Temporal Graph Neural Networks for Irregular Data

Joel Oskarsson
Linköping University

Per Sidén
Linköping University
Arriver Software AB

Fredrik Lindsten
Linköping University

Abstract

This paper proposes a temporal graph neural network model for forecasting of graph-structured irregularly observed time series. Our TGNN4I model is designed to handle both irregular time steps and partial observations of the graph. This is achieved by introducing a time-continuous latent state in each node, following a linear Ordinary Differential Equation (ODE) defined by the output of a Gated Recurrent Unit (GRU). The ODE has an explicit solution as a combination of exponential decay and periodic dynamics. Observations in the graph neighborhood are taken into account by integrating graph neural network layers in both the GRU state update and predictive model. The time-continuous dynamics additionally enable the model to make predictions at arbitrary time steps. We propose a loss function that leverages this and allows for training the model for forecasting over different time horizons. Experiments on simulated data and real-world data from traffic and climate modeling validate the usefulness of both the graph structure and time-continuous dynamics in settings with irregular observations.

1 INTRODUCTION

Many real-world systems can be modeled as graphs. When data about such systems is collected over time, the resulting time series has additional structure induced by the graph topology. Examples of such temporal graph data is the traffic speed in the road network (Li et al., 2018) and the spread of disease in different regions (Rozemberczki et al., 2021). Building accurate machine learning models in this setting requires taking both the temporal and graph structure into account.

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206. Copyright 2023 by the author(s).

While many works have studied the problem of modeling temporal graph data (Wu et al., 2020a), these approaches generally assume a constant sampling rate and no missing observations. In real data it is not uncommon to have irregular or missing observations due to non-synchronous measurements or errors in the data collection process. Dealing with such irregularities is especially challenging in the graph setting, as node observations are heavily interdependent. While observations in one node could be modeled using existing approaches for irregular time series (Rubanova et al., 2019; Schirmer et al., 2022), the situation becomes complicated when irregular observations in different nodes occur at different times.

In this paper we tackle two kinds of irregular observations in graph-structured time series: (1) irregularly spaced observation times, and (2) only a subset of nodes being observed at each time point. We propose the TGNN4I model for time series forecasting. The model uses a time-continuous latent state in each node, which allows for predictions to be made at any time point. The latent dynamics are motivated by a linear Ordinary Differential Equation (ODE) formulation, which has a closed form solution. This ODE solution corresponds to an exponential decay (Che et al., 2018) together with an optional periodic component. New observations are incorporated into the state by a Gated Recurrent Unit (GRU) (Cho et al., 2014). Interactions between the nodes are captured by integrating Graph Neural Network (GNN) layers both in the latent state updates and predictive model. To train our model we introduce a loss function that takes into account the time-continuous model formulation and irregularity in the data. We evaluate the model on forecasting problems using traffic and climate data of varying degrees of irregularity and a simulated dataset of periodic signals.

2 RELATED WORK

GNNs are deep learning models for graph-structured data (Gilmer et al., 2017; Wu et al., 2020a). By learning representations of nodes, edges or entire graphs GNNs can be used for many different machine learning tasks. These models have been successfully applied to diverse areas such as weather forecasting (Lam et al., 2022) molecule generation (Zang and Wang, 2020) and video classification (Kosman and Di Castro, 2022). Temporal GNNs model also time-varying signals in the graph. This extension to graph-structured time series is achieved by combining GNN layers with recurrent (Li et al., 2018), convolutional (Wu et al., 2019; Yu et al., 2018) or attention (Guo et al., 2019) architectures. While the graph is commonly assumed to be known a priori, some approaches also explore learning the graph structure jointly with the temporal GNN model (Zhang et al., 2022; Wu et al., 2020b).

Time series forecasting is a well-studied problem and a vast amount of methods exist in the literature. Traditional methods in the area include ARIMA models, vector auto-regression and Gaussian Processes (Box et al., 2015; Roberts et al., 2013). Many deep learning approaches have also been applied to time series forecasting. This includes Recurrent Neural Networks (RNNs) (Lazzeri, 2020), temporal convolutional neural networks (Chen et al., 2020) and Transformers (Giuliari et al., 2021).

The latent state of an RNN can be extended to continuous time by letting the state decay exponentially in between observations (Che et al., 2018). Such decay mechanisms have been used for modeling data with missing observations (Che et al., 2018), doing imputation (Cao et al., 2018) and parametrizing point processes (Mei and Eisner, 2017). Another way to define time-continuous states is by learning a more general ODE. In neural ODE models (Chen et al., 2018; Kidger, 2021) the latent state is the solution to an ODE defined by a neural network. Neural ODEs have been successfully applied to irregular time series (Rubanova et al., 2019) and can be used to define the dynamics of temporal GNNs (Fang et al., 2021; Poli et al., 2021). Poli et al. (2021) use such a model for graph-structured time series with irregular time steps, but consider the full graph to be observed at each observation time. Also the GraphCON framework of Rusch et al. (2022) combines GNNs with a second order system of ODEs. In GraphCON the time-axis of the ODE is however aligned with the layers of the GNN. This makes the framework suitable for node- and graph-level predictive tasks, rather than time-series modeling.

Another related body of work is concerned with using GNNs for data imputation in time series (Cini et al., 2022; Gordon et al., 2021; Omidshafiei et al., 2022). These methods generally do not assume that the time series come with some known graph structure. Instead, the GNN is defined on some graph specifically constructed for the purpose of performing imputation.

The closest work to ours found in the literature is the LG-ODE model of Huang et al. (2020). They consider the same types of irregularities, but are motivated more by a multi-agent systems perspective. LG-ODE is based on an encoder-decoder architecture and trained by maximizing the Evidence Lower Bound (ELBO). The encoder builds a spatio-temporal graph of observations and aggregates information using an attention mechanism (Vaswani et al., 2017). The decoder then extrapolates to future times by solving a neural ODE. The encoder-decoder setup differs from our TGNN4I model that sequentially incorporates observations and auto-regressively makes predictions at every time point. Because of the multi-agent motivation Huang et al. (2020) are also more focused on smaller graphs with few interacting entities, but longer forecasting horizons. An extended version of LG-ODE, called CG-ODE (Huang et al., 2021), also aims to learn the graph structure in the form of a weighted adjacency matrix.

3 A TEMPORAL GNN FOR IRREGULAR OBSERVATIONS

3.1 Setting

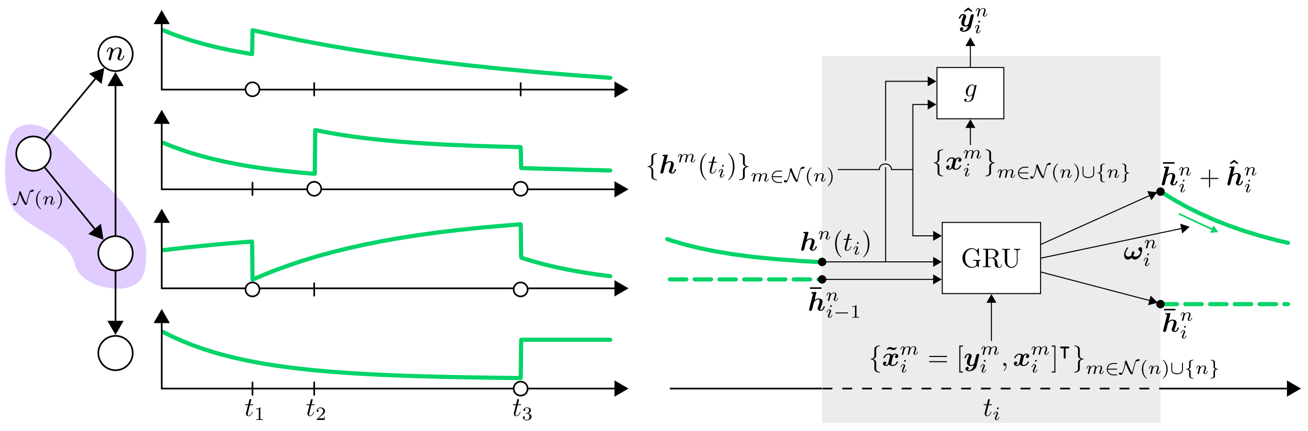
Consider a directed or undirected graph $\mathcal{G} = (V, E)$ with node set V and edge set E. Let $\{t_i\}_{i=1}^{N_t}$ be a set of (possibly irregular) time points s.t. $0 < t_1 < \cdots < t_{N_t}$. We will here present our model for a single time series, but in general we have a dataset containing multiple time series. Let $\mathcal{O}_i \subseteq V$ be the set of nodes observed at time t_i. If $n \in \mathcal{O}_i$, we denote the observed value as $\boldsymbol{y}_i^n \in \mathbb{R}^{d_y}$ and any accompanying input features as $\boldsymbol{x}_i^n \in \mathbb{R}^{d_x}$. We let $\boldsymbol{y}_i^n = \boldsymbol{x}_i^n = \boldsymbol{0}$ if $n \notin \mathcal{O}_i$. Note that this general setting encompasses a spectrum of irregularity, from single node observations ($|\mathcal{O}_i| = 1 \; \forall i$) to fully observed graphs ($\mathcal{O}_i = V \; \forall i$). A table of notation is given in appendix A.

The problem we consider is that of forecasting. At future time points we want to predict the value at each node, given all earlier observations. Since observations are irregular and we want to make predictions at arbitrary times, we need to consider models that can make predictions for any time in the future.

We consider a model where at each node n a latent state $\boldsymbol{h}^n(t) \in \mathbb{R}^{d_h}$ evolves over continuous time. We define the dynamics of $\boldsymbol{h}^n(t)$ by: (1) how $\boldsymbol{h}^n(t)$ evolves in between observations, and (2) how $\boldsymbol{h}^n(t)$ is updated when node n is observed. If node n is observed at time t_i we incorporate this observation into the latent state using a GRU cell (Cho et al., 2014). This information can then be used for making predictions at future time points. An overview of our model is given in Figure 1.

3.2 Time-continuous Latent Dynamics

Consider a time interval $]t_i, t_j]$ where node n is not observed. During this interval we define the latent state of node n by the sum $\boldsymbol{h}^n(t) = \bar{\boldsymbol{h}}_i^n + \tilde{\boldsymbol{h}}^n(t)$. The first part $\bar{\boldsymbol{h}}_i^n$ is constant over the time interval, constituting a base level around which the state evolves. The dynamics of $\tilde{\boldsymbol{h}}^n(t)$ are



n
$\mathcal{N}(n)$
t_1
t_2
t_3
$\hat{\boldsymbol{y}}_i^n$
g
$\{\boldsymbol{h}^m(t_i)\}_{m\in\mathcal{N}(n)}$
$\{\boldsymbol{x}_i^m\}_{m\in\mathcal{N}(n)\cup\{n\}}$
$\bar{\boldsymbol{h}}_i^n + \hat{\boldsymbol{h}}_i^n$
$\boldsymbol{h}^n(t_i)$
GRU
$\boldsymbol{\omega}_i^n$
$\bar{\boldsymbol{h}}_{i-1}^n$
$\bar{\boldsymbol{h}}_i^n$
$\{\tilde{\boldsymbol{x}}_i^m = [\boldsymbol{y}_i^m, \boldsymbol{x}_i^m]^\intercal\}_{m\in\mathcal{N}(n)\cup\{n\}}$
t_i

Figure 1: **Left:** Example graph with four nodes and their latent states, here one-dimensional for illustration purposes. Node observations are indicated with ◯. **Right:** Schematic diagram of the GRU update and predictive model g. Everything inside the shaded area happens instantaneously at time t_i. The GRU cell outputs the new initial state $\bar{\boldsymbol{h}}_i^n + \hat{\boldsymbol{h}}_i^n$, the static component $\bar{\boldsymbol{h}}_i^n$ and the ODE parameters $\boldsymbol{\omega}_i^n$. The parameters $\boldsymbol{\omega}_i^n$ define the dynamics until the next observation. The prediction $\hat{\boldsymbol{y}}_i^n$ here is based on information from all earlier time points.

dictated by a linear ODE of the form

$$d\tilde{\boldsymbol{h}}^n(t) = A\tilde{\boldsymbol{h}}^n(t)\,dt \tag{1}$$

with $A \in \mathbb{R}^{d_h \times d_h}$ and initial condition $\tilde{\boldsymbol{h}}^n(t_i) = \hat{\boldsymbol{h}}_i^n$. Over this interval the ODE has a closed form solution (Arrowsmith and Place, 1992) given by

$$\tilde{\boldsymbol{h}}^n(t) = \exp(\delta_t A)\hat{\boldsymbol{h}}_i^n \tag{2}$$

where exp is the matrix exponential function and we define $\delta_t = t - t_i$. Assuming that all eigenvalues of A are unique, we can use its eigen-decomposition[1] $A = Q\Lambda Q^{-1}$ to write

$$\tilde{\boldsymbol{h}}^n(t) = Q\exp(\delta_t\Lambda)Q^{-1}\hat{\boldsymbol{h}}_i^n. \tag{3}$$

Since Q contains an eigen-basis of $\mathbb{R}^{d_h}$ it can be viewed as a transition matrix, changing the basis of the latent space. While we could in principle learn Q, we note that the basis of the latent space has no physical interpretation and we can without loss of generality choose it such that $Q = I$.

Next, if we make the assumption that A has real eigenvalues the resulting dynamics are given by

$$\tilde{\boldsymbol{h}}^n(t) = \exp(-\delta_t\,\mathrm{diag}(\boldsymbol{\omega}_i^n))\hat{\boldsymbol{h}}_i^n \tag{4}$$

where $\boldsymbol{\omega}_i^n > 0$ is a parameter vector representing the negation of the eigenvalues. We arrive at an exponential decay in-between observations, a type of dynamics used with GRU-updates in existing works (Che et al., 2018; Cao et al., 2018). The positive restriction on $\boldsymbol{\omega}_i^n$ ensures the stability of the dynamical system in the limit, which for the complete state means that $\boldsymbol{h}^n(t_j) \to \bar{\boldsymbol{h}}_i^n$ as $t_j \to \infty$.

On the other hand, if we instead allow A to have complex eigenvalues we can decompose A using the real Jordan form (Horn and Johnson, 2012). This makes Λ block-diagonal with 2×2 blocks

$$C_j = \begin{bmatrix} a_j & -b_j \\ b_j & a_j \end{bmatrix} \tag{5}$$

corresponding to complex conjugate eigenvalues $a_j \pm b_j i$. If we compute the matrix exponential for this Λ we end up with a combination of exponential decay and periodic dynamics (Arrowsmith and Place, 1992). The solution gives dynamics that couple each pair of dimensions as

$$\begin{aligned}\left[\tilde{\boldsymbol{h}}^n(t)\right]_{j:j+1} &= \exp(\delta_t a_j)\times \\ &\begin{bmatrix} \cos(b_j\delta_t) & -\sin(b_j\delta_t) \\ \sin(b_j\delta_t) & \cos(b_j\delta_t) \end{bmatrix}\left[\hat{\boldsymbol{h}}_i^n\right]_{j:j+1}.\end{aligned} \tag{6}$$

We parametrize also these dynamics with $\boldsymbol{\omega}_i^n = [-a_1, -a_3, \ldots, -a_{d_h-1}, b_1, b_3, \ldots, b_{d_h-1}]^\intercal > 0$. Note that when $b_j \to 0$ these periodic dynamics reduce to the exponential decay in Eq. 4, but with the parameter for $-a_j$ shared across pairs of dimensions. We consider both the exponential decay and the more general periodic dynamics as options for our model. The periodic dynamics can naturally be seen as advantageous for modeling periodic data, as we will explore empirically in section 4.4.

3.3 Incorporating Observations from the Graph

When node n is observed at time t_i the observation is incorporated into the latent state by a GRU cell, incurring an

[1]Recall that the eigen-decomposition of a diagonalizable matrix A is given by $A = Q\Lambda Q^{-1}$, where Λ is a diagonal matrix containing the eigenvalues of A and the columns of Q are the corresponding eigenvectors (Searle and Khuri, 1982).

instantaneous jump in the latent dynamics to a new value $\bar{\boldsymbol{h}}_i^n + \hat{\boldsymbol{h}}_i^n$. Inspired by the continuous-time LSTM of Mei and Eisner (2017), we extend the GRU cell to output also the parameters $\boldsymbol{\omega}_i^n$, which define the dynamics of $\tilde{\boldsymbol{h}}^n(n)$ for the next time interval.

So far we have considered each node of the graph as separate entities, but in a graph-based system future observations of a node can depend on the history of the entire graph. To capture this we let the state of each node depend on observations and states in its graph neighborhood. This is achieved by introducing GNNs (Gilmer et al., 2017; Wu et al., 2020a) in the GRU update (Zhao et al., 2018). We replace matrix multiplications with GNNs, taking inputs both from the node n itself and from its neighborhood $\mathcal{N}(n) = \{m | (m,n) \in E\}$. The type of GNN we use is a simple version of a message passing neural network (Gilmer et al., 2017), defined as

$$\begin{aligned}&\mathrm{GNN}\Big(\boldsymbol{h}^n, \{\boldsymbol{h}^m\}_{m\in\mathcal{N}(n)}\Big)\\ &\quad= W_1\boldsymbol{h}^n + \frac{1}{|\mathcal{N}(n)|}\sum_{m\in\mathcal{N}(n)} e_{m,n} W_2 \boldsymbol{h}^m\end{aligned} \tag{7}$$

where the matrices W_1, W_2 are learnable parameters shared among all nodes and $e_{m,n}$ is an edge weight associated with the edge (m,n). The use of edge weights allows for incorporating prior information about the strength of connections in the graph. We can additionally stack multiple such GNN layers, append fully-connected layers and include non-linear activation functions in between. The inclusion of multiple GNN layers makes the GRU update dependent on a larger graph neighborhood.

In a standard GRU cell the input and previous state are first mapped to three new representations using matrices U and W (Cho et al., 2014). These representations are then used to compute the state update. In our GNN-based GRU update the matrices are replaced by GNNs and we require seven such intermediate representations to update $\hat{\boldsymbol{h}}_i^n$, $\bar{\boldsymbol{h}}_i^n$ and $\boldsymbol{\omega}_i^n$, as shown below. The node states are combined as

$$[\boldsymbol{u}_{i,1}^n, \dots, \boldsymbol{u}_{i,7}^n]^\intercal = \mathrm{GNN}^U\Big(\boldsymbol{h}^n(t_i), \{\boldsymbol{h}^m(t_i)\}_{m\in\mathcal{N}(n)}\Big) \tag{8}$$

where the resulting vector is split into seven equally sized chunks. Another GNN is then used for the combined observations and input features $\tilde{\boldsymbol{x}}_i^n = [\boldsymbol{y}_i^n, \boldsymbol{x}_i^n]^\intercal$,

$$[\boldsymbol{v}_{i,1}^n, \dots, \boldsymbol{v}_{i,7}^n]^\intercal = \mathrm{GNN}^W\Big(\tilde{\boldsymbol{x}}_i^n, \{\tilde{\boldsymbol{x}}_i^m\}_{m\in\mathcal{N}(n)}\Big). \tag{9}$$

Note that while all nodes might not be observed at time t_i, $\tilde{\boldsymbol{x}}_i^m = \boldsymbol{0}$ for any unobserved m and thus do not contribute to the sum over neighbors in the GNN. With the combined information from the graph neighborhood the full GRU update is computed as

$$\boldsymbol{r}_i^n = \sigma(\boldsymbol{v}_{i,1}^n + \boldsymbol{u}_{i,1}^n + \boldsymbol{b}_1) \tag{10a}$$
$$\boldsymbol{z}_i^n = \sigma(\boldsymbol{v}_{i,2}^n + \boldsymbol{u}_{i,2}^n + \boldsymbol{b}_2) \tag{10b}$$
$$\boldsymbol{q}_i^n = \tanh(\boldsymbol{v}_{i,3}^n + (\boldsymbol{r}_i^n \odot \boldsymbol{u}_{i,3}^n) + \boldsymbol{b}_3) \tag{10c}$$
$$\bar{\boldsymbol{h}}_i^n + \hat{\boldsymbol{h}}_i^n = (\boldsymbol{1} - \boldsymbol{z}_i^n) \odot \boldsymbol{h}^n(t_i) + \boldsymbol{z}_i^n \odot \boldsymbol{q}_i^n \tag{10d}$$

$$\bar{\boldsymbol{r}}_i^n = \sigma(\boldsymbol{v}_{i,4}^n + \boldsymbol{u}_{i,4}^n + \boldsymbol{b}_4) \tag{11a}$$
$$\bar{\boldsymbol{z}}_i^n = \sigma(\boldsymbol{v}_{i,5}^n + \boldsymbol{u}_{i,5}^n + \boldsymbol{b}_5) \tag{11b}$$
$$\bar{\boldsymbol{q}}_i^n = \tanh(\boldsymbol{v}_{i,6}^n + (\bar{\boldsymbol{r}}_i^n \odot \boldsymbol{u}_{i,6}^n) + \boldsymbol{b}_6) \tag{11c}$$
$$\bar{\boldsymbol{h}}_i^n = (\boldsymbol{1} - \bar{\boldsymbol{z}}_i^n) \odot \bar{\boldsymbol{h}}_{i-1}^n + \bar{\boldsymbol{z}}_i^n \odot \bar{\boldsymbol{q}}_i^n \tag{11d}$$

$$\boldsymbol{\omega}_i^n = \log(\boldsymbol{1} + \exp(\boldsymbol{v}_{i,7}^n + \boldsymbol{u}_{i,7}^n + \boldsymbol{b}_7)) \tag{12}$$

where σ is the sigmoid function and $\boldsymbol{b}_1$–$\boldsymbol{b}_7$ learnable bias parameters. In Eq. 11d we let $\bar{\boldsymbol{h}}_k^n = \bar{\boldsymbol{h}}_{k-1}^n$ if $n \notin \mathcal{O}_k$. Eq. 10a–10d correspond to one GRU update, using the decayed state $\boldsymbol{h}^n(t_i)$. Eq. 11a–11d define a separate GRU update, but for the decay target $\bar{\boldsymbol{h}}_i^n$. Note that we get $\hat{\boldsymbol{h}}_i^n$, the initial value of $\tilde{\boldsymbol{h}}^n(t)$, implicitly from the difference between Eq. 10d and Eq. 11d. Finally Eq. 12 computes the parameters $\boldsymbol{\omega}_i^n$ defining the dynamics of $\tilde{\boldsymbol{h}}^n(t)$ up until the next observation of node n. The parameters of the GRU cell are shared for all nodes in the graph. To capture any node-specific properties we parametrize initial states $\boldsymbol{h}^n(0)$ separately for all nodes and learn these jointly with the rest of the model.

3.4 Predictions

The time-continuous dynamics ensure that there is a well-defined latent state $\boldsymbol{h}^n(t)$ in each node at each time point t. The value of the time series can then be predicted at any time by applying a mapping $g\colon \mathbb{R}^{d_h} \to \mathbb{R}^{d_y}$ from this latent state to the prediction $\hat{\boldsymbol{y}}_j^n$.

The addition of GNNs into the GRU update makes the latent state dynamics of each node dependent on historical observations in its neighborhood. However, since the GRU updates happen only when a node is observed, information from observed neighbors might not be incorporated immediately in the latent state. Consider three consecutive time points $t_i < t_{i+1} < t_{i+2}$ s.t. $n \in \mathcal{O}_i$, $n \notin \mathcal{O}_{i+1}$. Then any observation $\boldsymbol{y}_{i+1}^m$ for $m \in \mathcal{O}_{i+1} \cap \mathcal{N}(n)$ will not be taken into account by the model for the prediction $\hat{\boldsymbol{y}}_{i+2}^n$, as that prediction is based on a latent state with only information from time t_i. To remedy this we choose also the predictive model g to contain one or more GNN layers,

$$\hat{\boldsymbol{y}}_j^n = \mathrm{GNN}^g\Big([\boldsymbol{h}^n(t_j), \boldsymbol{x}_j^n]^\intercal, \big\{[\boldsymbol{h}^m(t_j), \boldsymbol{x}_j^m]^\intercal\big\}_{m\in\mathcal{N}(n)}\Big). \tag{13}$$

This way g takes the latent states and input features of the whole neighborhood into account for prediction. We name the full proposed model Temporal Graph Neural Network for Irregular data (**TGNN4I**).

3.5 Loss Function

In order to make predictions for arbitrary future time points we introduce a suitable loss function based on the time-continuous nature of the model. Let $\hat{\boldsymbol{y}}^m_{i\to j}$ be the prediction for node m at time t_j, based on observations of all nodes at times $t \leq t_i$. Define also a time-continuous weighting function $w \colon \mathbb{R}^+ \to \mathbb{R}^+$ and the set $\tau_{m,i} = \{j : m \in \mathcal{O}_j \wedge i < j\}$ containing the indices of all times after t_i where node m is observed. We do not include predictions from the first N_{init} time steps in the loss, treating this as a short warm-up phase. The loss function for one graph-structured time series is then

$$\mathcal{L}_\ell = \frac{1}{N_{\text{obs}}} \sum_{m\in V} \sum_{i=N_{\text{init}}+1}^{N_t} \sum_{j\in\tau_{m,i}} \frac{\ell(\hat{\boldsymbol{y}}^m_{i\to j}, \boldsymbol{y}^m_j) w(t_j - t_i)}{j - N_{\text{init}} - 1} \tag{14}$$

where ℓ is any loss function for a single observation and $N_{\text{obs}} = \sum_{i=N_{\text{init}}+2}^{N_t} |\mathcal{O}_i|$ the total number of node observations. We use $\mathcal{L}_{\text{MSE}}$ with Mean Squared Error (MSE) as ℓ, but the framework is fully compatible with other loss functions as well. This includes general probabilistic predictions with a negative log-likelihood loss. Dividing by $j - N_{\text{init}} - 1$, the number of times observation j has been predicted, guarantees that later observations are not given a higher total weight.

The weighting function w allows for specifying which time-horizons that should be prioritized by the model. This choice is highly application-dependent and should capture which predictions that are of interest when later deploying the model in some real-world setting. If we care about all time horizons, but want to prioritize predictions close in time, a suitable choice could be $w(\Delta_t) = \exp\left(-\frac{\Delta_t}{\Omega}\right)$. It might also be desirable to focus predictive capabilities on a specific Δ_t. If we want predictions around $\Delta_t = \mu$ to be prioritized we can for instance use a Gaussian kernel as

$$w(\Delta_t) = \exp\left(-\left(\frac{\Delta_t - \mu}{\Omega}\right)^2\right). \tag{15}$$

A limitation of the proposed loss function is the quadratic scaling in the number of time steps, as predictions are made from all times to all future observations. This especially requires large amounts of memory for nodes that are observed at many time steps. However, for many sensible choices of w predictions far into the future have a close to 0 impact on the loss. In practice we can utilize this to approximate $\mathcal{L}_\ell$ by only making predictions N_{max} time steps into the future. This approximation explicitly corresponds to setting $\tau_{m,i} = \{j : m \in \mathcal{O}_j \wedge i < j \leq i + N_{\text{max}}\}$ and changing the denominator in Eq. 14 to $\min(N_{\text{max}}, j - N_{\text{init}} - 1)$. Alternatively, we can select a weight function with finite support which implies that many terms in Eq. 14 will be exactly zero. This does however require more bookkeeping than the aforementioned truncation method.

4 EXPERIMENTS

The TGNN4I model was implemented[2] using PyTorch and PyG (Fey and Lenssen, 2019). We evaluate the model on a number of different datasets. See appendix D and E for details on the pre-processing and experimental setups used. As the loss function $\mathcal{L}_{\text{MSE}}$ captures errors throughout an entire time series we adopt this also as our evaluation metric. Given that the loss weighting w used for training accurately represents how we value predictions at different time horizons, it is natural to use the same choice for evaluation. In our experiments we rescale each time series so that $t \in [0, 1]$ and use $w(\Delta_t) = \exp\left(-\frac{\Delta_t}{0.04}\right)$. By inspecting w and the time steps in the data we also choose a suitable $N_{\text{max}} = 10$.

We consider three versions of our TGNN4I model: (**static**) with a constant latent state in-between observations ($\tilde{\boldsymbol{h}}^n(t) = \hat{\boldsymbol{h}}^n_i \ \forall t \in]t_i, t_j]$), (**exponential**) with the exponential decay dynamics from Eq. 4 and (**periodic**) with the combined decay and periodic dynamics from Eq. 6. In all our experiments the training time of a single model on an NVIDIA A100 GPU is less than an hour and for the smallest dataset (METR-LA) not more than 20 minutes.

4.1 Baselines

We compare TGNN4I to multiple baseline models. As a simple starting point we consider a model that always predicts the last observed value in each node for all future time points (**Predict Previous**). Che et al. (2018) propose the GRU-D model for irregular time series, which we extend with our parametrization of the exponential decay and include as a baseline. GRU-D does not use the graph structure explicitly, so there are two ways to adapt this model to our setting. We can view the entire graph-structured time series as one series with $(|V|d_y)$-dimensional vectors at each time step (**GRU-D (joint)**). Alternatively, we can view the time series in each node as independent (**GRU-D (node)**), which is essentially the same as TGNN4I with all edges in the graph removed. Two **Transformer** baselines are also included, used in the same (joint) and (node) configurations. In these models the irregular observations are handled through attention masks and the use of timestamps in the sinusoidal positional encodings. We also compare against the **LG-ODE** model of Huang et al. (2020) using the code provided by the authors. We follow their proposed training procedure, where the model encodes the first half of each time series and has to predict the second half. When computing $\mathcal{L}_{\text{MSE}}$ using LG-ODE we encode all observations up to t_i and decode from that time point in order to get each $\hat{\boldsymbol{y}}^m_{i\to j}$. More details on the baseline models are given in appendix C. An attempt was also made to adapt the RAINDROP model of Zhang et al. (2022) to our fore-

[2]Our code and datasets are available at `https://github.com/joeloskarsson/tgnn4i`.

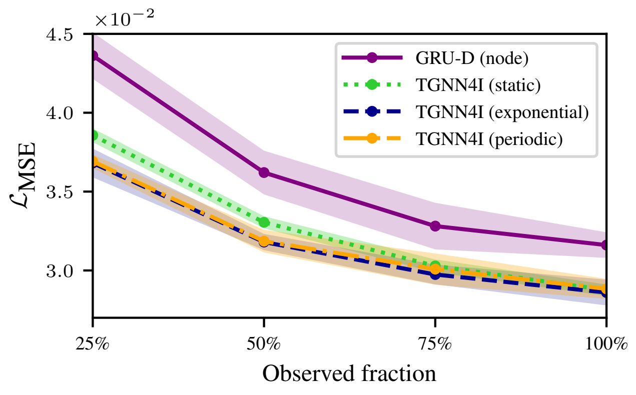


Figure 2: Test $\mathcal{L}_{\text{MSE}}$ on METR-LA traffic data for the best performing models from Table 1. Shaded areas correspond to 95% confidence intervals based on re-training models with 5 random seeds.

casting setting. We were however unable to get useful predictions without making major changes to the model and it is therefore not included here.

4.2 Traffic Data

We experiment on the PEMS-BAY and METR-LA datasets, containing traffic speed sensor data from the California highway system (Li et al., 2018; Chen et al., 2001). To be able to control the degree of irregularity, we start from regularly sampled data and choose subsets of observations. We use the versions of the datasets pre-processed by Li et al. (2018). Each dataset is split up into time series of 288 observations (1 day). PEMS-BAY contains 180 such time series with 325 nodes and METR-LA 119 time series with 207 nodes. We include the time of day and the time since the node was last observed as input features $\boldsymbol{x}_i^n$. In order to introduce irregularity in the time steps we next subsample each time series by keeping only 72 of the 288 observations. These $N_t = 72$ observation times are the same for all nodes. However, from these subsampled time series we furthermore sample subsets containing 25%–100% of all $N_t \times |V|$ individual node observations. This results in irregular observation time points and a fraction of nodes observed at each time. Our additional pre-processing prevents us from a direct comparison with Li et al. (2018), as their method does not handle irregular observations.

We report results for both datasets in Table 1 and highlight the best performing models on METR-LA in Figure 2. GRU-D (joint) has a hard time modeling all nodes jointly, often not performing better than the simple Predict Previous baseline. The Transformer models achieve somewhat better results, but still not competitive with TGNN4I. We additionally note that the Transformers can be highly sensitive to the random seed used for initialization, something that we have not observed for other models. Comparing TGNN4I and GRU-D (node) in Figure 2 it can be noted that the importance of using the graph structure increases when there are fewer observations. Out of the different versions of TGNN4I the exponential and periodic dynamics show a clear advantage over the static one, with the largest difference for the most sparsely observed data. We have observed that the periodic models output only low frequencies, resulting in dynamics and results similar to the model with exponential decay. While the periodic dynamics in Eq. 6 have fewer degrees of freedom when reduced to pure exponential decay, this does not seem to hurt the performance in this example.

We found that the training time of LG-ODE scales poorly to large graphs, limiting us to only training a single model for each dataset. The predictions are however quite poor, especially on the PEMS-BAY data. While the model seems to learn something more than just predicting the mean, it is not competitive with our TGNN4I model. We believe that the poor performance of LG-ODE can be explained by a combination of multiple things: (1) The LG-ODE model is primarily designed for data with clear continuous underlying dynamics, which might not match this type of traffic data. (2) When the model is trained as proposed by Huang et al. (2020), it can require large amounts of data. For some of the experiments in the original paper 20 000 sequences are used for training, while we use less than 150. (3) The slow training has limited possibilities for exhaustive hyperparameter tuning on our datasets. Training one LG-ODE model on the PEMS-BAY data takes us over 50 hours.

4.3 USHCN Climate Data

Irregular and missing observations are common problems in climate data (Schneider, 2001). The United State Historical Climatology Network (USHCN) daily dataset contains over 50 years of measurements of multiple climate variables from sensor stations in the United States (Menne et al., 2015). We use the pre-processing of De Brouwer et al. (2019) to clean and subsample the data. The target variables chosen are minimum and maximum daily temperature ($T_{\min}$ and $T_{\max}$), which we model as separate datasets. While existing works (De Brouwer et al., 2019; Schirmer et al., 2022) have treated time series from different sensor stations as independent, we model also the spatial correlation by constructing a 10-nearest-neighbor graph using the sensor positions. Each full dataset contains 186 time series of length $N_t = 100$ on a graph of 1123 nodes. The pre-processed USHCN data is sparsely observed with only around 5% of potential node observations present.

We report results on both datasets in Table 2. Due to the large size of the graph it was not feasible to apply the LG-ODE model here. We note that for these datasets the (joint) baselines clearly outperform the (node) versions. For GRU-D this is the opposite of what we saw in the traffic data. This can be explained by the fact that climate data

Table 1: Test $\mathcal{L}_{\text{MSE}}$ (multiplied by 10^2) for the traffic datasets with different fractions of node observations. Where applicable we report mean ± one standard deviation across 5 runs with different random seeds. The lowest mean $\mathcal{L}_{\text{MSE}}$ for each dataset and observation percentage is marked in bold.

	PEMS-BAY			
Model	25%	50%	75%	100%
Predict Previous	26.32	18.60	15.25	13.50
GRU-D (joint)	18.79±0.07	18.27±0.10	17.93±0.08	17.75±0.12
GRU-D (node)	8.79±0.06	6.62±0.02	5.82±0.06	5.49±0.06
Transformer (joint)	12.05±1.19	13.13±2.59	12.21±1.95	11.09±1.38
Transformer (node)	16.49±0.17	14.44±0.48	13.20±0.56	13.16±1.23
LG-ODE	27.00	24.93	24.71	23.52
TGNN4I (static)	7.41±0.09	5.98±0.07	5.29±0.08	4.89±0.05
TGNN4I (exponential)	**7.10±0.07**	**5.78±0.05**	5.23±0.03	**4.87±0.09**
TGNN4I (periodic)	**7.10±0.09**	5.80±0.08	**5.22±0.09**	**4.87±0.02**
	METR-LA			
Model	25%	50%	75%	100%
Predict Previous	9.86	7.54	6.52	6.04
GRU-D (joint)	8.38±0.05	8.03±0.04	7.89±0.03	7.80±0.02
GRU-D (node)	4.36±0.08	3.62±0.07	3.28±0.08	3.16±0.04
Transformer (joint)	5.70±1.41	7.17±1.66	5.95±1.90	6.11±1.80
Transformer (node)	7.01±0.31	6.34±0.24	5.84±0.23	5.96±0.50
LG-ODE	8.51	7.35	6.71	6.24
TGNN4I (static)	3.86±0.02	3.31±0.02	3.03±0.02	2.88±0.02
TGNN4I (exponential)	**3.68±0.05**	**3.18±0.03**	**2.97±0.03**	**2.86±0.04**
TGNN4I (periodic)	3.69±0.02	3.19±0.04	3.01±0.05	2.88±0.03

Table 2: Test $\mathcal{L}_{\text{MSE}}$ (multiplied by 10^2) for the two USHCN climate datasets.

	T_{min}	T_{max}
Predict Previous	16.88	17.18
GRU-D (joint)	8.03±0.23	7.97±0.19
GRU-D (node)	13.12±0.03	13.67±0.04
Transformer (joint)	7.36±0.41	7.37±0.28
Transformer (node)	15.68±0.32	15.74±0.34
TGNN4I (static)	6.97±0.05	6.86±0.04
TGNN4I (exponential)	**6.72±0.04**	**6.60±0.04**
TGNN4I (periodic)	**6.72±0.05**	6.63±0.03

has strong spatial dependencies. The (joint) models can to some extent learn to pick up on these, while for (node) no information can flow between nodes. The best results are however achieved by TGNN4I, showing the added benefit of utilizing the spatial graph.

4.4 Synthetic Periodic Data

In the previous experiments, using periodic dynamics with TGNN4I has not added any value. Instead, the learned dynamics have been largely similar to just using exponential decay. This should to some extent be expected, as none of the previous datasets show any clear periodic patterns at the considered time scales. To investigate the possible benefits of the periodic dynamics we instead create a synthetic dataset with periodic signals propagating over a graph.

The synthetic dataset is based on a randomly sampled directed acyclic graph with 20 nodes. We define a periodic base signal

$$\rho^n(t) = \sin(\phi^n t + \eta^n) \tag{16}$$

with random parameters ϕ^n and η^n for each node n. The target signal $y^n(t)$ in each node is then defined through

$$\kappa^n(t) = \rho^n(t) + \frac{0.5}{|\mathcal{N}(n)|} \sum_{m \in \mathcal{N}(n)} \kappa^m(t - 0.05) \tag{17a}$$

$$y^n(t) = \kappa^n(t) + \epsilon^n(t) \tag{17b}$$

where $\epsilon^n(t)$ is Gaussian white noise with standard deviation 0.01. The target signal in each node depends on the base signal in the node itself and the signals in neighboring nodes at a time lag of 0.05. To construct one time series we sample $y^n(t)$ at 70 irregular time points on $[0, 1]$. In total we sample 200 such time series and keep 50% of the node observations in each.

We train versions of the GRU-D (node) and TGNN4I models with different latent dynamics on the synthetic data.

Table 3: Test $\mathcal{L}_{\text{MSE}}$ (multiplied by 10^2) for synthetic data.

	Static	Exponential	Periodic
GRU-D (node)	8.88±0.36	3.13±0.06	2.81±0.04
TGNN4I	15.12±0.05	2.91±0.17	**1.95±0.11**
Predict Prev.		27.52	
Transformer (joint)		23.19±0.38	
Transformer (node)		15.39±0.05	
LG-ODE		16.61±0.23	

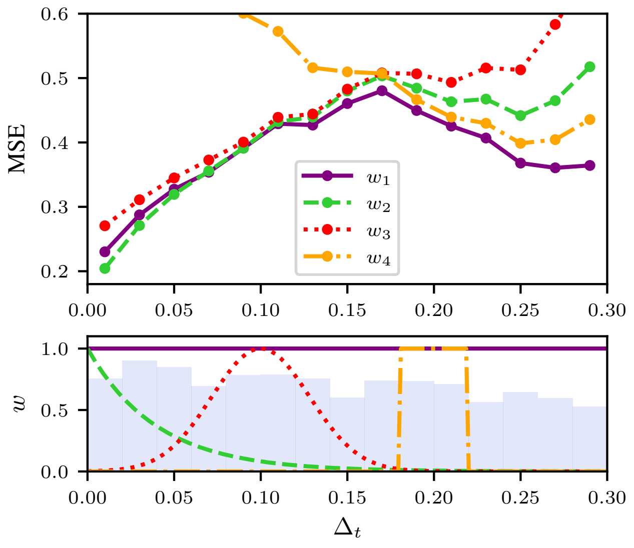


Figure 3: MSE (Top) for predictions at time Δ_t in the future for models trained with different loss weighting w (Bottom). The weighting functions are described in Eq. 18a–18d. To compute the MSE all predictions in the test set were binned based on their Δ_t, with bin width 0.02. The bottom subplot also shows a histogram of the number of predictions in each bin.

Also our other baselines are included for comparison. Results are reported in Table 3. For both GRU-D (node) and TGNN4I we see a large difference between the different types of latent dynamics. The periodic dynamics seem to help the model to keep track of the base signal in the node and its neighborhood in order to achieve accurate future predictions. While this is a synthetic example, periodic behavior is prevalent in much time series data and being able to explicitly model this in the latent state can be highly advantageous. Attempts were made to also train the GRU-D (joint) model on this dataset, but it failed to pick up on any patterns and ended up only predicting a constant value for all nodes and times.

4.5 Loss Weighting

To investigate the impact of the loss weighting function w we trained four TGNN4I models on the subsampled PEMS-BAY dataset with 25% observations. We used exponential dynamics and considered the weighting functions

$$w_1(\Delta_t) = 1 \tag{18a}$$

$$w_2(\Delta_t) = \exp\left(-\frac{\Delta_t}{0.04}\right) \tag{18b}$$

$$w_3(\Delta_t) = \exp\left(-\left(\frac{\Delta_t - 0.1}{0.02}\right)^2\right) \tag{18c}$$

$$w_4(\Delta_t) = \mathbb{I}_{\{\Delta_t \in [0.18, 0.22]\}}. \tag{18d}$$

Figure 3 shows the test MSE for the trained models at different Δ_t in the future, as well as plots of the weighting functions. As an example, the prediction $\hat{\boldsymbol{y}}^m_{i\to j}$ has $\Delta_t = t_j - t_i$.

We note that the choice of w can have substantial impact on the error of the model at different time horizons. The exponential weighting in w_2 makes the model focus heavily on short-term predictions. This results in better predictions for low Δ_t. At the shortest time horizon the exponential weighting yields an 11% improvement over the model trained with constant w_1, but this comes at the cost of far higher errors for long-term predictions. Interestingly the w_1 model gives better predictions at all time horizons than the models with w_3 and w_4, which focus on predictions at some specific time ahead. We believe that there can be a feedback effect benefiting the constant weighting, where learning to make good short-term predictions also aid the learning of long-term prediction, for example by finding useful intermediate representations. A drawback of weighting with w_1 is however that since the loss never approaches 0 there is no properly motivated choice of N_{max} for our loss approximation. As the model trained with w_4 still gives good predictions in the interval $[0.18, 0.22]$, there can still be practical reasons to choose such a weighting. With this choice we could reduce the innermost sum in Eq. 14 to only those j:s that lie in the interval of interest.

5 DISCUSSION

We have proposed a temporal GNN model that can handle both irregular time steps and partially observed graphs. By defining latent states in continuous time our model can make predictions for arbitrary time points in the future. In this section we discuss some details and limitations of the approach, and also give some pointers to interesting directions for future work.

5.1 Efficient Implementation

In order to efficiently implement the training and inference of TGNN4I there is a key design choice between (1) storing everything in dense matrices and utilizing massively parallel GPU-computations, and (2) utilizing sparse representations in order to avoid computing values that will never be used. Which of these is to be preferred depends on the sparsity of observations in the data. Our implementation follows the massively parallel approach, using binary masks for keeping track of $\mathcal{O}_i$ at each time point. In order to scale TGNN4I to massive graphs in real world scenarios it could be interesting to consider a version of the model distributed over multiple machines, perhaps directly connected to sensors producing the input data. A sparse implementation would be strongly preferred for such an extension.

5.2 Linear and Neural ODEs

The dynamics of TGNN4I are defined by a linear ODE and additionally restricted by the assumption of unique eigenvalues in A. This has the benefit of a closed form solution that is efficient to compute, but also limits the types of dynamics that can be learned. Our approach can be contrasted with Neural ODEs (Chen et al., 2018), that allow for learning more expressive dynamics. Neural ODEs do however lack closed form solutions and require using numerical solvers (Kidger, 2021). This incurs a trade-off between speed and numerical accuracy. In experiments we have compared TGNN4I with the LG-ODE model (Huang et al., 2020), which uses a Neural ODE decoder. The slow training of the LG-ODE model can to a large extent be attributed to the numerical ODE solver. While more complex latent dynamics can be useful for some datasets, it can also be argued that simpler dynamics can be compensated with a high enough latent dimension d_h and a flexible enough predictive model g (Schirmer et al., 2022).

5.3 Societal and Sustainability Impact

While our contributions are purely methodological, many applications of graph-based and spatio-temporal data analysis have a clear societal impact. Our example applications of traffic and climate modeling both have potential to aid efforts of transforming society in more sustainable directions, such as those described in the United Nations sustainable development goals 11 and 13 (Rolnick et al., 2022; United Nations, 2015). Traffic modeling allows us to both understand travel behavior and predict future demand. This can enable optimizations of transport systems, both improving the experience of travelers and reducing the environmental impact. Integrating machine learning methods with climate modeling has potential to speed up simulations and increase our understanding of the climate around us. However, it is surely also possible to find applications of our method with a damaging impact on society, for example through undesired mass-surveillance.

5.4 Future Work

We consider a setting where the graph structure is both known and constant over time. In some practical applications it is not obvious how to construct the graph describing the system. To tackle this problem our model could be combined with approaches for also learning the graph structure (Stanković et al., 2020; Zhang et al., 2022). Extending our method to dynamic graphs, that evolve over time, would not require any major changes and could be an interesting direction for future experiments.

Our focus has been on forecasting, but the model could also be trained for other tasks. The time-continuous latent state in each node could be used for imputing missing observations or performing sequence segmentation. Also classification tasks are possible, either classifying each node separately or the entire graph-structured time series.

While our model can produce predictions at arbitrary time points, an extension would be to also predict the time until the next observation occurs. One way to achieve this would be to let the latent state parametrize the intensity of a point process (Mei and Eisner, 2017; Jia and Benson, 2019). Building such point process models on graphs could be an interesting future application of our model.

Acknowledgments

This research is financially supported by the Swedish Research Council via the project *Handling Uncertainty in Machine Learning Systems* (contract number: 2020-04122), the Excellence Center at Linköping–Lund in Information Technology (ELLIIT), and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by the Berzelius resource at the National Supercomputer Centre, provided by the Knut and Alice Wallenberg Foundation. All datasets available online were accessed from the Linköping University network.

References

Arrowsmith, D. and Place, C. M. (1992). *Dynamical systems: differential equations, maps, and chaotic behaviour*. CRC Press.

Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

Cao, W., Wang, D., Li, J., Zhou, H., Li, L., and Li, Y. (2018). BRITS: Bidirectional recurrent imputation for time series. In *Advances in Neural Information Processing Systems*, volume 31.

Che, Z., Purushotham, S., Cho, K., Sontag, D., and Liu, Y. (2018). Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, 8(1):6085.

Chen, C., Petty, K., Skabardonis, A., Varaiya, P., and Jia, Z. (2001). Freeway performance measurement system: Mining loop detector data. *Transportation Research Record*, 1748(1):96–102.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In *Advances in neural information processing systems*, volume 31.

Chen, Y., Kang, Y., Chen, Y., and Wang, Z. (2020). Probabilistic forecasting with temporal convolutional neural network. *Neurocomputing*, 399:491–501.

Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111.

Cini, A., Marisca, I., and Alippi, C. (2022). Filling the g_ap_s: Multivariate time series imputation by graph neural networks. In *International Conference on Learning Representations (ICLR)*.

De Brouwer, E., Simm, J., Arany, A., and Moreau, Y. (2019). GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series. In *Advances in Neural Information Processing Systems*, volume 32.

Fang, Z., Long, Q., Song, G., and Xie, K. (2021). Spatial-temporal graph ODE networks for traffic flow forecasting. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 364–373.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1263–1272.

Giuliari, F., Hasan, I., Cristani, M., and Galasso, F. (2021). Transformer networks for trajectory forecasting. In *25th International Conference on Pattern Recognition (ICPR)*.

Gordon, D., Petousis, P., Zheng, H., Zamanzadeh, D., and Bui, A. A. (2021). TSI-GNN: Extending graph neural networks to handle missing data in temporal settings. *Frontiers in Big Data*, 4.

Guo, S., Lin, Y., Feng, N., Song, C., and Wan, H. (2019). Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 922–929.

Horn, R. A. and Johnson, C. R. (2012). *Matrix analysis*. Cambridge university press.

Huang, Z., Sun, Y., and Wang, W. (2020). Learning continuous system dynamics from irregularly-sampled partial observations. In *Advances in Neural Information Processing Systems*, volume 33, pages 16177–16187.

Huang, Z., Sun, Y., and Wang, W. (2021). Coupled graph ODE for learning interacting system dynamics. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 705–715.

Jia, J. and Benson, A. R. (2019). Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, volume 32.

Kidger, P. (2021). *On neural differential equations*. PhD thesis, University of Oxford.

Kosman, E. and Di Castro, D. (2022). Graphvid: It only takes a few nodes to understand a video. In *Computer Vision – ECCV 2022*, pages 195–212.

Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Pritzel, A., Ravuri, S., Ewalds, T., Alet, F., Eaton-Rosen, Z., et al. (2022). Graphcast: Learning skillful medium-range global weather forecasting. *arXiv preprint arXiv:2212.12794*.

Lazzeri, F. (2020). *Machine learning for time series forecasting with Python*. John Wiley & Sons.

Li, Y., Yu, R., Shahabi, C., and Liu, Y. (2018). Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations (ICLR)*.

Mei, H. and Eisner, J. M. (2017). The neural hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems*, volume 30.

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

Omidshafiei, S., Hennes, D., Garnelo, M., Wang, Z., Recasens, A., Tarassov, E., Yang, Y., Elie, R., Connor, J. T., Muller, P., Mackraz, N., Cao, K., Moreno, P., Sprechmann, P., Hassabis, D., Graham, I., Spearman, W., Heess, N., and Tuyls, K. (2022). Multiagent off-screen behavior prediction in football. *Scientific Reports*, 12(1):1–13.

Poli, M., Massaroli, S., Rabideau, C. M., Park, J., Yamashita, A., Asama, H., and Park, J. (2021). Continuous-depth neural models for dynamic graph prediction. *arXiv preprint arXiv:2106.11581*.

Roberts, S., Osborne, M., Ebden, M., Reece, S., Gibson, N., and Aigrain, S. (2013). Gaussian processes for time-series modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*.

Rolnick, D., Donti, P. L., Kaack, L. H., Kochanski, K., Lacoste, A., Sankaran, K., Ross, A. S., Milojevic-Dupont, N., Jaques, N., Waldman-Brown, A., Luccioni, A. S., Maharaj, T., Sherwin, E. D., Mukkavilli, S. K., Kording, K. P., Gomes, C. P., Ng, A. Y., Hassabis, D., Platt, J. C., Creutzig, F., Chayes, J., and Bengio, Y. (2022). Tackling climate change with machine learning. *ACM Computing Surveys*, 55(2).

Rozemberczki, B., Scherer, P., Kiss, O., Sarkar, R., and Ferenci, T. (2021). Chickenpox cases in hungary: A benchmark dataset for spatiotemporal signal processing with graph neural networks. In *Workshop on Graph Learning Benchmarks@ TheWebConf 2021*.

Rubanova, Y., Chen, R. T. Q., and Duvenaud, D. K. (2019). Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, volume 32.

Rusch, T. K., Chamberlain, B., Rowbottom, J., Mishra, S., and Bronstein, M. (2022). Graph-coupled oscillator networks. In *Proceedings of the 39th International Conference on Machine Learning*, pages 18888–18909.

Schirmer, M., Eltayeb, M., Lessmann, S., and Rudolph, M. (2022). Modeling irregular time series with continuous recurrent units. In *Proceedings of the 39th International Conference on Machine Learning*, pages 19388–19405.

Schneider, T. (2001). Analysis of incomplete climate data: Estimation of mean values and covariance matrices and imputation of missing values. *Journal of Climate*, 14(5):853–871.

Searle, S. R. and Khuri, A. I. (1982). *Matrix algebra useful for statistics*. John Wiley & Sons.

Stanković, L., Mandic, D., Daković, M., Brajović, M., Scalzo, B., Li, S., and Constantinides, A. G. (2020). Data analytics on graphs part III: Machine learning on graphs, from graph topology to applications. *Foundations and Trends in Machine Learning*, 13(4):332–530.

United Nations (2015). Transforming our world: The 2030 agenda for sustainable development.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, volume 30.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2020a). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21.

Wu, Z., Pan, S., Long, G., Jiang, J., Chang, X., and Zhang, C. (2020b). Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 753–763.

Wu, Z., Pan, S., Long, G., Jiang, J., and Zhang, C. (2019). Graph wavenet for deep spatial-temporal graph modeling. In *Proceedings of the Twenty-Eight International Joint Conference on Artificial Intelligence*, page 1907–1913.

Yu, B., Yin, H., and Zhu, Z. (2018). Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 3634–3640.

Zang, C. and Wang, F. (2020). Moflow: An invertible flow model for generating molecular graphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, page 617–626.

Zhang, X., Zeman, M., Tsiligkaridis, T., and Zitnik, M. (2022). Graph-guided network for irregularly sampled multivariate time series. In *International Conference on Learning Representations (ICLR)*.

Zhao, X., Chen, F., and Cho, J.-H. (2018). Deep learning for predicting dynamic uncertain opinions in network data. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1150–1155.

A TABLE OF NOTATION

The notation used is listed in Table 4.

B IMPLEMENTATION AND TRAINING DETAILS FOR TGNN4I

The presentation of TGNN4I in Section 3 describes how the model processes a single graph-structured time-series. In practice we use batches of multiple such time series during training and inference. When working on batches there is an additional sum in the definition of $\mathcal{L}_\ell$, computing the mean over all samples in the batch. While we assume that all time series in a batch share the same N_t, the exact time points $\{t_i\}_{i=1}^{N_t}$ can differ. In all experiments we use $N_{\text{init}} = 5$ and minimize the loss using the Adam optimizer (Kingma and Ba, 2015).

Both GNN^U and GNN^W contain only GNN layers, while the predictive model g contains a sequence of GNN layers followed by a sequence of fully connected layers. We use ReLU activation functions in between all layers.

The input $\tilde{\boldsymbol{x}}_i^m$ to GNN^W is $\mathbf{0}$ for neighbors that are not observed at time t_i. However, there could be an observation of neighbor m where the observed value and input features are exactly $\mathbf{0}$. To help the model differentiate between these two cases we additionally include an indicator variable $\mathbb{I}_{\{m \in \mathcal{O}_i\}}$ in $\tilde{\boldsymbol{x}}_i^m$.

C BASELINE MODELS

C.1 Predict Previous

The Predict Previous baseline does not require any training. Predictions are computed as $\hat{\boldsymbol{y}}_{i \to j}^m = \boldsymbol{y}_{\text{prev}(i)}^m \; \forall j$, where $\text{prev}(i) = \max\{k : k \leq i \wedge m \in \mathcal{O}_k\}$. If there is no earlier observation of node m the predictions is just $\mathbf{0}$.

C.2 GRU-D (node)

GRU-D (node) is essentially a version of TGNN4I without the GNN components. The GNNs GNN^U and GNN^W in the GRU update are replaced by matrix multiplications and the predictive model g includes only fully connected layers. Because of this no information can flow between nodes, and time-series in different nodes are treated as independent. The GRU-D (node) model uses the exponential dynamics for the latent state. To stay consistent with the other models we still process all nodes in the graph concurrently. This means that a batch size of B for GRU-D (node) means that we process $B \times N$ independent node time-series.

C.3 GRU-D (joint)

The GRU-D (joint) model is defined similar to GRU-D (node), but modeling all nodes jointly. All node observations are concatenated into one long vector $\boldsymbol{y}_i = \left[\boldsymbol{y}_i^1, \dots, \boldsymbol{y}_i^{|V|}\right]^\intercal$ and similarly the features are concatenated as $\boldsymbol{x}_i = \left[\boldsymbol{x}_i^1, \dots, \boldsymbol{x}_i^{|V|}\right]^\intercal$. This time series is then modeled using a single latent state, also based on the exponential dynamics. In GRU-D (joint) there is however a GRU update at each t_i, as at least one of the nodes is observed at each time point. We can think of this setup as a graph with a single node, for which the observations are high-dimensional. The high-dimensional predictions are split up and re-assigned to the original nodes in the graph for computing the loss. Note also that many entries in each $\boldsymbol{y}_i$ and $\boldsymbol{x}_i$ will be zero, corresponding to the nodes that are not observed. We again include indicators $\mathbb{I}_{\{m \in \mathcal{O}_i\}}$ as input to the GRU, but here for all nodes.

C.4 LG-ODE

For the LG-ODE model we use the code provided by the authors (Huang et al., 2020)[3], making only small modifications to the data loading in order to correctly handle our datasets. The original code does not use a validation set, instead evaluating the model on the test set after each training epoch. We change this step to use validation data and save the model from the epoch with the lowest validation error. That model is then loaded and evaluated on the test data. Due to the high training time we have not been able to perform exhaustive hyperparameter tuning for the LG-ODE model. We have mainly used

[3]`https://github.com/ZijieH/LG-ODE`

Table 4: List of notation

Notation	Description	Defined in
$\mathrm{diag}(\boldsymbol{v})$	Diagonal matrix with entries of vector $\boldsymbol{v}$ on the diagonal	
$[\boldsymbol{v}]_{j:j+1}$	Entries j and $j+1$ of vector $\boldsymbol{v}$	
$\mathbb{I}_{\{\cdot\}}$	Indicator function	
$\mathcal{G}$	The underlying graph of the time series	Section 3.1
V, E	Node and edge set of $\mathcal{G}$	
t_i	Time point where at least one node is observed	
N_t	Number of time points in one graph-structured time series	
$\mathcal{O}_i$	Set of nodes observed at t_i	
$\boldsymbol{y}_i^n$	Observed value in node n at time point t_i. Equal to $\mathbf{0}$ if node n is not observed at time t_i.	
$\boldsymbol{x}_i^n$	Features of node n at time point t_i. Equal to $\mathbf{0}$ if node n is not observed at time t_i.	
d_x, d_y, d_h	Dimensionality of $\boldsymbol{x}_i^n$, $\boldsymbol{y}_i^n$ and $\boldsymbol{h}^n(t)$	
$\boldsymbol{h}^n(t)$	Time-continuous latent state of node n	
$\bar{\boldsymbol{h}}_i^n$	Static part of $\boldsymbol{h}^n(t)$ after time t_i	Section 3.2
$\tilde{\boldsymbol{h}}^n(t)$	Dynamic part of $\boldsymbol{h}^n(t)$ defined by a linear ODE	
A	Coefficient matrix in the ODE defining $\tilde{\boldsymbol{h}}^n(t)$	
$\hat{\boldsymbol{h}}_i^n$	Initial value of $\tilde{\boldsymbol{h}}^n(t_i)$ in the ODE from time t_i onward	
δ_t	Elapsed time since t_i	
Q	Matrix containing the eigenvectors of A	
Λ	Diagonal matrix containing the eigenvalues of A	
$\boldsymbol{\omega}_i^n$	Parameters defining the dynamics of $\tilde{\boldsymbol{h}}^n(t)$ after t_i	
C_j	2×2 block matrix in real Jordan form of A	
$a_j \pm b_j i$	Complex conjugate eigenvalue pair of A	
$\mathcal{N}(n)$	Neighbors (parents) of node n	Section 3.3
W_1, W_2	Matrices used in GNN	
$e_{m,n}$	Edge weight associated with the edge (m, n)	
GNN^U	GNN combining latent states in neighborhood of node n	
$\boldsymbol{u}_{i,1}^n - \boldsymbol{u}_{i,7}^n$	Output of GNN^U for node n at time t_i. Intermediate representations used in GRU update.	
$\tilde{\boldsymbol{x}}_i^n$	Concatenation of $\boldsymbol{y}_i^n$ and $\boldsymbol{x}_i^n$	
GNN^W	GNN combining features and observations in neighborhood of node n	
$\boldsymbol{v}_{i,1}^n - \boldsymbol{v}_{i,7}^n$	Output of GNN^W for node n at time t_i. Intermediate representations used in GRU update.	
$\boldsymbol{r}_i^n, \boldsymbol{z}_i^n, \boldsymbol{q}_i^n$	Intermediate representations in GRU update for $\boldsymbol{h}^n(t)$	
σ	Sigmoid function	
$\boldsymbol{b}_1$–$\boldsymbol{b}_7$	Bias parameters in GRU cell	
$\bar{\boldsymbol{r}}_i^n, \bar{\boldsymbol{z}}_i^n, \bar{\boldsymbol{q}}_i^n$	Intermediate representations in GRU update for $\bar{\boldsymbol{h}}_i^n$	
g	Predictive model	Section 3.4
$\hat{\boldsymbol{y}}_j^n$	Predicted value of node n at time t_j	
GNN^g	GNN used as predictive model g	
$\hat{\boldsymbol{y}}_{i\to j}^m$	Predicted value of node m at time t_j based on observations before or at time t_i	Section 3.5
w	Loss weighting function	
$\tau_{m,i}$	Set of indices of all times after t_i where node m is observed	
N_{init}	Length of warm-up phase where predictions are not included in loss	
$\mathcal{L}_\ell$	Loss function for one whole graph-structured time-series	
ℓ	General loss function for a single observation	
$\mathcal{L}_{\text{MSE}}$	$\mathcal{L}_\ell$ with Mean Squared Error as loss for each observation	
N_{obs}	Number of node observations in time-series	
Δ_t	Time difference between last observation and prediction time	
$N_{\max}$	Maximum number of time steps to predict ahead in approximation of $\mathcal{L}_\ell$	

the default parameters of Huang et al. (2020) or a slightly smaller version of the model. The smaller version has halved dimensions for hidden layers in the GNN (from 128 to 64) and augmentation (from 64 to 32) used in the ODE decoder.

For computing $\mathcal{L}_{\text{MSE}}$ using the LG-ODE model we need to compute each necessary $\hat{\boldsymbol{y}}^m_{i \to j}$. This is done by encoding all observations up to time t_i and then decoding N_{max} time steps into the future. It should be noted that the model is still trained as proposed by Huang et al. (2020), by encoding the first half of each time series and predicting the second half. During training each encoded time series is $N_t/2$ long, but when computing $\mathcal{L}_{\text{MSE}}$ on the test set the length of the sequence being encoded varies. We have however not noticed any higher errors for time steps with a shorter or longer encoded sequence than that used during training.

C.5 Transformers

The Transformer (Vaswani et al., 2017) baselines use an encoder-decoder approach similar to LG-ODE. During training a prediction time t_i is however randomly sampled as $i \sim \mathcal{U}(\{N_{\text{init}}, \dots, N_t - N_{\text{max}}\})$. Each sequence is then encoded up to time t_i and decoded over the next N_{max} time steps to produce predictions. The $\mathcal{L}_{\text{MSE}}$ loss is used also for the Transformer models, but only based on this one prediction per time series.

The irregular time steps are handled by sinusoidal encodings concatenated to the input of both the encoder and decoder. Instead of basing these on the sequence index i, the exact timestamp t_i is used in

$$\theta_i = \frac{t_i}{0.1^{2i/d_h}} \tag{19}$$

to then compute the full encoding vector $\left[\sin(\theta_1), \dots, \sin(\theta_{\lfloor d_h/2 \rfloor}), \cos(\theta_1), \dots, \cos(\theta_{\lfloor d_h/2 \rfloor})\right]^\intercal$.

Unobserved nodes are in Transformer (node) handled by masking the attention mechanism. For each node n, this prevents the model from attending to encoded time steps j s.t. $n \notin \mathcal{O}_j$. In Transformer (joint) the same approach as in GRU-D (joint) is instead used, where indicator variables are included as input.

The hyperparameters defining the Transformer architectures differ somewhat from the other models. We still let d_h represent the dimensionality of hidden representations, but here also tune the number of transformer layers stacked together.

D DATASETS

D.1 Traffic Data

For the PEMS-BAY and METR-LA datasets we use the versions pre-processed by Li et al. (2018), where weighted graphs are created based on thresholded road-network distances. We additionally remove edges from any node to itself and drop nodes not connected to the rest of the graph. The original time series is then split into sequences of length 288, corresponding to one day of observations at 5 minute intervals. In each such sequence we uniformly sample only $N_t = 72$ time points to keep, introducing irregularity between the time steps. Next we create the set $\{(n,i)\}_{n \in V, i=1,\dots,N_t}$ with indices of all single node observations. We uniformly sample a fraction of this set as the observations to keep, independently for each sequence. The percentages in Table 1 refer to the percentage of observations kept from this set. This step introduces further irregularity as all nodes will generally not be observed at each time step. The METR-LA dataset has some observations missing initially, meaning that we can never get to exactly 100% observations for that dataset. From this pre-processing we end up with 180 time series in the PEMS-BAY dataset and 119 time series in the METR-LA dataset. We randomly assign 70% of these to the training set, 10% to the validation set and 20% to the test set.

D.2 USHCN Climate Data

The USHCN daily data is openly available[4] together with the positions of all sensor stations. We use the pre-processing of De Brouwer et al. (2019) to clean and subsample the data[5]. We choose the longer version of the time series, with observations between the years 1950 and 2000, but split this into multiple sequences of 100 days.

In order to build the spatial graph we perform an equirectangular map projection of the sensor station coordinates and then connect each station to its 10 nearest neighbors based on euclidean distance. While there are many options for how

[4]`https://cdiac.ess-dive.lbl.gov/ftp/ushcn_daily/`

[5]A script for pre-processing the USHCN data is available together with their code at `https://github.com/edebrouwer/gru_ode_bayes`.

to create this type of spatial graph, we have found the 10-nearest-neighbor approach to work sufficiently well in practice. Further investigating different methods for building spatial graphs is outside the scope of this paper. We additionally add edge weights to this graph following a similar method as Li et al. (2018) did for the traffic data. For sensor stations m and n at a distance $d_{m,n}$ we assign the edge (m, n) the weight

$$e_{m,n} = \exp\left(-\left(\frac{d_{m,n}}{4\sigma_e}\right)^2\right) \tag{20}$$

where σ_e is the standard deviation of all distances associated with edges. The constant 4 is chosen such that the furthest distance gets a weight close to 0 and the nearest distance a weight close to 1. For the USHCN data the only input feature is elapsed time since the node was last observed. We use the same 70%/10%/20% training/validation/test split as for the traffic data.

We build two datasets from the USHCN data, one with the daily minimum temperature $T_{\min}$ as target variable and one with the daily maximum temperature $T_{\max}$. The reason for separating these target variables, instead of using $d_y = 2$, is that the pre-processed data contains time points where only one of these is observed. Our model is not designed for such a setting where we do not observe all dimensions of $\boldsymbol{y}_i^n$. Extending TGNN4I to handle this is left as future work. The USHCN data also contains other climate variables, related to precipitation and snow coverage. These time series are less interesting to directly evaluate our model on, as many entries are just 0. Properly modeling these would require designing a suitable likelihood function and taking into account that some sensor stations never get any snow. As this would shift the focus from the core problem we choose to restrict our attention to $T_{\min}$ and $T_{\max}$.

D.3 Synthetic Periodic Data

To create the graph for the synthetic data we first sample 20 node positions uniformly over $[0, 1]^2$. We then create an undirected graph using a Delaunay triangulation (De Loera et al., 2010) based on these positions. This undirected graph is turned into a directed acyclic graph by choosing a random ordering of the nodes and removing edges going from nodes later in the ordering to those earlier.

Based on this graph 200 irregular time series are sampled according to Eq. 16 and 17. An example is shown in Figure 4. The irregular time steps are created by first discretizing $[0, 1]$ into 1000 time steps and then sampling 70 of these independently for each time series. Out of all node observations we keep 50%, sampled in the same way as for the traffic data. The parameters of the periodic signals are sampled according to $\phi^n \sim \mathcal{U}([20, 100])$ and $\eta^n \sim \mathcal{U}([0, 2\pi])$. We resample all η^n for each sequence, but sample the node frequencies ϕ^n only once, treating these as underlying properties of the nodes. Out of the 200 sampled sequences we use 100 for training, 50 for validation and 50 for testing.

E DETAILS ON EXPERIMENT SETUPS

We perform hyperparameter tuning by exhaustive grid search over combinations of parameter values. The configuration that achieves the lowest validation $\mathcal{L}_{\text{MSE}}$ is then used for the final experiment. An overview of hyperparameter values considered and the final configurations for all experiments is given in Table 5. All hyperparamter tuning for GRU-D and TGNN4I is done on the versions of the models with exponential dynamics. For these models we generally observe that larger architectures perform better, but the memory usage limits how much we can increase the model size. While the exact model architecture of TGNN4I does impact the results, the model does not seem particularly sensitive to learning rate or batch size.

We evaluate $\mathcal{L}_{\text{MSE}}$ on the validation set after each training epoch, stopping the training early if the validation error does not improve for 20 epochs. Except for LG-ODE we train all models until this early stopping occurs, which typically takes less than 150 epochs.

E.1 Traffic Data

For the traffic data we perform hyperparameter tuning on the versions of the datasets with 25% observations. The same hyperparameter configuration, shown in Table 5, performed the best for both PEMS-BAY and METR-LA.

We used the slightly downscaled version of the LG-ODE model for the traffic data. We tried also the default hyperparameters on the METR-LA dataset, but this did not improve the results. LG-ODE was trained for 50 epochs with a batch size of 8 and we observed no further improvements when trying to train the model for longer.

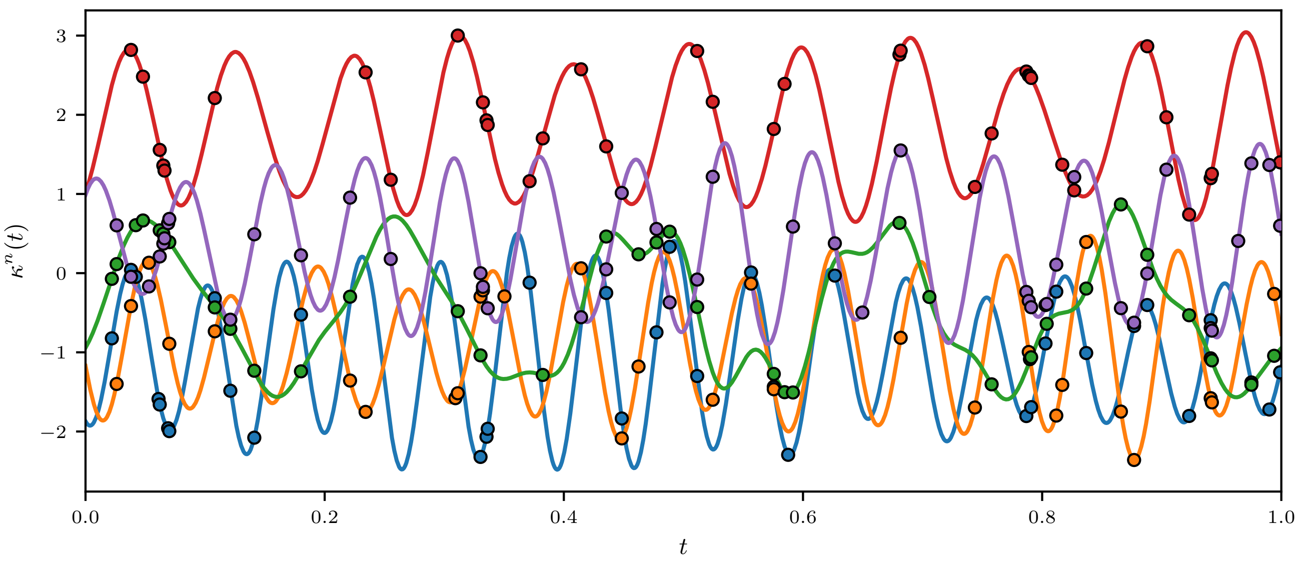


Figure 4: Examples of signals in the synthetic periodic data for 5 of the 20 nodes. The lines show clean signals $\kappa^n(t)$ and each dot a (noisy) observation $y^n(t_i)$. Note that the time steps are irregular and that not all signals are observed at each observation time.

E.2 USHCN Climate Data

For the USHCN data we perform the hyperparameter tuning on the T_{min} dataset. Since the USHCN graph contains many nodes we are somewhat more restricted in how large we can scale up the models.

E.3 Synthetic Periodic Data

On the periodic data we tried the same hyperparameters for GRU-D (joint) as for the other models, but no options gave any useful results. Because of this we exclude the model from this experiment. The number of iterations until the validation $\mathcal{L}_{\text{MSE}}$ stops decreasing is higher for the periodic data, with models training up to 500 epochs.

We used default hyperparameters for the LG-ODE model on the periodic data, but with a batch size of 16. As this graph only contains 20 nodes we were here able to train 5 LG-ODE models with different random seeds.

E.4 Loss Weighting

For the loss weighting experiment we do not perform any new hyperparameter tuning, but use the best configuration from the traffic data experiment. One TGNN4I model was trained for each weighting function. We here use a batch size of 4 and $N_{\text{init}} = 25$, which allows us to study predictions to higher Δ_t in the future.

Appendix References

De Loera, J. A., Rambau, J., and Santos, F. (2010). *Triangulations*, volume 25 of *Algorithms and Computation in Mathematics*. Springer.

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

Table 5: Values considered in hyperparameter tuning for the different datasets. Bold numbers represent the best performing configuration, that was then used in the final experiment.

	Learning rate	d_h	Fully connected layers in g	GNN layers in g	GNN layers in GRU	Batch size
Traffic Data						
GRU-D (joint)	**0.001**, 0.0005	64, 128, **256**, 512	1, **2**	-	-	**16**
GRU-D (node)	**0.001**, 0.0005	64, 128, **256**	1, **2**	-	-	**8**
TGNN4I	**0.001**, 0.0005	64, 128, **256**	1, **2**	1, **2**	1, **2**	**8**
Transformer (joint)	**0.001**	**64**, 256, 512, 2048	**2**, 4 (Transformer layers)			**8**
Transformer (node)	**0.001**	**64**, 128, 256	**2**, 4 (Transformer layers)			**8**
USHCN Data						
GRU-D (joint)	**0.001**	64, 128, 256, **512**	**2**	-	-	**16**
GRU-D (node)	**0.001**	32, 64, **128**	**2**	-	-	**4**
TGNN4I	**0.001**	32, 64, **128**	**2**	1, **2**	1, **2**	**4**
Transformer (joint)	**0.001**	**64**, 256, 512, 2048	2, **4** (Transformer layers)			**8**
Transformer (node)	**0.001**	**64**, 128	**2**, 4 (Transformer layers)			**8**
Periodic Data						
GRU-D (node)	**0.001**	64, **128**, 256	**2**	-	-	**16**
TGNN4I	**0.001**	64, **128**, 256	**2**	**2**	**2**	**16**
Transformer (joint)	**0.001**	64, **256**, 512, 2048	**2**, 4 (Transformer layers)			**8**
Transformer (node)	**0.001**	64, **128**, 256	2, **4** (Transformer layers)			**8**