Journal of Machine Learning Research 24 (2023) 1-36 Submitted 10/22; Revised 7/23; Published 8/23

Sparse Graph Learning from Spatiotemporal Time Series

Andrea Cini ANDREA.CINI@USI.CH
The Swiss AI Lab IDSIA
Università della Svizzera italiana
Lugano, CH

Daniele Zambon DANIELE.ZAMBON@USI.CH
The Swiss AI Lab IDSIA
Università della Svizzera italiana
Lugano, CH

Cesare Alippi CESARE.ALIPPI@USI.CH
The Swiss AI Lab IDSIA
Università della Svizzera italiana
Lugano, CH
Politecnico di Milano
Milan, IT

Editor: Silvia Chiappa

Abstract

Outstanding achievements of graph neural networks for spatiotemporal time series analysis show that relational constraints introduce an effective inductive bias into neural forecasting architectures. Often, however, the relational information characterizing the underlying data-generating process is unavailable and the practitioner is left with the problem of inferring from data which relational graph to use in the subsequent processing stages. We propose novel, principled—yet practical—probabilistic score-based methods that learn the relational dependencies as distributions over graphs while maximizing end-to-end the performance at task. The proposed graph learning framework is based on consolidated variance reduction techniques for Monte Carlo score-based gradient estimation, is theoretically grounded, and, as we show, effective in practice. In this paper, we focus on the time series forecasting problem and show that, by tailoring the gradient estimators to the graph learning problem, we are able to achieve state-of-the-art performance while controlling the sparsity of the learned graph and the computational scalability. We empirically assess the effectiveness of the proposed method on synthetic and real-world benchmarks, showing that the proposed solution can be used as a stand-alone graph identification procedure as well as a graph learning component of an end-to-end forecasting architecture.

Keywords: graph learning, spatiotemporal data, graph-based forecasting, time series forecasting, score-based learning, graph neural networks

1. Introduction

Traditional statistical and signal processing methods to time series analysis leverage on temporal dependencies to model data generating processes (Harvey et al., 1990). Graph signal processing methods extend these approaches to dependencies observed both in time and space, i.e., to the setting where temporal signals are observed over the nodes of a graph (Ortega et al., 2018; Stanković et al., 2020; Di Lorenzo et al., 2018; Isufi et al., 2019). The key ingredient here is the use of graph shift operators, constructed from the graph adjacency matrix, that localizes learned filters on the graph structure. The same holds true for graph deep learning methods that have revolutionized the landscape of machine learning for graphs (Bruna et al., 2014; Bronstein et al., 2017; Bacciu et al., 2020; Bronstein et al., 2021). However, it is often the case that no prior topological information about the reference graph is available, or that dependencies in the dynamics observed at different locations are not well modeled by the available spatial information (e.g., the physical proximity of the sensors). Examples can be found in social networks, smart grids, and brain networks, just to name a few relevant application domains.

The interest in the graph learning problem, in the context of spatiotemporal time series processing, indeed arises from many practical and theoretical concerns. In the first place, learning existing relationships among time series that better explain an observed phenomenon is worth the investigation on its own; as a matter of fact, graph identification is a well-known problem in graph signal processing (Mei and Moura, 2016; Variddhisai and Mandic, 2020). In the deep learning setting, several methods train, end-to-end, a graph learning module with a neural forecasting architecture to maximize performance on the downstream task (Shang and Chen, 2021; Wu et al., 2020). A typical deep learning approach consists in exploiting spatial attention mechanisms to discover the reciprocal salience of different spatial locations at each layer (Satorras et al., 2022; Rampášek et al., 2022). Graph learning, in this context, can then be seen as a regularization of Transformer-like models (Vaswani et al., 2017); regularization that comes in the form of the relational inductive biases typical of graph processing methods: namely, the sparsity of the pairwise relationships between nodes and the locality of the learned representations. In fact, despite their effectiveness, pure attention-based approaches impair two major benefits of graph-based learning: they (1) do not allow for the sparse computation enabled by the discrete nature of graphs and (2) do not take advantage of the structure, introduced by the graph topology, as an inductive bias for the learning system. Indeed, sparse computation allows graph neural networks (GNNs; Scarselli et al. 2008, Bacciu et al. 2020) with message-passing architectures (Gilmer et al., 2017) to scale in terms of network depth and the dimension of the graphs that are possible to process. At the same time, sparse graphs constrain learned representations to be localized in node space and mitigate over-fitting spurious correlations in the training data. Graph learning approaches that do attempt to learn relational structures from time series exist, but often rely on continuous relaxations of the binary adjacency matrix and, as a consequence, on dense computations to enable automatic reverse-mode differentiation through any subsequent processing (Shang and Chen, 2021; Kipf et al., 2018). Conversely, other solutions make the computation sparse (Wu et al., 2020; Deng and Hooi, 2021) at the expense of the quality of the gradient estimates as shown by Zügner et al. (2021). The challenge is, then, to provide accurate gradients while, at the same time, allowing for sparse computations in the downstream message-passing operations, typical of modern GNNs.

In this paper, we address the graph learning problem and model it from a probabilistic perspective which, besides naturally accounting for uncertainty and the embedding of priors, enables the learning of sparse graphs as realizations of a discrete probability distribution. In particular, given a set of time series, we seek to learn a parametric distribution $\boldsymbol{p}_\theta$ such that graphs sampled from $\boldsymbol{p}_\theta$ maximize the performance on the given downstream task, e.g., multistep-ahead forecasting. As an example, consider a cost function $\delta_t(\,\cdot\,)$ (e.g., the forecasting accuracy) associated with each time step t and dependant on the inferred graph. The core challenge in learning $\boldsymbol{p}_\theta$ to minimize the expected cost is associated with estimating the gradient

$$\nabla_\theta \mathbb{E}_{\boldsymbol{A}\sim\boldsymbol{p}_\theta}[\delta_t(\boldsymbol{A})] \tag{1}$$

of the expected value of the cost function $\delta_t(\boldsymbol{A})$ w.r.t the distributional parameters θ, the sampling of a random graph (adjacency matrix $\boldsymbol{A}$) from $\boldsymbol{p}_\theta$ and given batch of input-output data pairs corresponding to observations at time step t. Previous works proposing probabilistic methods (Shang and Chen, 2021; Kipf et al., 2018) learn $\boldsymbol{p}_\theta$ with *path-wise* gradient estimators (Glasserman and Ho, 1991; Kingma and Welling, 2013), i.e., by reparametrizing $\boldsymbol{A}\sim\boldsymbol{p}_\theta$ as $\boldsymbol{A}=g(\varepsilon,\theta)$, with deterministic function g decoupling parameters θ from the (parameter-free) random component $\varepsilon\sim\boldsymbol{p}_0$. However, these approaches imply approximating discrete distributions with a softmax continuous relaxation (Paulus et al., 2020) which makes *all* the downstream computations dense and quadratic in the number of nodes. Differently, here, we adopt the framework of *score-function* (SF) gradient estimators (Rubinstein, 1969; Williams, 1992; Mohamed et al., 2020) by relying on the rewriting of Equation (1) as

$$\nabla_\theta \mathbb{E}_{\boldsymbol{A}\sim\boldsymbol{p}_\theta}[\delta_t(\boldsymbol{A})] = \mathbb{E}_{\boldsymbol{A}\sim\boldsymbol{p}_\theta}[\delta_t(\boldsymbol{A})\nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A})] \tag{2}$$

which, as we detail in Section 5.1, allows us for preserving the sparsity of the sampled graphs and the scalability of the subsequent processing steps (e.g., the forward and backward passes of a message-passing network). In particular, our contributions are as follows.

- We provide an end-to-end methodological framework for probabilistic graph learning in spatiotemporal data, based on SF gradient estimators [Section 5] and design associated Monte Carlo (MC) estimators for stochastic message-passing architectures [Section 5.1].

- We introduce two parametrizations of $\boldsymbol{p}_\theta$ as 1) a set of Bernoulli distributions and as 2) the sampling *without* replacement of edges under a sparsity constraint [Section 5.2]. We show how to sample graphs from both distributions and derive the associated differentiable log-likelihood functions. Both distributions allow us to deal with an adaptive number of neighboring nodes.

- We propose a novel and effective, yet simple to implement, variance reduction method for the estimators [Section 6] based on the Fréchet mean graph w.r.t. the proposed distributions, for which we provide closed-form solutions [Propositions 1 and 3]. Our method does not require the estimation of additional parameters and, differently from more general-purpose approaches (e.g., see Mnih and Gregor (2014)), is as expensive as taking a sample from the considered distributions and evaluating the corresponding cost function.

- Finally, we present an approximate surrogate loss function [Section 7] derived from a convenient rewriting of the gradient for the considered settings [Proposition 5] which provides a considerable improvement in convergence rate.

Empirical results demonstrate that the techniques introduced here enable the use of score-based estimators to learn graphs from spatiotemporal time series; furthermore, experiments on time series forecasting benchmarks show that our approach compares favorably w.r.t. the state of the art. We strongly believe that our approach constitutes an effective method in the toolbox of the practitioner for designing new, even more effective, classes of novel graph-based time series processing architectures.

The paper is organized as follows. Section 2 discusses related works. Section 3 introduces relevant background material; Section 4 provides the formulation of the problem. We present the proposed parametrizations of $\boldsymbol{p}_\theta$ and related gradient estimators in Section 5 and the associated variance reduction techniques in Section 6. The proposed rewriting of the gradient and approximated objective are derived and discussed in Section 7. Finally, the empirical evaluation of the proposed method is given in Section 8 and conclusions are presented in Section 9.

2. Related Works

Graph neural networks have become increasingly popular in spatiotemporal time series processing (Seo et al., 2018; Li et al., 2018; Yu et al., 2018; Wu et al., 2019; Deng and Hooi, 2021; Cini et al., 2022; Marisca et al., 2022; Wu et al., 2022) and the graph learning problem is well-known within this context. Wu et al. (2019) propose Graph WaveNet, an architecture for time series forecasting that learns a weighted adjacency matrix $\boldsymbol{A} = \sigma\left(\boldsymbol{E}_1\boldsymbol{E}_2^\top\right)$ learned from the factorization with node embedding matrices $\boldsymbol{E}_1, \boldsymbol{E}_2$. Several other methods follow this direction (Bai et al., 2020; Oreshkin et al., 2021). Satorras et al. (2022) showed that hierarchical attention-based architectures are effective to account for dependencies among spatiotemporal time series to obtain accurate predictions in the downstream task. However, all the aforementioned approaches generally lead to dense graphs and cannot, therefore, exploit the sparsity and locality priors—and computational scalability—typical of graph-based machine learning. To address this issue, MTGNN (Wu et al., 2020) and GDN (Deng and Hooi, 2021) sparsify the learned factorized adjacency by selecting, for each node, the K edges associated with the largest weights. Using hard top-k operators, however, results in sparse gradients and has differentiability issues that can undermine the effectiveness of the learning procedure. More recently, Zhang et al. (2022) proposed a different approach based on the idea of sparsifying the learned graph by thresholding the average of learned attention scores across time steps.

Among probabilistic models, Franceschi et al. (2019) tackle the graph learning problem for non-temporal data by using a bi-level optimization routine and a straight-through gradient trick (Bengio et al., 2013) which, nonetheless, requires dense computations. The NRI approach, introduced by Kipf et al. (2018), learns a latent variable model predicting the interactions of physical objects by learning edge attributes of a fully connected (dense) graph. GTS (Shang and Chen, 2021) simplifies the NRI module by considering binary relationships only and integrates graph inference in a spatiotemporal recurrent graph neural network (Li et al., 2018). Both NRI and GTS exploit path-wise gradient estimators based on the categorical *Gumbel trick* (Maddison et al., 2017; Jang et al., 2017) and, as such, rely on continuous relaxations of discrete distributions and suffer from the computational setbacks anticipated in the introduction. Finally, the graph learning module proposed by

Kazi et al. (2022) uses the Gumbel-Top-K trick (Kool et al., 2019) to sample a K-nearest neighbors (K-NN) graph, where node scores are learned by using a heuristic for increasing the likelihood of sampling edges that contribute to correct classifications.

Besides applications in graph-based processing, the problem of learning discrete structures has been widely studied in deep learning and general machine learning (Niculae et al., 2023). As alternatives to methods relying on continuous relaxations and path-wise estimators (Jang et al., 2017; Maddison et al., 2017; Paulus et al., 2020), several approaches tackled the problem by exploiting score-based estimators and variance reduction techniques, e.g., based on control variates derived from continuous relaxations (Tucker et al., 2017; Grathwohl et al., 2018) and data-driven baselines (Mnih and Gregor, 2014). In particular, related to our method, Rennie et al. (2017) use a greedy baseline based on the mode of the distribution being learned, while Kool et al. (2020) constructs a variance-reduced estimator based on sampling without replacement from the discrete distribution. Beyond score-based and path-wise methods, Correia et al. (2020) take a different approach by considering *sparse* distributions where analytically computing the gradient becomes tractable. Niepert et al. (2021) introduce a class of (biased) estimators, based on maximum-likelihood estimation, that generalize the straight-through estimator (Bengio et al., 2013) to more complex distributions; Minervini et al. (2023) make such estimators adaptive to balance the bias of the estimator and the sparsity of the gradients. We refer to Mohamed et al. (2020) and Niculae et al. (2023) for an in-depth discussion of the topic. None of these method target specifically graph distributions, nor consider sparsity of the downstream computations as a requirement.

To the best of our knowledge, we are the first to propose a spatiotemporal graph learning module that relies on variance-reduced score-based gradient estimators specifically tailored for graph-based processing, and allowing for sparse computation in both training and inference phases of message-passing neural networks.

3. Preliminaries

The section introduces some preliminary concepts and provides the reference models and the notions regarding distributions over graphs needed to support the theoretical and technical derivations presented in the next sections.

3.1 Spatiotemporal Time Series with Graph Side Information

As reference case study, we consider spatiotemporal time series acquired from a sensor network. More specifically, consider a set $\mathcal{S} = \{1, 2, \dots, N\}$ of N sensors and indicate with $\boldsymbol{x}_t^i \in \mathbb{R}^{d_o}$ the d_o-dimensional observation acquired by the i-th sensor at discrete time step t. We denote by $\boldsymbol{X}_t \in \mathbb{R}^{N \times d_o}$ the matrix collecting all sensor observations $\{\boldsymbol{x}_t^i : i \in \mathcal{S}\}$ at time step t. Similarly, whenever available, $\boldsymbol{U}_t \in \mathbb{R}^{N \times d_u}$ indicates the d_u-dimensional exogenous variables and with $\boldsymbol{V} \in \mathbb{R}^{N \times d_v}$ static node attributes, e.g., sensor specific features. Assume that nodes (sensors) are available at all time steps and are identified, i.e., a node identifier can be paired to each sensor measurement over time. We also assume node features to be homogeneous across nodes, i.e., to correspond to the same types of sensor readings; an assumption that, however, can easily be relaxed in practice (e.g., see Schlichtkrull et al. 2018).

To account for dependencies among measurements at different nodes, observations can be paired with side relational information encoded by an edge set $\mathcal{E} \subseteq \mathcal{S} \times \mathcal{S}$ or, equivalently, by a (binary) adjacency matrix $\boldsymbol{A} \in \{0,1\}^{N \times N}$. Edges of the resulting graph can represent functional dependencies among the different time series that are instrumental for modeling the monitored system and solving the downstream task. To consider relations that change over time, e.g., as those between users of a social network, we can consider a *dynamic* adjacency matrix $\boldsymbol{A}_t$ (or edge set $\mathcal{E}_t$) representing the variable topology, differently from the *static* case. Finally, $\boldsymbol{y}_t \in \mathbb{R}^{d_y}$ denotes the target vector at every time step, i.e., the task-dependant value to be predicted; targets can also be associated with each node in which case we write $\boldsymbol{Y}_t \in \mathbb{R}^{N \times d_y}$. Often, we are interested in making predictions for a time horizon up to H steps ahead: notation $\boldsymbol{Y}_{t:t+H}$ denotes the multi-step targets in the interval $[t, t+H)$. Targets define the nature of *downstream task*, which can be either regression or classification, either at the graph or node level. In the following, we consider multi-step node-level tasks as the default setting.

The above framework is flexible enough to account for several application settings involving sensor measurements; the example below is provided to ease intuition for the reader.

Example 1 *Consider a sensor network monitoring the speed of vehicles at crossroads. In this case, $\boldsymbol{X}_{1:T}$ refers to traffic speed measurements sampled at a certain frequency. Exogenous variables $\boldsymbol{U}_t$ account for time-of-the-day and day-of-the-week identifiers and, eventually, the current state of traffic lights. The node-attribute matrix $\boldsymbol{V}$ reports static features regarding the type of road a sensor is placed in. An adjacency matrix $\boldsymbol{A}$ can be obtained by considering each pair of sensors connected if and only if they are connected by a road segment. Targets $\boldsymbol{Y}_t$ provide labels for the task of predicting whether a traffic jam will happen in a fixed number of future time steps or simply one could consider the task of forecasting the next H measurements at each sensor, i.e., $\boldsymbol{Y}_{t:t+H} = \boldsymbol{X}_{t:t+H}$.*

3.2 Spatiotemporal Graph Neural Networks

The subsection provides an overview of the architectures considered in the sequel. We look at a general class of message-passing operators as well as spatiotemporal graph neural network (STGNN) architectures.

3.2.1 Message-Passing Neural Networks

We consider the family of message-passing (MP; Gilmer et al. 2017) operators where representations are updated at each layer l such as

$$\boldsymbol{z}_t^{i,(l)} = \rho^{(l)}\Big(\boldsymbol{z}_t^{i,(l-1)}, \textsc{Aggr}\Big\{\gamma^{(l)}\big(\boldsymbol{z}_t^{j,(l-1)}, \boldsymbol{z}_t^{i,(l-1)}, \boldsymbol{e}_{i,j}\big); j \in \mathcal{N}(i)\Big\}\Big) \tag{3}$$

where $\boldsymbol{z}_t^{i,(l)}$ indicates the representation of the i-th node at layer l; $\mathcal{N}(i)$ is the set of its neighboring nodes, and $\boldsymbol{e}_{i,j}$ are the features associated with the edge connecting the j-th to the i-th node. Update and message functions, ρ and γ, respectively, can be implemented by any differentiable function—e.g., a multilayer perceptron—while $\textsc{Aggr}\{\cdot\}$ indicates a generic permutation invariant aggregation function. By considering a graph-wise operator, the l-th message-passing neural network layer (MPNN) of the—possibly deep—architecture can be represented in a compact way as

$$\boldsymbol{Z}_t^{(l)} = \text{MPNN}^{(l)}\left(\boldsymbol{Z}_t^{(l-1)}, \boldsymbol{A}\right). \tag{4}$$

3.2.2 Spatiotemporal Architectures

STGNNs process input spatiotemporal data by considering operators that use the underlying graph to impose inductive biases in the representation learning process. By adopting a terminology similar to the one introduced in (Gao and Ribeiro, 2022), we distinguish between time-then-space (TTS) and time-and-space (T&S) STGNNs, depending on whether message-passing is carried out after or in-between a temporal encoding step.

Time-then-space models. TTS models are based on an encoder-decoder architecture where the encoder embeds each input time series $\boldsymbol{x}^i_{t-W:t}$ associated with a graph node to a vector representation, while the decoder, implemented as a multilayer GNN, propagates information across the spatial dimension. In particular, we consider the family of models s.t.

$$\boldsymbol{Z}_t^{(0)} = \textsc{TemporalEncoder}\left(\boldsymbol{X}_{t-W:t}, \boldsymbol{U}_{t-W:t}, \boldsymbol{V}\right), \tag{5}$$

$$\boldsymbol{Z}_t^{(l)} = \text{MPNN}^{(l)}\left(\boldsymbol{Z}_t^{(l-1)}, \boldsymbol{A}\right), \quad \forall\, l = 1, \dots, L \tag{6}$$

$$\widehat{\boldsymbol{Y}}_{t:t+H} = \textsc{Readout}\left(\boldsymbol{Z}_t^{L}\right), \tag{7}$$

where the notation is consistent with that of Equation (3). Examples of spatiotemporal graph processing models that fall into the time-then-space category are NRI (Kipf et al., 2018) and the encoder-decoder architecture introduced by Satorras et al. (2022).

Time-and-space models. Time-and-space models are a general class of STGNNs where space and time are processed by operators that process representation along the time and space dimensions. A large subset of this family of models can be seen as performing the following operations

$$\boldsymbol{Z}_{t-W:t}^{(0)} = \left[\boldsymbol{X}_{t-W:t} || \boldsymbol{U}_{t-W:t} || \boldsymbol{V}\right], \tag{8}$$

then, for every layer $l = 1, \dots, L$,

$$\boldsymbol{H}_{t-W:t}^{(l)} = \textsc{TemporalLayer}^{(l)}\left(\boldsymbol{Z}_{t-W:t}^{(l-1)}\right), \tag{9}$$

$$\boldsymbol{Z}_k^{(l)} = \text{MPNN}^{(l)}\left(\boldsymbol{H}_k^{(l)}, \boldsymbol{A}\right), \qquad \forall\, k = t-W, \dots, t-1 \tag{10}$$

finally followed by

$$\widehat{\boldsymbol{Y}}_{t:t+H} = \textsc{Readout}\left(\textsc{Aggr}\left\{\boldsymbol{Z}_{t-W}^{(L)}, \dots, \boldsymbol{Z}_{t-1}^{(L)}\right\}\right), \tag{11}$$

where $\textsc{TemporalLayer}\left(\,\cdot\,\right)$ indicates a generic (parametric) operator processing representations across the different time steps, e.g., a 1-D convolutional layer. Note that predictions, here, are obtained in Equation (11) by pooling representations along the temporal dimension and then using, e.g., a linear readout. Other architectures are possible, e.g., by exploiting recurrent neural networks (Seo et al., 2018; Li et al., 2018).

3.3 Mean Adjacency Matrices

In this section, we recall some definitions related to probability distributions over graph data. The discrete nature of graphs makes a large part of the well-established results from probability and statistics unsuitable for objects that do not adhere to Euclidean geometry. An example is the notion of "expected" graph that is of interest to the present paper [Section 6] and whose definition needs to be extended. Here, we do so by following Fréchet (1948).

For a random vector $\boldsymbol{x} \in \mathbb{R}^d$ characterized by probability density function $\boldsymbol{p}$, expectation $\mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{p}}[\boldsymbol{x}] = \int \boldsymbol{x}\, \boldsymbol{p}(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}$ is a weighted average over $\boldsymbol{x}$; we interchangeably adopt forms $\mathbb{E}_{\boldsymbol{p}}[\boldsymbol{x}]$ and $\mathbb{E}[\boldsymbol{x}]$. Notably, $\mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{p}}[\boldsymbol{x}]$ can be equivalently written as

$$\mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{p}}[\boldsymbol{x}] = \operatorname*{arg\,min}_{\boldsymbol{x}'\in\mathbb{R}^d} \mathfrak{F}_2(\boldsymbol{x}'), \tag{12}$$

where $\mathfrak{F}_2(\,\cdot\,)$ denotes the *Fréchet function*

$$\mathfrak{F}_2(\boldsymbol{x}') \triangleq \mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{p}}\left[\|\boldsymbol{x}' - \boldsymbol{x}\|_2^2\right] \tag{13}$$

associated with distribution $\boldsymbol{p}$ and the squared Euclidean distance $\|\cdot\|_2^2$. Following Equations 12 and 13, we can derive a generalized definition of mean applicable to non-Euclidean data, like graphs and sparse adjacency matrices. We comment that, following this line, we can extend these results also to the sample mean $1/M \sum_{m=1}^{M} \boldsymbol{x}_m$ of a finite sample $\mathcal{D} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_M\}$, and define accordingly the Fréchet sample mean of a sample of non-Euclidean data.

Consider, then, the space $\mathcal{A} \subseteq \{0,1\}^{N\times N}$ of adjacency matrices $\boldsymbol{A}$ over the node (sensor) set $\mathcal{S}$, each of which representing a graph topology over $\mathcal{S}$; for instance, for undirected graphs, $\mathcal{A}$ is the subset of $\{0,1\}^{N\times N}$ of symmetric matrices, whereas for directed k-NN graphs

$$\mathcal{A} = \left\{ \boldsymbol{A} \in \{0,1\}^{N\times N} : \sum_{j=1}^{N} \boldsymbol{A}_{i,j} = k,\ \forall\, i \right\}. \tag{14}$$

By equipping $\mathcal{A}$ with a metric distance, we define a Fréchet function analogous to that of Equation (13), applicable to random adjacency matrices. In this paper, we consider the Hamming distance

$$H(\boldsymbol{A}, \boldsymbol{A}') \triangleq \sum_{i,j=1}^{N} I(\boldsymbol{A}_{i,j} \neq \boldsymbol{A}'_{i,j}), \tag{15}$$

where $\boldsymbol{A}, \boldsymbol{A}' \in \mathcal{A}$ and I is the indicator function such that $I(a) = 1$, if a is true, 0 otherwise. The Hamming distance counts the number of mismatches between the entries of $\boldsymbol{A}$ and $\boldsymbol{A}'$, and is then a natural choice to measure the dissimilarity between two graphs.

We define the Fréchet function over space $(\mathcal{A}, H)$, and the random adjacency matrix $\boldsymbol{A} \sim \boldsymbol{p}$, for all $\boldsymbol{A}' \in \mathcal{A}$ as

$$\mathfrak{F}_H(\boldsymbol{A}') \triangleq \mathbb{E}_{\boldsymbol{A}\sim\boldsymbol{p}}\left[H(\boldsymbol{A}', \boldsymbol{A})\right]. \tag{16}$$

According to Equation 12, we then define *Fréchet mean adjacency matrix* any matrix

$$\boldsymbol{A}^{\mu} \in \operatorname*{arg\,min}_{\boldsymbol{A}'\in\mathcal{A}} \mathfrak{F}_H(\boldsymbol{A}'). \tag{17}$$

A matrix $\boldsymbol{A}^{\mu}$ always exists in $\mathcal{A}$, as $\mathcal{A}$ is a finite set, but, in general, is not unique. Conditions for the uniqueness of the Fréchet mean in the context of graph-structured data have been studied in the literature, e.g., by Jain (2016). Throughout the paper, we use the term "Fréchet mean" referring to *any* Fréchet mean of a given distribution.

4. Problem Formulation

This section provides a probabilistic formulation of the graph learning problem in spatiotemporal time series and defines the operational framework in which we operate.

4.1 Graph Learning from Spatiotemporal Time Series

Given a window of W past observations $\mathcal{X}_{t-W:t} = \langle \boldsymbol{X}_{t-W:t}, \boldsymbol{U}_{t-W:t}, \boldsymbol{V} \rangle$ open on the time series, we consider the problem of predicting H future targets $\boldsymbol{Y}_{t:t+H}$ associated with the graph nodes. The notation $t:T$ denotes the time steps in interval $[t, T)$; when not differently specified, we consider the multistep-ahead forecasting task $\boldsymbol{Y}_{t:t+H} = \boldsymbol{X}_{t:t+H}$.

Consider the family of predictive models F_{ψ} and parametric probability distribution $\boldsymbol{p}_{\theta}$ over graphs

$$\widehat{\boldsymbol{Y}}_{t:t+H} = F_{\psi}\left(\mathcal{X}_{t-W:t}\,,\, \boldsymbol{A}_t\right), \qquad \boldsymbol{A}_t \sim \boldsymbol{p}_{\theta}\left(\boldsymbol{A} | \mathcal{X}_{t-W:t}\right), \tag{18}$$

where ψ, θ are the model parameters. The joint graph and model learning problem consists in jointly learning parameters ψ, θ by solving the optimization problem

$$\hat{\psi}, \hat{\theta} = \underset{\psi,\theta}{\arg\min} \frac{1}{T} \sum_{t=1}^{T} \mathcal{L}_t\left(\psi, \theta\right), \qquad \mathcal{L}_t\left(\psi, \theta\right) \triangleq \mathbb{E}_{\boldsymbol{A} \sim \boldsymbol{p}_{\theta}}\left[\delta_t(\boldsymbol{A}_t; \psi)\right], \tag{19}$$

where $\mathcal{L}_t\left(\psi, \theta\right)$ is the optimization objective at time step t expressed as the expectation, over the graph distribution $\boldsymbol{p}_{\theta}$, of a cost—loss—function $\delta_t(\boldsymbol{A}_t; \psi)$, typically a p-norm

$$\delta_t(\boldsymbol{A}_t; \psi) = \|\boldsymbol{Y}_{t:t+H} - F_{\psi}\left(\mathcal{X}_{t-W:t}, \boldsymbol{A}_t\right)\|_p^p \tag{20}$$

with, e.g., $p = 1$ or 2. Note that in Equation (18) the distribution of $\boldsymbol{A}_t$ at time step t is conditioned on the most recent observations $\mathcal{X}_{t-W:t}$, hence modeling a scenario associated with a dynamic graph distribution [Section 3.1]. A static graph scenario follows by simply removing the conditioning on $\mathcal{X}_{t-W:t}$. We consider a generic family of predictive models F_{ψ} implemented by STGNNs based on the message-passing framework and following either the TTS or the T&S paradigm to process information along space and time. Other architectures can be considered. Notably, F_{ψ} can be suitably designed in order to exchange messages w.r.t. a different graph $\boldsymbol{A}^{(l)}$ at each MP layer. Section 7 provides a thorough discussion of this setup.

In this setting, the model family and the downstream task impact on the type of relationships being learned. For example, linear and nonlinear models will yield different results that depend also on the number of layers and the choice of MP operators, e.g., standard graph convolutions against anisotropic message-passing layers such as those used in graph attention networks (Veličković et al., 2018). Ultimately, the learned graph distribution is the one that best explains the observed data given the architecture of the predictive model and the family of graph distributions. Different parametrizations of $\boldsymbol{p}_{\theta}$ allow the practitioner for embedding different inductive biases (such as sparsity) as structural priors into the processing.

4.2 Core Challenge

Minimizing the sum of expectations $\mathcal{L}_t\left(\psi,\theta\right)$, $t=1,\dots,T$, is challenging, as it involves estimating the gradients $\nabla_\theta\mathcal{L}_t\left(\psi,\theta\right)$ w.r.t. the parameters of the discrete distribution $\boldsymbol{p}_\theta$ over (binary) adjacency matrices. Sampling matrices (graphs) $\boldsymbol{A}\sim\boldsymbol{p}_\theta$ throughout the learning process results in a stochastic computational graph (CG) and, while automatic differentiation of CGs is a core component of modern deep learning libraries (Paszke et al., 2019; Abadi et al., 2015), dealing with stochastic nodes introduces additional challenges as the gradients have to be estimated w.r.t. expectations over the sampling of the associated random variables (Schulman et al., 2015; Weber et al., 2019; Mohamed et al., 2020). Tools for automatic differentiation of stochastic CGs are being developed (Foerster et al., 2018; Bingham et al., 2019; van Krieken et al., 2021; Dillon et al., 2017); however, general approaches can be ineffective and prone to failure, especially in the case of discrete distributions (see also Mohamed et al. 2020).

In our setup, having a stochastic message-passing graph (MPG) emerges as problematic: the MP paradigm constrains the flow of spatial information, making the CG dependent on the MPG. Moreover, a stochastic input MPG introduces N^2 stochastic nodes in the resulting CG (i.e., one for each potential edge in MPG), leading to a large number of paths data can flow through. For instance, by considering an L-layered architecture, the number of stochastic nodes can increase up to $\mathcal{O}(LN^2)$, making the design of reliable, low-variance—i.e., effective—MC gradient estimators inherently challenging. Furthermore, as already mentioned, computing gradients associated with each stochastic edge introduce additional challenges w.r.t. time and space complexity; further discussion and actionable directions are given in the next section.

5. Score-based Sparse Graph Learning from Spatiotemporal Time Series

In this section, we present our approach to probabilistic graph learning. After introducing score-based gradient estimators [Section 5.1], we propose two graph distribution models [Section 5.2] and comment on their practical implementations [Section 5.3]. The problem of controlling the variance of the estimators is discussed together with novel and principled variance reduction techniques tailored to graph-based architectures [Section 6]. Finally, we provide a convenient rewriting of the gradient for L-layered MP architectures leading to a novel surrogate loss [Section 7]. Figure 1 provides a schematic overview of the framework. In particular, the block on the left shows the graph learning module, where $\boldsymbol{A}$ is sampled from $\boldsymbol{p}_\theta$; as the figure suggests, depending on the parametrization of $\boldsymbol{p}_\theta$, some components of $\boldsymbol{A}$ can be sampled independently. The bottom of the figure, instead, shows the predictive model F_ψ that, given the sampled graph and the input window, outputs the predictions used to estimate $\mathcal{L}_t\left(\psi,\theta\right)$, whose gradient provides the learning signals.

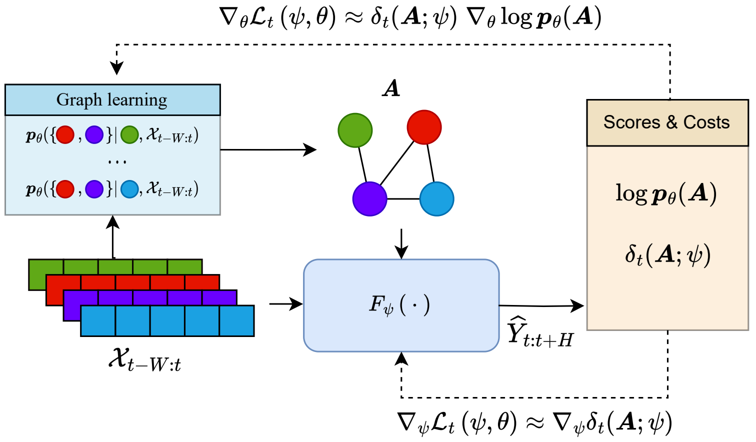

$\nabla_\theta \mathcal{L}_t\left(\psi, \theta\right) \approx \delta_t(\boldsymbol{A};\psi)\ \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A})$
Graph learning
$\boldsymbol{p}_\theta(\{●,●\}|●, \mathcal{X}_{t-W:t})$
…
$\boldsymbol{p}_\theta(\{●,●\}|●, \mathcal{X}_{t-W:t})$
$\boldsymbol{A}$
Scores & Costs
$\log \boldsymbol{p}_\theta(\boldsymbol{A})$
$\delta_t(\boldsymbol{A};\psi)$
$F_\psi(\,\cdot\,)$
$\widehat{Y}_{t:t+H}$
$\mathcal{X}_{t-W:t}$
$\nabla_\psi \mathcal{L}_t\left(\psi, \theta\right) \approx \nabla_\psi \delta_t(\boldsymbol{A};\psi)$

Figure 1: Overview of the learning architecture. The graph learning module samples a graph used to propagate information along the spatial dimension in F_ψ; predictions and samples are used to compute costs and log-likelihoods. Gradient estimates are propagated back to the respective modules.

5.1 Estimating Gradients for Stochastic Message-Passing Networks

SF estimators are based on the identity

$$\nabla_\theta \mathbb{E}_{\boldsymbol{p}_\theta}[f(x)] = \nabla_\theta \int f(x)\boldsymbol{p}_\theta(x)\,dx = \int f(x)\nabla_\theta \boldsymbol{p}_\theta(x)\,dx \tag{21}$$

$$= \int f(x)\boldsymbol{p}_\theta(x)\nabla_\theta \log \boldsymbol{p}_\theta(x)\,dx = \mathbb{E}_{\boldsymbol{p}_\theta}[f(x)\nabla_\theta \log \boldsymbol{p}_\theta(x)], \tag{22}$$

which holds—under mild assumptions[1]—for generic cost functions f and distributions $\boldsymbol{p}_\theta$. The rewriting of $\nabla_\theta \mathbb{E}_{\boldsymbol{p}_\theta}[f(x)]$ in terms of the gradient of the *score function* $\log \boldsymbol{p}_\theta(\,\cdot\,)$ allows for estimating the gradients easily by MC sampling and backpropagating them through the computation of the score function. SF estimators are black-box optimization methods, i.e., they only require to *evaluate* pointwise the *cost* $f(x)$ which does not necessary need to be differentiable w.r.t. parameters θ. In our setup, by assuming disjoint ψ and θ, Equation (22) becomes

$$\nabla_\theta \mathcal{L}_t\left(\psi, \theta\right) = \nabla_\theta \mathbb{E}_{\boldsymbol{p}_\theta}\left[\delta_t(\boldsymbol{A};\psi)\right] = \mathbb{E}_{\boldsymbol{p}_\theta}\left[\delta_t(\boldsymbol{A};\psi)\nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A})\right], \tag{23}$$

allowing for computing gradients w.r.t. the graph generative process without requiring a full evaluation of all the stochastic nodes in the CG.

Sparse computation. Path-wise gradient estimators tackle the problem of estimating the gradient $\nabla_\theta \mathbb{E}_{\boldsymbol{p}_\theta}\left[\delta_t(\boldsymbol{A};\psi)\right]$ by exploiting continuous relaxations of the discrete $\boldsymbol{p}_\theta$, thus estimating the gradient by differentiating through all nodes of the stochastic CG. Defined

[1] The identity is valid as long as $\boldsymbol{p}_\theta$ and f allow for the interchange of differentiation and integration in Equation (21); see L'Ecuyer (1995); Mohamed et al. (2020).

E to be the number of edges in a realization of $\boldsymbol{p}_\theta$, the cost of learning a graph with a path-wise estimator is that of making any subsequent MP operation scale with $\mathcal{O}(LN^2)$, instead of the $\mathcal{O}(LE)$ complexity that would have been possible with a sparse computational graph. The outcome is even more dramatic if we consider T&S models where MP is used for propagating information at each time step, thus making the computational and memory complexity $\mathcal{O}(LTN^2)$, which would be unsustainable for any practical application at scale. Conversely, the proposed score-based methods allow for the implementation of MP operators with efficient scatter-gather operations that exploit the sparsity of $\boldsymbol{A}$, thus resulting in an $\mathcal{O}(LE)$ complexity.

5.2 Graph Distributions, Graphs Sampling, and Graphs Likelihood

The distribution $\boldsymbol{p}_\theta$ should be chosen to (i) efficiently sample graphs and evaluate their likelihood and (ii) backpropagate the errors through the computation of the score [Equation (23)] to parameters θ. In the following, we consider graph distributions s.t. each stochastic edge $j \rightarrow i$ is associated with a weight $\Phi_{i,j}$. The considered distributional parameters $\Phi \in \mathbb{R}^{N \times N}$ can then be learned as a function of the learnable parameters θ. In the case of static graphs, we can directly consider $\Phi = \theta$; however, to account for the dynamic case, more complex parametrizations are possible, e.g., by exploiting amortized inference to condition distribution $\boldsymbol{p}_\theta$ on the observed values. Further discussion is deferred to the end of the section.

5.2.1 Binary Edge Sampler

A straightforward approach considers a Bernoulli random variable of parameter $\sigma(\Phi_{i,j})$ associated with each potential edge $j \rightarrow i$. We refer to this graph learning module as *binary edge sampler* (BES).

Sampling. For all pairs of sensors $i, j \in \mathcal{S}$, the corresponding entries $\boldsymbol{A}_{i,j}$ of $\boldsymbol{A}$ can be sampled independently from the associated distribution since $\boldsymbol{A}_{i,j} \sim \text{Bernoulli}(\sigma(\Phi_{i,j}))$. Here, the sampling from $\boldsymbol{p}_\theta$ can be done efficiently and is highly parallelizable.

Log-likelihood evaluation. Computing the log-likelihood of a sample is cheap and differentiable as it corresponds to evaluating the binary cross-entropy between the sampled entries of $\boldsymbol{A}$ and the corresponding parameters $\sigma(\Phi)$ of the Bernoulli distributions, i.e,

$$\log \boldsymbol{p}_\theta(\boldsymbol{A}) = \sum_{i,j}^{N} \boldsymbol{A}_{i,j} \log(\sigma(\Phi_{i,j})) + (1 - \boldsymbol{A}_{i,j}) \log(1 - \sigma(\Phi_{i,j})). \tag{24}$$

Sparsity priors can then be imposed by regularizing Φ, e.g., by adding a Kullback-Leibler regularization term to the loss (Shang and Chen, 2021; Kipf et al., 2018). Graph generators like BES are a common choice in the literature (Franceschi et al., 2019; Shang and Chen, 2021) as the independence assumption makes the mathematics amenable and avoids the often combinatorial complexity of dealing with more structured distributions. In the experimental sections, we demonstrate that even simple parametrizations like BES can be effective with the proposed score-based learning.

5.2.2 Subset Neighborhood Sampler

Encoding structural priors about the sparseness of the graphs directly into $\boldsymbol{p}_\theta$ is often desirable and might allow—depending on the problem—to remarkably reduce sample complexity. In this section, we use the score matrix $\Phi \in \mathbb{R}^{N \times N}$ to parametrize a stochastic top-k sampler that we dub *subset neighborhood sampler* (SNS). For each n-th node, we sample a subset $S_K \subset \mathcal{S} = \{1, \dots, N\}$ of K neighboring nodes by sampling *without replacement* from a categorical distribution parametrized by *normalized log-probabilities* $\Phi_{n,:}$. The probability of sampling neighborhood S_K for each node n is given by

$$\boldsymbol{p}_\theta(S_K|n) = \sum_{\vec{S}_K \in \mathcal{P}(S_K)} \boldsymbol{p}_\theta(\vec{S}_K|n) = \sum_{\vec{S}_K \in \mathcal{P}(S_K)} \prod_{j \in \vec{S}_K} \frac{\exp(\Phi_{n,j})}{1 - \sum_{k<j} \exp(\Phi_{n,k})}, \tag{25}$$

where $\vec{S}_K$ denotes an ordered sample without replacement and $\mathcal{P}(S_K)$ is the set of all the permutations of S_K.

Sampling. Sampling can be done efficiently by exploiting the *Gumbel-top-k trick* (Kool et al., 2019). Accordingly, we consider the parameter vector $\phi = \Phi_{n,:}$ and denote with $[G_{\phi_1}, \dots, G_{\phi_N}]$ the associated random vector of independent Gumbel random variables $G_{\phi_j} \sim \text{Gumbel}(\phi_j)$; given a realization thereof $[g_1, \dots, g_N]$, it is possible to show that $S_K = \arg\text{top-K}\{g_i : i \in \mathcal{S}\}$ follows the desired distribution (Kool et al., 2019).

Log-likelihood evaluation. Evaluating the score function is more challenging; in fact, Equation (25) shows that directly computing $\boldsymbol{p}_\theta(S_K|n)$ requires marginalizing over all the possible $K!$ orderings of S_K. While exploiting the Gumbel-max trick can bring down computation to $\mathcal{O}(2^K)$ (Huijben et al., 2022; Kool et al., 2020), exact computation remains untractable for any practical application. Luckily, $\boldsymbol{p}_\theta(S_K|n)$ can be approximated efficiently using numerical integration. Following the notation of Kool et al. (2019, 2020), for a subset $B \in \mathcal{S}$ we define

$$\underset{i \in B}{\textsc{LogSumExp}}(\phi_i) \triangleq \log\left(\sum_{i \in B} \exp \phi_i\right), \tag{26}$$

we use the notation $\phi_B = \textsc{LogSumExp}_{i \in B}\, \phi$, and indicate with f_u and $\mathcal{F}_u$ the p.d.f. and c.d.f., respectively, of a Gumbel random variable Gumbel(u) with location parameter u. Recall that $\mathcal{F}_u(z) = \exp(-\exp(-z+u))$ and the following property of Gumbel random variables:

$$G_{\phi_B} \triangleq \max_{i \in B} G_{\phi_i} \sim \text{Gumbel}(\phi_B). \tag{27}$$

With a derivation analogous to that of Kool et al. (2020), Equation (25) can be conveniently rewritten by exploiting the property shown in Equation (27) as:

$$\boldsymbol{p}_\theta(S_K|n) = \mathbb{P}\left(\min_{i \in S_K} G_{\phi_i} > \max_{i \in \mathcal{S} \setminus S_k} G_{\phi_i}\right) \tag{28}$$

$$= \mathbb{P}\left(G_{\phi_i} > G_{\phi_{\mathcal{S} \setminus S_k}}, \forall i \in S_K\right) \tag{29}$$

$$= \int_{-\infty}^{\infty} \prod_{i \in S_K} \left(1 - \mathcal{F}_{\phi_i}(g)\right) f_{\phi_{\mathcal{S} \setminus S_k}}(g)\, dg \tag{30}$$

With an appropriate change (details in Appendix B), the integral can be rewritten as

$$\boldsymbol{p}_\theta(S_K|n) = \exp\left(\phi_{\mathcal{S}\setminus S_K} + c\right) \int_0^1 u^{\exp\left(\phi_{\mathcal{S}\setminus S_K}+c\right)-1} \prod_{i\in S_k} \left(1 - u^{\exp(\phi_i + c)}\right) du, \tag{31}$$

where c is a conditioning constant. We then approximate the integral in Equation (31) by using the trapezoidal rule as

$$\begin{aligned}\log \boldsymbol{p}_\theta(S_K|n) \approx{} & \log(\Delta u) + \phi_{\mathcal{S}\setminus S_K} + c \\ & + \underset{m=1,\dots,M-1}{\textsc{LogSumExp}} \left(\left(\exp\left(\phi_{\mathcal{S}\setminus S_K} + c\right) - 1\right) \log(u_m) + \sum_{i\in S_K} \log\left(1 - u_m^{\exp(\phi_i+c)}\right)\right),\end{aligned} \tag{32}$$

with M trapezoids and equally spaced intervals of length Δu; the integrands are computed in log-space—with a computational complexity of $\mathcal{O}(MK)$—for numeric stability. The expression in Equation (32) provides, then, a differentiable numeric approximation of the SNS log-likelihood which can be used for backpropagation.

As previously discussed, the proposed SNS method allows for embedding structural priors on the sparsity of the latent graph directly into the generative model. Fixing the number K of neighbors might, however, introduce an irreducible approximation error when learning graphs with nodes characterized by a variable number of neighbors. We solve this problem by adding dummy nodes.

Adaptive number of neighbors. Given K, we add up to $K-1$ dummy nodes to set $\mathcal{S}$ (i.e. the set of candidate neighbors) and expand matrix Φ accordingly. At this point, a neighborhood of exactly K nodes can be sampled and the log-likelihood evaluated according to the procedure described above; however, dummy nodes are discarded to obtain the $N \times N$ adjacency matrix $\boldsymbol{A}$. By doing so, hyperparameter K can also be used to cap the maximum number of edges and set a minimum sparsity threshold. The resulting computational complexity in the subsequent MP layers is at most $\mathcal{O}(NK)$.

5.3 Learning the Parameters of the Graph Distribution $\boldsymbol{p}_\theta$

As previously mentioned, for both BES and SNS, we can parametrize $\boldsymbol{p}_\theta$ by associating a score $\Phi_{i,j}$ to each edge $j \to i$; i.e., by setting $\Phi = \theta$. Similarly, one could reduce the number of parameters to estimate from N^2 to $2dN$, with $d \ll N$, by using amortized inference and learning some factorization of Φ, e.g., $\Phi = \theta_s\theta_t^\top$ where $\theta_s, \theta_t \in \mathbb{R}^{N\times d}$ (e.g., see Kipf and Welling 2016; Kipf et al. 2018). Modeling dynamic graphs instead requires accounting for observations $\mathcal{X}_{t-W:t}$ at each considered time step t. For example, one can consider models s.t.

$$\boldsymbol{h}_t^i = \textsc{Encoder}\left(\boldsymbol{x}_{t-W:t}^i, \boldsymbol{u}_{t-W:t}^i, \boldsymbol{v}^i\right), \qquad \phi_{i,j} = \boldsymbol{a}^\top \sigma\left(\boldsymbol{W}[\boldsymbol{h}_t^i || \boldsymbol{h}_t^j] + \boldsymbol{b}\right), \tag{33}$$

where $\textsc{Encoder}(\cdot)$ indicates a generic encoding function for in the input window (e.g., an MLP or an RNN), σ a nonlinear activation function, $\boldsymbol{W} \in \mathbb{R}^{d\times 2d_h}$ is a learnable weight matrix, $\boldsymbol{b} \in d$ a learnable bias and $\boldsymbol{a} \in \mathbb{R}^d$ the learnable parameters of the output linear transformation.

6. Reducing the Variance of the Estimator

MC estimation is the most commonly used technique to approximate the gradient in Equation (23). Although MC estimators are unbiased, the quality of the estimate can be dramatically impacted by its variance: as such, variance reduction is a critical step in the use of score-based estimators. As for any MC estimator, a direct method to reduce the variance consists in increasing the number M of independent samples used to compute the estimator, which results in reducing the variance by a factor $1/M$ w.r.t. the one-sample estimator. In our setting, sampling M adjacency matrices results in M evaluations of the cost and the associated score and, in turn, to an often non-negligible computational overhead. In this section, we provide more sample-efficient alternatives, based on the control variates method. Our approach grants a significant variance reduction while requiring only one extra evaluation of the cost function. That being said, our approach to variance reduction is orthogonal to increasing the sample size, which remains viable to further improve the quality of the gradient estimator.

6.1 Control Variates and Baselines

The control variates method provides a variance reduction method for MC estimator of $\mathbb{E}_{\boldsymbol{p}_\theta}[g(x)]$. It consists in introducing an auxiliary quantity $h(x)$ for which we know how to efficiently compute the expectation under the sampling distribution $\boldsymbol{p}_\theta$ (Mohamed et al., 2020). Then, a function $\tilde{g} = g - \beta(h - \mathbb{E}[h])$ is defined, for some constant β, such that $\tilde{g}$ has the same expected value of g, i.e., $\mathbb{E}[\tilde{g}(x)] = \mathbb{E}[g(x)]$, but lower variance ($\text{Var}[\tilde{g}(x)] < \text{Var}[g(x)]$). Quantity h is called *control variate*, while β is often referred to as *baseline*. In score-based methods, a computationally cheap choice is to use the score function itself as control variate, i.e., referring to our case where $g(\boldsymbol{A}) \triangleq \delta_t(\boldsymbol{A};\psi)\nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A})$ (Equation (23)), we set $h(\boldsymbol{A}) \triangleq \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A})$, for which $\mathbb{E}_{\boldsymbol{p}_\theta}[h(\boldsymbol{A})] = 0$, and obtain

$$\nabla_\theta \mathcal{L}_t(\psi, \theta) = \mathbb{E}_{\boldsymbol{p}_\theta}\left[\left(\delta_t(\boldsymbol{A};\psi) - \beta\right)\nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A})\right]. \tag{34}$$

This narrows the problem to finding appropriate values for baseline β.

Since for any f_1, f_2, $\text{Var}[f_1 + f_2] = \text{Var}[f_1] + \text{Var}[f_2] + 2\text{Cov}[f_1, f_2]$, the optimal baseline β_* in Equation (34) is given by

$$\beta_* \triangleq \frac{\text{Cov}[\delta_t(\boldsymbol{A};\psi)\nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}), \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A})]}{\text{Var}_{\boldsymbol{p}_\theta}[\nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A})]} = \frac{\mathbb{E}_{\boldsymbol{p}_\theta}[\delta_t(\boldsymbol{A};\psi)(\nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}))^2]}{\mathbb{E}_{\boldsymbol{p}_\theta}[(\nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}))^2]}. \tag{35}$$

Unfortunately, finding the optimal β_* can be as hard as estimating the desired gradient in Equation (23); moreover, note also that $\beta_* = \beta_*(\mathcal{X}_t)$, as δ_t depends on the observations $\mathcal{X}_t$.

Therefore, we opt for the approximation

$$\mathbb{E}_{\boldsymbol{p}_\theta}[\delta_t(\boldsymbol{A};\psi)(\nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}))^2] \approx \mathbb{E}_{\boldsymbol{p}_\theta}[\delta_t(\boldsymbol{A};\psi)]\mathbb{E}_{\boldsymbol{p}_\theta}[(\nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}))^2], \tag{36}$$

and obtain $\beta_* \approx \mathbb{E}_{\boldsymbol{p}_\theta}[\delta_t(\boldsymbol{A};\psi)]$. Note that a similar choice of baseline is very popular, for instance, in reinforcement learning applications (e.g., see advantage actor-critic estimators, Sutton et al. 1999; Mnih et al. 2016). However, since approximating $\mathbb{E}_{\boldsymbol{p}_\theta}[\delta_t(\boldsymbol{A};\psi)]$ would require the introduction of an additional estimator, we rely on a different approximation by moving the expectation inside the cost function and obtaining $\beta_* \approx \delta_t(\boldsymbol{\mu};\psi)$, where $\boldsymbol{\mu} = \mathbb{E}_{\boldsymbol{p}_\theta}[\boldsymbol{A}]$.

We recall that, in general, $\boldsymbol{\mu}$ is dense and its components are real numbers, therefore computing $\delta_t(\boldsymbol{\mu};\psi)$ would require evaluating the output of the model w.r.t. a dense adjacency matrix, potentially outside the well-behaved region of the input space, and to compute messages w.r.t. each node pair, thus negating any computational complexity benefit. Accordingly, we substitute $\boldsymbol{\mu}$ with the Fréchet mean adjacency matrix $\boldsymbol{A}^\mu$, relying on the generalized notion of mean for binary adjacency matrices introduced in Section 3.3. We then choose as $\hat{\beta}$ such that

$$\hat{\beta} \triangleq \delta_t(\boldsymbol{A}^\mu;\psi). \tag{37}$$

The computational cost of evaluating $\hat{\beta}$ corresponds then to that of a single evaluation of the cost function δ_t w.r.t. the binary and eventually sparse adjacency matrix $\boldsymbol{A}^\mu$.

Finally, we point out that, even though $\hat{\beta}$ may differ from β_*, the variance is reduced as long as $0 < \hat{\beta} < 2\beta_*$. We indicate the modified cost, i.e., the cost minus the baseline as $\tilde{\delta}_t(\boldsymbol{A};\psi) = \delta_t(\boldsymbol{A};\psi) - \delta_t(\boldsymbol{A}^\mu;\psi)$; the modified cost is computed after each forward pass and used to update the parameters of $\boldsymbol{p}_\theta$. In next Sections 6.2 and 6.3 we derive analytic solutions for finding $\boldsymbol{A}^\mu$ for BES and SNS, respectively.

6.2 Baseline for BES

We start by recalling the notation from previous sections. Denote expectation $\mathbb{E}_{\boldsymbol{p}_\theta}[\boldsymbol{A}]$ with respect to BES as $\boldsymbol{\mu} \in [0,1]^{N\times N} \subset \mathbb{R}^{N\times N}$ and with $\boldsymbol{A}^\mu$ the binary Fréchet mean adjacency matrix with respect to the support $\mathcal{A} = \{0,1\}^{N\times N}$ of the distribution $\boldsymbol{p}_\theta$ associated with BES. The main result of the section is the following proposition which allows us to provide a baseline as

$$\hat{\beta}_{\text{BES}} \triangleq \delta_t\left(\lfloor\sigma(\Phi)\rceil;\psi\right), \tag{38}$$

where $\lfloor\Phi\rceil$ indicates the element-wise rounding of the components of the real matrix Φ to the closest integer (either 0 or 1).

Proposition 1 *Consider BES with associated distribution $\boldsymbol{p}_\theta$ and support $\mathcal{A}$. Then,*

- *the expected matrix $\mathbb{E}_{\boldsymbol{p}_\theta}[\boldsymbol{A}]$ is $\boldsymbol{\mu} = \sigma(\Phi)$, with σ applied element-wise;*
- *the Fréchet mean adjacency matrix $\boldsymbol{A}^\mu = \lfloor\boldsymbol{\mu}\rceil = I(\Phi > 0)$.*

Proof As each component of $\boldsymbol{A} \sim \boldsymbol{p}_\theta$ is independent from the others, $\boldsymbol{\mu}_{i,j}$ can be considered element-wise as $\mathbb{E}_{\boldsymbol{p}_\theta}[\boldsymbol{A}_{i,j}] = \sigma(\Phi_{i,j})$, for all $i,j = 1,\dots,N$. Similarly, each component of $\boldsymbol{A}^\mu$ can be computed independently as well, by relying on Lemma 2.

Lemma 2 *The minimum of the Fréchet function $\mathfrak{F}_H$ can be expressed as*

$$\min_{\boldsymbol{A}\in\mathcal{A}} \mathfrak{F}_H(\boldsymbol{A}) = \min_{\boldsymbol{A}\in\mathcal{A}} \sum_{i,j=1}^{N} \left(\boldsymbol{\mu}_{i,j} - \boldsymbol{A}_{i,j}\right)^2. \tag{39}$$

To conclude the proof of Preposition 1, we observe that the minimum of Equation (39) is attained at $\boldsymbol{A}^\mu = \lfloor\boldsymbol{\mu}\rceil$, that is $\boldsymbol{A}^\mu_{i,j} = 1$ for all $\boldsymbol{\mu}_{i,j} > 1/2$ (or $\Phi > 0$), and 0 elsewhere. The proof of the Lemma 2 is deferred to Appendix A. ■

6.3 Baseline for SNS

Similarly to what has been done for BES in Proposition 1, we provide analogous results for SNS, with the added technical complexity that, in this case, edges $j \rightarrow i$ and $j' \rightarrow i$ are not independent. Nevertheless, the result remains intuitive:

$$\hat{\beta}_{\text{SNS}} \triangleq \delta_t \left(\boldsymbol{A}^{\mu}; \psi \right), \qquad \text{with } \boldsymbol{A}^{\mu}_{i,j} = I \left(\Phi_{i,j} \in \text{top-K}\{\Phi_{i,:}\} \right), \ \forall\, i, j \in \mathcal{S}. \tag{40}$$

The proof that $\boldsymbol{A}^{\mu}$ is indeed the Fréchet mean for SNS follows Preposition 3. Recall that, for SNS, the support of $\boldsymbol{p}_{\boldsymbol{\theta}}$ is that of directed K-NN graphs in Equation (42), where the neighborhood of each node is sampled independently. Equation (40) is derived by considering a neighborhood of fixed size K; however, the analysis remains valid for the adaptive case discussed in Section 5.2.2.

In the SNS case, each entry $\boldsymbol{\mu}_{n,i}$ of $\boldsymbol{\mu}$ is

$$\boldsymbol{\mu}_{n,i} = \boldsymbol{p}_{\boldsymbol{\theta}}(i \in S_K | n) = \sum_{S'_K : \, i \, \in \, S'_K} \boldsymbol{p}_{\boldsymbol{\theta}}(S'_K | n), \tag{41}$$

where the sum is taken over all subsets S'_K of $\mathcal{S}$ of K elements containing node i. Even if marginalizing over all possible sampled subsets of S'_K has combinatorial complexity, we show that $\boldsymbol{A}^{\mu}$ can be derived without directly computing $\boldsymbol{\mu}$ as stated in Proposition 3.

Proposition 3 *Consider an SNS distribution with support*

$$\mathcal{A} = \left\{ \boldsymbol{A} \in \{0,1\}^{N \times N} : \sum\nolimits_{j=1}^{N} \boldsymbol{A}_{i,j} = K, \ \forall\, i \right\}. \tag{42}$$

Then, the Frechét mean $\boldsymbol{A}^{\mu}$ *is given by*

$$\boldsymbol{A}^{\mu}_{i,j} = I \left(\Phi_{i,j} \in \text{top-K}\{\Phi_{i,:}\} \right), \ \forall\, i, j \in \mathcal{S}. \tag{43}$$

Proof Computing $\boldsymbol{A}^{\mu}$ corresponds to solving the optimization problem

$$\min_{\boldsymbol{A} \in \mathcal{A}} \mathfrak{F}_H \left(\boldsymbol{A} \right) = \min_{\boldsymbol{A} \in \mathcal{A}} \mathbb{E}_{\boldsymbol{A}' \sim \boldsymbol{p}_{\boldsymbol{\theta}}} \left[H(\boldsymbol{A}, \boldsymbol{A}') \right]. \tag{44}$$

Start by rewriting the Fréchet function as

$$\begin{aligned}
\mathfrak{F}_H(\boldsymbol{A}) &= \mathbb{E}_{\boldsymbol{A}' \sim \boldsymbol{p}_{\boldsymbol{\theta}}} \left[H(\boldsymbol{A}, \boldsymbol{A}') \right] && (45)\\
&= \mathbb{E}_{\boldsymbol{A}' \sim \boldsymbol{p}_{\boldsymbol{\theta}}} \left[\sum_{n,i=1}^{N} \boldsymbol{A}_{n,i} - 2\boldsymbol{A}_{n,i}\boldsymbol{A}'_{n,i} + \boldsymbol{A}'_{n,i} \right] && (46)\\
&= \sum_{n,i=1}^{N} \boldsymbol{A}_{n,i} \underbrace{\left(1 - 2\boldsymbol{\mu}'_{n,i}\right)}_{w_{n,i}} + \underbrace{\sum_{n,j=1}^{N} \boldsymbol{\mu}_{n,i}}_{c}. && (47)
\end{aligned}$$

where $\boldsymbol{\mu}_{n,i} = \boldsymbol{p}_{\boldsymbol{\theta}} \left(i \in S_K | n \right) = \boldsymbol{p}_{\boldsymbol{\theta}} \left(\boldsymbol{A}_{n,i} = 1 \right)$ and c is a constant. The proof follows from Lemma 4.

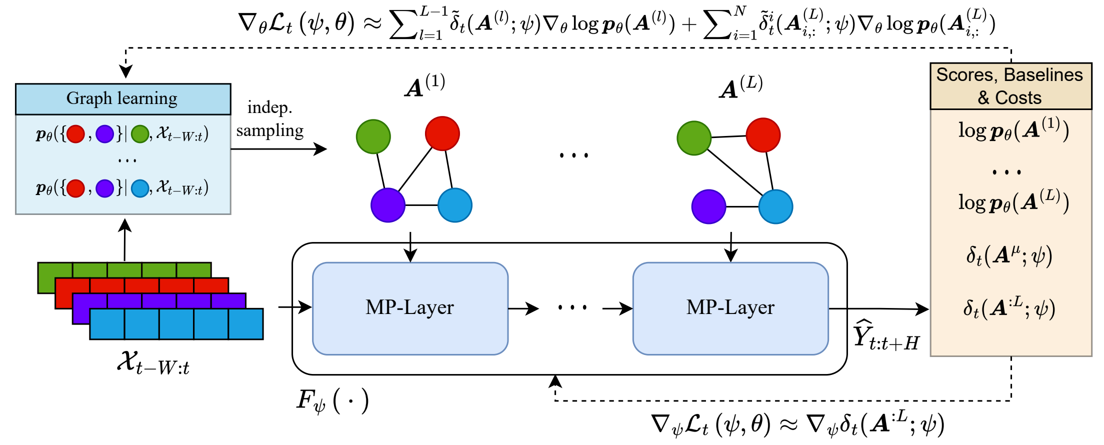

$\nabla_\theta \mathcal{L}_t(\psi, \theta) \approx \sum_{l=1}^{L-1} \tilde{\delta}_t(\boldsymbol{A}^{(l)}; \psi) \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}^{(l)}) + \sum_{i=1}^{N} \tilde{\delta}_t^i(\boldsymbol{A}^{(L)}_{i,:}; \psi) \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}^{(L)}_{i,:})$
Graph learning
$\boldsymbol{p}_\theta(\{$●, ●$\}|$ ●, $\mathcal{X}_{t-W:t})$
...
$\boldsymbol{p}_\theta(\{$●, ●$\}|$ ●, $\mathcal{X}_{t-W:t})$
indep. sampling
$\boldsymbol{A}^{(1)}$
...
$\boldsymbol{A}^{(L)}$
Scores, Baselines & Costs
$\log \boldsymbol{p}_\theta(\boldsymbol{A}^{(1)})$
...
$\log \boldsymbol{p}_\theta(\boldsymbol{A}^{(L)})$
$\delta_t(\boldsymbol{A}^{\mu}; \psi)$
$\delta_t(\boldsymbol{A}^{:L}; \psi)$
MP-Layer
...
MP-Layer
$\widehat{Y}_{t:t+H}$
$\mathcal{X}_{t-W:t}$
$F_\psi(\cdot)$
$\nabla_\psi \mathcal{L}_t(\psi, \theta) \approx \nabla_\psi \delta_t(\boldsymbol{A}^{:L}; \psi)$

Figure 2: Overview of the learning architecture with layer-wise sampling and surrogate objective. The graph module samples a graph for each MP layer of predictor F_ψ.

Lemma 4 *Let $\boldsymbol{p}_\theta$ be an SNS distribution with associated log-probabilities Φ. Then $\forall n, i, j \in \mathcal{S}$*

$$\boldsymbol{p}_\theta(\boldsymbol{A}_{n,i} = 1) \geq \boldsymbol{p}_\theta(\boldsymbol{A}_{n,j} = 1) \iff \Phi_{n,i} \geq \Phi_{n,j}. \tag{48}$$

The proof of Lemma 4 is provided in Appendix A. Following Equation (47), the optimization problem in Equation (44) becomes the linear program

$$\begin{aligned} \text{minimize} & \sum_{i=1}^{N} \sum_{j=1}^{N} w_{i,j} \boldsymbol{A}_{i,j} \\ \text{s.t.} & \sum_{j=1}^{N} \boldsymbol{A}_{i,j} = K; \\ & \boldsymbol{A}_{i,j} \in \{0, 1\} \qquad \forall i = 1, \dots, N, \end{aligned} \tag{49}$$

where $w_{i,j} = 1 - 2\boldsymbol{p}_\theta(\boldsymbol{A}_{i,j} = 1)$. Since Lemma 4 grants that, for each i, the K-smallest $w_{i,j}$ weights correspond row-wise to the top-K scores $\Phi_{i,j}$, the solution $\boldsymbol{A}^{\mu}$ to the linear program is given by $\boldsymbol{A}^{\mu}_{i,j} = I(\Phi_{i,j} \in \text{top-K}\{\Phi_{i,:}\})$ and, hence, the thesis. ∎

7. Layer-wise Sampling and Surrogate Objective

As a final step, we can leverage on the structure of MP neural networks to rewrite the gradient $\nabla_\theta \mathcal{L}_t(\psi, \theta)$. This formulation allows for obtaining a different estimator for the case where we sample a different $\boldsymbol{A}^{(l)}$ at each of the L MP layers of F_ψ (e.g., see Equation (10)). A schematic overview of the procedure is shown in Figure 2 where $\boldsymbol{A}^{:L} = \{\boldsymbol{A}^{(l)}\}_{l=1}^{L}$.

Proposition 5 *Consider family of models $F_\psi(\,\cdot\,; \boldsymbol{A}^{:L})$ with exactly L message-passing layers propagating messages w.r.t. different adjacency matrices $\boldsymbol{A}^{(l)}$, $l = 1, \dots, L$, sampled from $\boldsymbol{p}_\theta$ (either BES or SNS). Assume that the cost function δ_t can be written as the summation over node-level costs δ_t^i. Then*

$$\nabla_\theta \mathcal{L}_t(\psi, \theta) = \mathbb{E}_{\boldsymbol{p}_\theta}\left[\sum_{l=1}^{L-1} \delta_t(\boldsymbol{A}^{:L}; \psi) \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}^{(l)}) + \sum_{i=1}^{N} \delta_t^i(\boldsymbol{A}^{:L}; \psi) \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}^{(L)}_{i,:})\right], \quad (50)$$

where $\boldsymbol{A}^{(L)}_{i,:}$ denotes the i-th row of adjacency matrix $\boldsymbol{A}^{(L)}$, i.e., the row corresponding to the neighborhood of the i-th node.

Proposition 5 holds for all parametrizations of $\boldsymbol{p}_\theta$ as long as the neighborhood of each node (i.e., the rows of $\boldsymbol{A}$) are sampled independently. Furthermore, note that almost all of the cost functions typically used for node-level tasks satisfy the assumption, e.g.,

$$\widehat{\boldsymbol{Y}}_{t:t+H} = F_\psi\left(\mathcal{X}_{t-W:t}; \boldsymbol{A}^{:L}\right), \qquad \delta_t(\boldsymbol{A}^{:L}; \psi) = \sum_{i=1}^{N} \left\| \boldsymbol{y}^i_{t:t+H} - \widehat{\boldsymbol{y}}^i_{t:t+H} \right\|_p^p = \sum_{i=1}^{N} \delta_t^i(\boldsymbol{A}^{:L}; \psi).$$

The following provides proof of Proposition 5 and presents a surrogate objective function inspired by Equation (50).

Proof A proof can be derived by noticing the independence of $\delta_t^i(\boldsymbol{A}^{:L}; \psi)$ and $\boldsymbol{p}_\theta(\boldsymbol{A}^{(L)}_{j,:})$ for $i \neq j$, and by exploiting the fact that with both BES and SNS rows of each $\boldsymbol{A}^{(l)}$ are sampled independently. For the sake of readability, we omit the dependency of δ_t and δ_t^i from $\boldsymbol{A}^{:L}$ and ψ. The proof follows:

$$\begin{aligned}
\nabla_\theta \mathcal{L}_t(\psi, \theta) &= \mathbb{E}_{\boldsymbol{p}_\theta}\left[\delta_t \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}^{:L})\right] && (51)\\
&= \mathbb{E}_{\boldsymbol{p}_\theta}\left[\sum_{l=1}^{L-1} \delta_t \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}^{(l)})\right] + \underbrace{\mathbb{E}_{\boldsymbol{p}_\theta}\left[\delta_t \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}^{(L)})\right]}_{(*)}. && (52)
\end{aligned}$$

By considering the second term:

$$\begin{aligned}
(*) &= \mathbb{E}_{\boldsymbol{p}_\theta}\left[\delta_t \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}^{(L)})\right] && (53)\\
&= \mathbb{E}_{\boldsymbol{p}_\theta}\left[\sum_{i=1}^{N} \delta_t^i \sum_{j=1}^{N} \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}^{(L)}_{j,:})\right] && (54)\\
&= \mathbb{E}_{\boldsymbol{p}_\theta}\left[\sum_{i=1}^{N} \delta_t^i \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}^{(L)}_{i,:})\right] + \underbrace{\mathbb{E}_{\boldsymbol{p}_\theta}\left[\sum_{i=1}^{N} \delta_t^i \sum_{j \neq i} \nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}^{(L)}_{j,:})\right]}_{(**)}. && (55)
\end{aligned}$$

The two factors in $(**)$ are independent since δ_t^i depends only on $\boldsymbol{A}^{:L-1}$ and $\boldsymbol{A}^{L}_{i,:}$, hence

$$(**) = \sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{p}_\theta}\left[\delta_t^i\right] \sum_{j \neq i} \underbrace{\mathbb{E}_{\boldsymbol{p}_\theta}\left[\nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}^{(L)}_{j,:})\right]}_{=0} = 0. \quad (56)$$

Putting everything together, we get Equation (50) and the proof is completed. ■

7.1 Surrogate Objective

Intuitively, the second term in Equation (50) can be interpreted as directly rewarding connections that lead to accurate final predictions w.r.t. the local cost δ^i. Besides providing a more general MC estimator, Preposition 5 motivates us in considering a similar surrogate approximate loss $\widehat{\mathcal{L}}_t\,(\psi,\theta)$ for the case where we use a single $\boldsymbol{A}$ for all layers, i.e., we consider

$$\nabla_\theta \widehat{\mathcal{L}}_t\,(\psi,\theta) = \mathbb{E}_{\boldsymbol{p}_\theta}\Big[\lambda\delta_t(\boldsymbol{A};\psi)\nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}) + \sum\nolimits_{i=1}^{N} \delta_t^i(\boldsymbol{A};\psi)\nabla_\theta \log \boldsymbol{p}_\theta(\boldsymbol{A}_{i,:})\Big], \tag{57}$$

as gradient to learn $\boldsymbol{p}_\theta$. Equation (57) is developed from Equation (50) by considering a single sample $\boldsymbol{A} \sim \boldsymbol{p}_\theta$ and introducing the hyperparameter λ. Note that, in this case, $\widehat{\mathcal{L}}_t\,(\psi,\theta)$ is an approximation of the true objective with a reweighting of the contribution of each $\delta^i(\boldsymbol{A};\psi)$. Following this consideration, λ can be interpreted as a trade-off between local and global cost. In practice, we set $\lambda = 1/N$, so that the two terms are roughly on the same scale. Empirically, we observed that using the modified objective consistently leads to faster convergence; see Section 8.

8. Experiments

To validate the effectiveness of the proposed framework, we carried out experiments in several settings on both synthetic and real-world datasets. In particular, a set of experiments focuses on the task of graph identification where the objective is that of retrieving graphs that better explain a set of observations given a (fixed) predictive model. The second collection of experiments shows instead how the proposed approach can be used as a graph-learning module in an end-to-end forecasting architecture.

8.1 Datasets

Dataset	# nodes	# edges	# steps
GPVAR	30	98	30000
PEMS-BAY	325	2369	52128
METR-LA	207	1515	34272
AQI (Beijing)	36	180	8760
AQI (Tianjin)	27	135	8760

Table 1: Additional information on the considered datasets.

We consider one synthetic dataset and 3, openly available, real-world benchmarks.

- **GPVAR** – The GPVAR synthetic dataset consists of signals generated by recursively applying a polynomial Graph VAR filter (Isufi et al., 2019) and adding Gaussian noise at each time step: this results in complex, yet known and controllable, spatiotemporal dynamics. In particular, analogously to Zambon and Alippi (2022), we consider the data generating process

$$\boldsymbol{X}_t = \tanh\left(\sum_{l=0}^{L}\sum_{q=1}^{Q}\Theta_{l,q}\widetilde{\boldsymbol{A}}^l\boldsymbol{X}_{t-q}\right) + \eta_t \tag{58}$$

where $\widetilde{\boldsymbol{A}} = \boldsymbol{I} + \boldsymbol{A}$ (with $\boldsymbol{I}$ being the identity matrix), $\Theta \in \mathbb{R}^{(L+1)\times Q}$ denotes the model parameter and $\eta_t \sim \mathcal{N}(0, \boldsymbol{I})$ is a Gaussian noise vector. Model parameters, with $L = Q = 2$, are set as described in (Zambon and Alippi, 2022) and used to generate a trajectory of $T = 30000$ steps. We use 70/10/20% data split for training, validation, and testing, respectively.

- **AQI** – The Air Quality Index (AQI) dataset consists of hourly readings from air quality monitoring stations scattered over different Chinese cities. AQI has been previously used as a benchmark for time series imputation methods (Yi et al., 2016; Cini et al., 2022; Marisca et al., 2022). We use the same preprocessing and data splits of previous works (Yi et al., 2016). The ground-truth graph is obtained by considering the pairwise distance of the sensors, following the procedure used in (Cini et al., 2022).
- **METR-LA** and **PEMS-BAY** – The METR-LA and PEMS-BAY datasets from (Jagadish et al., 2014; Li et al., 2018) are two popular benchmarks in the traffic forecasting literature. The datasets consist of traffic speed measurements taken at crossroads in Los Angeles and San Francisco, respectively. We use the same preprocessing and data splits of previous works (Wu et al., 2019). The underlying graphs are extracted from the geographic position of the sensors following the same steps of Wu et al. (2019).

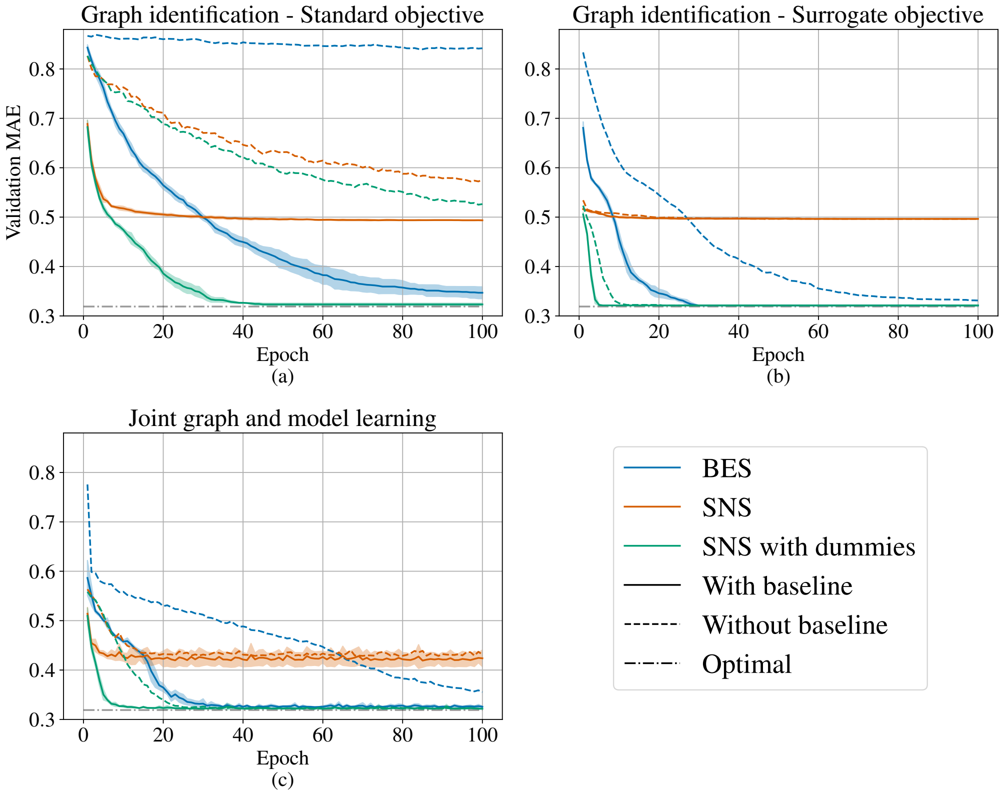
Additional relevant information about the datasets is provided in Table 1.

8.2 Controlled Environment Experiments

To gather insights on the impact of each aspect of the methods introduced so far, we start by using the controlled environment provided by the GPVAR dataset.

8.2.1 Graph Identification and Time Series Forecasting

In the first setup, we consider a GPVAR filter as the predictor and assume known the true model parameters, i.e., the coefficients of the filter, to decouple the assessment of the graph-learning module from that of the forecasting module. Then, in a second scenario, we learn the graph while, at the same time, fitting the filter's parameters. Figure 3 shows the validation mean absolute error (MAE) after each training epoch by using BES and SNS samplers, with and without baseline $\hat{\beta}$ for variance reduction, and when SNS is run with dummy nodes for adaptive node degrees. The number of maximum neighbors is set to $K = 5$, which is the maximum degree of the ground truth graph. In particular, Figure 3a and Figure 3b show results in the graph identification task for the vanilla gradient estimator derived from Equation (22) and for the surrogate objective from Equation (57), respectively. To match the optimal prediction, models have to perfectly retrieve the underlying graph. During the evaluation, we used $\boldsymbol{A}^{\mu}$ as input to the predictor instead of sampling $\boldsymbol{p}_{\theta}$. Results allow us to make the following comments.



(a)

(b)

(c)

Figure 3: Experiments on GPVAR. All the curves show the validation MAE after each training epoch.

Impact of the Baseline The first striking outcome is the effect of baseline $\hat{\beta}$ in both the considered configurations which dramatically accelerates the learning process.

Graph distribution The second notable result is that, although both SNS and BES are able to retrieve the underlying graph, the sparsity prior in SNS yields faster convergence w.r.t. the number of samples seen during training, as the validation curves are steeper for SNS; note that the approximation error induced by having a fixed number of neighbors is effectively removed with the dummy nodes.

Surrogate objective Figure 3b shows that the surrogate objective contributes to accelerating learning even further for all considered methods.

Joint training Finally, Figure 3c reports the results for the joint training of the predictor and graph module with the surrogate objective. The curves, in this case, were obtained by initializing the parameters of the filter randomly and specifying an order of the filter higher than the real one; nonetheless, the learning procedure was able to quickly converge to the optimum when using as baseline the cost evaluated w.r.t. $\boldsymbol{A}^{\mu}$.

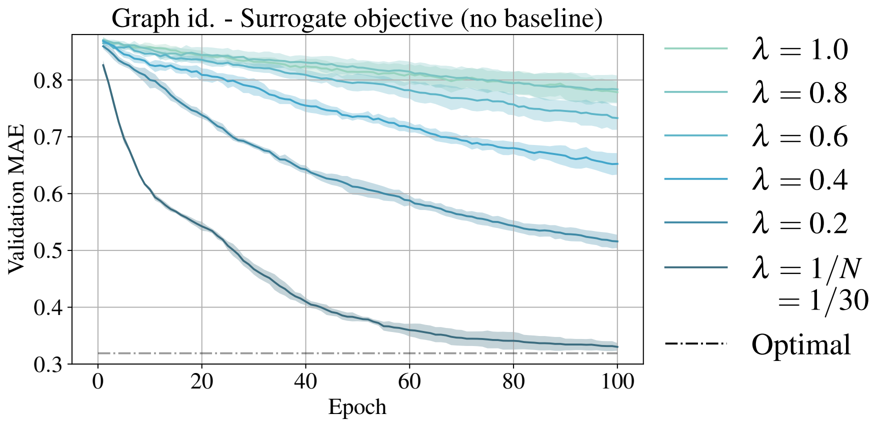


Figure 4: Sensitivity analysis on λ for the surrogate objective.

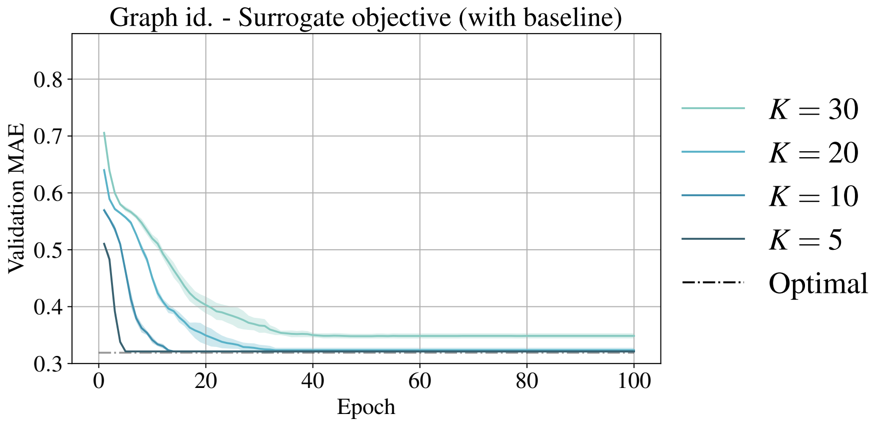


Figure 5: Sensitivity analysis on K for SNS.

8.2.2 Sensitivity Analysis

To further assess the impact of the surrogate objective and that of the structural priors embedded into the SNS parametrization, we run a sensitivity analysis on both these aspects.

Regarding the surrogate objective, we run a sensitivity analysis on the hyperparameter λ, which was kept fixed to $\lambda = 1/N$ in the experiments in Figure 3. In particular, we repeated the experiment on graph identification setting by considering the BES parametrization and values for λ in the range $[1/N = 1/30, 1]$. We did not use the baseline to accentuate the sensitivity to λ. Results, shown in Figure 4, demonstrate the effectiveness of the surrogate loss in accelerating learning by introducing and reweighting the local cost term and how decreasing the weight of the global cost leads to faster convergence.

Finally, we assess the impact of the value of the hyperparameter K on the learning speed for the SNS sampler. In this case, we consider the graph identification experiment with the baseline for variance reduction. We run experiments with $K \in (5, 10, 20, 30)$ and

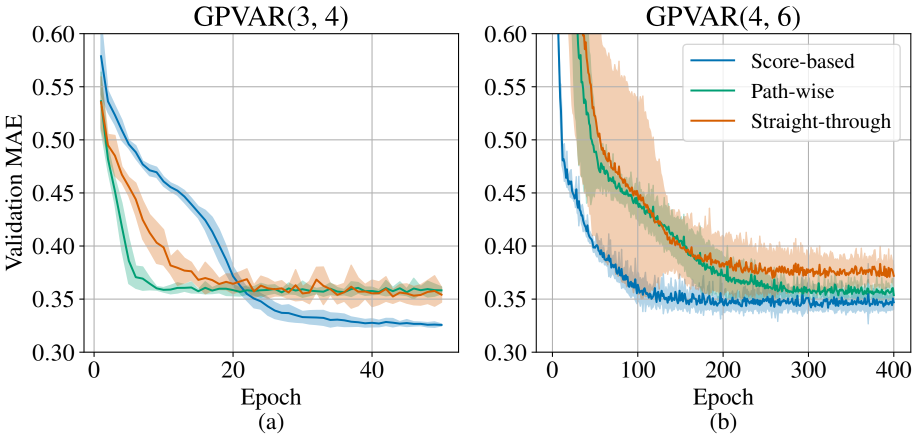


Figure 6: Comparison of different estimators on the joint training settings in GPVAR.

a number of dummy nodes equal to $K-1$. Results in Figure 5 show that while the use of dummy nodes reduces the impact of a wrong assessment of K, overestimating the maximum number of neighbors can nonetheless lead to slower convergence. In particular, given these settings and hyperparameters, SNS fails to converge to the optimal solution for $K = 30$, i.e., a number of neighbors equal to the number of nodes. As a general recommendation, we argue that using SNS can be beneficial as long as $K < N/2$, while for larger values of K a BES parametrization is preferable due to the reduced overhead in sampling and likelihood evaluation.

8.2.3 Comparison with Path-wise and Straight-through Estimators

In this section, we assess the effectiveness of the proposed score-function estimator (with baseline and surrogate objective) against both the path-wise estimator, based on the Concrete continuous relaxation of Bernoulli random variables (Maddison et al., 2017), and the straight-through estimator (Bengio et al., 2013). We consider the controlled joint graph and model learning scenario from Section 8.2.1. In particular, for all estimators, we consider the BES parametrization for the graph distribution and the model family of Graph VAR filters of spatial order 3 and temporal order 4—as in the joint training experiment of Section 8.2.1—and a more difficult scenario corresponding to filters up to orders 4 and 6, respectively.

The results of the experiment are shown in Figure 6. In the simpler setting (Figure 6a), both the path-wise and straight-through estimators appear to converge faster than the score-based approach, yet they reach sub-optimal results—a side-effect that we attribute to the bias of the path-wise and straight-through estimators. In the harder setting (Figure 6b), instead, our method achieves better performance both in terms of forecasting accuracy and sample complexity. This behavior might be associated with the complex dynamics of learning the relational structure given a larger family of predictive models.

8.3 Real-World Datasets

The following discusses the application of the proposed method w.r.t. data coming from real-world scenarios.

8.3.1 Graph Identification in AQI

For graph identification, we set up the following scenario. From the AQI dataset, we extract 2 subsets of sensors that correspond to monitoring stations in the cities of Beijing and Tianjin, respectively. We build a graph for both subsets of data by constructing a K-NN graph of the stations based on their distance; we refer to these as ground-truth graphs. Then, we train a different predictor for each of the two cities, based on the ground-truth graph. In particular, we use a TTS STGNN with a simple architecture consisting of a GRU (Chung et al., 2014) encoder followed by 2 MP layers. As a reference value (sanity check), we also report the performance achieved by a GRU trained on all sensors, without using any spatial information. Performance is measured in terms of 1-step-ahead MAE. Results for the two models, trained with early stopping on the validation set and tested on the hold-out test set for the same city (i.e., in a transductive learning setting) are shown in the main diagonal of Table 2. In the second stage of the experiment, we consider an inductive setting: we train the model above on one of the two cities as a source, freeze its parameters, discard the ground-truth graph w.r.t. the left-out city, and train our graph learning module (with the SNS parametrization) to maximize the forecasting accuracy. The idea is to show that our module is able to recover a topology that gives performance close to what would be achievable with the ground-truth graph. Results, reported in the off-diagonal elements of Table 2, show that our approach is able to almost match the performance that would have been possible to achieve by fitting the model directly on the target dataset with the ground-truth adjacency matrix; moreover, the performance is significantly better than that of the reference GRU.

	Tested on	
Trained on	Beijing	Tianjin
Beijing	9.43 ± 0.03	10.62 ± 0.05
Tianjin	9.55 ± 0.06	10.56 ± 0.03
Baseline	10.21 ± 0.01	11.25 ± 0.04

Table 2: AQI experiment.

8.3.2 Joint Training and Forecasting in Traffic Datasets

Finally, we test our approach on 2 widely used traffic forecasting benchmarks. Here we took the full-graph attention architecture proposed in (Satorras et al., 2022), removed the attention gating mechanism, and used the graph learned by our module to sparsify the learned attention coefficients; in particular, we considered the SNS sampler with $K = 30$, 10 dummy nodes and surrogate objective ($\lambda = 1/N$). We used the same hyperparameters of (Satorras et al., 2022), except for the learning rate schedule and batch size (see supplemental material). As a reference, we also tested results using the ground-truth graph, a graph with only self-loops (i.e., with $\boldsymbol{A}$ set to the identity matrix), as well as a random graph sampled from the Erdös-Rényi model with $p = 0.1$. For MTGNN (Wu et al., 2020) and GTS we report results obtained by running the authors' code. More details are provided in Appendix C. Note that GTS is considered the state of the art for methods based on path-wise estimators (Zügner et al., 2021). Results in Table 3 show the MAE performance for 15, 30 and 60 minutes time horizons achieved over multiple independent runs. Our approach is always competitive w.r.t. the state-of-the-art alternatives, and statistically better than all the baselines with reference adjacency matrices. Note that, using a random adjacency matrix—which essentially corresponds to randomly sparsifying the attention coefficients—is often competitive with more

	METR-LA			PEMS-BAY		
Model	MAE @ 15	MAE @ 30	MAE @ 60	MAE @ 15	MAE @ 30	MAE @ 60
Full attention	2.727 ± .005	3.049 ± .009	3.411 ± .007	1.335 ± .003	1.655 ± .007	1.929 ± .007
GTS	2.750 ± .005	3.174 ± .013	3.653 ± .048	1.360 ± .011	1.715 ± .032	2.054 ± .061
MTGNN	2.690 ± .012	3.057 ± .016	3.520 ± .019	1.328 ± .005	1.655 ± .010	1.951 ± .012
Our (SNS)	2.725 ± .005	3.051 ± .009	3.412 ± .013	1.317 ± .002	1.620 ± .003	1.873 ± .005
Adjacency						
–Truth	2.720 ± .004	3.106 ± .008	3.556 ± .011	1.335 ± .001	1.676 ± .004	1.993 ± .008
–Random	2.801 ± .006	3.160 ± .008	3.517 ± .009	1.327 ± .001	1.636 ± .002	1.897 ± .003
–Identity	2.842 ± .002	3.264 ± .002	3.740 ± .004	1.341 ± .001	1.684 ± .001	2.013 ± .003

Table 3: Results on the traffic datasets.

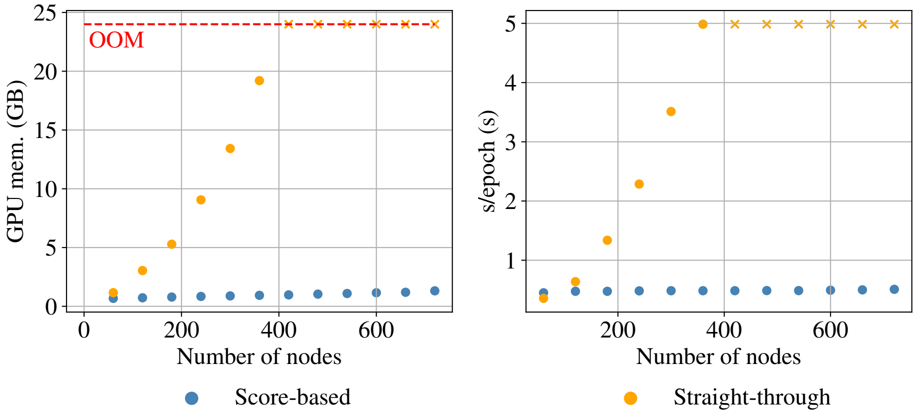


Figure 7: Computational scalability of the proposed estimator against the straight-through method.

complex approaches which suggests that, in some datasets, having access to the ground-truth graph is not decisive for achieving high performance. That being said, our graph learning methods consistently improve performance w.r.t. the naïve baselines.

8.4 Scalability

To assess the scalability of the proposed method, we consider a T&S model consisting of a message-passing GRU (MPGRU, Cini et al. 2022), i.e., a GRU with gates implemented by MPNNs. In particular, we consider a simple MP scheme s.t.

$$\boldsymbol{z}_t^{i,(l)} = \sum_{j \in \mathcal{N}(i)} \text{MLP}\left(\boldsymbol{z}_t^{i,(l-1)}, \boldsymbol{z}_t^{j,(l-1)}\right). \tag{59}$$

The resulting model has a space and time complexity that scales as $\mathcal{O}(LTE)$. By considering the same controlled environment of the experiments in Section 8.2 and varying the number of nodes in the graph underlying the generated data, we empirically assessed the time and memory cost of learning a graph distribution with our SNS approach against the straight-through approach. Note that, while the straight-through estimator allows for a sparse forward pass at inference, the processing is nonetheless dense at training time—thus requiring $\mathcal{O}(LTN^2)$ time and space, instead of $\mathcal{O}(LTE)$.

The resulting models are trained on mini-batches of 4 samples with a window size of 8 steps for 50 epochs, each consisting of 5 mini-batches. The empirical results in Figure 8.4 show measured GPU usage and latency for the above settings. The computational advantages of the sparse message-passing operations of our method are evident.

9. Conclusions

In this paper, we propose a methodological framework for learning graph distributions from spatiotemporal data. Our novel probabilistic framework relies upon score-function gradient estimators that allow us for keeping the computation sparse throughout both the training and inference phases. We then develop variance-reduction techniques for our method to obtain accurate estimates for the training gradient. The proposed graph learning modules are applied to the time series forecasting task where they can be used for both graph identification and as components of an end-to-end architecture. Empirical results support our claims, showing the effectiveness of the framework. Notably, we achieve forecasting performance on par with state-of-the-art alternatives, while maintaining the benefits of graph-based processing. Possible directions for future research include the assessment of the proposed method w.r.t. inference of dynamic adjacency matrices, distribution agnostic variance reduction methods, and, in particular, the design of advanced forecasting architectures to achieve accurate predictions at scale. Furthermore, it would interesting to assess the combination of the proposed estimators with orthogonal variance reduction techniques (e.g., Kool et al. 2020) and data-driven baselines. Finally, future works might investigate the application of the recently proposed implicit maximum likelihood estimators (Niepert et al., 2021; Minervini et al., 2023) to the settings explored in this paper.

Acknowledgements

This work was supported by the Swiss National Science Foundation project FNS 204061: *HigherOrder Relations and Dynamics in Graph Neural Networks.* The authors wish to thank the Institute of Computational Science at USI for granting access to computational resources.

Appendix

Appendix A. Deferred Proofs

This Appendix provides the proofs for Lemma 2 and Lemma 4.

A.1 Proof of Lemma 2

Note that for all $\boldsymbol{A}, \boldsymbol{A'} \in \mathcal{A} \triangleq \{0,1\}^{N\times N}$ the Fréchet function $\mathfrak{F}_H$ can be expressed as

$$\mathfrak{F}_H(\boldsymbol{A'}) = \mathfrak{F}_F(\boldsymbol{A'}) \triangleq \mathbb{E}_{\boldsymbol{A}\sim \boldsymbol{p}_\theta}\left[\|\boldsymbol{A} - \boldsymbol{A'}\|_F^2\right] \tag{60}$$

w.r.t. the Frobenius norm, therefore we have also

$$\min_{\boldsymbol{A'}\in\mathcal{A}} \mathfrak{F}_H(\boldsymbol{A'}) = \min_{\boldsymbol{A'}\in\mathcal{A}} \mathfrak{F}_F(\boldsymbol{A'}). \tag{61}$$

Note now that

$$\begin{aligned}
\mathfrak{F}_F(\boldsymbol{A'}) &= \mathbb{E}_{\boldsymbol{p}_\theta}\left[\|\boldsymbol{A} - \boldsymbol{A'}\|_F^2\right] = \mathbb{E}_{\boldsymbol{p}_\theta}\left[\|\boldsymbol{A} \pm \boldsymbol{\mu} - \boldsymbol{A'}\|_F^2\right] && (62)\\
&= \mathbb{E}_{\boldsymbol{p}_\theta}\left[\|\boldsymbol{A} - \boldsymbol{\mu}\|_F^2\right] + 2\mathbb{E}_{\boldsymbol{p}_\theta}\left[\langle \boldsymbol{A} - \boldsymbol{\mu}, \boldsymbol{\mu} - \boldsymbol{A'}\rangle_F\right] + \mathbb{E}_{\boldsymbol{p}_\theta}\left[\|\boldsymbol{\mu} - \boldsymbol{A'}\|_F^2\right] && (63)\\
&= \mathbb{E}_{\boldsymbol{p}_\theta}\left[\|\boldsymbol{A} - \boldsymbol{\mu}\|_F^2\right] + 2\underbrace{\langle \mathbb{E}_{\boldsymbol{p}_\theta}[\boldsymbol{A}] - \boldsymbol{\mu}, \boldsymbol{\mu} - \boldsymbol{A'}\rangle_F}_{=0} + \|\boldsymbol{\mu} - \boldsymbol{A'}\|_F^2. && (64)
\end{aligned}$$

Moreover, as the first term does not depend on $\boldsymbol{A'}$, the minimum of $\mathfrak{F}_F(\boldsymbol{A'})$ is achieved at the minimum of

$$\|\boldsymbol{\mu} - \boldsymbol{A'}\|_F^2 = \sum_{i,j=1}^{N} (\boldsymbol{\mu}_{i,j} - \boldsymbol{A'}_{i,j})^2. \tag{65}$$

A.2 Proof of Lemma 4

The neighborhood of each node n is sampled independently from the others, so we derive the proof for a reference node n and denote $\phi = \Phi_{n,:}$.

Note that, for every pair of node $i, j \in \mathcal{S}$ and scalar $g \in \mathbb{R}$

$$\begin{aligned}
&\mathbb{P}(G_{\phi_i} \geq g) \geq \mathbb{P}(G_{\phi_j} \geq g) && (66)\\
&\qquad \iff e^{-e^{-(g-\phi_i)}} = \mathcal{F}_{\phi_i}(g) \leq \mathcal{F}_{\phi_j}(g) = e^{-e^{-(g-\phi_j)}} && (67)\\
&\qquad \iff \left(e^{-e^{-g}}\right)^{e^{\phi_i}} \leq \left(e^{-e^{-g}}\right)^{e^{\phi_i}}. && (68)
\end{aligned}$$

Being $e^{-e^{-g}} < 1$ and the e^x monotone we obtain

$$\mathbb{P}(G_{\phi_i} \geq g) \geq \mathbb{P}(G_{\phi_j} \geq g) \iff e^{\phi_i} \geq e^{\phi_j} \iff \phi_i \geq \phi_j. \tag{69}$$

$P(\boldsymbol{A}_{n,i} = 1)$ can then be rewritten as

$$\begin{aligned}
\mathbb{P}(\boldsymbol{A}_{n,i} = 1) &= \mathbb{P}(G_{\phi_i} \in \text{top-K}\{G_{\phi_l} : l \in \mathcal{S}\}) = \mathbb{P}(G_{\phi_i} \geq \overline{G}) && (70)\\
&= \int \mathbb{P}(G_{\phi_i} \geq g) f_{\overline{G}}(g)\, dg && (71)
\end{aligned}$$

with $\overline{G}$ being the random variable associated with the K-th largest realization in $\{G_{\phi_l} : l \in \mathcal{S}\}$ and $f_{\overline{G}}$ its p.d.f., we obtain

$$\mathbb{P}(\boldsymbol{A}_{n,i} = 1) \geq \mathbb{P}(\boldsymbol{A}_{n,j} = 1) \overset{\text{(Eq. 71)}}{\iff} \mathbb{P}(G_{\phi_i} \geq g) \geq \mathbb{P}(G_{\phi_j} \geq g) \overset{\text{(Eq. 69)}}{\iff} \phi_i \geq \phi_j, \tag{72}$$

concluding the proof.

Appendix B. Details on the Computation of the SNS Likelihood

In this appendix, we provide all the steps to obtain the rewriting of the likelihood on an SNS sample introduced in Equation (31). The derivations provided here follow from the results of Kool et al. (2020).

$$
\begin{aligned}
\boldsymbol{p}_{\theta}(S_K|i) &= \mathbb{P}\left(\min_{i\in S_K} G_{\phi_i} > \max_{i\in \mathcal{S}\setminus S_k} G_{\phi_i}\right) \\
&= \mathbb{P}\left(\min_{i\in S_K} G_{\phi_i} > G_{\phi_{\mathcal{S}\setminus S_k}}\right) \\
&= \mathbb{P}\left(G_{\phi_i} > G_{\phi_{\mathcal{S}\setminus S_k}}, \forall i \in S_K\right) \\
&= \int_{-\infty}^{\infty} f_{\phi_{\mathcal{S}\setminus S_k}}(g)\mathbb{P}\left(G_{\phi_i} > g, \forall i \in S_K\right)\, dg \\
&= \int_{-\infty}^{\infty} \prod_{i\in S_K} \left(1 - \mathcal{F}_{\phi_i}\left(g\right)\right) f_{\phi_{\mathcal{S}\setminus S_k}}(g)\, dg \\
&= \int_{0}^{1} \prod_{i\in S_K} \left(1 - \mathcal{F}_{\phi_i}\left(\mathcal{F}^{-1}_{\phi_{\mathcal{S}\setminus S_k}}(v)\right)\right)\, dv \quad \left\{{\scriptstyle v=\mathcal{F}_{\phi_{\mathcal{S}\setminus S_k}}(g)}\right\} \\
&= \int_{0}^{1} \prod_{i\in S_k} \left(1 - v^{\exp(\phi_i - \phi_{\mathcal{S}\setminus S_K})}\right) dv \\
&= \exp\left(b\right) \int_{0}^{1} u^{\exp(b)-1} \prod_{i\in S_k} \left(1 - u^{\exp(\phi_i - \phi_{\mathcal{S}\setminus S_k} + b)}\right)\, du \quad \left\{{\scriptstyle u=v^{\exp\,(-b)}}\right\} \\
&= \exp\left(\phi_{\mathcal{S}\setminus S_K} + c\right) \int_{0}^{1} u^{\exp\left(\phi_{\mathcal{S}\setminus S_K}+c\right)-1} \prod_{i\in S_k} \left(1 - u^{\exp(\phi_i + c)}\right)\, du \quad \left\{{\scriptstyle c=b-\phi_{\mathcal{S}\setminus S_K}}\right\},
\end{aligned}
$$

which corresponds to the desired rewriting.

Appendix C. Experiments Details

All the code for the experiments has been developed in Python using the following open-source libraries:

- PyTorch (Paszke et al., 2019);
- PyTorch Geometric (Fey and Lenssen, 2019);
- Torch Spatiotemporal (Cini and Marisca, 2022);
- PyTorch Lightning (Falcon and The PyTorch Lightning team, 2019);
- numpy (Harris et al., 2020);

furthermore, we relied on Neptune[2] (neptune.ai, 2021) for logging experiments. For GTS, we used the code provided by the authors[3] to obtain the results shown in the table, however we fixed a bug in the performance evaluation present in the official implementation[4].

Experiments were run on a cluster equipped with Nvidia Titan V and GTX 1080 GPUs. The code to reproduce the experiments of the paper is available online[5].

C.1 Synthetic Experiments

For the graph identification experiments, we simply trained the different graph identification modules using the Adam optimizer with a learning rate of 0.05 to minimize the absolute error. For the joint graph identification and forecasting experiment, we train on the generated dataset a GPVAR filter with $L = 3$ and $Q = 4$ with parameters randomly initialized and fitted with Adam using the same learning rate for the parameters of both graph filter and graph generator. To avoid numeric instability, scores Φ were soft-clipped to the interval $(-5, 5)$ by using the *tanh* function.

C.2 AQI Experiment

For the experiments on AQI we use a simple TTS model with a GRU encoder with 2 hidden layers, followed by a GNN decoder with 2 graph convolutional layers updating representations as:

$$\boldsymbol{Z}^{(l)} = \sigma\left(\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{Z}^{(l-1)}\boldsymbol{W} + \boldsymbol{V}\boldsymbol{Z}^{(l-1)}\right) \tag{73}$$

where $\boldsymbol{W}, \boldsymbol{V} \in \mathbb{R}^{d_z \times d_z}$ are learnable weight matrices and σ is a nonlinear activation function (in particular we use Swish (Ramachandran et al., 2017)). All layers have a hidden size of 64 units. We use an input window size of 24 steps and train for 100 epochs the models with the Adam optimizer with an initial learning rate of 0.005 and a multi-step learning rate scheduler. For the GRU baseline, we use a single recurrent layer of size 64 followed by an MLP decoder with 1 hidden layer with 32 units. For the graph module, we use SNS with $K = 5$ and 4 dummy nodes and train with Adam with a learning rate of 0.01 for 200 epochs. At test time, we used models with weights corresponding to the lowest validation error across epochs.

C.3 Traffic Experiment

As reported in the paper, we use the same architecture and hyperparameters of the full graph model of Satorras et al. (2022), except for the gating mechanism which was removed for the graph-based baselines. We train the models for a maximum of 200 epochs with Adam and an initial learning rate of 0.003 and a multi-step scheduler (analogously to Satorras et al. (2022). Note that we used an initial learning rate lower than the one used in (Satorras et al., 2022) as we observed it was on average leading to better performance. In each epoch, we used 200 mini-batches of size 64 for all the model variations, except for the full-attention model for which –on PEMS-BAY– we had to limit the batch size to 16 due to GPU memory limitations. For the graph learning module, we used SNS with $K = 30$ and 10 dummy nodes. We also

[2] `https://neptune.ai/`

[3] `https://github.com/chaoshangcs/GTS`

[4] `https://github.com/chaoshangcs/GTS/issues/19`

[5] `https://github.com/andreacini/sparse-graph-learning`

used a temperature $\tau = 0.5$ to make the sampler more deterministic. During evaluation, we used the $\boldsymbol{A}^{\mu}$ to obtain test-time predictions.

References

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

D. Bacciu, F. Errica, A. Micheli, and M. Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221, 2020.

L. Bai, L. Yao, C. Li, X. Wang, and C. Wang. Adaptive graph convolutional recurrent network for traffic forecasting. *Advances in Neural Information Processing Systems*, 33: 17804–17815, 2020.

Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. A. Szerlip, P. Horsfall, and N. D. Goodman. Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20:28:1–28:6, 2019. URL `http://jmlr.org/papers/v20/18-403.html`.

M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and deep locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

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

A. Cini and I. Marisca. Torch Spatiotemporal, 3 2022. URL `https://github.com/TorchSpatiotemporal/tsl`.

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

G. Correia, V. Niculae, W. Aziz, and A. Martins. Efficient marginalization of discrete and structured latent variables via sparsity. *Advances in Neural Information Processing Systems*, 33:11789–11802, 2020.

A. Deng and B. Hooi. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4027–4035, 2021.

P. Di Lorenzo, P. Banelli, E. Isufi, S. Barbarossa, and G. Leus. Adaptive graph signal processing: Algorithms and optimal sampling strategies. *IEEE Transactions on Signal Processing*, 66(13):3584–3598, 2018.

J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, and R. A. Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.

W. Falcon and The PyTorch Lightning team. PyTorch Lightning, 3 2019. URL `https://github.com/PyTorchLightning/pytorch-lightning`.

M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

J. Foerster, G. Farquhar, M. Al-Shedivat, T. Rocktäschel, E. Xing, and S. Whiteson. Dice: The infinitely differentiable monte carlo estimator. In *International Conference on Machine Learning*, pages 1529–1538. PMLR, 2018.

L. Franceschi, M. Niepert, M. Pontil, and X. He. Learning discrete structures for graph neural networks. In *International conference on machine learning*, pages 1972–1982. PMLR, 2019.

M. Fréchet. Les éléments aléatoires de nature quelconque dans un espace distancié. In *Annales de l'institut Henri Poincaré*, volume 10, pages 215–310, 1948.

J. Gao and B. Ribeiro. On the equivalence between temporal and static equivariant graph representations. In *International Conference on Machine Learning*, pages 7052–7076. PMLR, 2022.

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

P. Glasserman and Y.-C. Ho. *Gradient estimation via perturbation analysis*, volume 116. Springer Science & Business Media, 1991.

W. Grathwohl, D. Choi, Y. Wu, G. Roeder, and D. Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In *International Conference on Learning Representations*, 2018.

C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

A. C. Harvey et al. Forecasting, structural time series models and the Kalman filter. *Cambridge Books*, 1990.

I. A. Huijben, W. Kool, M. B. Paulus, and R. J. Van Sloun. A review of the gumbel-max trick and its extensions for discrete stochasticity in machine learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

E. Isufi, A. Loukas, N. Perraudin, and G. Leus. Forecasting time series with VARMA recursions on graphs. *IEEE Transactions on Signal Processing*, 67(18):4870–4885, 2019.

H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. Big data and its technical challenges. *Communications of the ACM*, 57 (7):86–94, 2014.

B. J. Jain. Statistical graph space analysis. *Pattern Recognition*, 60:802–812, 2016.

E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. 2017.

A. Kazi, L. Cosmo, S.-A. Ahmadi, N. Navab, and M. Bronstein. Differentiable graph module (dgm) for graph convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *Proceedings of International Conference on Learning Representations*, 2013.

T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pages 2688–2697. PMLR, 2018.

T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

W. Kool, H. Van Hoof, and M. Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pages 3499–3508. PMLR, 2019.

W. Kool, H. van Hoof, and M. Welling. Estimating gradients for discrete random variables by sampling without replacement. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rklEj2EFvB`.

P. L'Ecuyer. Note: On the interchange of derivative and expectation for likelihood ratio derivative estimators. *Management Science*, 41(4):738–747, 1995.

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

C. Maddison, A. Mnih, and Y. Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

I. Marisca, A. Cini, and C. Alippi. Learning to reconstruct missing data from spatiotemporal graphs with sparse observations. *Advances in Neural Information Processing Systems*, 35: 32069–32082, 2022.

J. Mei and J. M. Moura. Signal processing on graphs: Causal modeling of unstructured data. *IEEE Transactions on Signal Processing*, 65(8):2077–2092, 2016.

P. Minervini, L. Franceschi, and M. Niepert. Adaptive perturbation-based gradient estimation for discrete latent variable models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 9200–9208, 2023.

A. Mnih and K. Gregor. Neural variational inference and learning in belief networks. In *International Conference on Machine Learning*, pages 1791–1799. PMLR, 2014.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement rearning. In *International Conference on Machine Learning*, pages 1928–1937. PMLR, 2016.

S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih. Monte carlo gradient estimation in machine learning. *J. Mach. Learn. Res.*, 21(132):1–62, 2020.

neptune.ai. Neptune: Metadata store for mlops, built for research and production teams that run a lot of experiments, 2021. URL `https://neptune.ai`.

V. Niculae, C. F. Corro, N. Nangia, T. Mihaylova, and A. F. Martins. Discrete latent structure in neural networks. *arXiv preprint arXiv:2301.07473*, 2023.

M. Niepert, P. Minervini, and L. Franceschi. Implicit mle: backpropagating through discrete exponential family distributions. *Advances in Neural Information Processing Systems*, 34: 14567–14579, 2021.

B. N. Oreshkin, A. Amini, L. Coyle, and M. J. Coates. FC-GAGA: Fully connected gated graph architecture for spatio-temporal traffic forecasting. In *AAAI*, 2021.

A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5): 808–828, 2018.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. 2019.

M. Paulus, D. Choi, D. Tarlow, A. Krause, and C. J. Maddison. Gradient estimation with stochastic softmax tricks. *Advances in Neural Information Processing Systems*, 33: 5691–5704, 2020.

P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini. Recipe for a general, powerful, scalable graph transformer. *arXiv preprint arXiv:2205.12454*, 2022.

S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel. Self-critical sequence training for image captioning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7008–7024, 2017.

R. Y. Rubinstein. *Some problems in Monte Carlo optimization.* PhD thesis, University of Riga, 1969.

V. G. Satorras, S. S. Rangapuram, and T. Januschowski. Multivariate time series forecasting with latent graph inference. *arXiv preprint arXiv:2203.03423*, 2022.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

J. Schulman, N. Heess, T. Weber, and P. Abbeel. Gradient estimation using stochastic computation graphs. *Advances in Neural Information Processing Systems*, 28, 2015.

Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer, 2018.

C. Shang and J. Chen. Discrete graph structure learning for forecasting multiple time series. In *Proceedings of International Conference on Learning Representations*, 2021.

L. Stanković, D. Mandic, M. Daković, M. Brajović, B. Scalzo, S. Li, A. G. Constantinides, et al. 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, 2020.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12, 1999.

G. Tucker, A. Mnih, C. J. Maddison, J. Lawson, and J. Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. *Advances in Neural Information Processing Systems*, 30, 2017.

E. van Krieken, J. M. Tomczak, and A. T. Teije. Storchastic: A framework for general stochastic automatic differentiation. In *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=KAFyFabsK88`.

T. Variddhisai and D. Mandic. Methods of adaptive signal processing on graphs using vertex-time autoregressive models. *arXiv preprint arXiv:2003.05729*, 2020.

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

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

T. Weber, N. Heess, L. Buesing, and D. Silver. Credit assignment techniques in stochastic computation graphs. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2650–2660. PMLR, 2019.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang. Graph wavenet for deep spatial-temporal graph modeling. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 1907–1913, 2019.

Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang. 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, 2020.

Z. Wu, D. Zheng, S. Pan, Q. Gan, G. Long, and G. Karypis. Traversenet: Unifying space and time in message passing for traffic forecasting. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

X. Yi, Y. Zheng, J. Zhang, and T. Li. St-mvl: filling missing values in geo-sensory time series data. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, 2016.

B. Yu, H. Yin, and Z. Zhu. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 3634–3640, 2018.

D. Zambon and C. Alippi. Az-whiteness test: a test for uncorrelated noise on spatio-temporal graphs. *To appear in Advances in Neural Information Processing Systems*, 2022.

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

D. Zügner, F.-X. Aubet, V. G. Satorras, T. Januschowski, S. Günnemann, and J. Gasthaus. A study of joint graph inference and forecasting. *arXiv preprint arXiv:2109.04979*, 2021.