
cross-validation folds	5	5	5
number of timesteps	350,665	552,327	2,485,769
number of channels	5	96	102
missing percentage	77.9%	94.2%	97.8%
Δt range (min/median/max)	0.1/0.4/137	1/1/80	1/9/2848

E.2 Hyperparameter choices

Table 5: Hyperparameter Grid

	USHCN	MIMIC-III	MIMIC-IV
max epochs	$\{100\}$	$\{25\}$	$\{25\}$
batch-size	$\{64\}$	$\{64\}$	$\{64\}$
hidden-size	$\{64, 128\}$	$\{64, 128\}$	$\{64, 128\}$
latent-size	$\{128, 192\}$	$\{128, 192\}$	$\{128, 192\}$
kernel-init	{skew-symmetric}	{skew-symmetric}	{skew-symmetric}
encoder	5-block ResNet with 2 RELU pre-activated layers each		
filter	StackedFilter of 3 KalmanCells, the first is linear		
system	LinODECell with skew-symmetric initialization		
total parameters	97k-357k-	510k-704k	510k-704k
learning-rate	$\{0.001\}$	$\{0.001\}$	$\{0.001\}$
beta-parameters	$\{(0.9, 0.999)\}$	$\{(0.9, 0.999)\}$	$\{(0.9, 0.999)\}$
weight-decay	$\{0.001\}$	$\{0.001\}$	$\{0.001\}$
GPU	RTX 3090 / A400	RTX 3090 / A4000	RTX 3090 / A4000