Graph Deep Learning for Time Series Forecasting

Doctoral Dissertation submitted to the
Faculty of Informatics of the Università della Svizzera italiana
In partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Andrea Cini

under the supervision of
Cesare Alippi

August 2024

Dissertation Committee

Prof. Michael Bronstein	Università della Svizzera italiana, CH University of Oxford, UK
Prof. Luca Maria Gambardella	Università della Svizzera italiana, CH
Prof. Davide Bacciu	University of Pisa, IT
Prof. Mathias Niepert	University of Stuttgart, DE
Prof. Peter Tiňo	University of Birmingham, UK

Dissertation accepted on 13 August 2024

Research Advisor	PhD Program Director
Cesare Alippi	**Prof. Walter Binder / Prof. Stefan Wolf**

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Andrea Cini
Lugano, 13 August 2024

Abstract

Neural networks have been used to forecast time series for decades. One of the key elements enabling most of the field's recent achievements is the training of a single neural network on large collections of related time series. This approach, however, often considers each time series independently from the others and, consequently, discards dependencies that might be instrumental for accurate predictions. Nonetheless, the alternative of modeling the full collection as a large multivariate time series cannot scale as it does not exploit the structure of the data to reduce computational and sample complexity. In this research, we aim to address the shortcomings of the state of the art in correlated time series forecasting by relying on graph representations and graph deep learning methods. We propose graph-based predictors that model pairwise relationships among time series by conditioning forecasts on a (possibly dynamic) graph spanning the collection. The conditioning is implemented as an architectural bias directly embedded into the processing; as we will show, this inductive bias enables the training of global forecasting models on large sensor networks by accounting for local correlations (graph edges) among time series (graph nodes). Our research introduces a comprehensive methodological framework characterizing the family of these predictive models and provides design principles for graph-based forecasting. Within this context, we propose methods to tackle the inherent challenges of the field, i.e., dealing with missing data, sparse observations, local effects, and latent relational dependencies. The computational scalability of the proposed framework is also addressed, together with methodologies enabling transfer learning and hierarchical forecasting. Extensive empirical results validate the introduced methodologies and place graph deep learning methods among the most important tools available in modern forecasting.

Contents

Contents iii

List of Figures xi

List of Tables xiii

List of Acronyms xv

1 Introduction 1
- 1.1 Goals and challenges . . . 3
- 1.2 Contributions . . . 4
- 1.3 Publications and dissemination . . . 7
- 1.4 Outline . . . 10

2 Forecasting correlated time series 11
- 2.1 Problem settings . . . 12
  - 2.1.1 Correlated time series . . . 12
- 2.2 Multi-step time series forecasting . . . 14
  - 2.2.1 Point predictors . . . 14
- 2.3 Global and local models . . . 15
  - 2.3.1 Global and local predictors . . . 15
  - 2.3.2 Global and local models for correlated time series . . . 17
- 2.4 Performance metrics and model selection . . . 18
- 2.5 Related work . . . 19
  - 2.5.1 Deep learning architectures for sequence modeling . . . 20
  - 2.5.2 Neural forecasting architectures . . . 22

3 Graph deep learning for time series forecasting 25
- 3.1 Graph-based representations for collections of time series . . . 26
  - 3.1.1 Extensions to the reference settings . . . 27

3.2 Forecasting with relational side information 28
3.3 Spatiotemporal graph neural networks 29
3.3.1 Message-passing neural networks 30
3.3.2 Spatiotemporal message passing 31
3.3.3 A template architecture 32
3.3.4 Taxonomy 32
3.3.5 Globality and locality in STGNNs 35
3.4 Related work 35
3.4.1 Graph deep learning for temporal data 36

4 Benchmarks and baselines 39
4.1 Benchmarks 40
4.1.1 Data from real sensor networks 40
4.1.2 Synthetic data 41
4.2 Baselines 42
4.3 Some empirical results 44

5 Local effects 47
5.1 Dealing with local effects 48
5.2 Hybrid global-local STGNNs 49
5.3 Node embeddings 50
5.3.1 Amortized specialization 51
5.3.2 Structuring the embedding space 51
5.4 Transferability 53
5.5 Related work 54
5.6 Empirical results 54
5.6.1 Synthetic data 55
5.6.2 Benchmarks 56
5.6.3 Transfer learning 58
5.7 Discussion and future directions 60

6 Missing data 61
6.1 Dealing with missing data 63
6.1.1 Related work 63
6.2 Problem definition 65
6.3 Graph Recurrent Imputation Network 66
6.3.1 Spatiotemporal encoder 67
6.3.2 Spatial decoder 68
6.3.3 Bidirectional model 69

6.3.4 Discussion and limitations 70
6.4 Spatiotemporal Point Imputation Network 71
6.4.1 Model conceptualization 71
6.4.2 Sparse spatiotemporal attention 72
6.4.3 Spatiotemporal positional encoding 74
6.4.4 Discussion and limitations 75
6.5 Empirical results 75
6.5.1 In-sample and out-of-sample imputation 76
6.5.2 Imputation benchmarks 77
6.5.3 Virtual sensing 79
6.6 Discussion and future directions 80

7 Latent graph learning 83
7.1 Latent graph learning 84
7.1.1 Learning an adjacency matrix 84
7.1.2 Learning distributions over graphs 86
7.1.3 Related work 87
7.2 Preliminaries 89
7.2.1 Reference settings 89
7.2.2 Mean adjacency matrices 89
7.3 Problem formulation 91
7.3.1 Graph learning from correlated time series 91
7.3.2 Core challenge 92
7.4 Score-based graph learning from correlated time series 93
7.4.1 Estimating gradients for stochastic message-passing networks 94
7.4.2 Graph distributions, graphs sampling, and graphs likelihood 95
7.4.3 Parametrizing the graph distribution 98
7.5 Reducing the variance of the estimator 99
7.5.1 Control variates and baselines 99
7.6 Layer-wise sampling and surrogate objective 103
7.6.1 Surrogate objective 104
7.7 Empirical results 105
7.7.1 Datasets 105
7.7.2 Controlled environment experiments 106
7.7.3 Real-world datasets 110
7.7.4 Scalability 112
7.8 Conclusions and future directions 113

8 Computational scalability **115**
8.1 Dealing with large time series collection . . . 116
8.1.1 Computational scalability in STGNNs . . . 116
8.2 Preliminaries . . . 118
8.2.1 Echo state networks . . . 118
8.3 Scalable spatiotemporal GNNs . . . 118
8.3.1 Scalable spatiotemporal representation . . . 119
8.3.2 Multi-scale decoder . . . 121
8.3.3 Training and sampling . . . 122
8.4 Related work . . . 123
8.5 Empirical results . . . 123
8.5.1 Experimental setup . . . 125
8.5.2 Results . . . 126
8.6 Discussion and future directions . . . 129

9 Graph-based hierarchical forecasting **131**
9.1 Hierarchical time series and graph clustering . . . 133
9.1.1 Related work . . . 134
9.2 Preliminaries . . . 135
9.2.1 Operational settings . . . 135
9.2.2 Hierarchical time series . . . 136
9.3 Graph-based hierarchical clustering and forecasting . . . 137
9.3.1 Graph-based hierarchical forecasting . . . 137
9.3.2 End-to-end clustering and forecasting . . . 140
9.3.3 Forecast reconciliation . . . 141
9.4 Empirical results . . . 142
9.4.1 End-to-end hierarchical clustering and forecasting . . . 143
9.4.2 Cluster analysis . . . 146
9.5 Discussion and future directions . . . 147

10 Conclusion **149**
10.1 Future directions . . . 150
10.2 Final remarks . . . 151

A Torch Spatiotemporal **153**
A.1 Related work . . . 154

B Performance metrics **155**

C Experimental setup **157**

D Appendix to Chapter 4 **159**
D.1 Additional details on the experimental setup 159
D.1.1 Reference architectures 159
D.1.2 Hyperparameters 160

E Appendix to Chapter 5 **161**
E.1 Transfer learning experiment 161
E.1.1 Additional results 162

F Appendix to Chapter 6 **165**
F.1 Additional details on the experimental setup 165
F.1.1 Hyperparameters 165

G Appendix to Chapter 7 **167**
G.1 Deferred proofs 167
G.1.1 Proof of Lemma 1 167
G.1.2 Proof of Lemma 2 168
G.2 Details on the computation of the SNS likelihood 168
G.3 Additional details on the experimental setup 169
G.3.1 Synthetic experiments 169
G.3.2 AQI experiment 170
G.3.3 Traffic experiment 170

H Appendix to Chapter 8 **171**
H.1 Additional details on the experimental setup 171
H.1.1 Hardware platform 171
H.1.2 Datasets 171
H.1.3 Additional details on the SGP architecture 172
H.1.4 Training and evaluation procedure 172

I Appendix to Chapter 9 **175**
I.1 Additional details on the experimental setup 175
I.1.1 Reference architecture 175
I.1.2 Hyperparameters and training details 175

Bibliography **177**

Figures

1.1 Thesis outline 4

2.1 Operational settings 12

3.1 Graph-based representation of a time series collection 26

4.1 GPVAR community graph 42

5.1 Time series clusters obtained by clustering node embeddings. . . 58

6.1 The architecture of GRIN 67
6.2 The architecture of SPIN 72
6.3 Sparse spatiotemporal attention block 73
6.4 Virtual sensing with GRIN 81

7.1 Graph learning architecture. 94
7.2 Graph learning on GPVAR 107
7.3 Sensitivity analysis (graph learning surrogate objective) 108
7.4 Sensitivity analysis (number of neighbors) 108
7.5 Comparison of different gradient estimators. 109
7.6 Computational scalability of different gradient estimators 112

8.1 Overview of SGP forecasting framework 117
8.2 Overview of the SGP encoder 119
8.3 Training curves on PV-US 129

9.1 Hierarchical time series 136
9.2 Time series with a hierarchical relational structure 137
9.3 Cluster assignments learned by HiGP 146

A.1 Torch Spatiotemporal logo. 153

Tables

4.1 Details on the adopted datasets. 40
4.2 Evaluation of reference architectures on GPVAR-G. 43
4.3 Evaluation of reference architectures on benchmark data. 45

5.1 Evaluation of different RNN+IMP baselines. 50
5.2 Evaluation of different models on GPVAR-L. 55
5.3 Evaluation of global-local forecasting architectures with learnable node embeddings. 57
5.4 PEMS datasets 58
5.5 Evaluation of different transfer learning approaches with fine-tuning on a week of observations 59
5.6 Sensitivity analysis of transfer learning approaches w.r.t. fine-tuning set size 59

6.1 Evaluation of GRIN in in-sample and out-of-sample imputation on 78
6.2 Evaluation of imputation models. 79
6.3 Evaluation of imputation models under increasing levels of data sparsity. (Point missing) 80
6.4 Evaluation of imputation models under increasing levels of data sparsity. (Block missing) 80

7.1 Graph identification on AQI. 110
7.2 Evaluation of different forecasting architectures equipped with a graph structure learning module. 111

8.1 Scalability benchmarks. 124
8.2 Evaluation of SGP against the state of the art. 127
8.3 Evaluation of different architectures on scalability benchmarks. . 128

9.1 Evaluation of HiGP on traffic benchmark datasets. 144

9.2 Evaluation of HiGP on traffic forecasting benchmarks. 145

E.1 Additional transfer learning results on PEMS03 162
E.2 Additional transfer learning results on PEMS04 163
E.3 Additional transfer learning results on PEMS07 163
E.4 Additional transfer learning results on PEMS08 163

Acronyms

BES *binary edge sampler*. 95, 96, 98, 100, 101, 103, 106, 108, 109

CG computational graph. 92, 93, 95

DS deep set. 17

ESN echo state network. 20, 117–120, 126, 172

FR *forecast reconciliation*. 133, 141, 142

GCRNN graph convolutional recurrent neural network. 32, 33, 42–44, 63, 66, 67, 124

GDL graph deep learning. 2–5, 7, 8, 10, 36, 54, 61, 63, 64, 79, 80, 84, 133, 149

GNN graph neural network. 2, 24, 29, 30, 32, 36, 53, 65, 67, 84, 88, 133, 134, 143, 151, 154, 170

GRIN *Graph Recurrent Imputation Network*. xi, xiii, 63, 64, 66, 67, 69–71, 75–81, 149, 165

GRU gated recurrent unit. 32, 42, 43, 67, 110, 112, 159, 170

HiGP *Hierarchical Graph Predictor*. xi, xiii, xiv, 131, 133, 138, 141–147, 150, 175, 176

MAE mean absolute error. 18, 43–45, 55, 56, 70, 77, 78, 80, 106, 110, 111, 126, 155, 161, 174

MAPE mean absolute percentage error. 18, 126, 155, 156

MC Monte Carlo. 87, 92–94, 99, 104, 134

MIMO multiple-input-multiple-output. 17

MISO multiple-input-single-output. 17

MLP multilayer perceptron. 20, 21, 23, 30, 32, 66, 67, 69, 72, 76, 117, 119, 121–123, 135, 139, 143, 145, 159, 170, 172, 173

MP message-passing. 2, 29–37, 42, 43, 51, 56, 61, 67, 68, 76, 83–85, 87, 92, 93, 95, 98, 103, 110, 112, 116, 121, 135, 138, 139, 143–145, 159, 173

MPG message-passing graph. 93

MPGRU *message-passing GRU*. 68, 69, 76–78, 165

MRE mean relative error. 18, 78, 156

MSE mean squared error. 18, 78, 155

NARX nonlinear autoregressive exogenous. 20

RMSE root mean squared error. 18

RNN recurrent neural network. 6, 16, 20–23, 32–34, 42–45, 50, 55, 57, 61, 64, 76, 85, 118, 143, 159, 160

SF *score-function*. 87, 94

SGP *Scalable Graph Predictor*. xiii, 115, 117–119, 123, 125–130, 150, 172–174

SNS *subset neighborhood sampler*. 96, 98, 100–103, 106–112

SPIN *Spatiotemporal Point Inference Network*. 71–75, 77–80, 149, 166

SRC *select, reduce, connect*. 137

SSM state-space models. 21, 23, 37

STGNN spatiotemporal graph neural network. vi, 2, 7, 10, 25, 29, 31–33, 35, 47–49, 51, 53, 54, 56–58, 61, 63, 64, 83–86, 92, 115, 116, 122, 127, 129, 134, 151, 153, 159

STMP spatiotemporal message-passing. 31–33, 35, 43, 49, 72–74, 154, 159

STT space-then-time. 32, 34, 35, 116

T&S time-and-space. 32–35, 42–45, 55, 92, 95, 112, 116, 145, 159

TCN temporal convolutional network. 16, 20, 21, 23, 24, 33, 34, 44

TTS time-then-space. 32, 34, 35, 42, 43, 45, 49, 55, 92, 110, 116, 123, 125, 127, 130, 135, 139, 143, 159, 170, 175

Chapter 1

Introduction

In modern cyber-physical systems, physical and virtual sensors continuously produce large amounts of data. The result is a large collection of correlated time series that learning systems should integrate and process to make accurate predictions. Exploiting temporal dependencies as well as relationships across time series is crucial to building models that can scale and make accurate predictions. Proper inductive biases, i.e., soft or hard structural assumptions steering the learning system toward the most plausible models, are and have been among the key ingredients enabling many of the successes of deep learning systems [Hochreiter and Schmidhuber, 1997; Sperduti and Starita, 1997; LeCun and Bengio, 1998]. Similarly, traditional statistical methods for time series analysis leverage the structure of the data generating process to obtain effective models [Harvey et al., 1990; Hyndman et al., 2002].

Deep learning models have become fundamental tools in modern forecasting practice [Benidis et al., 2022; Petropoulos et al., 2022]. The most successful approach consists of training a single deep network on collections of related time series while sharing parameters [Smyl, 2020; Salinas et al., 2020; Benidis et al., 2022]. Although this allows for training large (*global*) models on vast amounts of data, the resulting predictors process a time series at a time: they cannot take advantage of existing relationships among time series. Conversely, considering sequences in a collection as a single multivariate time series suffers from the curse of dimensionality, does not enjoy the benefits of parameter sharing, and does not take into account any structural (spatial) prior knowledge. Consider the setting we discussed in the first paragraph, i.e., the large variety of sensors that permeate any modern infrastructure (e.g., traffic networks and smart grids). The resulting sets of time series have inherently rich *spatiotemporal structure* and *spatiotemporal dynamics*. In these settings, the mentioned standard approaches

appear hopelessly inadequate. In particular, they cannot account for any prior on the existing relationships and would most likely fall short in capturing dependencies among time series. Pruning of spurious relationships by exploiting prior knowledge could help, provided a method for encoding the appropriate learning biases.

In this research, we argue that graph deep learning (GDL) [Bacciu et al., 2020; Bronstein et al., 2021] provides the appropriate framework and architectural biases to go beyond the limitations of the current state of the art in deep learning for time series forecasting. Within the GDL framework, dependencies can be modeled in terms of pairwise relationships among time series in the collection. The resulting representation is a graph where each time series is associated with a node and functional relationships among them are represented as edges. The conditioning of the predictor on observations at correlated time series can then be embedded as an inductive bias into the neural architecture. Graph neural networks (GNNs), e.g., based on the message-passing (MP) framework [Gilmer et al., 2017], provide the suitable neural operators and the proper computational and modeling framework. In particular, GNNs allow for both sharing parameters among time series and accounting for observations at neighboring nodes (related time series) [Cini et al., 2024].

Since the first applications of GDL to time series processing [Li et al., 2018; Yu et al., 2018], the resulting models, called spatiotemporal graph neural networks (STGNNs), have become well-established among practitioners [Jin et al., 2023b]. STGNNs implement global and inductive architectures for forecasting correlated time series, addressing the shortcomings of standard deep learning predictors and opening up a large design space of methodologies and architectures. Consequently, researchers have been proposing a large variety of STGNNs by integrating MP into popular sequence modeling architectures, e.g., by exploiting MP to implement the gates of recurrent cells [Seo et al., 2018; Li et al., 2018] and propagate representations in fully convolutional [Yu et al., 2018; Wu et al., 2019] and attention-based architectures [Zheng et al., 2020; Wu et al., 2022; Marisca et al., 2022]. The resulting STGNNs have been successful in a wide range of time series benchmarks ranging from traffic flow prediction [Li et al., 2018; Yu et al., 2018; Wu et al., 2019], air quality monitoring [Chen et al., 2021b; Iskandaryan et al., 2023], and energy analytics [Eandi et al., 2022; Cini et al., 2023a], to financial time series [Chen et al., 2018b; Matsunaga et al., 2019], and biomedical signal processing Zhang et al. [2022]. However, the foundations of the field had not been systematically laid out yet. In particular, observed empirical results had not been contextualized within a proper methodological framework. For example, global and local aspects of graph-based forecasting models have

been discussed, for the first time, only in the context of this thesis [Cini et al., 2023c]. Furthermore, methodologies to deal with the structural challenges in adopting such models in practical applications were lacking. These challenges, as discussed in the next section, include dealing with heterogeneous dynamics, missing data, latent dependencies, and computational scalability issues. In this research, we tackle such challenges and show that graph-based neural operators allow for building effective, principled, and scalable forecasting models.

1.1 Goals and challenges

The objective of the research project can be summarized as follows.

Goal The goal of the thesis is to introduce a comprehensive methodological framework instrumental to designing GDL methods for time series forecasting. We frame the problem from the appropriate perspective and provide the necessary tools and theory. We show that graph-based processing allows for building scalable global predictors while taking structural dependencies among time series into account.

To achieve this goal, we introduce methodologies to deal with challenges inherent to the adopted processing framework. Said challenges can be detailed as follows.

Ch. 1: Local effects. Time series in a collection might be heterogeneous and characterized by specific dynamics (*local effects*) not easily captured by a global model. This issue might be addressed by specializing graph-based forecasting models to such dynamics, which, however, makes the transferability of the learned model to different sensor networks more challenging.

Ch. 2: Missing data. Missing data and intermittent observations (across both time and space) are inherent to any application involving a network of sensors. This makes the processing challenging, particularly in those applications characterized by extremely sparse observations.

Ch. 3: Latent graph learning. Available relational information can be inaccurate, inadequate, or completely missing. Exploiting relational architectures in such a setting requires learning graphs from data.

Ch. 4: Scalability. In real-world applications, tens of thousands of sensors acquire data at high frequency. Processing the resulting observations

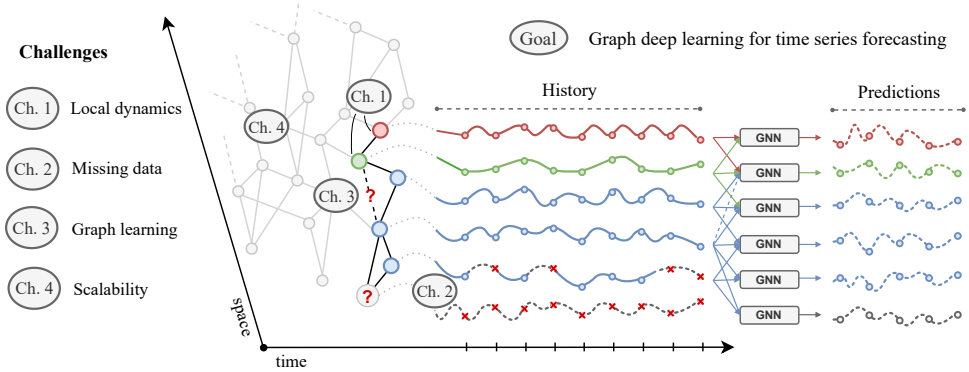


Figure 1.1. Outline of the research project highlighting the research goal and related challenges.

over both time and space has a high computational cost. Effectively exploiting the available spatiotemporal dependencies can then be extremely challenging and requires ad-hoc scalable operators.

A schematic outline of the project, its goal, and associated challenges is provided in Figure 1.1. To address each challenge, we designed several graph-based methodologies for time series analysis. The following section summarizes the main contributions of the thesis.

1.2 Contributions

In this research, we proposed GDL methods for time series forecasting addressing the foundations of the field. The following briefly details the main contributions; for each conceptual contribution, we report the associated reference papers.

Graph deep learning for time series forecasting We developed novel methods and techniques to build, understand, and scale graph-based predictors, i.e., methods to process and forecast collections of correlated time series given graph-based input representations. In particular, we provided a formalization of the problem settings and the associated graph-based representations and predictors. We discussed available design choices and their implications, providing guidelines to address associated challenges. This has been the overreaching goal of the thesis, and the main results of the research have been summarized in the following tutorial paper.

- Andrea Cini, Ivan Marisca, Daniele Zambon, and Cesare Alippi. Graph Deep Learning for Time Series Forecasting. *arXiv preprint arXiv:2310.15978*, 2023b

Global-local graph-based forecasting Concerning **(Challenge 1)**, we developed methodologies to address local effects and transfer learned models to different node sets. In particular, we considered the problem of learning graph-based forecasting architecture combining shared global components with local processing blocks, i.e., modules with node-specific parameters. The associated techniques and trade-offs were introduced and discussed in the context of global and local models for time series forecasting. In particular, we discussed and addressed the implications of introducing local components in both transductive and transfer learning scenarios. The results of this research has been presented in the following work.

- Andrea Cini, Ivan Marisca, Daniele Zambon, and Cesare Alippi. Taming Local Effects in Graph-based Spatiotemporal Forecasting. *Advances in Neural Information Processing Systems*, 2023c

Graph deep learning for time series imputation Within the thesis, we pioneered several state-of-the-art GDL methodologies for missing data imputation **(Challenge 2)** tackling the problem of processing irregular time series and sparse observations. The introduced methods exploit graph-based representations to reconstruct missing data by exploiting available observations at the target time series and neighboring nodes. In particular, we introduced a bidirectional recurrent architecture [Cini et al., 2022b] and a follow-up attention-based model [Marisca et al., 2022], complementing and addressing the limitations of the former. Our work opened up a line of research, i.e., that of graph-based neural imputation, which is currently very active and represents the state of the art in several applications. These methodologies have been introduced in the following papers.

- Andrea Cini, Ivan Marisca, and Cesare Alippi. Filling the G_ap_s: Multivariate Time Series Imputation by Graph Neural Networks. In *International Conference on Learning Representations*, 2022b

- Ivan Marisca, Andrea Cini, and Cesare Alippi. Learning to Reconstruct Missing Data from Spatiotemporal Graphs with Sparse Observations. In *Advances in Neural Information Processing Systems*, 2022

- Giovanni De Felice, Andrea Cini, Daniele Zambon, Vladimir Gusev, and Cesare Alippi. Graph-based Virtual Sensing from Sparse and Partial Multivariate Observations. In *International Conference on Learning Representations*, 2024

Score-based latent graph learning We tackled the problem of learning latent graph structures to process sets of time series with no prior relational information attached **(Challenge 3)**. In particular, we introduced methodologies to learn distributions over graphs while keeping downstream message-passing computations sparse and efficient at both inference and training time. We achieved this result by introducing novel variance-reduced score-based estimators to learn the parameters of a probabilistic model embedding sparsity priors. The resulting graph learning framework has also then allowed us to introduce state-space models where input, outputs, and states are represented as graphs [Zambon et al., 2023]. The methodology and the associated theoretical and technical results have been presented in the following journal paper.

- Andrea Cini, Daniele Zambon, and Cesare Alippi. Sparse graph learning from spatiotemporal time series. *Journal of Machine Learning Research*, 24(242):1–36, 2023d

Scalable graph-based forecasting Regarding the scalability challenge **(Challenge 4)**, we proposed a scalable forecasting architectures combining a propagation process on the graph structure and deep randomized recurrent neural networks (RNNs). The introduced architecture extracts spatiotemporal representations without requiring training; such representations can be precomputed and then sampled as if they were i.i.d. to train a decoder to map them to predictions. This precomputation strategy makes the cost of a training step independent of graph size and sequence length. The associated paper won the best paper award at the most prominent workshop on processing dynamic relational data.

- Andrea Cini, Ivan Marisca, Filippo Maria Bianchi, and Cesare Alippi. Scalable Spatiotemporal Graph Neural Networks. *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, 2023a

Graph-based hierarchical forecasting Besides addressing the identified challenges, we also made a first step toward graph-based models operating at adaptive spatial resolution. In particular, we introduced a methodology unifying

graph-based forecasting with hierarchical time series processing. The resulting framework allows for operating on aggregates rather than on single time series, thus allowing for modeling higher-order structures. The mechanism used to group (cluster) time series is learned directly by exploiting a self-supervised training mechanism. This end-to-end approach combines relational and hierarchical inductive biases into a single framework and allows for forecasting collections of time series while clustering them.

- Andrea Cini, Danilo Mandic, and Cesare Alippi. Graph-based Time Series Clustering for End-to-End Hierarchical Forecasting. *International Conference on Machine Learning*, 2024

Software Most of the methodologies designed within the thesis have been published together with open-source implementations of the associated time series processing pipelines. Furthermore, we developed and open-sourced *Torch Spatiotemporal*, a software library for prototyping STGNNs and processing time series collections with relational inductive biases (see Appendix A).

- Andrea Cini and Ivan Marisca. Torch Spatiotemporal, 2022. URL https://github.com/TorchSpatiotemporal/tsl

Applications The methods developed during the project have found several, practical, real-world applications in collaboration with Siemens (Zürich) and DXT Commodities (Lugano). In particular, we applied GDL methods to process data coming from sensor networks in smart grid and load forecasting applications and, more recently, to the processing of biomedical data.

1.3 Publications and dissemination

Within the thesis, I co-authored 17 papers. Most of these have been published in the top venues of the field such as NeurIPS [Marisca et al., 2022; Cini et al., 2023c], ICML [Cini et al., 2024], ICLR [Cini et al., 2022b; De Felice et al., 2024], AAAI [Cini et al., 2023a], JMLR [Cini et al., 2023d; D'Eramo et al., 2021], TMLR [Butera et al., 2024], and other international conferences and journals [Cini et al., 2020; Eandi et al., 2022; Cini et al., 2022a; Ferretti et al., 2022; Efkarpidis et al., 2023]; the remaining papers are currently under review or in the process of being submitted [Cini et al., 2023b; Zambon et al., 2023; Marzi et al., 2023]. The research on scalable STGNNs [Cini et al., 2023a], later published at AAAI, won the best paper award at the TGL workshop at

NeurIPS 2022, which is the most prominent workshop on the application of GDL to temporal data. I have organized a tutorial on graph deep learning for time series forecasting at ECML PKDD 2023 and given talks on my work in both industrial and academic settings. The following is the list of the papers I worked on during my PhD. The papers that constitute the core of the thesis have already been highlighted in Section 1.2; the full publication list is reported here.

Full publication list

Journal papers

1. Andrea Cini, Daniele Zambon, and Cesare Alippi. Sparse graph learning from spatiotemporal time series. *Journal of Machine Learning Research*, 24(242):1–36, 2023d

2. Carlo D'Eramo, Andrea Cini, Alessandro Nuara, Matteo Pirotta, Cesare Alippi, Jan Peters, and Marcello Restelli. Gaussian Approximation for Bias Reduction in Q-Learning. *Journal of Machine Learning Research*, 22:1–51, 2021

3. Luca Butera, Andrea Cini, Alberto Ferrante, and Cesare Alippi. Object-Centric Relational Representations for Image Generation. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856

4. Lorenzo Ferretti, Andrea Cini, Georgios Zacharopoulos, Cesare Alippi, and Laura Pozzi. Graph neural networks for high-level synthesis design space exploration. *ACM Transactions on Design Automation of Electronic Systems*, 28(2):1–20, 2022

5. Nikolaos A Efkarpidis, Stefano Imoscopi, Martin Geidl, Andrea Cini, Slobodan Lukovic, Cesare Alippi, and Ingo Herbst. Peak shaving in distribution networks using stationary energy storage systems: A Swiss case study. *Sustainable Energy, Grids and Networks*, 34:101018, 2023

Conference papers

6. Andrea Cini, Ivan Marisca, and Cesare Alippi. Filling the G_ap_s: Multivariate Time Series Imputation by Graph Neural Networks. In *International Conference on Learning Representations*, 2022b

7. Andrea Cini, Ivan Marisca, Daniele Zambon, and Cesare Alippi. Taming Local Effects in Graph-based Spatiotemporal Forecasting. *Advances in Neural Information Processing Systems*, 2023c

8. Andrea Cini, Ivan Marisca, Filippo Maria Bianchi, and Cesare Alippi. Scalable Spatiotemporal Graph Neural Networks. *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, 2023a

9. Andrea Cini, Danilo Mandic, and Cesare Alippi. Graph-based Time Series Clustering for End-to-End Hierarchical Forecasting. *International Conference on Machine Learning*, 2024

10. Ivan Marisca, Andrea Cini, and Cesare Alippi. Learning to Reconstruct Missing Data from Spatiotemporal Graphs with Sparse Observations. In *Advances in Neural Information Processing Systems*, 2022

11. Giovanni De Felice, Andrea Cini, Daniele Zambon, Vladimir Gusev, and Cesare Alippi. Graph-based Virtual Sensing from Sparse and Partial Multivariate Observations. In *International Conference on Learning Representations*, 2024

12. Andrea Cini, Carlo D'Eramo, Jan Peters, and Cesare Alippi. Deep reinforcement learning with weighted Q-Learning. *The Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2022a

13. Andrea Cini, Slobodan Lukovic, and Cesare Alippi. Cluster-based aggregate load forecasting with deep neural networks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020

14. Simone Eandi, Andrea Cini, Slobodan Lukovic, and Cesare Alippi. Spatio-Temporal Graph Neural Networks for Aggregate Load Forecasting. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022

Preprints

15. Andrea Cini, Ivan Marisca, Daniele Zambon, and Cesare Alippi. Graph Deep Learning for Time Series Forecasting. *arXiv preprint arXiv:2310.15978*, 2023b

16. Daniele Zambon, Andrea Cini, Lorenzo Livi, and Cesare Alippi. Graph state-space models. *arXiv preprint arXiv:2301.01741*, 2023

17. Tommaso Marzi, Arshjot Khehra, Andrea Cini, and Cesare Alippi. Feudal Graph Reinforcement Learning. *arXiv preprint arXiv:2304.05099*, 2023

1.4 Outline

In more detail, the thesis is structured as follows.

Chapter 2 introduces notation and problem settings. In particular, the section focuses on the problem of forecasting correlated time series and on related deep learning methods for time series forecasting. We provide a categorization of the forecasting approaches and discuss the main limitations of existing methods.

Chapter 3 presents the proposed graph-based forecasting framework by introducing the associated representations and proposing a taxonomy of STGNN architectures. We then discuss the available design choices and their implications with respect to existing methods. Finally, we provide an assessment of STGNNs in the context of global and local methods for time series forecasting.

Chapter 4 introduces the reference benchmark datasets used throughout the thesis and reports the results of an empirical evaluation of reference architectures.

Chapter 5 discusses the problem of accounting for local dynamics in time series collection and introduces hybrid global-local graph-based forecasting architectures. Furthermore, we address the introduced methodologies in the context of transfer learning.

Chapter 6 discusses the problem of dealing with missing data and introduces models for reconstructing the missing observations.

Chapter 7 addresses the problem of efficiently learning latent graph structures by introducing ad-hoc estimators and learning architectures.

Chapter 8 discusses the computational scalability of standard GDL forecasting architectures and introduces a scalable alternative based on reservoir computing and precomputed representations.

Chapter 9 presents a novel methodology unifying hierarchical and graph-based forecasting to obtain predictors able to predict the input time series while clustering them.

Chapter 10 draws final considerations and summarizes the main outcomes of the thesis. Finally, directions and perspectives for future research are presented and discussed.

Chapter 2

Forecasting correlated time series

This chapter introduces the reference problem settings for the thesis. In particular, we consider the problem of forecasting collections of correlated time series, i.e., time series, possibly multivariate, that are correlated among each other. Differently from generic collections of time series that might share some similarity, in correlated time series, observations at related sequences allow for reducing the uncertainty of the forecasts, i.e., for making more accurate predictions. Section 2.1 introduces the scenarios under which the methodologies introduced in the thesis operate. Then, Section 2.2 defines the forecasting problem and a broad model family of point predictors addressing it. Section 2.3 introduces a key distinction within time series forecasting methodologies by defining global and local model archetypes. Performance metrics and model selection procedures are presented in Section 2.4. Finally, related work is discussed at length in Section 2.5.

Reference papers The content of the chapter is partly based on material from the following papers.

- Andrea Cini, Ivan Marisca, Daniele Zambon, and Cesare Alippi. Graph Deep Learning for Time Series Forecasting. *arXiv preprint arXiv:2310.15978*, 2023b
- Andrea Cini, Ivan Marisca, Daniele Zambon, and Cesare Alippi. Taming Local Effects in Graph-based Spatiotemporal Forecasting. *Advances in Neural Information Processing Systems*, 2023c
- Andrea Cini, Daniele Zambon, and Cesare Alippi. Sparse graph learning from spatiotemporal time series. *Journal of Machine Learning Research*, 24(242):1–36, 2023d

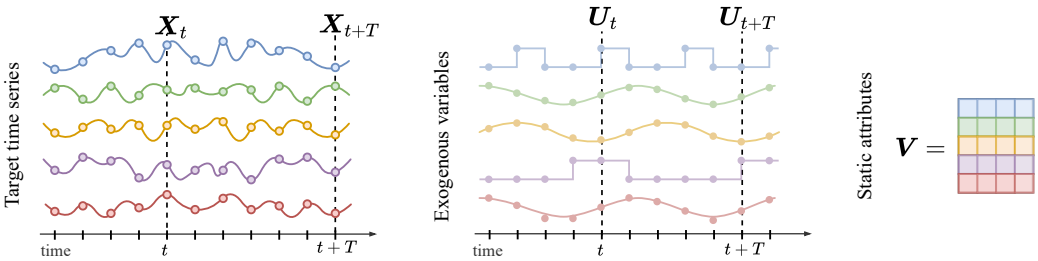


Figure 2.1. Correlated time series with exogenous variables and static attributes.

2.1 Problem settings

This section introduces the reference operational settings. Chapter 3 will extend the base scenario described here. Notably, several of the assumptions made in the following will be relaxed and challenged in Section 3.1.1. Differences in setup and additional assumptions adopted by methodologies introduced in later chapters will be made explicit whenever required.

2.1.1 Correlated time series

Consider a collection, denoted as $\mathcal{D}$, of N regularly and synchronously sampled correlated time series. The i-th time series is composed by a sequence of d_x-dimensional vectors $\boldsymbol{x}_t^i \in \mathbb{R}^{d_x}$ acquired at each time step t from sensors[1] with d_x channels. Time series are assumed to be *homogenous*, i.e., characterized by the same variables (observables). Matrix $\boldsymbol{X}_t \in \mathbb{R}^{N \times d_x}$ denotes the stacked N observations at time t, while $\boldsymbol{X}_{t:t+T}$ indicates the sequence of observations within time interval $[t, t+T)$; with the shorthand $\boldsymbol{X}_{<t}$ we indicate observations at time steps up to t (excluded). Exogenous variables (i.e., external covariates) associated with each time series are denoted by $\boldsymbol{U}_t \in \mathbb{R}^{N \times d_u}$, while static (time-independent) attributes are grouped in matrix $\boldsymbol{V} \in \mathbb{R}^{N \times d_v}$. Figure 2.1 summarizes the available data. We use tuple $\mathcal{X}_t \doteq \langle \boldsymbol{X}_t, \boldsymbol{U}_t, \boldsymbol{V} \rangle$ to denotes the observed variables associated with time step t.

System model We model each (multivariate) observation as generated by a *time-invariant stochastic process* such that

$$\boldsymbol{x}_t^i \sim p^i \left(\boldsymbol{x}_t^i \middle| \mathcal{X}_{<t}, \boldsymbol{U}_t \right) \tag{2.1}$$

[1]Here the term sensor has to be considered in a broad sense, as an entity producing a sequence of observations over time.

in particular, we assume the existence of a generic predictive causality *à la Granger* [Granger, 1969], i.e., we assume that forecasts for a single time series can benefit – in terms of accuracy – from accounting for the past values of (a subset of) other time series in the collection. These dependencies can vary in nature, e.g., from mere correlations to stronger functional constraints. Note that time series $\{\boldsymbol{x}_t^i\}_t$ might be generated by a different stochastic process, i.e., in general, $p^i \neq p^j$ if $i \neq j$. In other words, in general, each time series cannot be considered as an i.i.d. sample drawn from the same process.

Terminology In the sequel, the term *spatial* refers to the dimension of size N, that spans the time series collection, e.g., we will talk about *spatial dependendencies* to indicate dependencies among different time series. In the case of time series acquired from physical sensors, the term spatial reflects the fact that each time series might correspond to a different physical location. Analogously, we will talk about *relational dependency* to indicate pairwise dependencies among pairs of correlated time series. We will use the term *sensor* to indicate the entities generating the observed values over time.

Example 1. Consider a sensor network monitoring the speed of vehicles at crossroads. In this case, $\boldsymbol{X}_{1:t}$ refers to past traffic speed measurements sampled at a certain frequency. Exogenous variables $\boldsymbol{U}_{1:t}$ may account for time-of-the-day and day-of-the-week identifiers, and the current weather conditions. The attribute matrix $\boldsymbol{V}$ collects static features related to the sensor's position, e.g., the type of road the sensor is placed in or the number of lanes. In this scenario, time series in the collection would likely exhibit strong correlations, e.g., due to the traffic flow and how the different road segments of the traffic network are connected.

Example 2. Consider a modern smart metering infrastructure, with sensors measuring energy consumption at different households and facilities. Time series $\boldsymbol{X}_{1:t}$ correspond to sensor readings acquired over time w.r.t. each user. Exogenous variables $\boldsymbol{U}_{1:t}$, as in the previous example, may encompass weather conditions and calendar features. Static attributes $\boldsymbol{V}$ might indicate the type of customer (e.g., private or business) and type of contract. In this context, dependencies among time series might be due to relationships among users, e.g., private customers working for the same company, which would affect the observed energy consumption patterns.

2.2 Multi-step time series forecasting

We address the multi-step-ahead time-series forecasting problem [Hyndman and Athanasopoulos, 2018; Benidis et al., 2022], i.e., we are interested in predicting, for each time step t and some forecasting horizon $H \geq 1$, the h step-ahead observations $\boldsymbol{X}_{t+h}$ for all $h \in [0, H)$ given a window of available $W \geq 1$ past observations. In particular, we are interested in learning a model $p_{\boldsymbol{\theta}}$ approximating the unknown conditional probability such that

$$p_{\boldsymbol{\theta}}\left(\boldsymbol{x}^i_{t+h} \middle| \mathcal{X}_{t-W:t}, \boldsymbol{U}_{t:t+h+1},\right) \approx p^i\left(\boldsymbol{x}^i_{t+h} \middle| \mathcal{X}_{<t}, \boldsymbol{U}_{t:t+h+1}\right) \tag{2.2}$$

where $\boldsymbol{\theta}$ indicates the learnable parameters of the model which may or may not be specialized w.r.t. the i-th time series (see Section 2.3). Note that the considered models usually approximate the target distribution by conditioning the forecasts on a limited window of observations rather than the full history. Not all the exogenous variables $\boldsymbol{U}_{\leq t+h+1}$ might be available up to time step $t+h$ in practical applications and the conditioning would have to be adapted accordingly[2].

Learning a model means fitting parameters $\boldsymbol{\theta}$ to the available observations. In general, one might be interested in *probabilistic forecasts*, i.e., in modeling the full (conditional) probability distribution $p^i(\,\cdot\,)$. In the following, we will focus on *point forecasts*, i.e., on predicting specific values associated with the process such as, for example, the expected value of future observations. In particular, to keep the scope of the dissertation contained, the problem of estimating the uncertainty of the forecast will not be covered in-depth.

2.2.1 Point predictors

Limiting our analysis to point predictions and selecting the expected value as a target, we can consider predictive model families $\mathcal{F}(\,\cdot\,;\boldsymbol{\theta})$ such that

$$\widehat{\boldsymbol{X}}_{t:t+H} = \mathcal{F}\left(\mathcal{X}_{t-W:t}, \boldsymbol{U}_{t:t+H+1}; \boldsymbol{\theta}\right) \quad \text{s.t.} \quad \widehat{\boldsymbol{X}}_{t:t+H} \approx \mathbb{E}\left[\boldsymbol{X}_{t:t+H}\right]. \tag{2.3}$$

Model $\mathcal{F}\left(\,\cdot\,;\boldsymbol{\theta}\right)$ allows for several designs; for example, learnable parameters could be shared among time series which can be forecast individually – e.g., as $\boldsymbol{x}^i_t = \mathcal{F}(\boldsymbol{x}^i_{t-W:t}, \ldots; \boldsymbol{\theta})$ – or by considering the input as a single large multivariate time series. As one would expect, adopting any of these options

[2]Exogenous variables might contain, for example, actual weather conditions (available up to time step t) or estimates, e.g., weather forecasts, available for future time steps as well (up to time step $t+h$).

results in radically different modeling archetypes. Section 2.3 will discuss the different model families in-depth and 2.5 will present a selection of the resulting forecasting architectures.

Model fitting Parameters $\boldsymbol{\theta}$ can be learned by minimizing a cost function $\ell(\,\cdot\,)$ on a training set, i.e.,

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_{t=1}^{T} \ell\left(\widehat{\boldsymbol{X}}_{t:t+H}, \boldsymbol{X}_{t:t+H}\right), \tag{2.4}$$

where the cost is the *squared error*

$$\ell\left(\widehat{\boldsymbol{X}}_{t:t+H}, \boldsymbol{X}_{t:t+H}\right) = \frac{1}{N\,H} \sum_{i=1}^{N} \sum_{h=0}^{H-1} \left\| \widehat{\boldsymbol{x}}^{i}_{t+h} - \boldsymbol{x}^{i}_{t+h} \right\|_2^2. \tag{2.5}$$

Minimizing the squared loss results in forecasts of the mean (see Equation 2.3); considering different cost functions allows for obtaining point predictions of other values. For instance, forecasts of the median can be obtained by minimizing the *absolute error*, while other quantiles of the target distribution can be estimated by relying on the pinball loss [Koenker and Hallock, 2001]. Notably, forecasting multiple quantiles at the same time is a simple method to obtain probabilistic predictions [Wen et al., 2017; Gasthaus et al., 2019].

2.3 Global and local models

Having identified the reference settings, we delve into the problem of designing a forecasting model. We start by addressing a key distinction among the existing model archetypes.

2.3.1 Global and local predictors

A time series forecasting model is called *global* if its parameters are fitted to a group of time series (either univariate or multivariate); conversely, *local* models are specific to a single (possibly multivariate) time series. In different terms, a global model is trained to make predictions by learning from a set of time series, possibly generated by different stochastic processes, without learning any time-series-specific parameter. Conversely, a local model is obtained by minimizing the forecasting error on observations from a single time series. Both global and local models can be univariate or multivariate. More formally, following Benidis

et al. [2022] and *considering models that process one time series at a time*, a sequence-level local model would predict each target time series for $i = 1, \dots, N$ as

$$\hat{\boldsymbol{x}}^i_{t+h} = f^i\left(\boldsymbol{x}^i_{t-W:t}, \boldsymbol{u}^i_{t-W:t+h+1}, \boldsymbol{v}^i; \boldsymbol{\theta}^i\right) \tag{2.6}$$

where $\boldsymbol{\theta}^i$ indicates the model's parameters fitted on the i-th time series. Differently, in a global model, parameters would be shared among time series, i.e,

$$\hat{\boldsymbol{x}}^i_{t+h} = f\left(\boldsymbol{x}^i_{t-W:t}, \boldsymbol{u}^i_{t-W:t+h+1}, \boldsymbol{v}^i; \boldsymbol{\theta}\right) \tag{2.7}$$

where parameters $\boldsymbol{\theta}$ can be learned by minimizing the cost function w.r.t. the complete time series collection (see Equation 2.4). In the context of neural networks, both models can be implemented by using any sequence modeling architecture, e.g., a RNN [Hewamalage et al., 2021] or a temporal convolutional network (TCN) [Oord et al., 2016; Bai et al., 2018].

Trade-offs The advantages of global models have been discussed at length in the time series forecasting literature [Salinas et al., 2020; Januschowski et al., 2020; Montero-Manso and Hyndman, 2021; Benidis et al., 2022] and are mainly ascribable to the availability of large amounts of data that enable the use of models with a higher capacity w.r.t. single local models. Indeed, as noted by Montero-Manso and Hyndman [2021], given a large enough window of observations and model complexity, if a global model is a universal function approximator it could in principle output predictions identical to those of a set of local models individual to each time series. Training a single global model increases the effective sample size available to the learning procedure and, consequently, allows for exploiting models with a higher model complexity preventing overfitting. Finally, being trained on a set of time series, global models can extrapolate to related but unseen time series, i.e., they can be used in inductive learning scenarios where target time series (i.e., the ones to predict) can be potentially different from those in the training set[3]. Although the advantages of global models are evident, they might struggle to account for the peculiarities of each time series in the collection and might require impractically long observation windows and large memory capacity [Montero-Manso and Hyndman, 2021; Cini et al., 2023c]. Local models, conversely, naturally deal with such peculiarities and local dynamics. As a result, to enjoy the best of

[3]Such setting is relevant in many practical application domains and also known as the *cold-start* scenario [Benidis et al., 2022]; see Chapter 5 for more discussion on the topic.

both worlds, several hybrid models, combining global and local components have been proposed and analyzed (e.g., [Wang et al., 2019; Smyl, 2020; Cini et al., 2023c]) and will be further discussed in Section 2.5 and Chapter 5.

2.3.2 Global and local models for correlated time series

Models in Equation 2.6 and 2.7 discard dependencies among the synchronous time series in the collections. In the case of local models, it is possible to consider the other series in the collection as additional exogenous variables and build multiple-input-single-output (MISO) models such that

$$\hat{\boldsymbol{x}}^i_{t+h} = f^i\left(\boldsymbol{X}_{t-W:t}, \dots; \boldsymbol{\theta}^i\right). \tag{2.8}$$

Extending global models to correlated time series is not trivial. We could consider a multiple-input-multiple-output (MIMO) model simply regarding the input as a single multivariate time series as

$$\widehat{\boldsymbol{X}}_{t+h} = f\left(\boldsymbol{X}_{t-W:t}, \dots; \boldsymbol{\theta}\right). \tag{2.9}$$

However, the resulting predictor is essentially a local model operating on a highly-dimensional multivariate time series; models of this kind would not be able to exploit advantages coming from the global perspective as there would be only one such time series to learn from. Furthermore, predictors belonging to any of these two model families (Equation 2.8 and 2.9) would not be able to deal with new time series being added to the collection and would have to handle the high dimensionality of $\boldsymbol{X}_t$.

Global models for correlated time series Given the above, we are interested in global forecasting models able to process any subset of time series (potentially of variable size) from the collection while keeping parameters shared and taking dependencies among them into account. More formally, given a subset of target time series $\mathcal{S} \subset \mathcal{D}$ and denoted as $\boldsymbol{Y}^{\mathcal{S}}_t \in \mathbb{R}^{N_{\mathcal{S}} \times d_x}$ the resulting stack of observations at each time step t, we are interested in global models such that

$$\widehat{\boldsymbol{Y}}^{\mathcal{S}}_{t+h} = \mathcal{F}\left(\boldsymbol{Y}^{\mathcal{S}}_{t-W:t}, \dots; \boldsymbol{\theta}\right) \qquad \forall \mathcal{S} \in \mathcal{P}\left(\mathcal{D}\right) \tag{2.10}$$

where $\mathcal{P}\left(\mathcal{D}\right)$ denotes the power set of the time series collection $\mathcal{D}$. Models belonging to this family of forecasting architectures need a mechanism to share weights among the individual time series (thus keeping the model global) and, at the same time, exploit cross-correlations among them. Deep set (DS) [Zaheer

et al., 2017] and, in particular, Transformer [Vaswani et al., 2017; Grigsby et al., 2021] architectures offer a possible solution that we will discuss further in Section 2.5. However, they do not incorporate any available prior relational information into the processing, i.e., they do not exploit known existing (spatial) dependencies; a limitation that will motivate the framework introduced in Chapter 3. Section 2.5 delves into actual implementations of $\mathcal{F}(\,\cdot\,;\boldsymbol{\theta})$, focusing on deep learning methods for time series forecasting and surveying the state of the art.

2.4 Performance metrics and model selection

The quality of a forecasting model is primarily assessed in terms of its forecasting accuracy on a test set. As in the standard machine learning pipeline, the test set is made of observations that have been held out from the set used for training the model. In time series forecasting, data splits are usually obtained sequentially, by using observations up to a certain time step for training and the remaining for testing. The best model (among many) is usually selected by (statistically) comparing their performance on the test set [Hyndman and Athanasopoulos, 2018].

Metrics One might consider many performance metrics to evaluate point forecasts [Hyndman and Koehler, 2006; Gneiting, 2011]. Among scale-dependent metrics, the mean absolute error (MAE) is a commonly used performance metric in time series forecasting and is computed as

$$\mathrm{MAE}\left(\widehat{\boldsymbol{X}}_{t:t+T}, \boldsymbol{X}_{t:t+T}\right) \doteq \frac{1}{T\,N}\sum_{\tau=0}^{T-1}\sum_{i=1}^{N}\Big\|\underbrace{\hat{\boldsymbol{x}}^i_{t+\tau} - \boldsymbol{x}^i_{t+\tau}}_{\boldsymbol{r}^i_{t+\tau}}\Big\|_1 \tag{2.11}$$

where $\|\hat{\boldsymbol{x}}^i_t - \boldsymbol{x}^i_t\|_1$ is the 1-norm of the predictions residual $\boldsymbol{r}^i_t = \hat{\boldsymbol{x}}^i_t - \boldsymbol{x}^i_t$. Other common absolute metrics are the mean squared error (MSE) computed by considering the square of the 2-norm of the residuals instead of their 1-norm, and the root mean squared error (RMSE) (the square root of the MSE). Among scaled matrics, the mean absolute percentage error (MAPE) weights the 1-norm of the residuals by the 1-norm of the observed values before averaging them. The mean relative error (MRE) instead scales the sum of absolute errors by the sum of the observations. Details and exact formulas to compute performance metrics are reported in Appendix B.

Testing for residual correlations Besides performance-based evaluation, the fitness of a model can be assessed by checking for correlations among residuals. The underlying principle is that correlated residuals indicate the presence of structural information not captured by the model. Ad-hoc statistical hypothesis tests can be designed to detect such correlations in time series analysis [Durbin and Watson, 1950; Ljung and Box, 1978; Box et al., 1970]. However, accounting for correlations among both spatial and temporal residuals makes the process more challenging due to scalability issues and often requires ad-hoc techniques and priors [Zambon and Alippi, 2022, 2023].

2.5 Related work

There is a wide literature on how to build effective time series forecasting models. Statistical methods constitute powerful techniques in the toolbox of the practitioner [Box et al., 1970; Hamilton, 2020; Hyndman and Athanasopoulos, 2018]. These include, for example, ARIMA models [Box et al., 1970], exponential smoothing methods [Hyndman et al., 2008] and state space models [Durbin and Koopman, 2012]. These models often involve only a few trainable parameters, are usually fitted as local predictors, and can easily incorporate prior structural information (i.e., trends and seasonalities) [Montero-Manso and Hyndman, 2021]. However, being local models, as discussed in Section 2.3, fitting them to large groups of time series can be challenging due to scalability and sample complexity issues [Montero-Manso and Hyndman, 2021; Benidis et al., 2022]. Conversely, global approaches can enjoy a larger sample size that allows for adopting more complex models. In this context, deep learning methods are the natural candidate solution and have shown remarkable performance in practical and relevant forecasting applications [Benidis et al., 2022]. Given these considerations and the settings we are interested in, the following sections will mainly focus on deep learning forecasting methods.

We start by going through the main sequence modeling blocks adopted in deep learning architectures in Section 2.5.1 and continue by surveying a selection of modern neural forecasting methods in Section 2.5.2. The following is intended as an overview of widely popular architectures. Methods specific to each of the challenges addressed in the thesis will be discussed in the corresponding chapters.

2.5.1 Deep learning architectures for sequence modeling

Several deep learning architectures for sequence modeling have been studied in the literature.

Recurrent neural networks RNNs [Elman, 1990; Lin et al., 1996; Hochreiter and Schmidhuber, 1997; Tino et al., 2004; Cho et al., 2014] are and have been among the most widely adopted models for sequence modeling. Standard Elmann RNNs [Elman, 1990], usually trained with backpropagation through time [Werbos, 1990], process sequences by updating a state representation at each step with recurrent connections. Gated RNNs [Hochreiter and Schmidhuber, 1997; Cho et al., 2014] improve upon Elmann RNNs by introducing gates to control and regulate the state update mechanism with the intent of mitigating issues such as vanishing and exploding gradients [Hochreiter and Schmidhuber, 1997; Bengio et al., 1994]. Conversely, nonlinear autoregressive exogenous (NARX) models [Chen et al., 1990; Lin et al., 1996] extend the family of traditional autoregressive models by exploiting multilayer perceptrons (MLPs) with recurrent connections from the output to the input of the network. A radically different approach is instead adopted by reservoir computing architectures [Lukoševičius and Jaeger, 2009], such as echo state networks (ESNs) [Jaeger, 2001]. In ESNs, the weights of the RNN are randomly initialized and never trained. The idea behind this approach is to feed the input signal into a high-dimensional, randomized, and non-linear dynamical system, whose internal state can embed the input dynamics. In particular, such randomized RNNs are usually designed such that the state at each time step is asymptotically independent of the initial conditions[4]. Although RNNs are currently overshadowed by attention-based architectures [Vaswani et al., 2017], modern architectures have shown promising results even on tasks usually dominated by Transformers, such as processing very long sequences [Orvieto et al., 2023; Beck et al., 2024].

Temporal convolutional networks TCNs [LeCun and Bengio, 1998; Oord et al., 2016; Gehring et al., 2017; Bai et al., 2018], exploiting 1-d convolutional filters, are a popular alternative to RNNs. In particular, input sequences are usually padded to ensure that representations at a certain time step depend on past values only [Bai et al., 2018]. The main advantage of TCNs over RNNs is that computations can be easily parallelized on GPUs; furthermore, dilated convolutional kernels allow the receptive field of each filter to grow

[4]This requisite is known as *echo state property* [Yildiz et al., 2012].

exponentially at each layer [Bai et al., 2018]. These properties have made TCNs especially popular for processing signals acquired at high sampling rates, such as raw audio [Oord et al., 2016; Shen et al., 2018]. More recently, fully convolutional architectures have yielded outstanding results in processing long sequences [Fu et al., 2023; Li et al., 2023; Shi et al., 2023] challenging, in their turn, the performance of attention-based alternatives also in language processing tasks [Poli et al., 2023].

Attention-based models Transformers [Vaswani et al., 2017] have become one of the most successful sequence modeling architectures and have been applied in many domains from processing language [Brown et al., 2020] to even images [Dosovitskiy et al., 2021; Khan et al., 2022]. Transformers, based on attention [Bahdanau et al., 2015], update each token's representation by aggregating the representations of all other tokens in the set, weighted by normalized and adaptive attention scores. Since scores are computed directly among each pair of tokens, Transformers can avoid the memory bottlenecks and vanishing gradients of recurrent architectures [Vaswani et al., 2017]. Furthermore, being a feed-forward architecture, computation can easily be parallelized. The major drawback of attention operators is their space and time complexity, which is quadratic in the number of tokens. This has pushed the research community toward investigating more efficient, ideally linearized, attention operators [Tay et al., 2022]. The need for scalable sequence modeling also motivates the efforts to revamp recurrent and convolutional architectures discussed in previous paragraphs.

Neural structured state-space models Deep structured state-space modelss (SSMs) [Gu et al., 2022a] offer an alternative sequence modeling paradigm based on layers associating a continuous-time linear SSM to each feature. Deep SSMs can outperform standard Transforms in modeling long-range dependencies [Gu et al., 2022a] while being more computationally efficient. Thanks to proper reparametrizations, SSMs models can operate both as a TCN, which allows for efficient training, and autoregressively (as an RNN) which enables fast inference. Since their introduction, SSMs architectures have been improved and streamlined [Gupta et al., 2022; Gu et al., 2022b; Zhang et al., 2023] and have become a valid alternative to Transformers in many applications [Gu and Dao, 2023].

These sequence modeling operators, together with MLPs, constitute the backbone of most time series forecasting architectures. Nonetheless, directly

using these basic blocks is often not enough. Processing architectures need to be tailored to forecasting time series incorporating purposely designed operators and inductive biases. The following section discusses a selection of relevant approaches from the literature.

2.5.2 Neural forecasting architectures

Deep learning architectures have a long history of both successes [Benidis et al., 2022] and failures [Zhang et al., 1998] in time series forecasting. As already discussed, one of the main limiting factors has been the number of available samples when learning a model for a single, short, seasonal time series [Hewamalage et al., 2021]: a setting where traditional statistical methods often outperform more complex model architectures. The successes of deep learning in time series forecasting instead mostly come from applying the global approach [Salinas et al., 2020; Bandara et al., 2020; Oreshkin et al., 2020; Benidis et al., 2022]. The archetypal deep learning forecasting architecture consists of a stack of neural processing blocks *encoding* each (potentially preprocessed) time series. Representations can then be fed into a simple readout for outputting a multi-step point forecast [Oreshkin et al., 2020]. Alternatively, they can be processed by a more complex *decoder* that could provide, for example, probabilistic predictions [Salinas et al., 2020], or utilize the network's output to parameterize subsequent processing steps [Smyl, 2020; Rangapuram et al., 2018]. This paradigm has seen many implementations.

Forecasting architectures Among popular global predictors, the DeepAR model [Salinas et al., 2020], consists of a single RNN trained as a univariate predictor on data coming from collections of related time series. The output of the RNN cells is used to parameterize a probability distribution (e.g., Gaussian or negative binomial). Other popular methods, combine standard encoding layers with methods inspired by structured time series processing methods. For example, Rangapuram et al. [2018] uses the output of a stack of RNNs to parameterize a state-space model paired with a Kalman filter [Kalman, 1960]. More flexible probabilistic predictors have been obtained by combining sequence modeling architectures with quantile regression [Wen et al., 2017; Gasthaus et al., 2019] and generative models such as normalizing flows [Rasul et al., 2021b; Papamakarios et al., 2021] and diffusion models [Rasul et al., 2021a; Alcaraz and Strodthoff, 2023]. Across the different architectures, several approaches have obtained notable results by combining global models with local learnable components. For example, Smyl [2020] won the M4 forecasting

competition [Makridakis et al., 2020] by combining a global RNN with local exponential smoothing models fitted on each time series. Wang et al. [2019] introduced an architecture combining global RNNs with local models accounting for local random effects. Regarding the implementation of the encoding blocks, RNNs have historically been among the most widely adopted models [Mandic and Chambers, 2001; Wen et al., 2017; Salinas et al., 2020; Hewamalage et al., 2021]. Streamlined feed-forward architectures consisting of deep stacks of residual MLPs have also obtained remarkable results [Oreshkin et al., 2020; Challu et al., 2023]. TCNs, as mentioned in the previous section, constitute the core processing layers of many state-of-the-art forecasting architectures [Borovykh et al., 2017; Chen et al., 2020; Gasparin et al., 2022; Wu et al., 2023]. Transformers, and attention-based architectures in general, have become popular for modeling temporal dependendencies in time series [Li et al., 2019; Lim et al., 2021; Zhou et al., 2021; Nie et al., 2023], especially when large amounts of data are available [Kunz et al., 2023]. However, some skepticism on the effectiveness of fully attention-based forecasting architectures remains [Zeng et al., 2023]. Deep SSMs have also started being applied to forecasting [Alcaraz and Strodthoff, 2023] with the introduction of ad-hoc modifications for modeling autoregressive processes Zhang et al. [2023]. Finally, another trend worth discussing is that of foundation models for time series [Liang et al., 2024], i.e., models trained on large amounts of data engineered for being applied to forecast new time series zero-shot or by fine-tuning (part of) the model. These models have been shown capable of achieving zero-shot performance comparable to that of state-of-the-art general-purpose forecasting architectures fitted to the target time series [Garza and Mergenthaler-Canseco, 2023; Ansari et al., 2024; Das et al., 2024].

Models for correlated time series As discussed in Section 2.3, most state-of-the-art architectures consist of global models that forecast each time series individually, i.e., discarding spatial dependencies that might exist among input time series. Considering the input as a single multivariate time series results in local models with poor scalability and high sample complexity [Sen et al., 2019; Cini et al., 2023c]. Among methods tackling the problem, the DeepGLO architecture [Sen et al., 2019] extracts a set of latent covariates from a time series collection by temporally regularized matrix factorization [Yu et al., 2016] and uses them as additional input to a univariate TCN processing each time series separately. Another popular approach is that of relying on Transformers where attention is applied along both the tempora and spatial dimensions [Grigsby et al., 2021; Ma et al., 2019; Liu et al., 2023a]. However, *spatiotemporal*

Transformers are limited due to quadratic computational complexity potentially w.r.t. both the number of time series and length of the input window. If input time series can be arranged in a grid, multi-dimensional TCNs could exploit the structure of the spatial dependencies to make the processing more scalable [Shi et al., 2015; Tran et al., 2015]. However, this is rarely the case for data coming from, e.g., sensor networks where the spatial coverage is sparse and sensors are usually irregularly positioned. As we discuss in the following chapters, an effective approach to tackling the problem is offered by graph representations and graph neural networks [Seo et al., 2018; Cini et al., 2023b; Jin et al., 2023b].

Chapter 3

Graph deep learning for time series forecasting

This chapter introduces the proposed graph-based forecasting framework. Our methodology is aimed at overcoming the limitations of the forecasting approaches discussed in Chapter 2 by explicitly modeling relational dependencies among time series. In particular, we introduce graph-based representations of collections of correlated time series (Section 3.1) and the associated predictors (Section 3.2). Section 3.3 introduces STGNNs as a forecasting architecture implementing the framework and discusses a taxonomy of existing models. Further discussion on the state of the art is then provided in Section 3.4.

Reference papers The content of the chapter is partly based on material from the following papers.

- Andrea Cini, Ivan Marisca, Daniele Zambon, and Cesare Alippi. Graph Deep Learning for Time Series Forecasting. *arXiv preprint arXiv:2310.15978*, 2023b
- Andrea Cini, Ivan Marisca, Daniele Zambon, and Cesare Alippi. Taming Local Effects in Graph-based Spatiotemporal Forecasting. *Advances in Neural Information Processing Systems*, 2023c
- Andrea Cini, Daniele Zambon, and Cesare Alippi. Sparse graph learning from spatiotemporal time series. *Journal of Machine Learning Research*, 24(242):1–36, 2023d

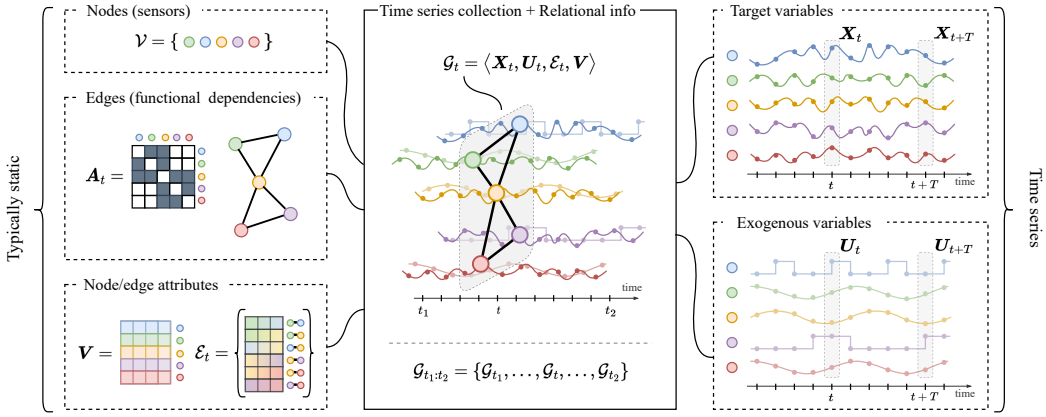


Figure 3.1. Graph-based representation of a time series collection.

3.1 Graph-based representations for collections of time series

We start by considering a collection of N correlated time series as described in Section 2.1. Relational dependencies among the time series can be modeled to inform the downstream processing and allow for, e.g., getting rid of spurious correlations in the observed sequences of data.

Graph-based representation The existence of pairwise relationships among time series can be accounted for by a (possibly dynamic) adjacency matrix $\boldsymbol{A}_t \in \{0,1\}^{N \times N}$ that encodes the (possibly asymmetric) dependencies at each time step t. Optional edge attributes $\boldsymbol{e}_t^{ij} \in \mathbb{R}^{d_e}$ can be associated to each non-zero entry of $\boldsymbol{A}_t$. In particular, we denote the set of attributed edges encoding all the available relational information by $\mathcal{E}_t \doteq \{\langle (i,j), \boldsymbol{e}_t^{ij} \rangle \mid \forall i,j : \boldsymbol{A}_t[i,j] \neq 0\}$. Whenever edge attributes are scalar, i.e., $d_e = 1$, edge set $\mathcal{E}_t$ can be simply represented as a weighted real-valued adjacency matrix $\boldsymbol{A}_t \in \mathbb{R}^{N \times N}$. Analogously to the homogeneity assumption for observations (Chapter 2), edges are assumed to indicate the same type of relational dependencies (e.g., physical proximity) and have the same type of attributes (either categorical or continuous) across the collection. Note that for many applications (e.g., traffic networks) changes in the topology happen slowly over time and the adjacency matrix – as well as edge attributes – can be considered as fixed within a short window of observations, i.e., $\mathcal{E}_t = \mathcal{E}$ and $\boldsymbol{e}_t^{ij} = \boldsymbol{e}^{ij}$ for all the (i,j) pairs. As discussed in Section 3.1.1, this representation is particularly flexible and can easily be tailored to several problem settings and practical applications. In particular, dependencies encoded

by $\boldsymbol{A}_t$ and $\mathcal{E}_t$ can be of different nature, from simple physical proximity and correlation to more complex relationships, e.g., the similarity of products in the retail market [Gandhi et al., 2021].

Example 3. Consider the sensor network monitoring the speed of vehicles introduced in Example 1. A static adjacency matrix $\boldsymbol{A}$ can be obtained by considering each pair of sensors connected by an edge – weighted by the road distance – if and only if a road segment directly connects them. Edge weights can be set as inversely proportional to the road distance, e.g., by considering radial basis functions [Shuman et al., 2013]. Conversely, road closures and traffic diversions can be accounted for by adopting a dynamic topology $\boldsymbol{A}_t$.

Example 4. Consider the smart metering infrastructure introduced in Example 2. A (weighted) adjacency matrix $\boldsymbol{A}$ can be obtained, for example, by modeling correlations in energy consumption patterns. In particular, $\boldsymbol{A}$ could be obtained from a thresholded similarity matrix of scores based, e.g., on Pearson correlation, correntropy [Liu et al., 2007], or dynamic time warping [Berndt and Clifford, 1994].

Terminology and notation We use interchangeably the terms *node* and *sensor* to indicate each of the N entities generating the time series and refer to the node set together with the relational information as *sensor network*. The tuple $\mathcal{G}_t \doteq \langle \boldsymbol{X}_t, \boldsymbol{U}_t, \mathcal{E}_t, \boldsymbol{V} \rangle$ indicates all the available information at time step t. A graphical representation of the problem settings is shown in Figure 3.1.

3.1.1 Extensions to the reference settings

This section offers extensions to the reference problem settings by discussing how the graph-based representation can be modified to account for peculiarities typical of a wide range of practical applications. Further assumptions will be challenged on Chapters 5 to 8.

New nodes, missing observations, and multiple collections It is often the case that the time frames of the time series in the collection, although synchronous and regularly sampled, do not overlap perfectly, i.e., some time series might become available at a later time and there might be windows with blocks of missing observations. For example, it is typical for the number of installed sensors to grow over time and many applications are affected by the presence of missing data, e.g., associated with readout and/or communication failures which result in transient or permanent faults. These scenarios can be

incorporated into the framework by setting N to the total maximum number of time series available, and, whenever needed, padding the time series appropriately to allow for a tabular representation of $\{\boldsymbol{X}_t\}_t$. An auxiliary binary exogenous variable $\boldsymbol{M}_t \in \{0,1\}^{N \times d_x}$, called *mask*, can be introduced at each time step as $\mathcal{G}_t \doteq \langle \boldsymbol{X}_t, \boldsymbol{U}_t, \boldsymbol{M}_t, \mathcal{E}_t, \boldsymbol{V} \rangle$ to model the availability of observations w.r.t. each node and time step. In particular, we set $\boldsymbol{m}_t^i[k] = 1$ if k-th channel in the corresponding observation $\boldsymbol{x}_t^i$ is valid, and $\boldsymbol{m}_t^i[k] = 0$ otherwise. If observations associated with the i-th node are completely missing at time step t, the associated mask vector will be null, i.e., $\boldsymbol{m}_t^i = \boldsymbol{0}$. The masked representation simplifies the presentation of concepts and, at the same time, is useful in data reconstruction tasks (see Chapter 6). Finally, if collections from multiple sensor networks are available, the problem can be formalized as learning from M disjoint sets of correlated time series $\mathcal{D} = \{\mathcal{G}^{(1)}_{t_1:t_1+T_1}, \mathcal{G}^{(2)}_{t_2:t_2+T_2}, \dots, \mathcal{G}^{(M)}_{t_m:t_m+T_m}\}$, potentially without overlapping time frames. In the latter case, we assume the absence of functional dependencies between time series in different sets and the homogeneity of node features and edge attributes across collections.

Heterogeneous time series and edge attributes Heterogeneous sets of correlated time series are commonly found in the real world (e.g., consider a set of weather stations equipped with different sensory packages) and result in collections where observations across time series in the set might correspond to different variables. Luckily, dealing with this setting is relatively straightforward and can be done in several ways. In particular, the masked representation introduced in the above paragraph can be used to pad each time series to the same dimension d_{max} and keep track of the available channels at each node; moreover, the sensor type of each sensor can be encoded in the attribute matrix $\boldsymbol{V}$. If the total number of variables is too large or is expected to change over time, one alternative strategy is to map each observation into a shared homogenous representation (see, e.g., relational models such as [Schlichtkrull et al., 2018]). Heterogeneous edge attributes can be dealt with analogously to heterogeneous node features.

3.2 Forecasting with relational side information

As discussed in Chapter 2, learning forecasting models for correlated time series as those in Equation 2.10 is challenging, and the main challenge is dealing with all the input time series at once. Notably, accounting for possible dependencies becomes increasingly difficult as the number of time series in the collection

grows. Intuitively, the high dimensionality of the problem can lead to spurious correlations among the observed time series impairing the effectiveness of the learning procedure. To address this issue, our proposal is that of embedding the available relational information as an inductive bias into the model. In particular, dependencies among the time series can be used to condition the prediction and, as discussed in Section 3.3, accounted for in the predictor through an architectural bias. Forecasting models, then, be conditioned on both past observations and the predefined relational structure. The considered family of models approximates the data-generating process as

$$p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t+h}^{i} \middle| \mathcal{G}_{t-W:t}, \boldsymbol{U}_{t:t+h+1}\right) \approx p^{i}\left(\boldsymbol{x}_{t+h}^{i} \middle| \mathcal{X}_{<t}, \boldsymbol{U}_{t:t+h+1}\right). \tag{3.1}$$

Notably, the conditioning on the sequence of attributed graphs $\mathcal{G}_{t-W:t}$ and, in particular, on the relationships encoded in $\mathcal{E}_{t-W:t}$, can localize predictions w.r.t. the neighborhood of each node and is intended to constrain the model to the most plausible ones. Analogously, the corresponding graph-based point predictors operate as

$$\widehat{\boldsymbol{X}}_{t:t+H} = \mathcal{F}\left(\mathcal{G}_{t-W:t}, \boldsymbol{U}_{t:t+h+1}; \boldsymbol{\theta}\right) \tag{3.2}$$

Graph-based predictors belonging to this family of models can implement global models of the type discussed in Section 2.3.2 (Equation 2.10). The following sections focus on the design of these *global* graph-based predictors and methods to embed relational inductive biases [Battaglia et al., 2018] into the processing architecture. In particular, Section 3.3 introduces a framework for designing STGNNs able to forecast any subset of time series from the collection by conditioning the predictions on the associated relational structure.

3.3 Spatiotemporal graph neural networks

This section introduces STGNNs [Seo et al., 2018; Li et al., 2018; Yu et al., 2018] and the MPs operators that constitute the core processing blocks of graph-based time series processing architectures (Section 3.3.1–3.3.3). Within this context, we also discuss a taxonomy of the design choices available to the practitioner when selecting a model architecture (Section 3.3.4). STGNNs are global forecasting models where parameters are shared among the target time series; the discussion on this fundamental aspect will take place in Section 3.3.5, while hybrid global-local STGNNs will be to focus of Chapter 5. Finally, Section 3.4 will then discuss graph deep learning methods for processing temporal data in a broader context. We refer to Jin et al. [2023b] for an in-depth survey on the existing GNN architectures across different tasks in time series processing.

3.3.1 Message-passing neural networks

Modern GNNs [Scarselli et al., 2008; Bacciu et al., 2020; Bronstein et al., 2021] embed architectural biases into the processing architecture by constraining the propagation of information w.r.t. a notion of neighborhood derived from the adjacency matrix. Most of the commonly used architectures fit into the MP framework [Gilmer et al., 2017], which provides a recipe for designing GNN layers; GNNs that fit within the MP framework are usually referred to as *spatial GNNs* [Sperduti and Starita, 1997; Scarselli et al., 2008], usually in opposition to *spectral GNNs*, which instead operate in the spectral domain[1] [Bruna et al., 2014; Wang and Zhang, 2022]. By taking as reference a graph with *static* node features $\boldsymbol{H}^0 \in \mathbb{R}^{N \times d_h}$ and edge set $\mathcal{E}$, we consider MP neural networks obtained by stacking MP layers that update each i-th node representation at each l-th layer as

$$\boldsymbol{h}^{i,l+1} = \textsc{Up}^l\Big(\boldsymbol{h}^{i,l}, \underset{j \in \mathcal{N}(i)}{\textsc{Aggr}}\Big\{\textsc{Msg}^l\big(\boldsymbol{h}^{i,l}, \boldsymbol{h}^{j,l}, \boldsymbol{e}^{ji}\big)\Big\}\Big), \tag{3.3}$$

where $\textsc{Up}^l(\,\cdot\,)$ and $\textsc{Msg}^l(\,\cdot\,)$ are respectively the update and message functions, e.g., implemented by MLPs. $\textsc{Aggr}\{\,\cdot\,\}$ indicates a generic permutation invariant aggregation function, while $\mathcal{N}(i)$ refers to the set of neighbors of node i, each associated with edge attribute $\boldsymbol{e}^{ji}$. In the following, we use the shorthand $\boldsymbol{H}^{l+1} = \mathrm{MP}^l\left(\boldsymbol{H}^l, \mathcal{E}\right)$ to indicate a MP step w.r.t. the full node set. MP GNNs are *inductive* models [Ruiz et al., 2020] which can process unseen graphs of variable sizes by sharing weights among nodes and localizing representations by aggregating features at neighboring nodes.

Isotropic and anisotropic message passing By following Dwivedi et al. [2023], we call *isotropic* those GNNs where the message function $\textsc{Msg}^l$ only depends on the features of the sender node $\boldsymbol{h}^{j,l}$; conversely, we use the term *anistropic* referring to GNNs where $\textsc{Msg}^l$ takes both $\boldsymbol{h}^{i,l}$ and $\boldsymbol{h}^{j,l}$ as input. For instance, a standard and commonly used isotropic MP layer for weighted graphs (with weighted adjacency matrix $\boldsymbol{A}$) is

$$\boldsymbol{h}^{i,l+1} = \xi\Big(\boldsymbol{W}_1^l \boldsymbol{h}^{i,l} + \underset{j \in \mathcal{N}(i)}{\textsc{Sum}}\Big\{a^{ji}\boldsymbol{W}_2^l \boldsymbol{h}^{j,l}\Big\}\Big), \tag{3.4}$$

where $\boldsymbol{W}_1^l$ and $\boldsymbol{W}_2^l$ are matrices of learnable parameters, $a^{ji} = \boldsymbol{A}[j,i]$, and $\xi(\,\cdot\,)$ is a generic activation function. Conversely, an example of an anisotropic MP

[1] Note that most of the so-called spectral GNNs can be seen as special instances of MP architectures nonetheless.

operator, based on [Bresson and Laurent, 2017], is

$$\boldsymbol{m}^{j\to i,l} = \boldsymbol{W}_2^l \xi\Big(\boldsymbol{W}_1^l\Big[\boldsymbol{h}^{i,l}||\boldsymbol{h}^{j,l}||\boldsymbol{e}^{ji}\Big]\Big), \qquad \alpha^{ji,l} = \sigma\Big(\boldsymbol{W}_0^l \boldsymbol{m}^{ji,l}\Big), \tag{3.5}$$

$$\boldsymbol{h}^{i,l+1} = \xi\Big(\boldsymbol{W}_3^l \boldsymbol{h}^{i,l} + \underset{j\in\mathcal{N}(i)}{\textsc{Sum}}\Big\{\alpha^{ji,l}\boldsymbol{m}^{j\to i,l}\Big\}\Big), \tag{3.6}$$

where matrices $\boldsymbol{W}_0^l \in \mathbb{R}^{1\times d_m}$, $\boldsymbol{W}_1^l$, $\boldsymbol{W}_2^l$ and $\boldsymbol{W}_3^l$ are learnable parameters, $\sigma(\,\cdot\,)$ is the sigmoid activation function and $||$ the concatenation operator applied along the feature dimension. Intuitively, isotropic MP operators compute and aggregate messages without taking into account the representations of sender and receiver nodes and rely entirely on the presence of edge weights to weigh the contribution of different neighbors. Conversely, anisotropic schemes allow for learning adaptive aggregation and message-passing schemes aware of the nodes involved in the computation. Popular anisotropic operators exploit multi-head attention mechanisms to learn rich propagations schemes where the information flowing from each neighbor is weighted and aggregated after multiple parallel transformations [Veličković et al., 2018; Vaswani et al., 2017]. We point out that selecting the proper MP operator, i.e., choosing the architectural bias for constraining the flow of information, is crucial for obtaining good performance for the problem at hand. Notably, many standard isotropic filters often assume homophily – i.e., that neighboring nodes behave similarly – and can suffer from over-smoothing [Rusch et al., 2023].

3.3.2 Spatiotemporal message passing

STGNNs can be designed by extending MP to aggregate, at each time step, spatiotemporal information from each node's neighborhood. In particular, we model the operations as spatiotemporal message-passing (STMP) blocks updating representations as

$$\boldsymbol{h}_t^{i,l+1} = \textsc{Up}^l\Big(\boldsymbol{h}_{\leq t}^{i,l}, \underset{j\in\mathcal{N}_t(i)}{\textsc{Aggr}}\Big\{\textsc{Msg}^l\big(\boldsymbol{h}_{\leq t}^{i,l}, \boldsymbol{h}_{\leq t}^{j,l}, \boldsymbol{e}_{\leq t}^{ji}\big)\Big\}\Big), \tag{3.7}$$

where $\mathcal{N}_t(i)$ indicates the neighbors of the i-th node at time step t (i.e., the nodes associated with incoming edges in $\mathcal{E}_t$). As in the previous case, in the following, the shorthand $\boldsymbol{H}_t^{l+1} = \text{STMP}^l\big(\boldsymbol{H}_{\leq t}^l, \mathcal{E}_{\leq t}\big)$ indicates an STMP step. As implicit from Equation 3.7, differently from standard MP, blocks of an STMP layer will have to integrate sequence modeling operators. The next section provides recipes for building STGNNs based on different implementations of the STMP blocks and on existing popular STGNN architectures.

3.3.3 A template architecture

We consider forecasting architectures consisting of an encoding step followed by STMP layers and a final readout mapping representations to predictions. In particular, we consider models, belonging to the family introduced in Equation 3.2, that consists of a sequence of three blocks:

$$\boldsymbol{h}_{t-1}^{i,0} = \textsc{Encoder}\left(\boldsymbol{x}_{t-1}^{i}, \boldsymbol{u}_{t-1}^{i}, \boldsymbol{v}^{i}\right), \tag{3.8}$$

$$\boldsymbol{H}_{t-1}^{l+1} = \mathrm{STMP}^{l}\Big(\boldsymbol{H}_{\leq t-1}^{l}, \mathcal{E}_{\leq t-1}\Big), \quad l = 0, \dots, L-1 \tag{3.9}$$

$$\hat{\boldsymbol{x}}_{t:t+H}^{i} = \textsc{Decoder}\left(\boldsymbol{h}_{t-1}^{i,L}, \boldsymbol{u}_{t:t+H}^{i}\right). \tag{3.10}$$

$\textsc{Encoder}(\,\cdot\,)$ and $\textsc{Decoder}(\,\cdot\,)$ indicate generic encoding and readout layers that can be implemented, as an example, as standard fully connected linear layers, or MLPs. Note that both encoder and decoder do not perform any propagation of information along time and space, a task which is delegated to the stack of STMP layers. For each paradigm, we discuss associated relevant architectures.

3.3.4 Taxonomy

We categorize STGNNs following the template of Section 3.3.3 in time-then-space (TTS), space-then-time (STT), and time-and-space (T&S) models. More specifically, in a TTS model each series of representations $\boldsymbol{h}_{<t}^{i,0}$ is processed by a sequence modeling operator, such as an RNN, before any MP operation along the spatial dimension [Gao and Ribeiro, 2022]; STT models are similarly obtained by inverting the order of the two operations. Conversely, in T&S models time and space are processed in a more integrated fashion, e.g., by a recurrent GNN [Seo et al., 2018] or by spatiotemporal convolutional operators [Yu et al., 2018].

Time-and-space models We include in this category any STGNN in which the processing of the temporal and spatial dimensions cannot be factorized in two separate steps. In T&S models, representations at every node and time step are the result of joint temporal and spatial processing, as in Equation 3.9. To the best of our knowledge, the first T&S STGNNs have been proposed by Seo et al. [2018], who introduced a popular family of recurrent architectures, hereby denoted as graph convolutional recurrent neural networks (GCRNNs), where standard fully-connected layers in (gated) RNNs are replaced by graph convolutions [Kipf and Welling, 2017; Defferrard et al., 2016]. As an example, by considering a gated recurrent unit (GRU) cell [Cho et al., 2014] and replacing

graph convolutions with generic MP layers, the resulting recurrent model updates representations at each time step t as

$$\boldsymbol{Z}_t^l = \boldsymbol{H}_t^{l-1} \tag{3.11}$$
$$\boldsymbol{R}_t^l = \sigma\left(\mathrm{MP}_r^l\left(\left[\boldsymbol{Z}_t^l || \boldsymbol{H}_{t-1}^l\right], \mathcal{E}_t\right)\right), \tag{3.12}$$
$$\boldsymbol{O}_t^l = \sigma\left(\mathrm{MP}_o^l\left(\left[\boldsymbol{Z}_t^l || \boldsymbol{H}_{t-1}^l\right], \mathcal{E}_t\right)\right), \tag{3.13}$$
$$\boldsymbol{C}_t^l = \tanh\left(\mathrm{MP}_c^l\left(\left[\boldsymbol{Z}_t^l || \boldsymbol{R}_t^l \odot \boldsymbol{H}_{t-1}^l\right], \mathcal{E}_t\right)\right), \tag{3.14}$$
$$\boldsymbol{H}_t^l = \boldsymbol{O}_t^l \odot \boldsymbol{H}_{t-1}^l + (1 - \boldsymbol{O}_t^l) \odot \boldsymbol{C}_t^l, \tag{3.15}$$

with $\odot$ denoting the element-wise (Hadamard) product and $||$ the concatenation operation. Note that we consider, for each gate, a single MP operation at each l-th layer for conciseness' sake (a stack of MP layers is often adopted in practice). Models following a similar approach have found widespread adoption replacing standard RNNs in the context of correlated time series processing [Li et al., 2018; Zhang et al., 2018; Bai et al., 2020; Cini et al., 2022b]. In particular, the DCRNN architecture Li et al. [2018], implementing the MP operators in Equation 3.12–3.15 with a diffusion convolution operator [Atwood and Towsley, 2016], has been among the first STGNN applied to time series forecasting. Apart from GCRNNs, an approach to building T&S models consists of integrating a temporal operator directly into the $\textsc{Msg}(\cdot)$ function. Among the others, Wu et al. [2022] and Marisca et al. [2022] use cross-node attention as a mechanism to propagate information among sequences of observations at neighboring nodes. As an additional example, an analogous model could be obtained by implementing the $\textsc{Up}(\cdot)$ and $\textsc{Msg}(\cdot)$ functions of the STMP layer in Equation 3.7 as TCNs:

$$\boldsymbol{h}_{t-W:t}^{i,l} = \mathrm{TCN}_1^l\left(\boldsymbol{h}_{t-W:t}^{i,l-1}, \underset{j\in\mathcal{N}_t(i)}{\textsc{Aggr}}\left\{\mathrm{TCN}_2^l\left(\boldsymbol{h}_{t-W:t}^{i,l-1}, \boldsymbol{h}_{t-W:t}^{j,l-1}, \boldsymbol{e}_{t-W:t}^{ji}\right)\right\}\right). \tag{3.16}$$

Note that the operator resulting from the MP processing defined in Equation 3.16 can be seen as operating on the product graph obtained from spatial and temporal relationships [Sabbaqi and Isufi, 2022]. Finally, a straightforward approach to build T&S architectures is that of stacking blocks of alternating spatial and temporal operators [Yu et al., 2018; Wu et al., 2019, 2020], e.g.,

$$\boldsymbol{z}_{t-W:t}^{i,l} = \mathrm{TCN}^l\left(\boldsymbol{h}_{t-W:t}^{i,l-1}\right) \quad \forall i, \qquad \boldsymbol{H}_t^l = \mathrm{MP}^l\left(\boldsymbol{Z}_t^l, \mathcal{E}_t\right) \quad \forall\, t, \tag{3.17}$$

where $\mathrm{TCN}^l(\cdot)$ indicates a temporal convolutional network layer. The first example of a similar architecture was introduced by Yu et al. [2018]. Among follow-up works, Wu et al. [2019] introduced the GraphWaveNet architecture, which exploits the same diffusion convolutions as Li et al. [2018] and residual

dilated TCNs [Oord et al., 2016]. One of the major drawbacks of T&S models is their time and space complexity which usually scale with the number of nodes and edges in the graph times the number of input time steps, i.e., with $\mathcal{O}\big(W\big(N + L|\mathcal{E}_{\max}|\big)\big)$, where $N \ll |\mathcal{E}_{\max}| = \max\{|\mathcal{E}_{t-k}|\}_{k=1}^{W}$ (see Chapter 8).

Time-then-space models The general recipe for a TTS model consists in 1) encoding time series associated with each node into a vector representation, obtaining an attributed graph, and 2) propagating the obtained representations throughout the graph with a stack of standard MP layers, i.e.,

$$\boldsymbol{h}_t^{i,1} = \textsc{SeqEnc}\left(\boldsymbol{h}_{\leq t}^{i,0}\right), \tag{3.18}$$

$$\boldsymbol{H}_t^{l+1} = \mathrm{MP}^l\left(\boldsymbol{H}_t^l, \mathcal{E}_t\right), \qquad \forall\, l = 1, \dots, L-1. \tag{3.19}$$

The sequence encoder $\textsc{SeqEnc}(\,\cdot\,)$ can be implemented by any modern deep learning architecture for sequence modeling (e.g, an RNNs, a TCNs or an attention-based operator). Note that this temporal encoder can consist of multiple layers, i.e., it can be a deep network by itself. Since MP is performed only w.r.t. representations corresponding to the last time step, in case of a dynamic topology the edge set used for propagation can be obtained as a function of $\mathcal{E}_{t-W:t}$ rather than simply using $\mathcal{E}_t$, i.e., $\widetilde{\mathcal{E}}_t = \textsc{Aggr}\{\mathcal{E}_{t-W:t}\}$. A possible choice would be to take the union of all the edge sets, which, however, requires further processing in the case of attributed edges [Gao and Ribeiro, 2022]. TTS models are relatively uncommon in the literature [Gao and Ribeiro, 2022; Satorras et al., 2022; Cini et al., 2023a, 2024] but are becoming more popular due to their efficiency and scalability compared to T&S alternatives [Gao and Ribeiro, 2022]. Differently from generic T&S models, in fact, the number of MP operations does not depend on the size of the window W. Indeed, TTS models have a time and space complexity that scales as $\mathcal{O}\big(NW + L|\mathcal{E}_t|\big)$, rather than $\mathcal{O}\big(W\big(N + L|\mathcal{E}_{\max}|\big)\big)$ of T&S models. However, the two-step encoding might introduce bottlenecks in the propagation of information. These considerations will be further expanded upon in Chapter 8, which will be dedicated to addressing scalability challenges.

Space-then-time models STT models can be built by simply inverting the order of Equation 3.18 and 3.19, i.e., by using MP layers to process static representations at each time step, then encoded along the temporal axis by a sequence model, i.e.,

$$\boldsymbol{H}_t^{i,l} = \mathrm{MP}^l\left(\boldsymbol{H}_t^{i,l-1}, \mathcal{E}_t\right), \qquad \forall\, l = 1, \dots, L-1 \tag{3.20}$$

$$\boldsymbol{h}_t^{i,L} = \textsc{SeqEnc}\left(\boldsymbol{h}_{t-W:t}^{i,L-1}\right). \tag{3.21}$$

The general idea behind STT approaches is to first enrich node observations by accounting for observations at neighboring nodes, and then process obtained sequences with a standard sequence model. Although they have seen some applications [Seo et al., 2018; Pareja et al., 2020; Zhao et al., 2019], STT models do not offer the same computational benefits of TTS models, having the same $\mathcal{O}\big(W\big(N+L|\mathcal{E}_{\max}|\big)\big)$ complexity of T&S models. Nonetheless, as in T&S models, dynamic edge sets $\mathcal{E}_{t-W:t}$ can be accounted for by performing MP operations w.r.t. the corresponding edges at each time step. Analogously to TTS models, the factorization of the processing in two steps might introduce bottlenecks.

3.3.5 Globality and locality in STGNNs

STGNNs introduced in Section 3.3 *are global models* that exploit relational architectural biases to account for related time series, addressing the limitations of the standard global approach. Indeed, by considering the STMP scheme of Equation 3.7, it is straightforward to see that STMP layers share the parameters used to process the time series in the collection. STMP layers condition the extracted representations on each node's neighborhood, thus accounting for spatial dependencies that would have been ignored by standard (univariate) global models. STGNNs are inductive and transferable as they do not rely upon node-specific parameters; such properties make them distinctively different from local multivariate approaches in Equation 2.8. Global models of the type implemented by STGNNs are akin to those formalized in Equation 2.10, i.e.,

$$\widehat{\boldsymbol{Y}}^{\mathcal{S}}_{t+h} = \mathcal{F}\left(\mathcal{G}^{[\mathcal{S}]}_{t-W:t}, \dots; \boldsymbol{\theta}\right) \qquad \forall \mathcal{S} \in \mathcal{P}\left(\mathcal{D}\right), \tag{3.22}$$

where $\mathcal{G}^{[\mathcal{S}]}$ indicates the sub-graph induced by the subset of nodes gS. The interplay between global and local aspects plays a major role in the context of graph-based forecasting models. Indeed, although the drawbacks of the local approach are evident, global STGNNs might struggle to effectively account for the peculiarities of each time series thus motivating the study of hybrid models which will be the focus of Chapter 5.

3.4 Related work

While we have already discussed architectures usually adopted in the literature, this section briefly overviews other methods to process dynamic relational data and the attempts to formalize such frameworks.

3.4.1 Graph deep learning for temporal data

GDL methods have found widespread application in the processing of dynamic relational data beyond time series processing [Kazemi et al., 2020; Longa et al., 2023; Gravina and Bacciu, 2024]. In particular, the term *temporal graph* (or temporal network) is used to indicate scenarios where nodes, attributes, and edges of a graph are dynamic and are given over time as a sequence of events localized at specific nodes [Kazemi et al., 2020; Rossi et al., 2020; Longa et al., 2023; Gravina and Bacciu, 2024]. A typical reference application is the processing of the dynamic relationships and user profiles that characterize social networks and recommender systems. Kazemi et al. [2020] propose an encoder-decoder framework to unify existing representation learning methods for dynamic graphs. Barros et al. [2021] compiled a rich survey of methods for embedding dynamic networks, while Skarding et al. [2021] focus on GNN approaches to the same problem. Longa et al. [2023] and Gravina and Bacciu [2024] introduce a taxonomy of tasks and models in temporal graph processing, with Gravina and Bacciu [2024] introducing at the same time a benchmark based on a diverse set of available datasets. Huang et al. [2023] build an alternative set of benchmarks and datasets with a focus on applications to large-scale temporal graphs. Besides temporal graphs, a large body of literature has been dedicated to the processing of sequences of arbitrary graphs, e.g., without assuming any correspondence between nodes across time steps [Zambon, 2022; Zambon et al., 2018; Paassen et al., 2020]. Although the problem addressed in this research could formally be seen as a sub-case of temporal graph processing, having actual time series associated with each node radically changes the available tools and methods, as well as the available model designs and target applications.

Graph deep learning for time series GNNs for time series processing have been pioneered in traffic forecasting [Li et al., 2018; Yu et al., 2018] and their application in such context has been extremely successful thereafter [Ye et al., 2020; Jiang and Luo, 2022; Jin et al., 2023a]. Current application domains include among the other air quality monitoring [Chen et al., 2021b; Iskandaryan et al., 2023], energy analytics [Eandi et al., 2022; Cini et al., 2023a], financial time series processing [Chen et al., 2018b; Matsunaga et al., 2019], and epidemiological data analysis [Kapoor et al., 2020; Fritz et al., 2022]. As already discussed, several sequence modeling architectures integrating MP into the processing have developed, from recurrent GNNs [Seo et al., 2018; Li et al., 2018; Micheli and Tortorella, 2022], convolutional models [Yu et al., 2018; Wu et al., 2019] and attention-based architectures [Zheng et al., 2020; Wu et al.,

2022; Marisca et al., 2022]. Recently, MP have been integrated into deep SSMs as well [Tang et al., 2023]. Outside of deep learning, graph-based methods for time series processing have been studied in the context of graph signal processing [Ortega et al., 2018; Stanković et al., 2020; Leus et al., 2023] and go under the name of *time-vertex signal processing* methods [Grassi et al., 2017]. Related methodologies addressing specific challenges will be further discussed in the dedicated chapters, we refer to Jin et al. [2023b] for a detailed and thorough survey of existing architectures.

Chapter 4

Benchmarks and baselines

Before analyzing challenges and starting addressing specific challenges, this chapter complements the discussion carried out so far by introducing benchmarks (Section 4.1) and baselines (Section 4.2) for the introduced forecasting architectures. Furthermore, we provide numerical simulations showing the impact of the transitioning from standard global and local deep learning predictors to graph-based architectures when forecasting collections of correlated time series (Section 4.3). This chapter serves as starting ad reference point for the empirical results discussed and presented throughout the thesis.

Reference papers The content of the chapter is partly based on material from the following papers.

- Andrea Cini, Ivan Marisca, Daniele Zambon, and Cesare Alippi. Graph Deep Learning for Time Series Forecasting. *arXiv preprint arXiv:2310.15978*, 2023b

Table 4.1. Statistics of datasets used in the experiments.

Datasets	Type	Time steps	Nodes	Edges	Rate
METR-LA	Directed	34,272	207	1515	5 minutes
PEMS-BAY	Directed	52,128	325	2369	5 minutes
CER-E	Directed	25,728	485	4365	30 minutes
AQI	Undirected	8,760	437	2699	1 hour
GPVAR-G	Undirected	30,000	120	199	N/A

4.1 Benchmarks

This section introduces the reference synthetic datasets and real-world benchmarks.

4.1.1 Data from real sensor networks

We consider datasets coming from traffic forecasting, energy analytics, and air quality monitoring. In particular, we use the following benchmarks.

METR-LA & PEMS-BAY METR-LA and PEMS-BAY, introduced by Li et al. [2018], are two popular traffic forecasting datasets consisting of measurements from loop detectors in the Los Angeles County Highway [Jagadish et al., 2014] and San Francisco Bay Area [Chen et al., 2001].

CER-E The CER-E dataset [Commission for Energy Regulation, 2016] consists of energy consumption readings, aggregated into 30-minutes intervals, from 485 smart meters monitoring small and medium-sized enterprises.

AQI The AQI [Zheng et al., 2015] dataset collects hourly measurements of pollutant PM2.5 from 437 air quality monitoring stations in China, spread across different cities.

For all datasets except AQI we use a $70\% - 10\% - 20$ for training, validation and testing. For AQI we use the same splits of Yi et al. [2016]. For simulations, window and horizon length are set as $W = 12, H = 12$ for the traffic datasets, $W = 48, H = 6$ for CER-E, and $W = 24, H = 3$ for AQI. Besides the input time series, we use exogenous variables consisting of 1) sinusoidal functions encoding the time of the day and 2) one-hot encodings of the day of the week. For datasets with a large number of missing values (METR-LA and AQI), we add

as an additional exogenous variable the binary mask introduced in Section 3.1.1. Additional details are reported in Table 4.1.

Relational information The adjacency matrices for the traffic and air quality monitoring datasets are obtained by applying a thresholded Gaussian kernel [Shuman et al., 2013] on the pairwise physical distances among sensors. In particular, for both datasets we consider a weighted adjacency matrix where entries $a_t^{i,j} = a^{i,j}$ corresponding to edges among the i-th and j-th node are computed as

$$a^{i,j} = \begin{cases} \exp\left(-\frac{\text{dist}(i,j)^2}{\gamma}\right) & \text{dist}\,(i,j) \leq \delta \\ 0 & \text{otherwise} \end{cases}, \tag{4.1}$$

where dist (i, j) indicates the distance between the i-th and j-th node, γ controls the width of the kernel and δ is the threshold. For CER-E the graph connectivity is derived from the correntropy [Liu et al., 2007] among time series. In particular, we build an adjacency matrix by extracting a K-nearest neighbor graph (with $K = 10$) from the similarity matrix built by computing the average weekly correntropy among time series.

4.1.2 Synthetic data

To evaluate forecasting architectures in a controlled environment, we adopt a modified version of the GP-VAR dataset introduced by Zambon and Alippi [2022].

System model Data are generated by the recursive application, starting from noise, of an autoregressive polynomial graph filter [Isufi et al., 2019] (with parameters shared across time series). Specifically, the underlying system model is specified by

$$\begin{aligned} \boldsymbol{H}_t &= \sum_{l=1}^{L} \sum_{q=1}^{Q} \Theta_{q,l} \boldsymbol{A}^{l-1} \boldsymbol{X}_{t-q}, \\ \boldsymbol{X}_{t+1} &= \boldsymbol{a} \odot \tanh\left(\boldsymbol{H}_t\right) + \boldsymbol{b} \odot \tanh\left(\boldsymbol{X}_{t-1}\right) + \eta_t, \end{aligned} \tag{4.2}$$

where $\boldsymbol{\Theta} \in \mathbb{R}^{Q \times L}$, $\boldsymbol{a} \in \mathbb{R}^N$, $\boldsymbol{b} \in \mathbb{R}^N$ and $\eta_t \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbb{I})$. In the basic version of the dataset, which we refer to as **GPVAR-G**, we fix $\boldsymbol{a} = \boldsymbol{b} = \mathbf{0.5}$ for all time series. The parameters of the spatiotemporal process are set as

$$\Theta = \begin{bmatrix} 2.5 & -2.0 & -0.5 \\ 1.0 & 3.0 & 0.0 \end{bmatrix}. \tag{4.3}$$

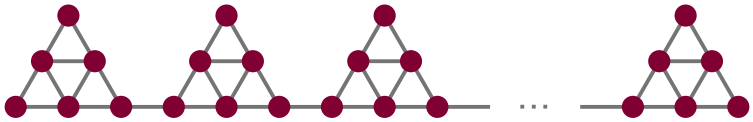


Figure 4.1. GPVAR community graph. We use a graph with 20 communities resulting in a network with 120 nodes.

The graph topology used to generate the data is the community graph shown in Fig. 4.1. In particular, we considered a network with 120 nodes with 20 communities. We use this environment to evaluate forecasting architectures in a setting where taking into account observations at related time series is needed for making optimal forecasts. A modified version of the dataset will be introduced in Chapter 5 to assess the impact of dynamics specific to each time series.

4.2 Baselines

As a case study, we consider *recurrent* architectures. In particular, starting from standard RNNs, implemented as GRUs [Chung et al., 2014], we compare the performance of a single global RNN sharing parameters across the collections against a set of local models and against the multivariate approach. In particular, we consider the following baselines.

RNN: a global node-level GRU conditioning predictions only on the history of the target as in Equation 2.7. This model does not take spatial dependencies into account.

FC-RNN: a GRU taking as input all of the time series concatenated along the spatial dimension as if they were a single multivariate sequence (as in Equation 2.9). This model lacks flexibility and does not exploit prior relational information.

LocalRNNs: a set of local GRUs (Equation 2.6). Each GRU is specialized on a specific time series and no parameter is shared. Similarly to the global node-level model, spatial dependencies are ignored.

Then, for what concerns graph-based architectures, we consider both TTS and T&S recurrent architectures. Specifically, we build TTS models by stacking MP layers after a RNN encoder and take GCRNNs as reference T&S architectures. For both architectures, we implement variants with both isotropic and

Table 4.2. One-step-ahead forecasting error (MAE) of on (5 runs).

		GPVAR-G			
Models		MAE	AZ-test		
			Time	T&S	Space
RNN		$.3999_{\pm.0000}$	$-3.0_{\pm1.3}$	$35.7_{\pm1.0}$	$53.5_{\pm0.5}$
FC-RNN		$.4388_{\pm.0027}$	$261.0_{\pm1.4}$	$252.2_{\pm6.3}$	$95.6_{\pm8.6}$
LocalRNNs		$.4047_{\pm.0001}$	$7.0_{\pm3.7}$	$43.4_{\pm4.2}$	$54.4_{\pm2.3}$
TTS	RNN+IMP	$.3193_{\pm.0000}$	$0.9_{\pm0.0}$	$0.5_{\pm0.7}$	$-0.3_{\pm0.1}$
TTS	RNN+AMP	$.3193_{\pm.0000}$	$1.2_{\pm1.6}$	$0.8_{\pm1.1}$	$-0.1_{\pm0.1}$
T&S	GCRNN-IMP	$.3194_{\pm.0000}$	$1.9_{\pm0.4}$	$1.2_{\pm0.4}$	$-0.3_{\pm0.2}$
T&S	GCRNN-AMP	$.3195_{\pm.0000}$	$2.6_{\pm2.0}$	$1.7_{\pm1.4}$	$-0.3_{\pm0.2}$
Optimal model		.3192	—	—	—

anisotropic message-passing. In particular, we compare the following model architectures.

RNN+IMP: a global TTS model composed by a GRU followed by a stack of isotropic MP layers. The MP operator is defined as in Equation 3.4.

RNN+AMP: a global TTS model composed by a GRU followed by anisotropic MP layers. The MP operator is defined as in Equation 3.5–3.6.

GCRNN-IMP: a global T&S gated GCRNN with isotropic MP. The recurrent cell implementation follows Equation 3.12–3.15, the MP operator is set up as in Equation 3.4.

GCRNN-AMP: a global T&S gated GCRNN with anisotropic MP. The recurrent cell implementation follows Equation 3.12–3.15, the MP operator is set up as in Equation 3.5–3.6.

All the considered architectures follow the template defined in Equation 3.8–3.10, and the different variants are obtained by changing the implementation of the STMP block. We stress that all the global models share the same parameters across the time series in the collection.

State-of-the-art architectures Besides reference architectures, we also consider the following state-of-the-art graph-based forecasting architectures.

DCRNN [Li et al., 2018]: a recurrent T&S model based on an isotropic graph convolutional operator aggregating representations form K-hop neighborhoods [Atwood and Towsley, 2016].

AGCRN [Bai et al., 2020]: a T&S hybrid global-local GCRNN where weights in each layer are dependent on the nodes being processed.

GraphWaveNet [Wu et al., 2019]: a deep T&S spatiotemporal convolutional network based on stack of dilated TCNs and multi-hop graph convolutions.

Additional baselines will be introduced, whenever appropriate, in the next chapters.

4.3 Some empirical results

In this section, we provide some preliminary empirical results to substantiate the discussion carried out in the previous chapters. Additional details and results are reported in Appendix D and in the reference paper [Cini et al., 2023b].

GP-VAR In the first experiment, we train the models on the task of one-step-ahead prediction in GPVAR-G. Besides MAE, we use AZ-whiteness test statistics [Zambon and Alippi, 2022] to assess the presence of temporal, spatial and spatiotemporal correlations on the prediction residuals. In particular, the test statistics allow for quantifying the residual correlation. The performance of the optimal model is obtained analytically by considering the variance of the noise η_t in Equation 4.2. As expected, models that do not exploit spatial dependencies (RNN, FC-RNN and LocalRNNs) struggle in both datasets, displaying large residual spatial correlation, as shown by the spatial and spatiotemporal statistics. Graph-based methods, instead, achieve performance close to the theoretical optimum in GPVAR-G, with the test statistics close to zero.

Benchmarks Table 4.3 shows the results of the empirical evaluation of the reference models on the selected datasets. Graph-based architectures outperform standard local and global predictors in all of the considered scenarios. Notably, as one might expect, local models perform and scale poorly. The performance

Table 4.3. Forecasting error (MAE) on 4 benchmark datasets (5 runs). The best result between each model and its variant with embeddings is in **bold**.

	Models	METR-LA	PEMS-BAY	CER-E	AQI
	RNN	$3.54_{\pm.00}$	$1.77_{\pm.00}$	$4.57_{\pm0.01}$	$14.02_{\pm.04}$
TTS	RNN+IMP	$3.34_{\pm.01}$	$1.72_{\pm.00}$	$4.39_{\pm0.01}$	$12.74_{\pm.02}$
TTS	RNN+AMP	$3.24_{\pm.01}$	$1.66_{\pm.00}$	$4.31_{\pm0.01}$	$12.30_{\pm.02}$
T&S	GCRNN-IMP	$3.35_{\pm.01}$	$1.70_{\pm.01}$	$4.44_{\pm0.01}$	$12.87_{\pm.02}$
T&S	GCRNN-AMP	$3.22_{\pm.02}$	$1.65_{\pm.00}$	$4.57_{\pm.04}$	$12.29_{\pm.02}$
Baselines	DCRNN	$3.22_{\pm.01}$	$1.64_{\pm.00}$	$4.28_{\pm0.01}$	$12.96_{\pm.03}$
Baselines	GraphWaveNet	$3.05_{\pm.03}$	$1.56_{\pm.01}$	$3.97_{\pm0.01}$	$12.08_{\pm.11}$
Baselines	AGCRN	$3.16_{\pm.01}$	$1.61_{\pm.00}$	$4.45_{\pm0.01}$	$13.33_{\pm.02}$

of state-of-the-art architectures shows that performance can be improved w.r.t. the vanilla TTS and T&S models; the next chapter will provide more insights on the source of these performance gains. As a final comment, anisotropic message-passing schemes outperform their isotropic counterparts in most scenarios, while reference TTS architectures perform on par or better than T&S models. However, these results do not necessarily generalize to all datasets and TTS/T&S architectures, e.g., as shown by the performance of GraphWaveNet.

Chapter 5

Local effects

Global STGNNs models have several advantages over standard multivariate models. However, explicitly accounting for the behavior of individual time series might be problematic **(Challenge 1)** [Montero-Manso and Hyndman, 2021]. In this chapter, we introduce hybrid global-local STGNNs and assess how to incorporate node-specific components in graph-based forecasting architecture. In particular, we identify learnable node-embeddings as and effective and efficient methodology to obtain such models. Furthermore, we show that node embeddings for time series outside the training dataset can be obtained by fitting a relatively small number of observations. The chapter is structured as follows. Section 5.1 introduces the problem and background,, while Section 5.2 discusses approaches to obtain hybrid predictors following the template architecture introduce in Section 3.3.3. Section 5.3 introduces learnable node embeddings as method to amortize the learning of node-specific components. The transferability of the resulting forecasting models to different node sets is then discussed in Section 5.4; Section 5.6 empirically evaluate the proposed methodology. Section 5.5 and Section 5.7 discuss related works and future directions, respectively.

Reference papers The content of the chapter is partly based on material from the following papers.

- Andrea Cini, Ivan Marisca, Daniele Zambon, and Cesare Alippi. Taming Local Effects in Graph-based Spatiotemporal Forecasting. *Advances in Neural Information Processing Systems*, 2023c

5.1 Dealing with local effects

Modeling the individual dynamics of each time series can be challenging for global models [Montero-Manso and Hyndman, 2021]. As an example, consider the problem of electric load forecasting: consumption patterns of single customers are influenced by shared factors, e.g., weather conditions and holidays, but are also determined by the daily routine of the individual users related by varying degrees of affinity. We refer to the dynamics proper to individual nodes as *local effects*. If local effects are present, global models might require an impractically large model capacity to account for all node-specific dynamics [Montero-Manso and Hyndman, 2021], thus losing some of the advantages of using a global approach. In particular, in the STGNN case, then, increasing the input window for each node would result in a large computational overhead.

Dealing with local effects Local effects might be tacked by providing the global model with a mechanism to promptly characterize the target time series. Ideally, as we will discuss in Section 5.3, this could be done by exploiting static attributes such as the positional encodings used by, e.g., graph Transformer architectures [Rampášek et al., 2022]. However, these are rarely available in practice, and extracting them might require thorough feature engineering. Furthermore, hand-crafted features could end up active as mere identifiers if the size of the collection does not allow for learning meaningful representations of static node attributes. A different approach is that of considering hybrid global-local architectures as discussed in Section 2.5.2 [Wang et al., 2019; Smyl, 2020]. By doing so, the designer accepts a compromise in transferability and model complexity that often empirically leads to higher forecasting accuracy. Notably, the added complexity and specialization might negate the benefits of using a global component. In this regard, it has become common to see node-specific trainable parameters being introduced as a means to extract node(sensor)-level features and then used as spatial identifiers within the processing [Bai et al., 2020; Deng and Hooi, 2021]. Although several of these architectures exist, none of the previous works discussed the implications of the adopted design choices in the context of global and local forecasting. In the following, we interpret learnable *node embeddings* as a method to amortize the learning of local processing blocks in hybrid forecasting architectures.

5.2 Hybrid global-local STGNNs

Combining global graph-based components with local node-level components has the potential for achieving a two-fold objective: 1) exploiting relational dependencies together with side information to learn flexible and efficient graph deep learning models and 2) making at the same time specialized and accurate predictions for each time series. In particular, global-local STGNNs approximate the data-generating process as

$$p^i_{\boldsymbol{\theta},\{\boldsymbol{\omega}^i\}}\left(\boldsymbol{x}^i_{t+h}\middle|\mathcal{G}_{t-W:t},\boldsymbol{U}_{t:t+h+1}\right)\approx p^i\left(\boldsymbol{x}^i_{t+h}\middle|\mathcal{X}_{<t},\boldsymbol{U}_{t:t+h+1}\right), \tag{5.1}$$

where parameter vector $\boldsymbol{\theta}$ is shared across all nodes, whereas $\{\boldsymbol{\omega}^i\}_{i=1}^N$ are time-series dependent parameters. The associated point predictor is

$$\widehat{\boldsymbol{X}}_{t:t+H}=\mathcal{F}\left(\mathcal{G}_{t-W:t},\dots;\boldsymbol{\theta},\{\boldsymbol{\omega}^i\}_{i=1}^N\right) \tag{5.2}$$

where $\mathcal{F}(\cdot)$ is shared among all nodes. Predictor $\mathcal{F}(\,\cdot\,;\boldsymbol{\theta},\{\boldsymbol{\omega}^i\}_{i=1}^N)$ could be implemented, for example, as a sum between a global model and a (simpler) local one:

$$\widehat{\boldsymbol{X}}^{(1)}_{t:t+H}=\mathcal{F}_{\text{G}}\left(\mathcal{G}_{t-W:t},\dots;\boldsymbol{\theta}\right),\qquad \hat{\boldsymbol{x}}^{i,(2)}_{t:t+H}=f_i\left(\boldsymbol{x}^i_{t-W:t},\dots;\boldsymbol{\omega}^i\right) \tag{5.3}$$

$$\hat{\boldsymbol{x}}^i_{t:t+H}=\hat{\boldsymbol{x}}^{i,(1)}_{t:t+H}+\hat{\boldsymbol{x}}^{i,(2)}_{t:t+H}, \tag{5.4}$$

or – with a more integrated approach – by using different weights for each time series at the encoding and/or decoding steps. The latter approach results in using a different encoder and/or decoder for each i-th node in the template STGNN architecture (Equation 3.8–3.10) to extract representations and, eventually, project them back into the input space:

$$\boldsymbol{h}^{i,0}_{t-1}=\textsc{Encoder}_i\left(\boldsymbol{x}^i_{t-1},\boldsymbol{u}^i_{t-1},\boldsymbol{v}^i;\boldsymbol{\omega}^i_{enc}\right), \tag{5.5}$$

$$\hat{\boldsymbol{x}}^i_{t:t+H}=\textsc{Decoder}_i\left(\boldsymbol{h}^{i,L}_{t-1},\boldsymbol{u}^i_{t:t+H};\boldsymbol{\omega}^i_{dec}\right). \tag{5.6}$$

STMP layers could in principle be modified as well to include specialized operators, e.g., by using a different local update function $\textsc{Up}_i(\,\cdot\,)$ for each node. However, this would be impractical unless subsets of nodes are allowed to share parameters to some extent (e.g., by clustering them).

To support our arguments, Table 5.1 shows empirical results for the reference TTS models with isotropic message passing (RNN+IMP) on METR-LA and PEMS-BAY, the two popular traffic forecasting benchmarks introduced in

Table 5.1. Perfomance (MAE) of RNN+IMP variants and number of associated trainable parameters (5-run average).

			METR-LA	[# params]	**PEMS-BAY**	[# params]
Global TTS			3.35 ±0.01	4.71×10^4	1.72 ±0.00	4.71×10^4
Global-local TTS	Weights	Encoder	3.15 ±0.01	2.05×10^5	1.66 ±0.01	2.75×10^5
		Decoder	3.09 ±0.01	2.08×10^5	1.58 ±0.00	3.00×10^5
		Enc. + Dec.	3.16 ±0.01	3.66×10^5	1.70 ±0.01	5.28×10^5
	Embed.	Encoder	3.08 ±0.01	5.59×10^4	1.58 ±0.00	5.96×10^4
		Decoder	3.13 ±0.00	5.59×10^4	1.60 ±0.00	5.96×10^4
		Enc. + Dec.	3.07 ±0.01	5.79×10^4	1.58 ±0.00	6.16×10^4
FC-RNN			3.56 ± 0.03	2.04×10^5	2.32 ± 0.01	3.04×10^5
LocalRNNs			3.69 ±0.00	7.04×10^6	1.91 ±0.00	1.10×10^7

Chapter 4. In particular, we compare the global approach with 3 hybrid global-local variants where local weights are used in the encoder, in the decoder, or in both of them (see Equation 5.5-5.6 and the light brown block in Table 5.1). While fitting a separate RNN to each individual time series fails (LocalRNNs), exploiting a local encoder and/or decoder significantly improves performance w.r.t. the fully global model. Note that the cost of specialization is paid in terms of the number of learnable parameters which is an order of magnitude higher in global-local variants. Remarkably, having both encoder and decoder implemented as local layers leads to an even larger number of parameters and has a marginal impact on forecasting accuracy. The table reports as a reference also results for FC-RNN, the multivariate RNN taking as input the concatenation of all time series.The light gray block in Table 5.1 anticipates the effect of replacing the local layers with the use of learnable *node embeddings*, an approach discussed in depth in the next section.

5.3 Node embeddings

Section 5.3.1 introduces node embeddings as a mechanism to amortize the learning of local components and discusses the supporting empirical results. Section 5.3.2 then proposes possible regularization techniques and discuss the advantages of embeddings in transfer learning scenarios.

5.3.1 Amortized specialization

As discussed in Section 5.1, static node features and positional encodings offer the opportunity to design and obtain node identification mechanisms across different time windows to tailor predictions to a specific node. However, in most settings, node features are either unavailable or insufficient to characterize the node dynamics. Furthermore, not having access to a large enough collection might prevent the model from using such encodings effectively. A possible solution consists of resorting to learnable node embeddings, i.e., a table of learnable parameters $\Theta = \boldsymbol{Q} \in \mathbb{R}^{N \times d_v}$. Rather than interpret these learned representations as positional encodings (such as those used in graph Transformers), our proposal is to consider them as a way of amortizing the learning of node-level specialized models. More specifically, instead of learning a local model for each time series, embeddings fed into modules of a global STGNN and learned end-to-end with the forecasting architecture allow for specializing predictions by simply relying on gradient descent to find a suitable encoding.

The most straightforward option for feeding embeddings into the processing is to update the template model by changing the encoder and decoder as

$$\boldsymbol{h}_t^{i,0} = \textsc{Encoder}\left(\boldsymbol{x}_{t-1}^i, \dots, \boldsymbol{q}^i\right), \tag{5.7}$$

$$\hat{\boldsymbol{x}}_{t:t+H}^i = \textsc{Decoder}\left(\boldsymbol{h}_t^{i,L}, \dots, \boldsymbol{q}^i\right). \tag{5.8}$$

which can be seen as amortized versions of the encoder and decoder in Equation 5.5–5.6. The encoding scheme of Equation 5.7 also facilitates the propagation of relevant information by identifying nodes, an aspect that can be particularly significant as message-passing operators – in particular isotropic ones – can act as low-pass filters that smooth out node-level features [Nt and Maehara, 2019; Oono and Suzuki, 2019].

Table 5.1 (light gray block) reports empirical results that show the effectiveness of embeddings in amortizing the learning of local components, with only a moderate increase in the number of trainable parameters w.r.t. the base global model. In particular, feeding embeddings to the encoder, instead of conditioning the decoding step only, results in markedly better performance, hinting at the impact of providing node identification ahead of MP (additional empirical results are provided in Section 5.6).

5.3.2 Structuring the embedding space

The latent space in which embeddings are learned can be structured and regularized to enjoy benefits in terms of interpretability and transferability. In

fact, accommodating new embeddings can be problematic, as they must fit in a region of the embedding space where the trained model can operate, and, at the same time, capture the local effects at the new nodes. In this setting, proper regularization can provide positive inductive biases and help transfer the learned model to different node sets. As an example, if domain knowledge suggests that neighboring nodes have similar dynamics, Laplacian regularization [Zhou et al., 2003; Kipf and Welling, 2017] can be added to the loss. Clearly, regularization needs to preserve the effectiveness of embeddings in specializing the predictions. In the following, we propose two general-purpose strategies based on variational inference and node clustering to impose soft constraints on the latent space. As shown in Section 5.6, the resulting structured space additionally allows us to gather insights into the features encoded in the embeddings.

Variational regularization As a probabilistic approach to structuring the latent space, we propose to consider learned embeddings as parameters of an approximate posterior distribution – given the training data – on the vector used to condition the predictions. In practice, we model each node embedding as a sample from a multivariate Gaussian $\boldsymbol{q}^i \sim r^i(\boldsymbol{q}^i|\mathcal{D}) = \mathcal{N}\left(\boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i^2)\right)$ where $(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i)$ are the learnable (local) parameters. Each node-level distribution is fitted on the training data by considering a standard Gaussian prior and exploiting the reparametrization trick [Kingma and Welling, 2013] to minimize

$$\delta_t \doteq \mathbb{E}_{\boldsymbol{Q}\sim R}\left[\ell\left(\widehat{\boldsymbol{X}}_{t:t+H}, \boldsymbol{X}_{t:t+H}\right)\right] + \beta D_{\mathrm{KL}}(R|P), \tag{5.9}$$

where $P = \mathcal{N}(\mathbf{0}, \mathbb{I})$ is the prior, D_{KL} the Kulback-Leibler divergence, and β controls the regularization strength. This regularization scheme pushes toward learning a smooth latent space where it is easier to interpolate between representations, thus providing a principled way for accommodating different node embeddings.

Clustering regularization A different (and potentially complementary) approach to structuring the latent space is to incentivize node embeddings to form clusters and, consequently, to self-organize into different groups. We do so by introducing a regularization loss inspired by deep K-means algorithms [Yang et al., 2017a]. In particular, besides the embedding table $\boldsymbol{Q} \in \mathbb{R}^{N\times d_q}$, we equip the embedding module with a matrix $\boldsymbol{C} \in \mathbb{R}^{K\times d_q}$ of $K \ll N$ learnable centroids and a cluster assignment matrix $\boldsymbol{S} \in \mathbb{R}^{N\times K}$ encoding scores associated to each node-cluster pair. We consider scores as logits of a categorical (Boltzmann)

distribution and learn them by minimizing the regularization term

$$\mathcal{L}_{reg} \doteq \mathbb{E}_{\boldsymbol{M}} \left[\|\boldsymbol{Q} - \boldsymbol{M}\boldsymbol{C}\|_2 \right], \quad p(\boldsymbol{M}_{ij} = 1) = \frac{e^{\boldsymbol{S}_{ij}/\tau}}{\sum e^{\boldsymbol{S}_{ik}/\tau}},$$

where τ is a hyperparameter. We minimize $\mathcal{L}_{reg}$ – which corresponds to the embedding-to-centroid distance – jointly with the forecasting loss by relying on the Gumbel softmax trick [Maddison et al., 2017]. Similarly to the variational inference approach, the clustering regularization gives structure to embedding space and allows for inspecting patterns in the learned local components (see Section 5.6).

5.4 Transferability

One of the main advantages of global models based on GNNs is that they can make predictions for never-seen-before node sets, and handle graphs of different sizes and variable topology. As we discussed, graph-based predictors can be used for zero-shot transfer and inductive learning and can easily handle new time series being added to the collection; this corresponds to the real-world scenario of new sensors being added to a network over time. This flexibility has several applications in time series processing besides forecasting, e.g., as models for performing spatiotemporal kriging [Stein, 1999] or virtual sensing [Cini et al., 2022b; Wu et al., 2021b; Zheng et al., 2023] (see Chapter 6), where inductive STGNNs can be used to perform graph-based spatial interpolation. However, as we will show in Section 5.6, performance in the inductive setting can quickly degrade as soon as the target time series exhibit dynamics that deviate from those observed in the training examples. Clearly, this flexibility is completely compromised in architectures that include node-specific local components and, as a result, cannot make zero-shot forecasts.

Transfer learning STGNNs can be adjusted to account for other sets of time series (with different dynamics) by fine-tuning on the available data a subset of the forecasting architectures weights. In particular, if local components are replaced by node embedding, adapting the specialized components is relatively cheap since the number of parameters to fit w.r.t. the new context is usually contained, and the structure of the embedding latent space can be exploited. In other words, node embeddings can amortize the cost of the transfer learning by limiting the fine-tuning of the model to fitting a new set of embeddings for the nodes in the target set while freezing the shared weights. Experiments in

Section 5.6 provide an in-depth empirical analysis of transferability within our framework and show that the discussed regularizations can be useful in this regard.

5.5 Related work

Although the interplay between globality and locality plays such a central role in graph-based forecasting, it has received only minor attention from the research community. In particular, while several architectures have included node-specific components, none of the previous works discussed the implications of the adopted design choices in the context of global and local forecasting. Among the methods that focus on modeling node-specific dynamics, Bai et al. [2020] use a factorization of the weight matrices in a recurrent STGNN to adapt the extracted representation to each node. Conversely, Chen et al. [2021a] use a model inspired by Wang et al. [2019] consisting of a global GNN paired with a local model conditioned on the neighborhood of each node. Node embeddings have been mainly used in structure-learning modules to amortize the cost of learning the full adjacency matrix [Wu et al., 2019; Shang and Chen, 2021; Deng and Hooi, 2021] and in attention-based approaches as positional encodings [Satorras et al., 2022; Marisca et al., 2022; Zheng et al., 2020]. Learned embeddings are indeed key components in several Transformer architectures [Liu et al., 2023a; Xiao et al., 2024]. Shao et al. [2022] observe how adding node and time embeddings to an otherwise global (univariate) architecture can outperform several state-of-the-art STGNNs. Conversely, Yin et al. [2022] used a cluster-based regularization to fine-tune an AGCRN-like model on different datasets. None of the previous works systematically directly addressed the problem of globality and locality in STGNNs, nor provided a comprehensive framework accounting for learnable node embeddings within different settings and architectures. Besides GDL methods, as we already discussed in Chapter 2, there are several examples of hybrid global and local time series forecasting models [Wang et al., 2019; Smyl, 2020], which however do not address the transfer learning scenarios.

5.6 Empirical results

This section reports salient results of an extensive empirical analysis of global and local models and combinations thereof in spatiotemporal forecasting benchmarks and different problem settings; additional results can be found in the reference

Table 5.2. One-step-ahead forecasting error (MAE) of on GPVAR-L (5 runs).

		GPVAR-L			
Models		MAE	AZ-test		
			Time	T&S	Space
	RNN	$.5441_{\pm.0002}$	$10.8_{\pm2.6}$	$0.5_{\pm1.9}$	$-10.1_{\pm0.3}$
	↪ + Emb.	$.4611_{\pm.0003}$	$6.1_{\pm1.4}$	$-1.1_{\pm1.1}$	$-7.7_{\pm0.8}$
	FC-RNN	$.5948_{\pm.0102}$	$108.4_{\pm8.1}$	$73.6_{\pm6.5}$	$-4.4_{\pm2.3}$
	LocalRNNs	$.4610_{\pm.0003}$	$3.2_{\pm1.1}$	$-2.3_{\pm1.1}$	$-6.5_{\pm1.1}$
STL	RNN+IMP	$.3808_{\pm.0031}$	$13.8_{\pm2.2}$	$7.9_{\pm1.6}$	$-2.6_{\pm0.9}$
	↪ + Emb.	$.3197_{\pm.0001}$	$1.4_{\pm1.0}$	$1.0_{\pm0.9}$	$-0.0_{\pm0.3}$
	RNN+AMP	$.3639_{\pm.0032}$	$13.1_{\pm2.6}$	$7.5_{\pm2.4}$	$-2.5_{\pm1.0}$
	↪ + Emb.	$.3199_{\pm.0001}$	$1.8_{\pm0.7}$	$1.0_{\pm0.6}$	$-0.3_{\pm0.3}$
T&S	GCRNN-IMP	$.3714_{\pm.0070}$	$15.2_{\pm2.9}$	$9.0_{\pm1.6}$	$-2.5_{\pm1.5}$
	↪ + Emb.	$.3204_{\pm.0001}$	$2.4_{\pm0.9}$	$1.8_{\pm0.7}$	$0.1_{\pm0.2}$
	GCRNN-AMP	$.3518_{\pm.0013}$	$10.5_{\pm2.5}$	$5.7_{\pm1.9}$	$-2.4_{\pm0.6}$
	↪ + Emb.	$.3204_{\pm.0002}$	$1.8_{\pm0.6}$	$0.9_{\pm0.4}$	$-0.4_{\pm0.5}$
Optimal model		.3192	—	—	—

paper [Cini et al., 2023c]. We consider the same models and hyperparameters introduced in Chapter 4; additionally, we consider the variations of the models augmented by the use of node embeddings ((Equation 5.7–5.8)) Note that among the baselines selected from the literature (namely DCRNN, GraphWaveNet, and AGCRN) only DCRNN can be considered fully global (see Section 5.5).

5.6.1 Synthetic data

We start by assessing the performance of hybrid global-local spatiotemporal models on a variant of GPVAR-G (Section 4.1), modified to include local effects. In particular, we keep the structure data-generating process unchanged, i.e.,

data are generated as

$$\begin{aligned} \boldsymbol{H}_t &= \sum_{l=1}^{L} \sum_{q=1}^{Q} \Theta_{q,l} \boldsymbol{A}^{l-1} \boldsymbol{X}_{t-q}, \\ \boldsymbol{X}_{t+1} &= \boldsymbol{a} \odot \tanh\left(\boldsymbol{H}_t\right) + \boldsymbol{b} \odot \tanh\left(\boldsymbol{X}_{t-1}\right) + \eta_t, \end{aligned} \tag{5.10}$$

but, instead of having $\boldsymbol{a}$ and $\boldsymbol{b}$ fixed, we sample them from a uniform distribution for each time series such that for each i-th node

$$\boldsymbol{a}^i, \boldsymbol{b}^i \sim \mathcal{U}\left(-2, 2\right). \tag{5.11}$$

Table 5.2 shows forecasting accuracy for reference architectures with a 6-steps window on data generated from the processes. In GPVAR-L, global and univariate models fail to match the performance of STGNNs that include local components; interestingly, the global model with anisotropic MP outperforms the isotropic alternative, suggesting that the more advanced MP schemes can lead to more effective state identification.

5.6.2 Benchmarks

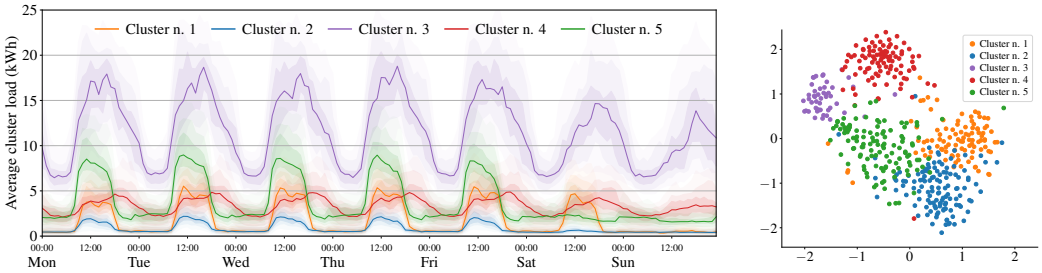
We then compare the performance of reference architectures and baselines with and without node embeddings at the encoding and decoding steps. Note that, while reference architectures and DCRNN are purely global models, the vanilla versions of GraphWaveNet and AGCRN already use node embeddings to obtain an adjacency matrix for MP. AGCRN, furthermore, uses embeddings to make the convolutional filters adaptive w.r.t. the node being processed. We evaluate all models on real-world datasets from three different domains (traffic networks, energy analytics, and air quality monitoring), by considering the settings introduced in Chapter 4. In particular, besides the already mentioned traffic forecasting benchmarks (**METR-LA** and **PEMS-BAY**), we run experiments on smart metering data from the **CER-E** dataset [Commission for Energy Regulation, 2016] and air quality measurements from **AQI** [Zheng et al., 2015]. Table 4.3 reports forecasting MAE averaged over the forecasting horizon. Global-local reference models outperform the fully global variants in every considered scenario. A similar observation can be made for the state-of-art architectures, where the impact of node embeddings (at encoding and decoding) is large for the fully global DCRNN and more contained in models already equipped with local components. Note that hyperparameters were not tuned to account for the change in architecture. Surprisingly, the simple RNN+IMP model equipped with node embeddings achieves results comparable to that of state-of-the-art

Table 5.3. Forecasting error (MAE) on 4 benchmark datasets (5 runs). The best result between each model and its variant with embeddings is in **bold**.

Models	METR-LA	PEMS-BAY	CER-E	AQI	METR-LA	PEMS-BAY	CER-E	AQI
Reference arch.	Global models				Global-local models (with embeddings)			
RNN	$3.54_{\pm.00}$	$1.77_{\pm.00}$	$4.57_{\pm0.01}$	$14.02_{\pm.04}$	$\mathbf{3.15}_{\pm.03}$	$\mathbf{1.59}_{\pm.00}$	$\mathbf{4.22}_{\pm0.02}$	$\mathbf{13.73}_{\pm.04}$
GCRNN-IMP	$3.35_{\pm.01}$	$1.70_{\pm.01}$	$4.44_{\pm0.01}$	$12.87_{\pm.02}$	$\mathbf{3.10}_{\pm.01}$	$\mathbf{1.59}_{\pm.00}$	$\mathbf{4.18}_{\pm0.01}$	$\mathbf{12.48}_{\pm.03}$
RNN+IMP	$3.34_{\pm.01}$	$1.72_{\pm.00}$	$4.39_{\pm0.01}$	$12.74_{\pm.02}$	$\mathbf{3.08}_{\pm.01}$	$\mathbf{1.58}_{\pm.00}$	$\mathbf{4.12}_{\pm0.03}$	$\mathbf{12.33}_{\pm.02}$
GCRNN-AMP	$3.22_{\pm.02}$	$1.65_{\pm.00}$	$4.57_{\pm0.04}$	$12.29_{\pm.02}$	$\mathbf{3.07}_{\pm.02}$	$\mathbf{1.59}_{\pm.00}$	$\mathbf{4.17}_{\pm0.02}$	$\mathbf{12.17}_{\pm.05}$
RNN+AMP	$3.24_{\pm.01}$	$1.66_{\pm.00}$	$4.31_{\pm0.01}$	$12.30_{\pm.02}$	$\mathbf{3.06}_{\pm.01}$	$\mathbf{1.58}_{\pm.01}$	$\mathbf{4.13}_{\pm0.01}$	$\mathbf{12.15}_{\pm.02}$
Baseline arch.	Original				Embeddings at Encoder and Decoder			
DCRNN	$3.22_{\pm.01}$	$1.64_{\pm.00}$	$4.28_{\pm.01}$	$12.96_{\pm.03}$	$\mathbf{3.07}_{\pm.02}$	$\mathbf{1.60}_{\pm.00}$	$\mathbf{4.13}_{\pm.02}$	$\mathbf{12.53}_{\pm.02}$
GraphWaveNet	$3.05_{\pm.03}$	$\mathbf{1.56}_{\pm.01}$	$\mathbf{3.97}_{\pm.01}$	$12.08_{\pm.11}$	$\mathbf{2.99}_{\pm.02}$	$1.58_{\pm.00}$	$4.01_{\pm.01}$	$\mathbf{11.81}_{\pm.04}$
AGCRN	$3.16_{\pm.01}$	$\mathbf{1.61}_{\pm.00}$	$4.45_{\pm.01}$	$13.33_{\pm.02}$	$\mathbf{3.14}_{\pm.00}$	$1.62_{\pm.00}$	$\mathbf{4.37}_{\pm.02}$	$\mathbf{13.28}_{\pm.03}$

STGNNs with a significantly lower number of parameters and a streamlined architecture. Interestingly, while both global and local RNNs models fail, the hybrid global-local RNN obtains remarkable performance.

Structured embeddings To test the hypothesis that structure in embedding space provides insights on the local effects at play, we consider the clustering regularization method (Section 5.3.2) and the reference RNN+IMP model trained on the CER-E dataset. We set the number of learned centroids to $K = 5$ and train the cluster assignment mechanism end-to-end with the forecasting architecture. Then, we inspect the clustering assignment by looking at intra-cluster statistics. In particular, for each load profile, we compute the weekly average load curve, and, for each hour, we look at quantiles of the energy consumption within each cluster. Figure 5.1a shows the results of the analysis by reporting the *median* load profile for each cluster; shaded areas correspond to quantiles with 10% increments. Results show that users in the different clusters have distinctly different consumption patterns. Figure 5.1b shows a 2D



(a) Average cluster load by day of the week. *(b)* Node embeddings.

Figure 5.1. Time series clusters in CER-E obtained by regularizing the embedding space. **(a)** Average load for each clusters. **(b)** t-SNE plot of the corresponding node embeddings.

Table 5.4. Datasets considered in the transfer learning experiments.

DATASETS	Type	Time steps	Nodes	Edges	Rate
PEMS03	Directed	26,208	358	546	5 minutes
PEMS04	Directed	16,992	307	340	5 minutes
PEMS07	Directed	28,224	883	866	5 minutes
PEMS08	Directed	17,856	170	277	5 minutes

t-SNE visualization of the learned node embeddings, providing a view of the latent space and the effects of the cluster-based regularization.

5.6.3 Transfer learning

In this experiment, we consider the scenario in which an STGNN for traffic forecasting is trained by using data from multiple traffic networks and then used to make predictions for a disjoint set of sensors sampled from the same region. We use the **PEMS03**, **PEMS04**, **PEMS07**, and **PEMS08** datasets [Guo et al., 2021b], which contain measurements from 4 different districts in California. We train models on 3 of the datasets, fine-tune on 1 week of data from the target left-out dataset, validate on the following week, and test on the week thereafter. Following previous works [Guo et al., 2021b], we split the datasets into 60%/20%/20% for training, validation, and testing, respectively. We compare variants of RNN+IMP with and without embeddings fed into encoder and decoder. Together with the unconstrained embeddings, we also consider the variational and clustering regularization approaches introduced

Table 5.5. Forecasting error (MAE) in the transfer learning setting (5 runs average). Results refer to a 1-week fine-tuning set size on all PEMS datasets.

RNN+IMP		PEMS03	PEMS04	PEMS07	PEMS08
Fine-tuning	Global	15.30 ± 0.03	21.59 ± 0.11	23.82 ± 0.03	15.90 ± 0.07
	Embeddings	14.64 ± 0.05	20.27 ± 0.11	**22.23** ± **0.08**	**15.45** ± **0.06**
	– Variational	**14.56** ± **0.03**	20.19 ± 0.05	22.43 ± 0.02	**15.41** ± **0.06**
	– Clustering	**14.60** ± **0.02**	**19.91** ± **0.11**	**22.16** ± **0.07**	**15.41** ± **0.06**
Zero-shot		18.20 ± 0.09	23.88 ± 0.08	32.76 ± 0.69	20.41 ± 0.07

Table 5.6. Forecasting error (MAE) on PEMS04 in the transfer learning setting by varying fine-tuning set size (5 runs average).

Model	Training set size			
RNN+IMP	2 weeks	1 week	3 days	1 day
Global	20.86 ± 0.03	21.59 ± 0.11	21.84 ± 0.06	22.26 ± 0.10
Embeddings	19.96 ± 0.08	20.27 ± 0.11	21.03 ± 0.14	21.99 ± 0.13
– Variational	19.94 ± 0.08	20.19 ± 0.05	20.71 ± 0.12	**21.20** ± **0.15**
– Clustering	**19.69** ± **0.06**	**19.91** ± **0.11**	**20.48** ± **0.09**	21.91 ± 0.21

in Section 5.3.2. At the fine-tuning stage, the global model updates all of its parameters, while in the hybrid global-local approaches only the embeddings are fitted to the new data. Table 5.5 reports results for the described scenario. The fully global approach is outperformed by the hybrid architectures in all target datasets. Besides the significant improvement in performance, adjusting only node embeddings retains performance on the source datasets. Furthermore, results show the positive effects of regularizing the embedding space in the transfer setting. This is further confirmed by results in Table 5.6, which reports, for PEMS04, how forecasting error changes in relation to the length of the fine-tuning window. We refer to Appendix E for an in-depth analysis of several additional transfer learning scenarios.

5.7 Discussion and future directions

This chapter investigated the impact of locality and globality in graph-based spatiotemporal forecasting architectures and introduced a framework for building hybrid global-local predictors. The introduced framework sheds light on the empirical results associated with using trainable node embeddings in the spatiotemporal forecasting literature. We also discussed different architectures and regularization techniques to account for local effects in different scenarios. The proposed methodologies are thoroughly empirically validated and, although not inductive, prove to be effective in a transfer learning context. We argue that the issues addressed here are of central importance for the understanding and design of effective graph-based spatiotemporal forecasting architectures.

Future directions Adding node specific components to the forecasting architecture, even if by exploiting node embeddings, makes the number of parameters scale linearly with the number of input time series (nodes). Future research could aim at tackling the issues by designing intermediate solutions, e.g., relying clustering to identify groups of time series that can share parameters. This idea has been, for example, explored by Bandara et al. [2020], which partition time series collections into groups and learn a global model specific to each group. Similarly, Xiao et al. [2024] introduced a Transformer architecture with a fixed number of learnable positional encodings. Nonetheless, assigning new target time series to a specific group if not trivial, especially if only few observations are available. In summary, future research should focus on 1) making the learning of specific components more scalable and 2) further tailoring the resulting methodologies to transfer (and possibly inductive) learning.

Chapter 6

Missing data

This chapter addresses the problem of missing data and introduces graph-based methodologies for data reconstruction **(Challenge 2)**. Dealing with missing values and incomplete time series is a labor-intensive and inevitable task when handling data from real-world applications. Here, we consider the problem in the context of collections of correlated time series and introduce methodologies to reconstruct missing observations by exploiting dependencies across time and space. In this research, we introduce a comprehensive GDL framework for missing data imputation. We propose two STGNN architectures for data reconstruction based on graph RNNs and attention-based MP operators. Empirical results show that graph-based models outperform state-of-the-art methods on relevant benchmarks with performance improvements often higher than 20%. Section 6.1 introduces the problem and discusses related works. Section 6.3 and 6.4 present the proposed methodologies. Experimental results are reported and discussed in Section 6.5. Section 6.6 summarizes the results and discusses future directions.

Reference papers The content of the chapter is partly based on material from the following papers.

- Andrea Cini, Ivan Marisca, and Cesare Alippi. Filling the G_ap_s: Multivariate Time Series Imputation by Graph Neural Networks. In *International Conference on Learning Representations*, 2022b

- Ivan Marisca, Andrea Cini, and Cesare Alippi. Learning to Reconstruct Missing Data from Spatiotemporal Graphs with Sparse Observations. In *Advances in Neural Information Processing Systems*, 2022

- Giovanni De Felice, Andrea Cini, Daniele Zambon, Vladimir Gusev, and Cesare Alippi. Graph-based Virtual Sensing from Sparse and Partial Multivariate Observations. In *International Conference on Learning Representations*, 2024

6.1 Dealing with missing data

Imputation of missing values is a prominent problem in multivariate time-series analysis from both theoretical and practical perspectives [Little and Rubin, 2019]. In fact, current systems acquire data through large and heterogeneous sensor networks often characterized by irregular sampling procedures which, together with hardware and software faults, result in incomplete observations. This means that the time series collections acquired by these systems can have considerable amounts of missing values. Missing values can appear uniformly at random over time and space or, e.g., be spatially and temporally correlated. In particular, we might observe contiguous blocks of missing data in time but also in space, with concurrent failures localized in a region of the network. In this context, effective imputation methods should aim at reconstructing the missing observations by possibly exploiting both temporal and spatial dependencies.

Dealing with missing data Reconstructing missing observations has relevant applications on its own as it provides insights on the monitored systems and, in many cases, is a required preprocessing step to enable further analysis, e.g., forecasting. Among different imputation methods, approaches based on deep learning have become increasingly popular [Yoon et al., 2018; Cao et al., 2018; Liu et al., 2019; Du et al., 2023]. However, as discussed in Section 6.1.1, these methods often disregard available relational information. As already argued in the thesis, GDL offers the tools needed to go beyond this limitation [Cini et al., 2022b; Marisca et al., 2022; Chen et al., 2022; Liu et al., 2023b; Wang et al., 2023; De Felice et al., 2024]. For the first time in the literature, the following introduces a comprehensive GDL framework for (multivariate) time series imputation by exploiting purposely designed STGNNs and relational side information [Cini et al., 2022b; Marisca et al., 2022]. In particular, we introduce *Graph Recurrent Imputation Network* (GRIN), a bidirectional GCRNNs model performing imputation by autoregressively integrating information across both time and space. We then extended the methodology to attention-based methods to tackle error-compounding issues that might occur in certain scenarios when relying on recurrent architectures [Marisca et al., 2022].

6.1.1 Related work

A large amount of literature addresses missing value imputation in time series. Besides interpolation based on polynomial curve fitting, popular approaches aim at filling up missing values by using standard statistical methods. For

example, several approaches rely on K-nearest neighbors [Troyanskaya et al., 2001; Beretta and Santaniello, 2016], expectation maximization [Ghahramani and Jordan, 1994] or linear predictors and state-space models [Durbin and Koopman, 2012]. Low-rank approximation methods, such as methods based on matrix factorization [Cichocki and Phan, 2009], are also widely popular and can incorporate temporal [Yu et al., 2016] and relational information [Kalofolias et al., 2014; Rao et al., 2015]. The idea of exploiting geometry and relational structures to perform dimensionality reduction and denoising is also related to manifold learning [Tenenbaum et al., 2000; Belkin and Niyogi, 2001; Coifman and Lafon, 2006]. Currently, deep learning methods are among the most commonly used approaches in time series imputation.

Deep learning for time series imputation Deep autoregressive models based on RNNs have been widely studied and adopted [Che et al., 2018; Yoon et al., 2017; Cao et al., 2018; Miao et al., 2021]. In this direction, BRITS [Cao et al., 2018] is archetypal of several related works which exploit bidirectional RNNs to perform imputation. Several approaches in the literature, then, rely on generative models [Yoon et al., 2018; Luo et al., 2018, 2019; Miao et al., 2021; Alcaraz and Strodthoff, 2023]. Attention-based imputation techniques have also been proposed [Du et al., 2023; Tashiro et al., 2021; Shukla and Marlin, 2021]. However, none of these prior works explicitly account for spatial dependencies within the graph processing framework and overlook the spatial dimension of the problem. Other works, instead, address this problem in the context of continuous-time models [Chen et al., 2018a; Rubanova et al., 2019; Biloš et al., 2023]. Huang et al. [2020], in particular, exploit graph representations to model dependencies among irregularly sampled spatiotemporal data. The limits of deep autoregressive approaches in data reconstruction have also been tackled by using hierarchical imputation methods [Liu et al., 2019]. GRIN [Cini et al., 2022b] has been the first among several GDL frameworks for general multivariate time series imputation [Chen et al., 2022; Liu et al., 2023b; Wang et al., 2023]. Other graph-based architectures have been used in application-specific settings, such as traffic data [Liang et al., 2022b; Ye et al., 2021], load profiles from smart grids [Kuppannagari et al., 2021], and trajectory reconstruction [Omidshafiei et al., 2022]. Many different architectures are being investigated and we refer to Jin et al. [2023b] for a recent survey. Related to imputation, STGNNs have been also used to perform virtual sensing [Stein, 1999; Wu et al., 2021b; Cini et al., 2022b], i.e., to infer observations at unmonitored locations, i.e., nodes with no valid observation attached. In particular, Wu et al. [2021b], Zheng et al. [2023], and Xu et al. [2023] directly tackle the virtual sensing

problem in inductive learning settings. In particular, we have recently explored the problem of virtual sensing from partial observations in [De Felice et al., 2024], where, besides considering relationships between different time series, we leveraged a nested graph structure to model dependencies among covariates. Besides applications to time series, GNN are also popular architectures for reconstructing missing features in static graphs [Spinelli et al., 2020; You et al., 2020] and for relational matrix completion [Monti et al., 2017; Berg et al., 2017]. In this context, Rossi et al. [2021], finally, adapt label propagation [Ghahramani and Jordan, 1994] to deal with missing node features.

6.2 Problem definition

To model the presence of missing values, as mentioned in Section 3.1.1, we consider, at each step, a binary mask $\boldsymbol{M}_t \in \{0,1\}^{N \times d_x}$, where $\boldsymbol{m}_t^i[k] = 1$ if k-th channel of $\boldsymbol{x}_t^i$ contains a valid observation, and $\boldsymbol{m}_t^i[k] = 0$ otherwise. In the following, for simplicity's sake, we focus on the case where missing value affect all the available channels at the same time, i.e., if there exist a k such that $\boldsymbol{m}_{t[k]}^i = 0$ then $\boldsymbol{m}_{t[k]}^i = 0$ at all the k-th channels[1]. In this setting, the notation can be simplified by considering mask $\boldsymbol{m}_t^i$ to be scalar, i.e., $\boldsymbol{m}_t^i \in \{0,1\}$. We denote by $\widetilde{\boldsymbol{X}}_t$ the *unknown* ground truth observations at time step t, i.e., the complete observation matrix without any missing data. We assume stationarity of missing data distribution and, in experiments, we mostly focus on the missing at random scenario [Rubin, 1976]. We do not make any assumption about the number of concurrent sensor failures or the length of missing data blocks; thus, multiple failures extended over time are accounted for. Nonetheless, one should expect imputation performance to scale with the number of concurrent faults and the time length of missing data blocks.

Time series imputation The objective of time series imputation is to reconstruct missing values in a sequence of input data. Considering again point predictions, we can define the missing data reconstruction error as

$$\mathcal{L}\left(\widehat{\boldsymbol{X}}_{t:t+T}, \widetilde{\boldsymbol{X}}_{t:t+T}, \boldsymbol{M}_{t:t+T}\right) = \sum_{\tau=t}^{t+T} \frac{\sum_{i=1}^{N} \overline{\boldsymbol{m}}_\tau^i \cdot \ell\left(\hat{\boldsymbol{x}}_\tau^i, \tilde{\boldsymbol{x}}_\tau^i\right)}{\sum_{i=1}^{N} \overline{\boldsymbol{m}}_\tau^i}, \tag{6.1}$$

[1]This assumption is removed in [De Felice et al., 2024] and can be easily circumvented by considering a heterogenous graph with multiple time series associated to each node (see Section 3.1.1).

where $\hat{\boldsymbol{x}}_h^i$ is the reconstructed $\tilde{\boldsymbol{x}}_h^i$ and $\overline{\boldsymbol{M}}_{t,t+T}$ and $\overline{\boldsymbol{m}}_h^i$ are respectively the logical binary complement of $\boldsymbol{M}_{t,t+T}$ and $\boldsymbol{m}_h^i$. Note that, in practice, it is impossible to have access to $\widetilde{\boldsymbol{X}}_{t,t+T}$ and, as a consequence, it is necessary to define a surrogate optimization objective, e.g., by using forecasting as a surrogate task or simulating the presence of additional missing values. In the context of trainable, parametric, imputation methods, we consider two different operational settings. In the first one, denoted by *in-sample imputation*, the model is trained to reconstruct missing values in a given *fixed* input sequence $\boldsymbol{X}_{t:t+T}$ w.r.t. specific time steps. Differently, in the second setting (referred to as *out-of-sample imputation*), the model is trained and evaluated w.r.t. different time steps. Clearly, in both cases, the model is prevented from accessing the ground-truth data used for the final evaluation. In other words, the first setting corresponds to a transductive learning setting (both spatially and temporally) while the second setting requires models that can perform inductive learning along the temporal dimension. The in-sample setting simulates the case where a practitioner fits the model directly on the sequence to fill up its gaps. The out-of-sample scenario, instead, simulates the case where one wishes to use a model fitted on a set of historical data to impute missing values in an unseen target sequence.

6.3 Graph Recurrent Imputation Network

In this section, we present GRIN, a graph-based, recurrent neural architecture for correlated time series imputation. Given a collection of time series $\boldsymbol{X}_{t:t+T}$ and associated mask $\boldsymbol{M}_{t:t+T}$, GRIN is trained to reconstruct missing values in the input sequence by combining the information coming from both the temporal and spatial dimensions. To do so, we design GRIN as a *bidirectional* GCRNN which progressively processes the input sequence both forward and backward in time by performing two stages of imputation for each direction. Then, an MLP processes the representation learned by the forward and backward models to obtain a final – refined – imputation for each point in time and space. An overview of the complete architecture is given in Figure 6.1. As shown in the figure, the two modules impute missing values iteratively, using at each time step previously imputed values as input. We start by detailing the processing carried out by the unidirectional model and then provide the bidirectional extension. To simplify the notation, we ignore the presence of exogenous variables, which can easily be integrated into the framework whenever present.

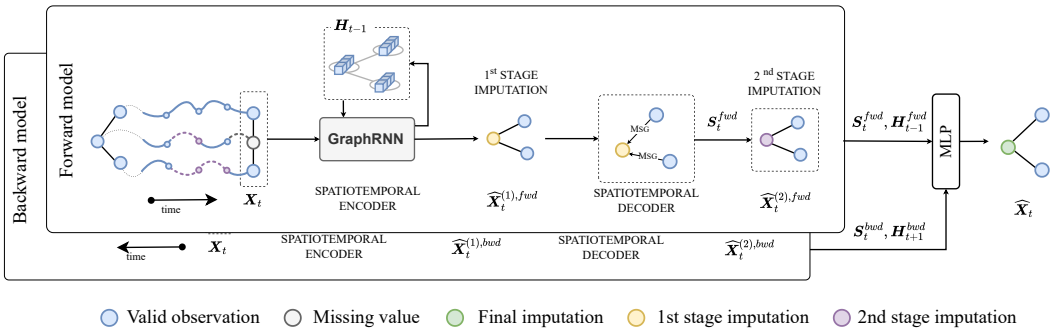


Figure 6.1. An overview of GRIN. Here, each unidirectional GRIN module is processing the t-th step of 3 input time series. There is one missing value at the considered time step. GRIN performs a first imputation, which is then processed and refined by the spatial decoder. The second-stage imputation is then used to continue the processing autoregressively. An MLP processes the representations obtained by the forward and backward models to obtain final imputations. The full architecture is trained end-to-end.

Unidirectional model Each GRIN module consists in two blocks, a spatiotemporal encoder and a spatial decoder, which process the input time series in two stages. The spatiotemporal encoder maps the input $\boldsymbol{X}_{t:t+T}$ into a spatiotemporal representation $\boldsymbol{H}_{t:t+T} \in \mathbb{R}^{N \times l}$ by exploiting a ad-hoc designed GCRNN. The spatial decoder, instead, takes advantage of the learned representations to perform two consecutive rounds of imputation. A first-stage imputation is obtained from the representation by using a linear readout; the second one exploits available relational, spatial, information at time step t. In particular, the decoder is implemented by a GNN which learns to infer the observed values at each i-th node – $\boldsymbol{x}_t^i$ – by refining first-stage imputations and processing representations $\boldsymbol{H}_{t-1}$ and values observed at neighboring nodes.

6.3.1 Spatiotemporal encoder

In the encoder, the input sequence $\boldsymbol{X}_{t:t+T}$ and mask $\boldsymbol{M}_{t:t+T}$ are processed sequentially one step at a time, by means of a recurrent neural network with gates implemented by message-passing layers. In particular, we leverage GRUs [Cho et al., 2014] and, similarly to Seo et al. [2018] and Li et al. [2018], we implement the GRU gates by relying on on MP operators (see Equation 3.12–3.15). The

resulting *message-passing GRU* (MPGRU) processes the input as:

$$\boldsymbol{R}_t = \sigma\left(\mathrm{MP}\left(\left[\hat{\boldsymbol{X}}_t^{(2)}||\boldsymbol{M}_t||\boldsymbol{H}_{t-1}\right], \mathcal{E}_t\right)\right) \tag{6.2}$$

$$\boldsymbol{O}_t = \sigma\left(\mathrm{MP}\left(\left[\hat{\boldsymbol{X}}_t^{(2)}||\boldsymbol{M}_t||\boldsymbol{H}_{t-1}\right], \mathcal{E}_t\right)\right) \tag{6.3}$$

$$\boldsymbol{C}_t = \tanh\left(\mathrm{MP}\left(\left[\hat{\boldsymbol{X}}_t^{(2)}||\boldsymbol{M}_t||\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1}\right], \mathcal{E}_t\right)\right) \tag{6.4}$$

$$\boldsymbol{H}_t = \boldsymbol{O}_t \odot \boldsymbol{H}_{t-1} + (1 - \boldsymbol{O}_t) \odot \boldsymbol{C}_t \tag{6.5}$$

where $\boldsymbol{R}_t, \boldsymbol{O}_t$ are the outputs of reset and update gates, respectively, $\boldsymbol{H}_t$ is the hidden representation at time t, and $\hat{\boldsymbol{X}}_t^{(2)}$ is the output of the decoding block at the previous time-step (see next paragraph). The symbols $\odot$ and $||$ denote the Hadamard product and the concatenation operator, respectively. The initial representations $\boldsymbol{H}_0$ can either be initialized as a constant or with a learnable embedding. Note that the encoder is fed with predictions from the decoder block for the steps where input data are missing, as explained in the next subsection. By autoregressively carrying out the above computations, the input sequences get encoded in $\boldsymbol{H}_{t:t+T}$.

6.3.2 Spatial decoder

As a first decoding step, we generate one-step-ahead predictions from hidden representations $\boldsymbol{H}_t$ by means of a global linear readout

$$\widehat{\boldsymbol{Y}}_t^{(1)} = \boldsymbol{H}_{t-1}\boldsymbol{V}_h + \boldsymbol{b}_h, \tag{6.6}$$

where $\boldsymbol{V}_h \in \mathbb{R}^{d_h \times d_x}$ is a learnable weight matrix and $\boldsymbol{b}_h \in \mathbb{R}^{d_x}$ is a learnable bias vector. We then define the *filler* operator as

$$\textsc{Fill}(\boldsymbol{Y}_t) \doteq \boldsymbol{M}_t \odot \boldsymbol{X}_t + \overline{\boldsymbol{M}}_t \odot \boldsymbol{Y}_t; \tag{6.7}$$

intuitively, the filler operator replaces the missing values in the input $\boldsymbol{X}_t$ with the values at the same positions in $\boldsymbol{Y}_t$. By feeding $\widehat{\boldsymbol{Y}}_t^{(1)}$ to the filler operator, we get the *first-stage imputation* $\widehat{\boldsymbol{X}}_t^{(1)}$ such that the output is $\boldsymbol{X}_t$ with missing values replaced by the one-step-ahead predictions $\widehat{\boldsymbol{Y}}_t^{(1)}$. The resulting node-level predictions are then concatenated to the mask $\boldsymbol{M}_t$ and the hidden representation $\boldsymbol{H}_{t-1}$, and processed by a final MP block which computes for each node an *imputation representation* $\boldsymbol{s}_t^i$ as

$$\boldsymbol{s}_t^i = \textsc{Up}\left(\boldsymbol{h}_{t-1}^i, \underset{j \in \mathcal{N}_t(i)/\{i\}}{\textsc{Aggr}}\left\{\textsc{Msg}\left(\hat{\boldsymbol{x}}_t^{j,(1)}, \boldsymbol{h}_{t-1}^j, \boldsymbol{m}_t^j\right)\right\}\right). \tag{6.8}$$

Notice that, as previously highlighted, the imputation representations only depend on messages received from neighboring nodes and the representation at the previous step. In fact, by aggregating only messages from the one-hop neighborhood, the representations $\boldsymbol{s}_t^i$ are independent of the input features $\boldsymbol{x}_t^i$ of the i-th node itself. This architectural bias forces the model to learn how to reconstruct a target input by considering spatial dependencies, thus constraining the model to focus on observations at related time series. Tmputation representation $\boldsymbol{S}_t$ is then concatenated to hidden representation $\boldsymbol{H}_{t-1}$ and used to obtain second-stage imputations by using a second linear readout:

$$\widehat{\boldsymbol{Y}}_t^{(2)} = [\boldsymbol{S}_t || \boldsymbol{H}_{t-1}] \, \boldsymbol{V}_s + \boldsymbol{b}_s; \qquad \widehat{\boldsymbol{X}}_t^{(2)} = \textsc{Fill}\left(\widehat{\boldsymbol{Y}}_t^{(2)}\right) \tag{6.9}$$

Finally, we feed $\widehat{\boldsymbol{X}}_t^{(2)}$ as input to the MPGRU (Equation 6.2–6.5) to update hidden representations and proceed to process the input; parameters can be fitted by training the model to output accurate imputations at both the first and second stage (see Equation 6.11 for more details).

6.3.3 Bidirectional model

Extending GRIN to account for both forward and backward dynamics is straightforward and can be achieved by duplicating the architecture described in the two previous sections. The first module will process the sequence in the forward direction (from the beginning of the sequence towards its end), while the second will go through the input in the opposite direction. The final imputation is then obtained with an MLP aggregating representations extracted by the two modules:

$$\widehat{\boldsymbol{y}}_t^i = \text{MLP}\left(\boldsymbol{s}_t^{i,fwd}, \boldsymbol{h}_{t-1}^{i,fwd}, \boldsymbol{s}_t^{i,bwd}, \boldsymbol{h}_{t+1}^{i,bwd}\right), \tag{6.10}$$

where fwd and bwd denote the forward and backward modules, respectively. The final output can then be easily obtained as $\widehat{\boldsymbol{X}}_{t:t+T} = \textsc{Fill}(\widehat{\boldsymbol{Y}}_{t:t+T})$. Note that, by construction, our model can exploit all the available relevant spatiotemporal information, since the only value explicitly masked out for each node is $\boldsymbol{x}_t^i$. At the same time, it is important to realize that our model does not merely reconstruct the input as an autoencoder, but it is specifically tailored for the imputation task due to its inductive biases.

Optimization objective The model is trained end-to-end by minimizing the reconstruction error of all imputation stages in both directions by minimizing

$$\begin{aligned}\mathcal{L} = &\mathcal{L}\left(\boldsymbol{Y}_{t,t+T}, \boldsymbol{X}_{t,t+T}, \boldsymbol{M}_{t,t+T}\right)\\ &+ \mathcal{L}\left(\boldsymbol{Y}^{(1),fwd}_{t,t+T}, \boldsymbol{X}_{t,t+T}, \boldsymbol{M}_{t,t+T}\right) + \mathcal{L}\left(\boldsymbol{Y}^{(2),fwd}_{t,t+T}, \boldsymbol{X}_{t,t+T}, \boldsymbol{M}_{t,t+T}\right)\\ &+ \mathcal{L}\left(\boldsymbol{Y}^{(1),bwd}_{t,t+T}, \boldsymbol{X}_{t,t+T}, \boldsymbol{M}_{t,t+T}\right) + \mathcal{L}\left(\boldsymbol{Y}^{(2),bwd}_{t,t+}, \boldsymbol{X}_{t,t+T}, \boldsymbol{M}_{t,t+T}\right),\end{aligned} \tag{6.11}$$

where each $\mathcal{L}(\,\cdot\,)$ is of the form of Equation 6.1 and the reconstruction error function is computed in terms of MAE. Note that here, compared to Equation 6.1, we are using $\boldsymbol{X}_{t,t+T}$ and $\boldsymbol{M}_{t,t+T}$ instead of $\widetilde{\boldsymbol{X}}_{t,t+T}$ and $\overline{\boldsymbol{M}}_{t,t+T}$: first stage imputations are obtained as one-step-ahead forecasts, while the second stage reconstructions include a spatial interpolation step.

6.3.4 Discussion and limitations

As we show in Section 6.5, compared to state-of-the-art imputation baselines, GRIN offers higher flexibility and achieves better reconstruction accuracy on several scenarios. However, it might suffer from issues typical of autoregressive reconstruction models [Liu et al., 2019; Marisca et al., 2022]. The autoregressive approach to imputation essentially models distributions $p^i(\boldsymbol{x}^i_t|\mathcal{X}_{<t})$ and uses one-step-ahead forecasting as a surrogate objective to learn how to recover missing observations. A bidirectional architecture allows for similarly modeling $p^i(\boldsymbol{x}^i_t|\mathcal{X}_{>t})$. Additional components are needed to account for spatial information at each step, i.e., modeling $p^i(\boldsymbol{x}^i_t|\{\boldsymbol{x}^{j\neq i}_t, \dots\})$. Architectures like BRITS [Cao et al., 2018] and GRIN, follow this scheme, with different components dedicated to modeling dependencies backward and forward in time as well as across time series. While being effective in practice, autoregressive methods have multiple drawbacks. Besides the computational overhead of having separate components, these models accumulate errors as predictions are used to bootstrap reconstruction at subsequent steps. This problem is particularly severe with sparse observations, where error accumulation can cause the hidden state to drift away [Bengio et al., 2015]. Additionally, integrating information from different modules can be challenging and may lead to further compounding of errors and information bottlenecks (e.g., w.r.t. representations $\boldsymbol{H}_t$).

6.4 Spatiotemporal Point Imputation Network

To address the limitations of GRIN and related autoregressive methods, we introduce a graph-based attention architecture, named *Spatiotemporal Point Inference Network* (SPIN). SPIN directly tackles the problem of reconstructing missing data from sparse observations and is specifically designed to perform imputation in such a setting. The following sections provide a high-level description of the approach; we refer to Marisca et al. [2022] for a more detailed technical description. We start by modeling the problem of learning a reconstruction model from a different perspective and then introduce the operators at the core of the proposed method.

6.4.1 Model conceptualization

We assume to have available covariates $\boldsymbol{U}_t \in \mathbb{R}^{N \times d_q}$ acting as spatiotemporal positional encodings to localize a point in time and space (e.g., date/time features and geographic location). Covariates $\boldsymbol{u}_t^i$ are assumed available for each node at each time step; if these covariates are not available they can be learned as discussed in Section 6.4.3. To simplify the discussion, we assume the input graph to be static. We denote as *observed (source) set*

$$\mathcal{S}_{t:t+T} = \left\{ \left\langle \boldsymbol{x}_\tau^i, \boldsymbol{u}_\tau^i \right\rangle \mid m_\tau^i = 1, \tau \in [t, t+T) \right\} \tag{6.12}$$

the set of all observations, paired with their spatiotemporal coordinates. Conversely, we name *target set*

$$\mathcal{T}_{t:t+T} = \left\{ \boldsymbol{u}_\tau^i \mid m_\tau^i = 0, \tau \in [t, t+T) \right\} \tag{6.13}$$

the complement set collecting the coordinates of the discrete spatiotemporal points for which we want to reconstruct an observation. We refer to the set of observed and target points of the i-th node as $\mathcal{S}_{t:t+T}^i$ and $\mathcal{T}_{t:t+T}^i$, respectively. Then, for all $\boldsymbol{u}_\tau^i \in \mathcal{T}_{t:t+T}$, we aim at learning a model for

$$p^i \left(\boldsymbol{x}_\tau^i | \boldsymbol{u}_\tau^i, \mathcal{S}_{t:t+T}, \mathcal{E} \right). \tag{6.14}$$

In particular,given disjoint observed and target sets $\mathcal{S}_{t:t+T}$ and $\mathcal{T}_{t:t+T}$, SPIN is trained to obtain point estimates as

$$\hat{\boldsymbol{x}}_\tau^i = \mathcal{F}(\boldsymbol{u}_\tau^i, \mathcal{S}_{t:t+T}, \mathcal{E}; \theta) \tag{6.15}$$

for all discrete $\boldsymbol{u}_\tau^i \in \mathcal{T}_{t:t+T}$. SPIN learns a parameterized propagation process where each representation, corresponding to a specific node and time step, is

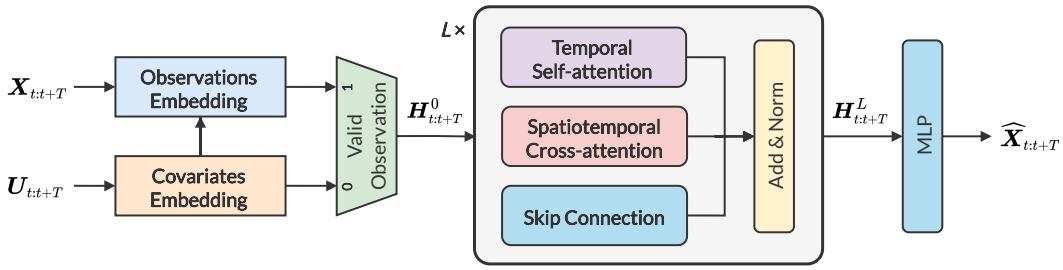


Figure 6.2. The architecture of SPIN. Observations $\boldsymbol{X}_{t:t+T}$ and spatiotemporal coordinates $\boldsymbol{U}_{t:t+T}$ are encoded into initial representations $\boldsymbol{H}^0_{t:t+T}$. Representations are processed by a stack of L sparse spatiotemporal attention blocks. Final imputations are obtained by feeding $\boldsymbol{H}^L_{t:t+T}$ into an MLP.

updated by aggregating information from all the available observations acquired at neighboring nodes weighted by input-dependent attention coefficients. Figure 6.2 shows an overview of the architecture. The next sections preset each component in detail and provide the motivations behind each design choice. We start by describing the propagation mechanism.

6.4.2 Sparse spatiotemporal attention

The core component of SPIN is a *sparse graph-based attention* block designed to propagate information among discrete points in time and space. Leveraging the attention mechanism, representations for at i-th node at each τ-th time step are learned by simultaneously aggregating information from (1) the observed set of i-th node $\mathcal{S}^i_{t:t+T}$ (2) the observed set $\mathcal{S}^j_{t:t+T}$ of its neighbors $j \in \mathcal{N}(i)$. Figure 6.3 shows a schematic representation of this procedure.

Input encoder Let $\boldsymbol{h}^{i,l}_\tau \in \mathbb{R}^{d_h}$ be the learned representation for the i-th node and time step τ at the l-th layer. The encoding is initialized as

$$\boldsymbol{h}^{i,(0)}_\tau = \begin{cases} \text{MLP}\left(\boldsymbol{u}^i_\tau\right) & \boldsymbol{u}^i_\tau \in \mathcal{T}_{t:t+T} \\ \text{MLP}\left(\boldsymbol{x}^i_\tau, \boldsymbol{u}^i_\tau\right) & \langle \boldsymbol{x}^i_\tau, \boldsymbol{u}^i_\tau \rangle \in \mathcal{S}_{t:t+T} \end{cases} \tag{6.16}$$

and then updated by joint temporal and spatiotemporal attention operations.

Spatiotemporal attention-based message passing The STMPs layers at the core of SPIN's architecture are based on a sparse attention mechanism. We adopt the terminology of Vaswani et al. [2017] and indicate as *query* the token

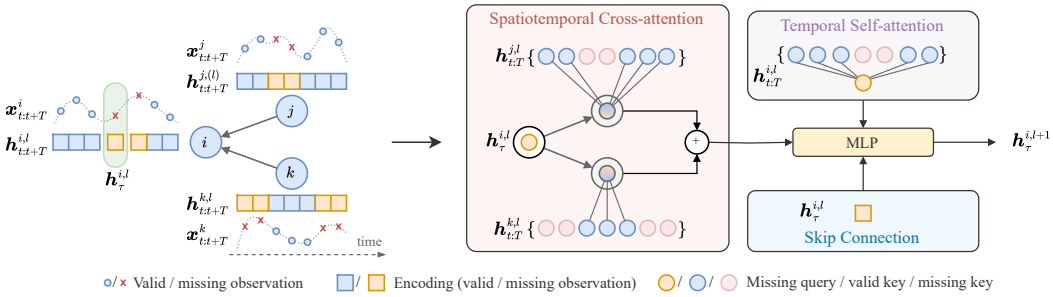


Figure 6.3. SPIN's sparse spatiotemporal attention block. The input time series (on the left) are processed by layers that update target representations (on the right). The figure shows the processing carried out to update the representation $\boldsymbol{h}_\tau^{i,l}$, i.e., the representation associated with the i-th node at time step τ.

for which we want to compute an updated representation, *key* a representation of the source tokens, and *value* the content representation of each token. By assuming that the value and key for the same token have the same representation, we introduce a Bahdanau-like [Bahdanau et al., 2015] masked attention operator

$$\boldsymbol{o}_{0:Q} = \textsc{MskAtt}\left(\boldsymbol{q}_{0:Q}, \boldsymbol{k}_{0:K}, \boldsymbol{m}_{0:K}\right) \tag{6.17}$$

operating as

$$\boldsymbol{r}_{qk} = \text{MLP}\left(\boldsymbol{q}_q, \boldsymbol{k}_k\right) \tag{6.18}$$

$$\alpha_{qk} = \frac{\boldsymbol{m}_k \exp(\boldsymbol{W}_\alpha \boldsymbol{r}_{qk})}{\sum_{k=0}^{K-1} \boldsymbol{m}_k \exp(\boldsymbol{W}_\alpha \boldsymbol{r}_{qk})} \tag{6.19}$$

$$\boldsymbol{o}_q = \sum_{k=0}^{K-1} \alpha_{qk} \boldsymbol{r}_{qk} \tag{6.20}$$

where $\boldsymbol{W}_\alpha^{1\times d_r}$ are learnable weights, $\boldsymbol{q}_q$ acts as query and $\boldsymbol{k}_k$ as key and value. Note that $\boldsymbol{m}_k \in \{0, 1\}$ allows for masking out representations of the corresponding value. STMP layers can then be implemented as

$$\boldsymbol{s}_\tau^{i,l} = \textsc{MskAtt}_1\left(\boldsymbol{h}_\tau^{i,l}, \boldsymbol{h}_{t:T}^{i,l}, \boldsymbol{m}_{t:T}^{i}\right), \qquad \text{Temporal self-attention} \tag{6.21}$$

$$\boldsymbol{c}_\tau^{i,l} = \underset{j\in\mathcal{N}(i)}{\textsc{Aggr}}\left\{\textsc{MskAtt}_2\left(\boldsymbol{h}_\tau^{i,l}, \boldsymbol{h}_{t:T}^{j,l}, \boldsymbol{m}_{t:T}^{j}\right)\right\}, \qquad \text{Spatial cross-attention} \tag{6.22}$$

$$\boldsymbol{h}_\tau^{i,l+1} = \text{MLP}\left(\boldsymbol{h}_\tau^{i,l}, \boldsymbol{s}_\tau^{i,l}, \boldsymbol{c}_\tau^{i,l}\right), \qquad \text{Update step} \tag{6.23}$$

where $\boldsymbol{s}_\tau^{i,l}$ is the output of a self-attention block while, $\boldsymbol{c}_\tau^{i,l}$ is the result of a cross-attention between the target and neighboring time series. The STMP

mechanism in Equation 6.21–6.23 implements a propagation mechanism allowing for information to flow without relying on any recurrent or autoregressive process.

Two-phase propagation Note that, due to masking, information does flow from observed to target set but not vice versa. Masking out tokens in the target set allows SPIN to propagate only representations corresponding to actual representations, avoiding bottlenecks. As a downside, this shuts down propagation paths going through points in the target set. This can be problematic when the input observations are extremely sparse. To avoid this, we assume that after a few layers, available information has already been partially propagated to locations with missing observations and we stop masking thereafter. In practice, we introduce a hyperparameter to control the number of layers with sparse propagation.

Decoder After L STMP layers, representations are mapped to predictions for all points in $\mathcal{T}_{t:t+T}$ with readout

$$\widehat{\mathcal{T}}_{t:t+T} = \{\hat{\boldsymbol{x}}_\tau^i = \mathrm{MLP}\big(\boldsymbol{h}_\tau^{i,l}\big) \mid \boldsymbol{u}_\tau^i \in \mathcal{T}_{t:t+T}\}. \tag{6.24}$$

SPIN can then be trained by considering a reconstruction loss (Equation 6.1) and by sampling a subset of the available observations to act as the target set for each sample in a training batch.

6.4.3 Spatiotemporal positional encoding

The above sections assumed that covariates $\boldsymbol{U}_{t:T}$ can act as positional encodings for each observation across time and space. These covariates can be directly extracted from available exogenous information (e.g., sensor location and date/-time features) or learned end-to-end jointly with the other model parameters. In practice, we use node-independent temporal encodings $\boldsymbol{w}_t \in \mathbb{R}^{d_w}$ combined with static node attributes $\boldsymbol{v}^i \in \mathbb{R}^{d_v}$ for the spatial components. Target covariates are then obtained as

$$\boldsymbol{u}_t^i = \mathrm{MLP}\left(\boldsymbol{w}_t, \boldsymbol{q}^i\right). \tag{6.25}$$

We obtain the temporal encodings $\boldsymbol{w}_t$ as sine and cosine transforms of the time step t w.r.t. seasonalities (e.g., day and/or week); while we use learnable node embeddings as node attributes (see Chapter 5). Several alternative methods to obtain the encodings can be considered [Kazemi et al., 2019; Dwivedi et al., 2022].

6.4.4 Discussion and limitations

While SPIN can account for the shortcomings of GRIN in processing sparse observations (see Section 6.5), it can result in severe computational setbacks in certain scenarios. The proposed spatiotemporal attention mechanism can be viewed as performing attention over the product graph between space and time dimensions – with some connections pruned w.r.t. unavailable data. Computing attention coefficient on such a graph has time and memory complexities that scale with $\mathcal{O}\left((N + |\mathcal{E}|)T^2\right)$. To enable applications of the proposed method to large graphs and long time horizons, we consider two different approaches. The straightforward approach consists of training the model by exploiting graph subsampling, using one of the many possible subsampling strategies from the literature (e.g., [Zeng et al., 2020]). In practice, at training time, we sample a k-hop subgraph centered on n target nodes and then compute the loss only w.r.t. these n nodes. Computational costs can then be reduced by acting on n and k.

Hierarchical attention An interesting and orthogonal approach to reduce computational costs is to rewire the attention mechanism to be hierarchical [Ravula et al., 2020]. In particular, a hierarchical masked attention block $\textsc{MskAtt-H}\left(\cdot\right)$ can be implemented by adding Z dummy tokens to act as hubs for propagation as

$$\boldsymbol{z}'_{0:Z} = \textsc{MskAtt}_1\left(\boldsymbol{z}_{0:Z}, \boldsymbol{k}_{0:K}, \boldsymbol{m}_{0:K}\right), \tag{6.26}$$

$$\boldsymbol{o}_{0:Q} = \textsc{MskAtt}_2\left(\boldsymbol{q}_{0:Q}, \boldsymbol{z}'_{0:Z}, \mathbf{1}_{0:Z}\right), \tag{6.27}$$

where $\mathbf{1}_{0:Z}$ simply indicates a sequence of ones. Replacing all the attention operators in Equation 6.21–6.23 with the above reduces the spatiotemporal attention complexity to $\mathcal{O}\left((N + |\mathcal{E}|)TZ\right)$ with $Z \ll T$, at the cost of introducing an information bottleneck. We initialize the representation of the attention hubs at layer $l = 0$ as trainable parameters. The performance of the hierarchical version of the model will be addressed in Section 6.5.

6.5 Empirical results

In the following, we evaluate the introduced imputation methods against baselines from the state of the art. We refer to the reference papers and Appendix F for more details and additional results [Cini et al., 2022b; Marisca et al., 2022; De Felice et al., 2024].

Datasets We run experiments on the **METR-LA**, **PEMS-BAY** and **AQI** datasets. We also a consider a smaller version of AQI (denoted by **AQI-36**) with only the 36 sensors scattered over the city of Beijing: a popular benchmark for imputation [Yi et al., 2016; Cao et al., 2018]. In METR-LA and PEMS-BAY, we inject missing data following two different policies: 1) *Block missing*, where we simulate at each time step a failure with 0.15% probability and sample its duration (in terms of steps) uniformly in the interval $[12, 48]$ (on top of this an additional 5% of the available data is randomly masked out); 2) *Point missing*, in which we simply randomly drop 25% of the available data. In all settings, the simulated missing data are masked out during training and are used as targets for evaluation. For the air quality datasets, we use the evaluation splits and masks of previous works, which are obtained by replicating the missing data distribution observed in certain months [Yi et al., 2016; Cao et al., 2018; Marisca et al., 2022].

Baselines We compare our methods against imputation methods from the deep learning literature: 1) **BRITS** [Cao et al., 2018], which is based on a bidirectional RNN; 2) **rGAIN** [Cini et al., 2022b], an adversarial approach based on GAIN [Yoon et al., 2018] and SSGAN [Miao et al., 2021]; 3) **STTr**, a standard Transformer [Vaswani et al., 2017] with attention being applied across both time and space; 4) **SAITS** [Du et al., 2023], a recent attention-based architecture based on diagonally-masked self-attention. Besides neural networks, we also consider simpler commonly used imputation methods: 5) **MEAN**, i.e., imputation using the node-level average; 6) **KNN**, i.e., imputation by averaging values of the $k = 10$ neighboring nodes with the highest weight in the adjacency matrix $\boldsymbol{W}_t$; 7) **MICE** [White et al., 2011], limiting the maximum number of iterations to 100 and the number of nearest features to 10; 8) Matrix Factorization (**MF**) with $rank = 10$ [Rubinsteyn and Feldman, 2016]; 9) **VAR**, i.e., a *Vector Autoregressive* one-step-ahead predictor [Hyndman et al., 2008]. Additionally, we also use as baseline 10) **MPGRU**, i.e., the unidirectional version of GRIN with the spatial decoder replaced by a simple MLP. For both MPGRU and GRIN we implement MP layers with diffusion convolutions [Atwood and Towsley, 2016] analogously to DCRNN [Li et al., 2018] and set the number of diffusion steps to $K = 2$.

6.5.1 In-sample and out-of-sample imputation

We start by considering in-sample and out-of-sample imputation on the air quality datasets. The objective of this experiment is to assess the performance

of GRIN against commonly used imputation methods. Table 6.1 shows the results of the experiment. In the in-sample settings, we compute metrics using as imputation the value obtained by averaging predictions over all the overlapping windows; in the out-of-sample settings, instead, we simply report results by averaging the reconstruction error over windows. Note that the matrix factorization baseline cannot be applied to the out-of-sample setting. GRIN largely outperforms other non-graph-based baselines in each scenario. In particular, in the latter case, GRIN decreases MAE w.r.t. the closest non-graph-based baseline by more than 20% in AQI. Performance improvements over MPGRU show the impact of the introduced designs. Interestingly, GRIN consistently outperforms BRITS in imputing missing values also for sensors corresponding to isolated (disconnected) nodes, i.e., nodes corresponding to stations more than 40 km away from any other station: this is empirical evidence of the positive regularizations brought by the global reconstruction approach implemented by GRIN. Our method achieves more accurate imputation also in the 36-dimensional dataset, where one could expect the graph representation to have a lower impact.

6.5.2 Imputation benchmarks

In the second set of experiments, we compare both GRIN and SPIN against state-of-the-art baselines including attention-based models. Additionally, we also report performance for the SPIN variant based on hierarchical attention (Section 6.4.4), denoted by **SPIN-H**. Here we focus on the out-of-sample scenario and consider both air quality and traffic datasets. Table 6.2 reports the results of the experiments. The introduced methods outperform the baselines in all considered setting. Unsurprisingly, SPIN's performance improvements over GRIN are more evident when entire blocks of data are missing, as in the AQI datasets and *Block missing* settings. Conversely, in the *Point missing* setting, SPIN performs on par with GRIN. With respect to STTr, i.e., the spatiotemporal Transformer baselines, SPIN and GRIN perform drastically better in most of the considered settings. In almost all cases, SPIN-H performs on par with the base mode (even better in some cases); these results qualify SPIN-H as a valid, less computationally demanding, alternative.

Sensitivity analysis We then compare the different methods in scenarios with increasing levels of data sparsity. In the first setting, the missing rate is progressively increased by associating with each observation an increasing probability of being removed; this corresponds to a sparser version of the *Point*

Table 6.1. Imputation results on the air datasets in the in-sample and out-of-sample settings. Performance averaged over 5 runs.

	Model	In-sample			Out-of-sample		
		MAE	MSE	MRE (%)	MAE	MSE	MRE (%)
AQI-36	Mean	53.48	4578.08	76.77	53.48	4578.08	76.77
	KNN	30.21	2892.31	43.36	30.21	2892.31	43.36
	MF	$30.54_{\pm 0.26}$	$2763.06_{\pm 63.35}$	$43.84_{\pm 0.38}$	–	–	–
	MICE	$29.89_{\pm 0.11}$	$2575.53_{\pm 07.67}$	$42.90_{\pm 0.15}$	$30.37_{\pm 0.09}$	$2594.06_{\pm 07.17}$	$43.59_{\pm 0.13}$
	VAR	$13.16_{\pm 0.21}$	$513.90_{\pm 12.39}$	$18.89_{\pm 0.31}$	$15.64_{\pm 0.08}$	$833.46_{\pm 13.85}$	$22.02_{\pm 0.11}$
	rGAIN	$12.23_{\pm 0.17}$	$393.76_{\pm 12.66}$	$17.55_{\pm 0.25}$	$15.37_{\pm 0.26}$	$641.92_{\pm 33.89}$	$21.63_{\pm 0.36}$
	BRITS	$12.24_{\pm 0.26}$	$495.94_{\pm 43.56}$	$17.57_{\pm 0.38}$	$14.50_{\pm 0.35}$	$662.36_{\pm 65.16}$	$20.41_{\pm 0.50}$
	MPGRU	$12.46_{\pm 0.35}$	$517.21_{\pm 41.02}$	$17.88_{\pm 0.50}$	$16.79_{\pm 0.52}$	$1103.04_{\pm 106.83}$	$23.63_{\pm 0.73}$
	GRIN	$\mathbf{10.51_{\pm 0.28}}$	$\mathbf{371.47_{\pm 17.38}}$	$\mathbf{15.09_{\pm 0.40}}$	$\mathbf{12.08_{\pm 0.47}}$	$\mathbf{523.14_{\pm 57.17}}$	$\mathbf{17.00_{\pm 0.67}}$
AQI	Mean	39.60	3231.04	59.25	39.60	3231.04	59.25
	KNN	34.10	3471.14	51.02	34.10	3471.14	51.02
	MF	$26.74_{\pm 0.24}$	$2021.44_{\pm 27.98}$	$40.01_{\pm 0.35}$	–	–	–
	MICE	$26.39_{\pm 0.13}$	$1872.53_{\pm 15.97}$	$39.49_{\pm 0.19}$	$26.98_{\pm 0.10}$	$1930.92_{\pm 10.08}$	$40.37_{\pm 0.15}$
	VAR	$18.13_{\pm 0.84}$	$918.68_{\pm 56.55}$	$27.13_{\pm 1.26}$	$22.95_{\pm 0.30}$	$1402.84_{\pm 52.63}$	$33.99_{\pm 0.44}$
	rGAIN	$17.69_{\pm 0.17}$	$861.66_{\pm 17.49}$	$26.48_{\pm 0.25}$	$21.78_{\pm 0.50}$	$1274.93_{\pm 60.28}$	$32.26_{\pm 0.75}$
	BRITS	$17.24_{\pm 0.13}$	$924.34_{\pm 18.26}$	$25.79_{\pm 0.20}$	$20.21_{\pm 0.22}$	$1157.89_{\pm 25.66}$	$29.94_{\pm 0.33}$
	MPGRU	$15.80_{\pm 0.05}$	$816.39_{\pm 05.99}$	$23.63_{\pm 0.08}$	$18.76_{\pm 0.11}$	$1194.35_{\pm 15.23}$	$27.79_{\pm 0.16}$
	GRIN	$\mathbf{13.10_{\pm 0.08}}$	$\mathbf{615.80_{\pm 10.09}}$	$\mathbf{19.60_{\pm 0.11}}$	$\mathbf{14.73_{\pm 0.15}}$	$\mathbf{775.91_{\pm 28.49}}$	$\mathbf{21.82_{\pm 0.23}}$

missing scenario of the previous experiment. In the second case, we instead operate in the *Block missing* setting by increasing the probability p_f of a failure at each step, i.e., the probability for each sensor of going offline for a random number $s \in [12, 36]$ of future (consecutive) time steps. In practice, we test the models on the same test split of the previous experiment but change the evaluation masks. Note that higher missing rates and failure probabilities correspond to longer blocks of contiguous missing values. In the *Block missing* case, failure probabilities $p_f 5\%$, $p_f = 10\%$, and $p_f = 15\%$ correspond to a missing rate of $\approx$ 70-75%, $\approx$ 90-92%, and $\approx$ 96-97%, respectively. We use the models trained for the experiments shown in Table 6.2; in particular, for the traffic datasets, we use models trained on the *Point missing* setting. Table 6.3 and Table 6.4 show the results of the experiment. Both GRIN and SPIN outperform the baselines; however, SPIN shows to be much more robust to changes in the missing data distribution. Notably, compared to GRIN, the performance of SPIN deteriorates at a much slower rate as the sparsity of the data increases.

Table 6.2. Reconstruction MAE averaged over 5 independent runs.

Model	Block missing		Point missing		Simulated failures	
	PEMS-BAY	METR-LA	PEMS-BAY	METR-LA	AQI-36	AQI
Mean	$5.46_{\pm.00}$	$7.48_{\pm.00}$	$5.42_{\pm.00}$	$7.56_{\pm.00}$	$53.48_{\pm.00}$	$39.60_{\pm.00}$
KNN	$4.30_{\pm.00}$	$7.79_{\pm.00}$	$4.30_{\pm.00}$	$7.88_{\pm.00}$	$30.21_{\pm.00}$	$34.10_{\pm.00}$
MICE	$2.94_{\pm.02}$	$4.22_{\pm.05}$	$3.09_{\pm.02}$	$4.42_{\pm.07}$	$30.37_{\pm.09}$	$26.98_{\pm.10}$
VAR	$2.09_{\pm.10}$	$3.11_{\pm.08}$	$1.30_{\pm.00}$	$2.69_{\pm.00}$	$15.64_{\pm.08}$	$22.95_{\pm.30}$
rGAIN	$2.18_{\pm.01}$	$2.90_{\pm.01}$	$1.88_{\pm.02}$	$2.83_{\pm.01}$	$15.37_{\pm.26}$	$21.78_{\pm.50}$
BRITS	$1.70_{\pm.01}$	$2.34_{\pm.01}$	$1.47_{\pm.00}$	$2.34_{\pm.00}$	$14.50_{\pm.35}$	$20.21_{\pm.22}$
SAITS	$1.56_{\pm.01}$	$2.30_{\pm.01}$	$1.40_{\pm.03}$	$2.26_{\pm.00}$	$18.16_{\pm.42}$	$21.33_{\pm.15}$
STTr	$1.70_{\pm.02}$	$3.54_{\pm.00}$	$0.74_{\pm.00}$	$2.16_{\pm.00}$	$11.98_{\pm.53}$	$18.11_{\pm.25}$
GRIN	$1.14_{\pm.01}$	$2.03_{\pm.00}$	$\mathbf{0.67}_{\pm.00}$	$\mathbf{1.91}_{\pm.00}$	$12.08_{\pm.47}$	$14.73_{\pm.15}$
SPIN	$\mathbf{1.06}_{\pm.02}$	$\mathbf{1.98}_{\pm.01}$	$0.70_{\pm.01}$	$\mathbf{1.90}_{\pm.01}$	$11.77_{\pm.54}$	$\mathbf{13.92}_{\pm.15}$
SPIN-H	$\mathbf{1.05}_{\pm.01}$	$2.05_{\pm.02}$	$0.73_{\pm.01}$	$1.96_{\pm.03}$	$\mathbf{10.89}_{\pm.27}$	$14.41_{\pm.13}$

6.5.3 Virtual sensing

As a final experiment, we also provide a quantitative and qualitative assessment of GRIN in virtual sensing (see related discussion in Section 6.1.1). The idea is to simulate the presence of a sensor by adding a node with no available data and, then, let the model reconstruct the corresponding time series. Note that for the approach to work, several assumptions are needed: 1) we have to assume that the physical quantity being monitored can be completely reconstructed from observations at neighboring sensors; 2) we should assume a high degree of homogeneity of sensors (e.g., in the case of air quality stations we should assume that sensors are placed at the same height) or that the features characterizing each neighboring sensor (e.g., placement) are available to the model. In this context, it is worth noting that, due to the inductive biases embedded in the model, GRIN performs reconstruction not only by minimizing reconstruction error at the single node, but by regularizing the reconstructed value for imputation at neighboring sensors (as the model autoregressively relies on its predictions). The objective of this experiment is to show the flexibility and application potential of the proposed methodology. As discussed in Section 6.1.1, GDL methods explicitly targeting virtual sensing applications exist [Wu et al., 2021b; Zheng et al., 2023; Xu et al., 2023] and are object of active research. A comprehensive treatment of the topic is however out of scope here; we refer to [De Felice et al., 2024] for a recent example of a comprehensive GDL framework for virtual sensing.

Table 6.3. Performance (MAE) with increasing data sparsity in the *Point missing* setting (averaged over 5 evaluation masks).

Model	METR-LA			PEMS-BAY			AQI		
	Missing rate			Missing rate			Missing rate		
	50 %	75 %	95 %	50 %	75 %	95 %	50 %	75 %	95 %
BRITS	$2.52_{\pm.00}$	$3.02_{\pm.00}$	$5.19_{\pm.02}$	$1.55_{\pm.00}$	$2.17_{\pm.00}$	$3.91_{\pm.02}$	$14.90_{\pm.03}$	$18.29_{\pm.03}$	$29.83_{\pm.07}$
SAITS	$2.48_{\pm.00}$	$3.74_{\pm.01}$	$6.72_{\pm.01}$	$1.50_{\pm.00}$	$2.96_{\pm.01}$	$7.40_{\pm.01}$	$15.36_{\pm.02}$	$20.64_{\pm.05}$	$34.57_{\pm.05}$
STTr	$2.31_{\pm.00}$	$2.71_{\pm.00}$	$5.13_{\pm.01}$	$0.85_{\pm.00}$	$1.13_{\pm.00}$	$2.70_{\pm.01}$	$9.11_{\pm.02}$	$12.56_{\pm.05}$	$25.65_{\pm.11}$
GRIN	$2.05_{\pm.00}$	$2.39_{\pm.00}$	$4.08_{\pm.02}$	$\mathbf{0.79}_{\pm.00}$	$1.09_{\pm.00}$	$2.70_{\pm.01}$	$8.43_{\pm.01}$	$10.97_{\pm.02}$	$20.38_{\pm.10}$
SPIN	$2.02_{\pm.00}$	$2.24_{\pm.00}$	$2.89_{\pm.01}$	$\mathbf{0.79}_{\pm.00}$	$1.00_{\pm.00}$	$1.71_{\pm.00}$	$\mathbf{8.15}_{\pm.01}$	$\mathbf{9.96}_{\pm.02}$	$\mathbf{15.51}_{\pm.08}$
SPIN-H	$\mathbf{2.01}_{\pm.00}$	$\mathbf{2.20}_{\pm.00}$	$\mathbf{2.82}_{\pm.00}$	$\mathbf{0.79}_{\pm.00}$	$\mathbf{0.97}_{\pm.00}$	$\mathbf{1.68}_{\pm.00}$	$8.67_{\pm.02}$	$10.27_{\pm.02}$	$15.75_{\pm.07}$

Table 6.4. Performance (MAE) with an increasing number of simulated failures in the *Block missing* setting (averaged over 5 evaluation masks).

Model	METR-LA			PEMS-BAY			AQI		
	Failure probability			Failure probability			Failure probability		
	5 %	10 %	15 %	5 %	10 %	15 %	5 %	10 %	15 %
BRITS	$5.87_{\pm.03}$	$7.26_{\pm.06}$	$8.29_{\pm.07}$	$4.14_{\pm.05}$	$5.41_{\pm.08}$	$5.84_{\pm.04}$	$24.09_{\pm.30}$	$31.90_{\pm.26}$	$37.62_{\pm.42}$
SAITS	$4.73_{\pm.07}$	$6.66_{\pm.05}$	$7.27_{\pm.03}$	$3.88_{\pm.09}$	$7.62_{\pm.21}$	$8.01_{\pm.11}$	$20.78_{\pm.30}$	$30.16_{\pm.39}$	$36.34_{\pm.33}$
STTr	$6.03_{\pm.04}$	$7.19_{\pm.05}$	$8.06_{\pm.05}$	$3.69_{\pm.06}$	$5.09_{\pm.05}$	$6.02_{\pm.04}$	$29.21_{\pm.33}$	$33.62_{\pm.16}$	$37.31_{\pm.14}$
GRIN	$3.05_{\pm.02}$	$4.52_{\pm.05}$	$5.82_{\pm.06}$	$2.26_{\pm.03}$	$3.45_{\pm.06}$	$4.35_{\pm.04}$	$15.62_{\pm.24}$	$22.08_{\pm.39}$	$29.03_{\pm.42}$
SPIN	$2.71_{\pm.02}$	$3.32_{\pm.02}$	$3.87_{\pm.05}$	$\mathbf{1.78}_{\pm.03}$	$\mathbf{2.15}_{\pm.03}$	$\mathbf{2.41}_{\pm.02}$	$\mathbf{14.29}_{\pm.24}$	$\mathbf{18.71}_{\pm.34}$	$\mathbf{24.34}_{\pm.46}$
SPIN-H	$\mathbf{2.64}_{\pm.02}$	$\mathbf{3.17}_{\pm.02}$	$\mathbf{3.61}_{\pm.04}$	$\mathbf{1.75}_{\pm.04}$	$\mathbf{2.16}_{\pm.03}$	$2.48_{\pm.02}$	$\mathbf{14.55}_{\pm.26}$	$\mathbf{19.37}_{\pm.36}$	$25.38_{\pm.37}$

We masked out observed values of the two nodes of AQI-36 with highest (station no. 1014) and lowest (no. 1031) connectivity, and trained GRIN on the remaining part of the data. Results in Figure 6.4 qualitatively show that GRIN can infer the trend and scale for unseen sensors. In terms of MAE, GRIN scores 11.74 for sensor 1014 and 20.00 for sensor 1031 (averages over 5 independent runs).

6.6 Discussion and future directions

GRIN and SPIN have been among the first GDL approaches for time series imputation. GDL approaches are now well-established and key tools for dealing with irregular spatiotemporal data with many applications in relevant domains.

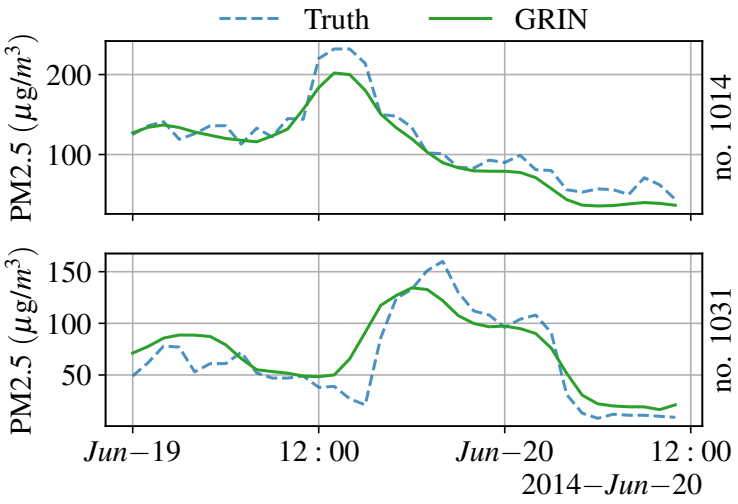


Figure 6.4. Reconstruction of observations from sensors removed from the training set. Plots show that GRIN might be used for virtual sensing.

Future directions Future research might explore the design of scalable imputation architectures (e.g., exploiting the approaches discussed in Chapter 8) and methods to quantify the uncertainty of the reconstruction. Additionally, it might be interesting to assess the theoretical conditions that would guarantee accurate imputation given given a characterization of the dependencies among time series. Future work might also explore methods to make forecasts directly from irregular data, bypassing the reconstruction step. Although research on the topic of graph-based methods in this context is limited [Zhong et al., 2021; Marisca et al., 2024], many of the operators used in imputation models can potentially be adapted to build forecasting architectures.

Chapter 7

Latent graph learning

This chapter deals with the problem of learning latent graph structures **(Challenge 3)**. As we discussed, standard STGNNs rely on a predefined pairwise relationships that come with the time series collection. The available relational information, however, can be inaccurate or inadequate for modeling the relevant dependencies. In other cases, relational information might be completely missing. Relational architectural biases can be exploited nonethless by learning a graph end-to-end with the forecasting architecture. In the following, we provide an in-depth discussion of possible methods to deal with the problem and identify a core challenge for applying such methods in practical applications, i.e., learning a graph while keeping MP operations sparse through the training procedure. For this purpose, within a probabilistic framework, we propose score-based gradient estimators to efficiently and effectively learn latent graph structures for time series forecasting. Section 7.1 introduces the topic and discusses different graph learning paradigms and related work. Section 7.2 provide preliminary concepts needed for introducing our approach. Section 7.3 then formally defines the problem setting in which we operate and discusses the core challenge in efficiently learning graph structures. The proposed score-based estimators and learning architecture are introduced in Section 7.4. We then address variance reduced estimators in Section 7.5 and 7.6.

Reference papers The content of the chapter is partly based on material from the following papers.

- Andrea Cini, Daniele Zambon, and Cesare Alippi. Sparse graph learning from spatiotemporal time series. *Journal of Machine Learning Research*, 24(242):1–36, 2023d

7.1 Latent graph learning

The interest in the graph learning problem arises from many practical 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 [Dong et al., 2016, 2019]. In GDL, graph learning modules are usually trained together with a GNN to maximize performance on a downstream task [Franceschi et al., 2019; Kipf et al., 2018; Kazi et al., 2022]. A widely adopted deep learning approach to the problem of modeling relationships consists of exploiting attention-like mechanisms to score the reciprocal salience of different spatial locations [Satorras et al., 2022; Rampášek et al., 2022]. However, despite their effectiveness, pure attention-based methods have two major downsides compared to 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 structure, introduced by the graph topology, as an inductive bias for the learning system. In our settings, sparse computation allows STGNNs to scale in terms of layers and number of time series that are possible to process. At the same time, sparse graphs constrain learned representations and mitigate over-fitting. Consequently, graph structure learning methods ought to learn meaningful dependencies while ideally allowing for sparse computations in the downstream MP layers.

Learning discrete graph structures Learning a graph, then, translates into learning a discrete relational structure among discrete entities. Learning discrete representations poses many challenges for deep learning methods [Niculae et al., 2023] due to issues in . The following subsections provide a critical overview of the most common approaches.

7.1.1 Learning an adjacency matrix

Most STGNNs rely on learning an adjacency matrix $\boldsymbol{A}$ as a function of a matrix of edge scores $\boldsymbol{\Phi} \in \mathbb{R}^{N \times N}$ as

$$\boldsymbol{A} = \xi\left(\boldsymbol{\Phi}\right) \quad \text{with} \quad \boldsymbol{\Phi} = \boldsymbol{\psi}, \tag{7.1}$$

where $\boldsymbol{\psi}$ are learnable parameters, and $\xi(\,\cdot\,)$ indicates a nonlinear activation function that can be used to induce sparsity in the resulting adjacency matrix, e.g., by thresholding the scores s.t. $\boldsymbol{A}[i,j] = 1$ if $\boldsymbol{\Phi}[i,j] > \varepsilon$ and 0 otherwise.

This approach results in exactly N^2 additional learning parameters; a cost that can be amortized, e.g., by factorizing the score matrix $\boldsymbol{\Phi}$ as

$$\boldsymbol{A} = \xi\left(\boldsymbol{\Phi}\right) \quad \text{with} \quad \boldsymbol{\Phi} = \boldsymbol{\Phi}_{src}\boldsymbol{\Phi}_{tgt}^{\top}, \tag{7.2}$$

where $\boldsymbol{\Phi}_{src}, \boldsymbol{\Phi}_{tgt} \in \mathbb{R}^{N \times d_z}$ are can node embeddings obtained, e.g., as tables of learnable parameters, which results in $\mathcal{O}(Nd_z)$ parameters. Such factorization approach has been pioneered in the context of STGNNs by the already cited GraphWaveNet architecture [Wu et al., 2019], where $\xi(\,\cdot\,)$ is implemented by a ReLU followed by a row-wise softmax activation. Several other methods have followed this direction which is quite flexible [Bai et al., 2020; Oreshkin et al., 2021; Liu et al., 2022]. To make the number of parameters independent from the number of nodes and condition the embeddings on the input window, scores can be computed similarly to attention coefficients as

$$\boldsymbol{A}_t = \xi\left(\boldsymbol{\Phi}_t\right) \quad \text{with} \quad \boldsymbol{\Phi}_t = \left(\boldsymbol{H}_t\boldsymbol{W}_{src}\right)\left(\boldsymbol{H}_t\boldsymbol{W}_{tgt}\right)^{\top}, \tag{7.3}$$

$$\boldsymbol{z}_t^i = \textsc{SeqEnc}\left(\boldsymbol{x}_{t-W:t}^i, \boldsymbol{u}_{t-W:t}^i, \boldsymbol{v}^i; \boldsymbol{\psi}\right), \tag{7.4}$$

where $\boldsymbol{W}_{src}, \boldsymbol{W}_{tgt} \in \mathbb{R}^{d_h \times d_z}$ are learnable weight matrices and $\textsc{SeqEnc}(\cdot)$ indicates a generic sequence encoder (e.g., an RNN) with learnable parameters $\boldsymbol{\psi}$. Dynamic edge scores can also be computed as a nonlinear function of source and target node representations $\boldsymbol{h}_t^i$ and $\boldsymbol{h}_t^j$ (similarly to Bahdanau attention attention scores [Bahdanau et al., 2015]), e.g, as

$$\boldsymbol{A}_t = \xi\left(\boldsymbol{\Phi}_t\right) \quad \text{with} \quad \boldsymbol{\Phi}_t[i,j] = \mathrm{MLP}(\boldsymbol{h}_t^i, \boldsymbol{h}_t^j). \tag{7.5}$$

Drawbacks of the direct approach The drawback of directly learning an adjacency matrix is that related methods often result in a dense $\boldsymbol{A}$ matrix which makes any subsequent MP operation scale with $\mathcal{O}(N^2)$ rather than with $\mathcal{O}(|\mathcal{E}|)$[1]. Among the existing methods, 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. Zhang et al. [2022] follow a different approach based on the idea of sparsifying the learned graph by thresholding the average of learned attention scores. However, making $\boldsymbol{A}$ sparse by using hard thresholds results in sparse gradients which may prevent the associated scores from being updated during training. A different approach consists of adopting a probabilistic perspective.

[1]Note that this is on top of the additional computation required to compute edge scores.

7.1.2 Learning distributions over graphs

Instead of directly learning a graph, probabilistic methods learn a probability distribution over graphs $\boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A})$ such that graphs sampled from $\boldsymbol{p}_{\boldsymbol{\Phi}}$ maximize the performance at task. The probabilistic approach allows for the embedding of priors directly into $\boldsymbol{p}_{\boldsymbol{\psi}}$, enabling the learning of sparse graphs as realizations of a discrete probability distribution. For example, one can consider graph distributions $\boldsymbol{p}_{\boldsymbol{\psi}}$ such that

$$\boldsymbol{A}_t \sim p_{\boldsymbol{\psi}_t}(\boldsymbol{A}_t), \tag{7.6}$$

where $\boldsymbol{p}_{\boldsymbol{\psi}}$ is parameterized by edge scores $\boldsymbol{\Phi}_t$ obtained by adopting any of the parameterizations discussed in the previous paragraph. For example, the graph distribution can be, e.g., implemented by considering a Bernoulli variable associated with each edge as

$$\boldsymbol{A}_t[i,j] \sim \text{Bernoulli}(\sigma(\boldsymbol{\Phi}_t[i,j])), \tag{7.7}$$

or by considering more complex distributions such as top-K samplers [Cini et al., 2023d; Kazi et al., 2022; Paulus et al., 2020; Ahmed et al., 2023] (see Section 7.4.2). As an example, consider a cost function $\delta_t(\cdot)$ (e.g., the forecasting accuracy) associated with each time step t and dependent on the inferred graph. The core challenge in learning $\boldsymbol{p}_{\boldsymbol{\psi}}$ is to estimate

$$\nabla_{\boldsymbol{\psi}} \mathbb{E}_{\boldsymbol{A} \sim p_{\boldsymbol{\psi}}} \left[\delta_t(\boldsymbol{A}) \right], \tag{7.8}$$

i.e., the gradient of the expected cost $\delta_t(\boldsymbol{A})$ with respect to the distributional parameters $\boldsymbol{\psi}$ under the sampling of $\boldsymbol{A}$ from $\boldsymbol{p}_{\boldsymbol{\psi}}$.

Path-wise and straight-through estimators Among existing probabilistic methods, Kipf et al. [2018] introduce NRI, a latent variable model for predicting the interactions of physical objects by learning discrete edge attributes of a fully connected graph. Shang and Chen [2021] introduce GTS by simplifying the NRI module by considering only binary relationships and integrates the graph inference module in a recurrent STGNN [Li et al., 2018]. To learn $\boldsymbol{p}_{\boldsymbol{\psi}}$ and estimate the gradient in Equation 7.8, both NRI and GTS exploit path-wise gradient estimators [Glasserman and Ho, 1991; Kingma and Welling, 2013] based on the categorical *Gumbel parametrization trick* [Maddison et al., 2017; Jang et al., 2017]. In particular, they reparametrize the sampling of $\boldsymbol{A}$ from $\boldsymbol{p}_{\boldsymbol{\psi}}$ as $\boldsymbol{A} = g(\varepsilon, \boldsymbol{\psi})$, with deterministic function g decoupling parameters $\boldsymbol{\psi}$ from the random component $\varepsilon \sim p_0$. In practice, these methods rely on

continuous relaxations of discrete distributions and, then, can suffer from the same computational setbacks of the methods discussed in Section 7.1.1. Similar arguments can be made for methods based on *straight-through estimators* [Bengio et al., 2013] that keep discrete representations sparse in the forward pass but would still require to compute messages associated with each potential edge for backpropagation.

Score-based sparse graph learning Differently from previous work, in the following we adopt the framework of *score-function* (SF) gradient estimators [Rubinstein, 1969; Glynn, 1990; Williams, 1992] by relying on the rewriting of Equation 7.8 as

$$\nabla_{\boldsymbol{\psi}} \mathbb{E}_{\boldsymbol{A} \sim \boldsymbol{p}_{\boldsymbol{\psi}}}[\delta_t(\boldsymbol{A})] = \mathbb{E}_{\boldsymbol{A} \sim \boldsymbol{p}_{\boldsymbol{\psi}}}[\delta_t(\boldsymbol{A}) \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A})] \tag{7.9}$$

which, as detailed in Section 7.4.1, will allow 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 MP network). In particular, as anticipated, the SF approach allows for obtaining unbiased Monte Carlo (MC) estimates where the gradient is computed only w.r.t. the likelihood of the graph being sampled and any downstream processing is treated like a black box. Within our context, we show that score-based estimators are particularly appealing as, differently from path-wise alternatives, they do not compute derivatives w.r.t. the MP procedure and, as such, allow for sparse computations at training time. The drawback of standard score-based estimators is, however, their inherently high variance. To address this issue, we will introduce variance-reduced estimators (Section 7.5) for specific graph distributions by exploiting control covariates and a surrogate training objective (Section 7.6), keeping the computation sparse.

7.1.3 Related work

Besides the already discussed forecasting applications, the problem of latent graph learning and relational inference has been the subject of extensive research. There have been several follow-ups to the NRI model, e.g., refinements of the MP architecture Chen et al. [2021c], or related to modeling dynamic [Graber and Schwing, 2020; Gong et al., 2021] and heterogeneous [Webb et al., 2019] dependencies. Alet et al. [2019] propose a meta-learning learning approach to infer graph structures at test time after training on data where ground truth graphs are available. Latent graph learning has also been studied in the context

of static graphs. Notably, Franceschi et al. [2019] propose a bi-level optimization routine to learn graphs based on the straight-through estimator. Kazi et al. [2022] uses the Gumbel-Top-K trick [Kool et al., 2019] to sample K-nearest-neighbors graphs and learn edge scores by using a heuristic that increases the likelihood of sampling edges contributing to correct classifications. This approach has been extended to learn higher-order structures as well [Battiloro et al., 2024]. More recently, Wren et al. [2022] learn DAGs end-to-end by exploiting implicit maximum likelihood estimation [Niepert et al., 2021]. As already mentioned, the problem has also been studied from the graph signal processing perspective [Dong et al., 2019], e.g., assuming that smoothness of the node features w.r.t. a learned graph Laplacian [Dong et al., 2016]. Related to the graph learning problem, *graph rewiring* consists of adding and/or removing edges of the input (predefined) graph to improve performance at task [Topping et al., 2022; Di Giovanni et al., 2023; Gutteridge et al., 2023; Barbero et al., 2024]. In particular, probabilistic methods have been recently applied to graph rewiring, e.g., by estimating distributions over edges that should dropped or added [Qian et al., 2024a] or to connect existing nodes to hubs that enable long-distance communication [Qian et al., 2024b]. Similarly, Errica et al. [2023] uses a probabilistic approach to design GNNs with adaptive receptive fields.

Along with methods explicitly targeting graphs, 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., relying 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 distribute'ion 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 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. Several structure learning methods rely on methods to sample *subsets* Kool et al. [2019]. In particular, Xie and Ermon [2019] introduce a continuous relaxation and a

reparametrization trick for subset sampling. Conversely, Ahmed et al. [2023] introduces gradient estimators based on the straight-through approach and a parametrization of the distribution that allows for efficient computation of the exact marginal distribution. We refer to Mohamed et al. [2020] and Niculae et al. [2023] for an in-depth discussion of the topic. None of these methods specifically target graph distributions, nor consider sparsity of the downstream computations (at training time) as a requirement.

7.2 Preliminaries

The section introduces some preliminary concepts and provides the reference settings to support the theoretical and technical derivations presented in the next sections.

7.2.1 Reference settings

We consider the problem settings introduced in Chapter 2 and indicate the set of N entities (sensors) generating the data as $\mathcal{S} = \{1, 2, \dots, N\}$. We focus on modeling binary relationships adjeciency matrices $\boldsymbol{A} \in \{0, 1\}^{N \times N}$, i.e., we do not account for possible weights associated to each edge. Extensions of the following to learning categorical or continuous edge attributes are possible (see Section 7.8), but out of scope. Dynamic relationships can be modeled by considering dynamic adjacency matrices $\boldsymbol{A}_t$; in the following, we omit the temporal index as the methodology can be applied to learn both static and temporal dependencies by exploiting the parameterizations anticipated in Section 7.1 and further discussed in Section 7.4.2. Furthermore, to keep the notation compact, we use interchangeably subscripts and square brackets for indexing matrices, i.e, $\boldsymbol{A}_{ij} \doteq \boldsymbol{A}[i, j]$ indicate the entry of matrix $\boldsymbol{A}$ corresponding to the i-th row and j-th column. Although we focus on multi-step-ahead time series forecasting, the following can be extended to any other downstream task (e.g., time series regression and/or classification) at both the graph and node level.

7.2.2 Mean adjacency matrices

In this section, we provide some tools to deal with probability distributions over graphs. Notably, many well-established probabilistic and statistical tools cannot directly be applied to graphs given they are not embedded in an Euclidean space. An example is the notion of "expected" graph that will be important

for the estimators introduced in Section 7.5 and whose definition needs to be extended. Here, we do so by following Fréchet [1948].

Given a generic random vector $\boldsymbol{x} \in \mathbb{R}_x^d$ characterized by probability density function $\boldsymbol{p}$, the expected value of $\boldsymbol{x}$ is denoted by

$$\mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{p}}[\boldsymbol{x}] = \int \boldsymbol{x}\, \boldsymbol{p}(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \tag{7.10}$$

is a weighted average over $\boldsymbol{x}$; we interchangeably use notation $\mathbb{E}_{\boldsymbol{p}}[\boldsymbol{x}]$ and $\mathbb{E}[\boldsymbol{x}]$ to indicate the same quantity. Notably, $\mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{p}}[\boldsymbol{x}]$ can be equivalently rewritten as

$$\mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{p}}[\boldsymbol{x}] = \underset{\boldsymbol{x}'\in\mathbb{R}_x^d}{\arg\min}\, \mathfrak{F}_2(\boldsymbol{x}'), \tag{7.11}$$

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

$$\mathfrak{F}_2(\boldsymbol{x}') \doteq \mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{p}}\left[\|\boldsymbol{x}' - \boldsymbol{x}\|_2^2\right] \tag{7.12}$$

associated with $\boldsymbol{p}$ and the squared Euclidean distance $\|\cdot\|_2^2$. Following Equation 7.11 and 7.12, we can derive a generalized definition of mean for non-Euclidean data by relying on alternative notions of distance. Similarly, it is possible to define the Fréchet sample mean as a generalization of the concept of sample mean [Jain, 2016].

Consider, then, a subset $\mathcal{A} \subseteq \{0,1\}^{N\times N}$ of adjacency matrices $\boldsymbol{A}$ over the node (sensor) set $\mathcal{S}$; each subset can be selected to satisfy specific constraints. For example, undirected graphs are characterized by the subset of symmetric matrices. Conversely, K-NN graphs correspond to the subset

$$\mathcal{A} = \left\{ \boldsymbol{A} \in \{0,1\}^{N\times N} : \sum_{j=1}^{N} \boldsymbol{A}_{ij} = k,\ \forall\, i \right\}. \tag{7.13}$$

Equipping $\mathcal{A}$ with a metric distance allows us to define a Fréchet function analogous to that of Equation 7.12. Here we consider the Hamming distance

$$H(\boldsymbol{A}, \boldsymbol{A}') \doteq \sum_{i,j=1}^{N} I(\boldsymbol{A}_{ij} \neq \boldsymbol{A}'_{ij}), \tag{7.14}$$

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.

For random matrices $\boldsymbol{A} \sim \boldsymbol{p}$ and for all $\boldsymbol{A}' \in \mathcal{A}$, we define the Fréchet function over $(\mathcal{A}, H)$ as

$$\mathfrak{F}_H(\boldsymbol{A}') \doteq \mathbb{E}_{\boldsymbol{A}\sim\boldsymbol{p}}\left[H(\boldsymbol{A}', \boldsymbol{A})\right]. \tag{7.15}$$

Then, we define as *Fréchet mean adjacency matrix* any matrix $\boldsymbol{A}^\mu$ such that

$$\boldsymbol{A}^\mu \in \arg\min_{\boldsymbol{A}'\in\mathcal{A}} \mathfrak{F}_H(\boldsymbol{A}'). \tag{7.16}$$

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

7.3 Problem formulation

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

7.3.1 Graph learning from correlated 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 multi-step ahead time series forecasting problem, the family of predictors $\mathcal{F}(\,\cdot\,;\boldsymbol{\theta})$, and parametric probability distribution $\boldsymbol{p}_{\boldsymbol{\psi}}$ over graphs such that

$$\widehat{\boldsymbol{X}}_{t:t+H} = \mathcal{F}\left(\mathcal{X}_{t-W:t}\,,\, \boldsymbol{A}_t; \boldsymbol{\theta}\right), \qquad \boldsymbol{A}_t \sim \boldsymbol{p}_{\boldsymbol{\psi}}\left(\boldsymbol{A}|\mathcal{X}_{t-W:t}\right), \tag{7.17}$$

where $\boldsymbol{\theta}$, $\boldsymbol{\psi}$ are the model parameters. The joint graph and model learning problem consists in jointly learning parameters $\boldsymbol{\theta}$, $\boldsymbol{\psi}$ by solving the optimization problem

$$\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\psi}} = \arg\min_{\boldsymbol{\theta},\boldsymbol{\psi}} \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}_t\left(\boldsymbol{\theta}, \boldsymbol{\psi}\right), \qquad \mathcal{L}_t\left(\boldsymbol{\theta}, \boldsymbol{\psi}\right) \doteq \mathbb{E}_{\boldsymbol{A}\sim\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\delta_t(\boldsymbol{A}_t; \boldsymbol{\theta})\right], \tag{7.18}$$

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

$$\delta_t(\boldsymbol{A}_t; \boldsymbol{\theta}) \doteq \ell\left(\mathcal{F}\left(\mathcal{X}_{t-W:t}, \boldsymbol{A}_t; \boldsymbol{\theta}\right), \boldsymbol{X}_{t:t+H}\right) \tag{7.19}$$

where $\ell(\,\cdot\,)$ can be any error function (see Section 2.2.1). Note that in Equation 7.17 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. In the case of static dependencies, it is enough to remove the conditioning on $\mathcal{X}_{t-W:t}$. As in the previous chapters, we consider predictors $\mathcal{F}(\,\cdot\,;\boldsymbol{\theta})$ implemented by STGNNs based on the MP framework and following either the TTS or T&S paradigm. $\mathcal{F}(\,\cdot\,;\boldsymbol{\theta})$ can also consider different graphs and graph distributions at each MP layer. Section 7.5 provides a thorough discussion of this setup.

Meaning of the graphs being learned Note that under this problem setting, the model family and the downstream task have an impact on the type of relationships being learned. For example, different models will yield different results that will depend also on the number of layers and the choice of MP operators. Ultimately, the learned graph distribution is the one that best explains the observed data given the the predictive model and the chosen family of graph distributions. Notably, different parametrizations of $\boldsymbol{p}_{\boldsymbol{\psi}}$ allow the practitioner for embedding different inductive biases (such as sparsity) as structural priors into the processing.

7.3.2 Core challenge

As anticipated in Section 7.1, minimizing the sum of expectations $\mathcal{L}_t\left(\boldsymbol{\theta},\boldsymbol{\psi}\right)$ over $t=1,\dots,T$, is challenging, as it involves estimating the gradients $\nabla_{\boldsymbol{\psi}}\mathcal{L}_t\left(\boldsymbol{\theta},\boldsymbol{\psi}\right)$ w.r.t. the parameters of the discrete distribution $\boldsymbol{p}_{\boldsymbol{\psi}}$ over (binary) adjacency matrices. The most common approach to estimate the gradient is to rely on the Monte Carlo method [Metropolis and Ulam, 1949], i.e., on sampling graphs from the distribution $\boldsymbol{p}_{\psi}$. However, estimating gradients w.r.t. the parameters of the distribution being sampled is not trivial and require ad-hoc techniques (see Section 7.4.1) which introduce trade-offs in the computational and sample complexity.

Stochastic computational graphs Sampling matrices (graphs) $\boldsymbol{A}\sim\boldsymbol{p}_{\boldsymbol{\psi}}$ 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).

Stochastic computational graphs and message passing In our setup, messages exchanged among nodes are stochastic themselves, i.e., sampling graph from $\boldsymbol{p}_\psi$ results in a stochastic message-passing graph (MPG). Having a stochastic MPG is problematic: the MP paradigm constrains the flow of spatial information, making the CG dependent on the MPG. Moreover, a stochastic 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. As already mentioned, computing gradients associated with each stochastic edge introduces additional challenges w.r.t. time and space complexity; further discussion and actionable directions are given in the next section.

7.4 Score-based graph learning from correlated time series

This section presents our approach to probabilistic graph learning. After introducing score-based gradient estimators (Section 7.4.1), we propose two learnable graph distributions (Section 7.4.2) and comment on their practical implementations (Section 7.4.3). The problem of controlling the variance of the estimators is discussed together with novel and principled variance reduction techniques tailored to graph-based architectures in Section 7.5. Finally, Section 7.6 provides a convenient rewriting of the gradient for L-layered MP architectures leading to a novel surrogate loss. Figure 7.1 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 $\boldsymbol{p}_\psi$; as the figure suggests, depending on the parametrization of $\boldsymbol{p}_\psi$, some components of $\boldsymbol{A}$ can be sampled independently. The bottom of the figure, instead, shows the predictive model $\mathcal{F}(\,\cdot\,;\boldsymbol{\theta})$ that, given the sampled graph and the input window, outputs the predictions used to estimate $\mathcal{L}_t(\boldsymbol{\theta}, \boldsymbol{\psi})$, whose gradient provides the learning signals.

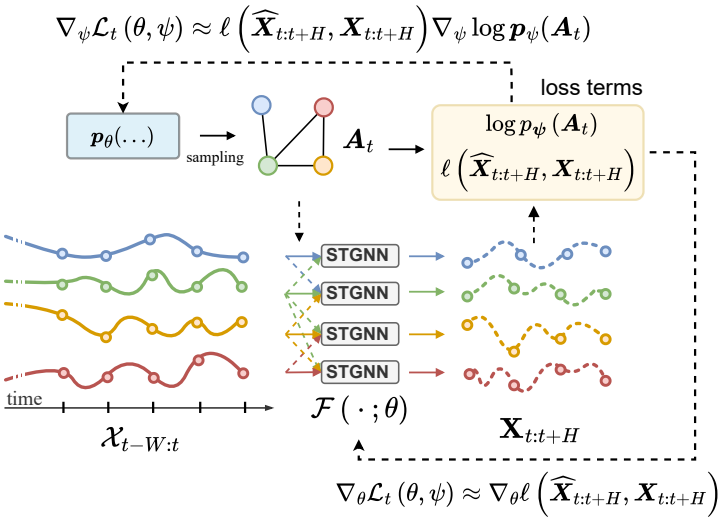


Figure 7.1. Overview of the learning architecture. The graph learning module samples a graph used to propagate information along the spatial dimension in $\mathcal{F}(\cdot;\boldsymbol{\theta})$; predictions and samples are used to compute costs and log-likelihoods. Gradient estimates are propagated back to the respective modules.

7.4.1 Estimating gradients for stochastic message-passing networks

SF estimators are based on the identity

$$\nabla_{\boldsymbol{\psi}} \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}[f(x)] = \nabla_{\boldsymbol{\psi}} \int f(x) \boldsymbol{p}_{\boldsymbol{\psi}}(x)\, dx = \int f(x) \nabla_{\boldsymbol{\psi}} \boldsymbol{p}_{\boldsymbol{\psi}}(x)\, dx \tag{7.20}$$

$$= \int f(x) \boldsymbol{p}_{\boldsymbol{\psi}}(x) \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(x)\, dx = \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}[f(x) \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(x)], \tag{7.21}$$

which holds – under mild assumptions[2] – for generic cost functions f and distributions $\boldsymbol{p}_{\boldsymbol{\psi}}$. Rewriting $\nabla_{\boldsymbol{\psi}} \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}[f(x)]$ in terms of the gradient of the *score function* $\log \boldsymbol{p}_{\boldsymbol{\psi}}(\,\cdot\,)$ allows for estimating the gradient through MC sampling and enables backpropagation through the computation of the score function. SF estimators are black-box optimization methods, i.e., they only require a pointwise *evaluation* of the *cost* $f(x)$ which does not necessarily need to be differentiable w.r.t. parameters $\boldsymbol{\psi}$. In our setup, assuming disjoint $\boldsymbol{\theta}$ and $\boldsymbol{\psi}$,

[2]The identity is valid as long as $\boldsymbol{p}_{\boldsymbol{\psi}}$ and f allow for the interchange of differentiation and integration in Equation 7.20; see L'Ecuyer [1995]; Mohamed et al. [2020].

Equation 7.21 becomes

$$\nabla_{\boldsymbol{\psi}} \mathcal{L}_t(\boldsymbol{\theta}, \boldsymbol{\psi}) = \nabla_{\boldsymbol{\psi}} \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}} \left[\delta_t(\boldsymbol{A}; \boldsymbol{\theta})\right] = \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}} \left[\delta_t(\boldsymbol{A}; \boldsymbol{\theta}) \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A})\right], \tag{7.22}$$

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 $\nabla_{\boldsymbol{\psi}} \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}} \left[\delta_t(\boldsymbol{A}; \boldsymbol{\theta})\right]$ by exploiting continuous relaxations of the discrete $\boldsymbol{p}_{\boldsymbol{\psi}}$, thus estimating the gradient by differentiating through all nodes of the stochastic CG. The cost of learning a graph with a path-wise estimator is making any subsequent MP operation scale with $\mathcal{O}(LN^2)$, instead of the $\mathcal{O}(L|\mathcal{E}|)$ 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 $\mathcal{O}(L|\mathcal{E}|)$ complexity.

7.4.2 Graph distributions, graphs sampling, and graphs likelihood

The distribution $\boldsymbol{p}_{\boldsymbol{\psi}}$ should be chosen to 1) efficiently sample graphs and evaluate their likelihood and 2) backpropagate the errors through the computation of the score (Equation 7.22) to parameters $\boldsymbol{\psi}$. In the following, we consider graph distributions s.t. each stochastic edge $j \to i$ is associated with a weight Φ_{ij}. The considered distributional parameters $\Phi \in \mathbb{R}^{N \times N}$ can be learned as a function of the learnable parameters $\boldsymbol{\psi}$ as discussed in Section 7.1.

7.4.2.1 Binary edge sampler

A simple graph distribution is the one introduced in Equation 7.7 and considers a Bernoulli random variable with parameter $\sigma(\Phi_{ij})$ associated with each potential edge $j \to i$. 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 $\boldsymbol{A}_{ij}$ of $\boldsymbol{A}$ can be sampled independently from the associated distribution since $\boldsymbol{A}_{ij} \sim \text{Bernoulli}(\sigma(\Phi_{ij}))$. Here, the sampling from $\boldsymbol{p}_{\boldsymbol{\psi}}$ 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 distribution, i.e,

$$\log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}) = \sum_{ij}^{N} \boldsymbol{A}_{ij} \log(\sigma(\Phi_{ij})) + (1 - \boldsymbol{A}_{ij}) \log(1 - \sigma(\Phi_{ij})). \tag{7.23}$$

Sparsity priors can then be added 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.

7.4.2.2 Subset neighborhood sampler

Encoding structural priors about the sparseness of the graphs directly into $\boldsymbol{p}_{\boldsymbol{\psi}}$ is often desirable and might allow – depending on the problem – to 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* $\Phi_{n,:}$, with $\Phi_{n,:}$ denoting the n-th row of matrix Φ. The probability of sampling neighborhood S_K is given, for each n node, by

$$\boldsymbol{p}_{\boldsymbol{\psi}}(S_K|n) = \sum_{\vec{S}_K \in \rho(S_K)} \boldsymbol{p}_{\boldsymbol{\psi}}(\vec{S}_K|n) = \sum_{\vec{S}_K \in \rho(S_K)} \prod_{j \in \vec{S}_K} \frac{\exp(\Phi_{nj})}{1 - \sum_{k<j} \exp(\Phi_{nk})}, \tag{7.24}$$

where $\vec{S}_K$ denotes an ordered sample without replacement and $\rho(S_K)$ is the set of all the permutations of S_K. Note that other parameterizations might be considered (e.g., see [Ahmed et al., 2023]).

Sampling Sampling can be done efficiently by exploiting the *Gumbel-top-K trick* [Kool et al., 2019]. Consider the row of parameters $\phi = \Phi_{n,:}$ and the associated random vector $[G_{\phi_1}, \dots, G_{\phi_N}]$ 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 7.24 shows that directly computing $\boldsymbol{p}_{\boldsymbol{\psi}}(S_K|n)$ 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, $\boldsymbol{p}_{\boldsymbol{\psi}}(S_K|n)$ 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

$$\underset{i \in B}{\textsc{LogSumExp}}(\phi_i) \doteq \log \left(\sum_{i \in B} \exp \phi_i \right), \tag{7.25}$$

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

$$G_{\phi_B} \doteq \max_{i \in B} G_{\phi_i} \sim \text{Gumbel}(\phi_B). \tag{7.26}$$

With a derivation analogous to that of Kool et al. [2020], Equation 7.24 can be conveniently rewritten by exploiting the property shown in Equation 7.26 as:

$$\boldsymbol{p}_{\boldsymbol{\psi}}(S_K|n) = \mathbb{P}\left(\min_{i \in S_K} G_{\phi_i} > \max_{i \in \mathcal{S} \setminus S_k} G_{\phi_i} \right) \tag{7.27}$$

$$= \mathbb{P}\left(G_{\phi_i} > G_{\phi_{\mathcal{S} \setminus S_k}}, \forall i \in S_K \right) \tag{7.28}$$

$$= \int_{-\infty}^{\infty} \prod_{i \in S_K} \left(1 - \text{cdf}_{\phi_i}(g)\right) \text{pdf}_{\phi_{\mathcal{S} \setminus S_k}}(g)\, dg \tag{7.29}$$

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

$$\boldsymbol{p}_{\boldsymbol{\psi}}(S_K|n) = \exp\left(\phi_{\mathcal{S} \setminus S_K} + c\right) \int_0^1 u^{\exp\left(\phi_{\mathcal{S} \setminus S_K} + c\right) - 1} \prod_{i \in S_k} \left(1 - u^{\exp(\phi_i + c)}\right) du, \tag{7.30}$$

where c is a conditioning constant. We then approximate the integral in Equation 7.30 by using the trapezoidal rule as

$$\log \boldsymbol{p}_{\boldsymbol{\psi}}(S_K|n) \approx \log(\Delta u) + \phi_{\mathcal{S}\setminus S_K} + c$$
$$+\underset{m=1,\dots,M-1}{\textsc{LogSumExp}}\left(\left(\exp\left(\phi_{\mathcal{S}\setminus S_K} + c\right) - 1\right)\log(u_m) + \sum_{i\in S_K}\log\left(1 - u_m^{\exp(\phi_i + c)}\right)\right), \tag{7.31}$$

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 7.31 provides, then, a differentiable numeric approximation of the SNS log-likelihood which can be used for backpropagation.

Adaptive number of neighbors 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. The problem can be solved by adding dummy nodes. 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 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 $\mathcal{O}(NK)$.

7.4.3 Learning the graph distribution $\boldsymbol{p}_{\boldsymbol{\psi}}$

For both BES and SNS, $\boldsymbol{p}_{\boldsymbol{\psi}}$ can be parametrized by following any of the strategies discussed in Section 7.1 to parameterize the score matrix $\boldsymbol{\Phi}$. Among the main available design choices, it is possible, for example, to associate a learnable parameter to each (edge) score Φ_{ij} by setting $\Phi = \boldsymbol{\psi}$. Similarly, one could reduce the number of parameters to estimate by factorizing the score matrix. Modeling dynamic graphs instead requires accounting for observations $\mathcal{X}_{t-W:t}$ at each considered time step t, e.g., by using a sequence modeling block to compute scores as a function of the input window. We refer to Section 7.1 for an in-depth discussion of the available alternatives and existing architectures.

7.5 Reducing the variance of the estimator

MC estimation is the most commonly used technique to approximate the gradient in Equation 7.22. Although the resulting 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 considerable 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 a viable to further improve the quality of the gradient estimator.

7.5.1 Control variates and baselines

The control variates method provides a variance reduction method for MC estimator of $\mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}[g(x)]$. It consists in introducing an auxiliary quantity $h(x)$ for which we know how to efficiently compute the expectation under the sampling distribution $\boldsymbol{p}_{\boldsymbol{\psi}}$ [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 ($\mathrm{Var}[\tilde{g}(x)] < \mathrm{Var}[g(x)]$). 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 $g(\boldsymbol{A}) \doteq \delta_t(\boldsymbol{A};\boldsymbol{\theta})\nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A})$ (Equation 7.22), we set $h(\boldsymbol{A}) \doteq \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A})$, for which $\mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}[h(\boldsymbol{A})] = 0$, and obtain

$$\nabla_{\boldsymbol{\psi}} \mathcal{L}_t\left(\boldsymbol{\theta}, \boldsymbol{\psi}\right) = \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\left(\delta_t(\boldsymbol{A};\boldsymbol{\theta}) - \beta\right) \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A})\right]. \tag{7.32}$$

This narrows the problem to finding appropriate values for baseline β. Since for any f_1, f_2, $\mathrm{Var}[f_1 + f_2] = \mathrm{Var}[f_1] + \mathrm{Var}[f_2] + 2\mathrm{Cov}[f_1, f_2]$, the optimal baseline β_* in Equation 7.32 is given by

$$\begin{aligned}\beta_* &\doteq \frac{\mathrm{Cov}[\delta_t(\boldsymbol{A};\boldsymbol{\theta})\nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}), \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A})]}{\mathrm{Var}_{\boldsymbol{p}_{\boldsymbol{\psi}}}[\nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A})]} \\ &= \frac{\mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}[\delta_t(\boldsymbol{A};\boldsymbol{\theta})(\nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}))^2]}{\mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}[(\nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}))^2]}.\end{aligned} \tag{7.33}$$

Unfortunately, finding the optimal β_* can be as hard as estimating the desired gradient in Equation 7.22; 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

$$\mathbb{E}_{\boldsymbol{p}_\psi}[\delta_t(\boldsymbol{A};\boldsymbol{\theta})(\nabla_\psi \log \boldsymbol{p}_\psi(\boldsymbol{A}))^2] \approx \mathbb{E}_{\boldsymbol{p}_\psi}[\delta_t(\boldsymbol{A};\boldsymbol{\theta})]\mathbb{E}_{\boldsymbol{p}_\psi}[(\nabla_\psi \log \boldsymbol{p}_\psi(\boldsymbol{A}))^2], \quad (7.34)$$

and obtain $\beta_* \approx \mathbb{E}_{\boldsymbol{p}_\psi}[\delta_t(\boldsymbol{A};\boldsymbol{\theta})]$. 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 $\mathbb{E}_{\boldsymbol{p}_\psi}[\delta_t(\boldsymbol{A};\boldsymbol{\theta})]$ would require the introduction of an additional estimator, we rely on a different approximation by moving the expectation inside the cost function and obtaining $\beta_* \approx \delta_t(\boldsymbol{\mu};\boldsymbol{\theta})$, where $\boldsymbol{\mu} = \mathbb{E}_{\boldsymbol{p}_\psi}[\boldsymbol{A}]$.

We recall that, in general, $\boldsymbol{\mu}$ is dense and its components are real numbers, therefore computing $\delta_t(\boldsymbol{\mu};\boldsymbol{\theta})$ 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 7.2.2. We then choose as $\hat{\beta}$ such that

$$\hat{\beta} \doteq \delta_t(\boldsymbol{A}^\mu;\boldsymbol{\theta}). \quad (7.35)$$

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 $\tilde{\delta}_t(\boldsymbol{A};\boldsymbol{\theta}) = \delta_t(\boldsymbol{A};\boldsymbol{\theta}) - \delta_t(\boldsymbol{A}^\mu;\boldsymbol{\theta})$; the modified cost is computed after each forward pass and used to update the parameters of $\boldsymbol{p}_\psi$. In the following, we derive analytical solutions for finding $\boldsymbol{A}^\mu$ for BES and SNS, respectively.

7.5.1.1 Baseline for BES

We start by recalling the notation from previous sections. Denote expectation $\mathbb{E}_{\boldsymbol{p}_\psi}[\boldsymbol{A}]$ 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 $\boldsymbol{p}_\psi$ associated with BES. The main result of the section is the following proposition which allows us to provide a baseline as

$$\hat{\beta}_{\text{BES}} \doteq \delta_t\left(\lfloor \sigma(\Phi) \rceil;\boldsymbol{\theta}\right), \quad (7.36)$$

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 $\boldsymbol{p}_{\boldsymbol{\psi}}$ and support $\mathcal{A}$. Then,

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

Lemma 1. The minimum of the Fréchet function $\mathfrak{F}_H$ can be expressed as

$$\min_{\boldsymbol{A} \in \mathcal{A}} \mathfrak{F}_H(\boldsymbol{A}) = \min_{\boldsymbol{A} \in \mathcal{A}} \sum_{i,j=1}^{N} \left(\boldsymbol{\mu}_{ij} - \boldsymbol{A}_{ij}\right)^2 . \tag{7.37}$$

To conclude the proof of Preposition 1, we observe that the minimum of Equation 7.37 is attained at $\boldsymbol{A}^{\mu} = \lfloor \boldsymbol{\mu} \rceil$, that is $\boldsymbol{A}^{\mu}_{ij} = 1$ for all $\boldsymbol{\mu}_{ij} > 1/2$ (or $\Phi > 0$), and 0 elsewhere. The proof of the Lemma 1 is deferred to Appendix G. □

7.5.1.2 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 $j \to i$ and $j' \to i$ are not independent. Nevertheless, the result remains intuitive:

$$\hat{\beta}_{\text{SNS}} \doteq \delta_t \left(\boldsymbol{A}^{\mu}; \boldsymbol{\theta}\right), \quad \text{with } \boldsymbol{A}^{\mu}_{ij} = I\left(\Phi_{ij} \in \text{top-K}\{\Phi_{i,:}\}\right), \ \forall\, i, j \in \mathcal{S}. \tag{7.38}$$

The proof that $\boldsymbol{A}^{\mu}$ is indeed the Fréchet mean for SNS follows Preposition 2. Recall that, for SNS, the support of $\boldsymbol{p}_{\boldsymbol{\psi}}$ is that of directed K-NN graphs in Equation 7.40, where the neighborhood of each node is sampled independently. Equation 7.38 is derived by considering a neighborhood of fixed size K; however, the analysis remains valid for the adaptive case discussed in Section 7.4.2.2.

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

$$\boldsymbol{\mu}_{n,i} = \boldsymbol{p}_{\boldsymbol{\psi}}(i \in S_K | n) = \sum_{S'_K : i \in S'_K} \boldsymbol{p}_{\boldsymbol{\psi}}(S'_K | n), \tag{7.39}$$

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 2.

Proposition 2. Consider an SNS distribution with support

$$\mathcal{A} = \left\{ \boldsymbol{A} \in \{0,1\}^{N\times N} : \sum\nolimits_{j=1}^{N} \boldsymbol{A}_{ij} = K,\ \forall\, i \right\}. \tag{7.40}$$

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

$$\boldsymbol{A}^\mu_{ij} = I\left(\Phi_{ij} \in \text{top-K}\{\Phi_{i,:}\}\right),\ \forall\, i,j \in \mathcal{S}. \tag{7.41}$$

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

$$\min_{\boldsymbol{A}\in\mathcal{A}} \mathfrak{F}_H\left(\boldsymbol{A}\right) = \min_{\boldsymbol{A}\in\mathcal{A}} \mathbb{E}_{\boldsymbol{A}'\sim\boldsymbol{p}_{\boldsymbol{\psi}}}\left[H(\boldsymbol{A},\boldsymbol{A}')\right]. \tag{7.42}$$

Start by rewriting the Fréchet function as

$$\begin{aligned}
\mathfrak{F}_H(\boldsymbol{A}) &= \mathbb{E}_{\boldsymbol{A}'\sim\boldsymbol{p}_{\boldsymbol{\psi}}}\left[H(\boldsymbol{A},\boldsymbol{A}')\right] && (7.43)\\
&= \mathbb{E}_{\boldsymbol{A}'\sim\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\sum_{n,i=1}^{N} \boldsymbol{A}_{n,i} - 2\boldsymbol{A}_{n,i}\boldsymbol{A}'_{n,i} + \boldsymbol{A}'_{n,i}\right] && (7.44)\\
&= \sum_{n,i=1}^{N} \boldsymbol{A}_{n,i} \underbrace{(1-2\boldsymbol{\mu}'_{n,i})}_{w_{n,i}} + \underbrace{\sum_{n,j=1}^{N} \boldsymbol{\mu}_{n,i}}_{c}. && (7.45)
\end{aligned}$$

where $\boldsymbol{\mu}_{n,i} = \boldsymbol{p}_{\boldsymbol{\psi}}\left(i \in S_K | n\right) = \boldsymbol{p}_{\boldsymbol{\psi}}\left(\boldsymbol{A}_{n,i} = 1\right)$ and c is a constant. The proof follows from Lemma 2.

Lemma 2. Let $\boldsymbol{p}_{\boldsymbol{\psi}}$ be an SNS distribution with associated log-probabilities Φ. Then $\forall n,i,j \in \mathcal{S}$

$$\boldsymbol{p}_{\boldsymbol{\psi}}\left(\boldsymbol{A}_{n,i} = 1\right) \geq \boldsymbol{p}_{\boldsymbol{\psi}}\left(\boldsymbol{A}_{n,j} = 1\right) \iff \Phi_{n,i} \geq \Phi_{n,j}. \tag{7.46}$$

The proof of Lemma 2 is provided in Appendix G. Following Equation 7.45, the optimization problem in Equation 7.42 becomes the linear program

$$\begin{aligned}
\text{minimize} & \sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}\boldsymbol{A}_{ij}\\
\text{s.t.} & \sum_{j=1}^{N} \boldsymbol{A}_{ij} = K;\\
& \boldsymbol{A}_{ij} \in \{0,1\} \qquad \forall i = 1,\ldots,N,
\end{aligned} \tag{7.47}$$

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

7.6 Layer-wise sampling and surrogate objective

As a final step, we can leverage on the structure of MP neural networks to rewrite the gradient $\nabla_{\boldsymbol{\psi}}\mathcal{L}_t(\boldsymbol{\theta}, \boldsymbol{\psi})$. 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 $\mathcal{F}(\,\cdot\,; \boldsymbol{\theta})$.

Proposition 3. Consider family of models $\mathcal{F}(\cdot, \boldsymbol{A}^{:L}; \boldsymbol{\theta})$ with exactly L message-passing layers propagating messages w.r.t. different adjacency matrices $\boldsymbol{A}^{(l)}$, $l = 1, \ldots, L$, sampled from $\boldsymbol{p}_{\boldsymbol{\psi}}$ (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

$$\nabla_{\boldsymbol{\psi}}\mathcal{L}_t(\boldsymbol{\theta}, \boldsymbol{\psi}) = \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\sum_{l=1}^{L-1} \delta_t(\boldsymbol{A}^{:L}; \boldsymbol{\theta}) \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}^{(l)}) + \sum_{i=1}^{N} \delta_t^i(\boldsymbol{A}^{:L}; \boldsymbol{\theta}) \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}^{(L)}_{i,:})\right], \tag{7.48}$$

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 3 holds for all parametrizations of $\boldsymbol{p}_{\boldsymbol{\psi}}$ 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.,

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

The following provides proof of Proposition 3 and presents a surrogate objective function inspired by Equation 7.48.

Proof. A proof can be derived by noticing the independence of $\delta_t^i(\boldsymbol{A}^{:L}; \boldsymbol{\theta})$ and $\boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}^{(L)}_{j,:})$ 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 $\boldsymbol{\theta}$. The proof follows:

$$\nabla_{\boldsymbol{\psi}} \mathcal{L}_t(\boldsymbol{\theta}, \boldsymbol{\psi}) = \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\delta_t \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}^{:L})\right] \tag{7.49}$$

$$= \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\sum_{l=1}^{L-1} \delta_t \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}^{(l)})\right] + \underbrace{\mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\delta_t \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}^{(L)})\right]}_{(*)}. \tag{7.50}$$

By considering the second term:

$$(*) = \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\delta_t \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}^{(L)})\right] \tag{7.51}$$

$$= \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\sum_{i=1}^{N} \delta_t^i \sum_{j=1}^{N} \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}_{j,:}^{(L)})\right] \tag{7.52}$$

$$= \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\sum_{i=1}^{N} \delta_t^i \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}_{i,:}^{(L)})\right] + \underbrace{\mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\sum_{i=1}^{N} \delta_t^i \sum_{j \neq i} \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}_{j,:}^{(L)})\right]}_{(**)}. \tag{7.53}$$

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

$$(**) = \sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\delta_t^i\right] \sum_{j \neq i} \underbrace{\mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}_{j,:}^{(L)})\right]}_{=0} = 0. \tag{7.54}$$

Putting everything together, we get Equation 7.48 and the proof is completed. □

7.6.1 Surrogate objective

Intuitively, the second term in Equation 7.48 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 3 motivates us in considering a similar surrogate approximate loss $\widehat{\mathcal{L}}_t(\boldsymbol{\theta}, \boldsymbol{\psi})$ for the case where we use a single $\boldsymbol{A}$ for all layers, i.e., we consider

$$\nabla_{\boldsymbol{\psi}} \widehat{\mathcal{L}}_t(\boldsymbol{\theta}, \boldsymbol{\psi}) = \mathbb{E}_{\boldsymbol{p}_{\boldsymbol{\psi}}}\left[\lambda \delta_t(\boldsymbol{A}; \boldsymbol{\theta}) \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}) + \sum\nolimits_{i=1}^{N} \delta_t^i(\boldsymbol{A}; \boldsymbol{\theta}) \nabla_{\boldsymbol{\psi}} \log \boldsymbol{p}_{\boldsymbol{\psi}}(\boldsymbol{A}_{i,:})\right], \tag{7.55}$$

as gradient to learn $\boldsymbol{p}_{\boldsymbol{\psi}}$. Equation 7.55 is developed from Equation 7.48 by considering a single sample $\boldsymbol{A} \sim \boldsymbol{p}_{\boldsymbol{\psi}}$ and introducing the hyperparameter λ. Note that, in this case, $\widehat{\mathcal{L}}_t(\boldsymbol{\theta}, \boldsymbol{\psi})$ is an approximation of the true objective with a reweighting of the contribution of each $\delta^i(\boldsymbol{A}; \boldsymbol{\theta})$. 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 7.7.

7.7 Empirical results

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. We refer to Appendix G and to the reference paper for additional details on the experimental setup [Cini et al., 2023d].

7.7.1 Datasets

As synthetic benchmark we use the original version of the GPVAR dataset introduce by Zambon and Alippi [2022] which consists of signals generated by recursively applying a polynomial Graph VAR filter [Isufi et al., 2019] and adding Gaussian noise at each time step. In particular, analogously to Zambon and Alippi [2022], we consider the data generating process

$$\boldsymbol{X}_t = \tanh\left(\sum_{l=0}^{L}\sum_{q=1}^{Q} \boldsymbol{\psi}_{l,q} \widetilde{\boldsymbol{A}}^l \boldsymbol{X}_{t-q}\right) + \eta_t \tag{7.56}$$

where $\widetilde{\boldsymbol{A}} = \boldsymbol{I} + \boldsymbol{A}$ (with $\boldsymbol{I}$ being the identity matrix), $\boldsymbol{\psi} \in \mathbb{R}^{(L+1)\times Q}$ denotes the model parameter and $\eta_t \sim \mathcal{N}(0, \boldsymbol{I})$ is a Gaussian noise vector. We use the community graph described in Chapter 4, but consider the configuration introduced in [Zambon and Alippi, 2022], which consists of 30 nodes. Model parameters, with $L = Q = 2$, are set as in previous works and used to generate a trajectory of $T = 30000$ steps. We use 70/10/20% data split for training, validation, and testing, respectively.

For what concerns benchmarks utilizing real-world data, we use the **AQI** dataset and traffic datasets (**METR-LA** and **PEMS-BAY**) as described in Chapter 4.

7.7.2 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.

7.7.2.1 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 7.2 shows the validation 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 7.2a and Figure 7.2b show results in the graph identification task for the vanilla gradient estimator derived from Equation 7.21 and for the surrogate objective from Equation 7.55, 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 $\boldsymbol{p}_{\boldsymbol{\psi}}$. Results allow us to make the following comments.

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 7.2b shows that the surrogate objective contributes to accelerating learning even further for all considered methods.

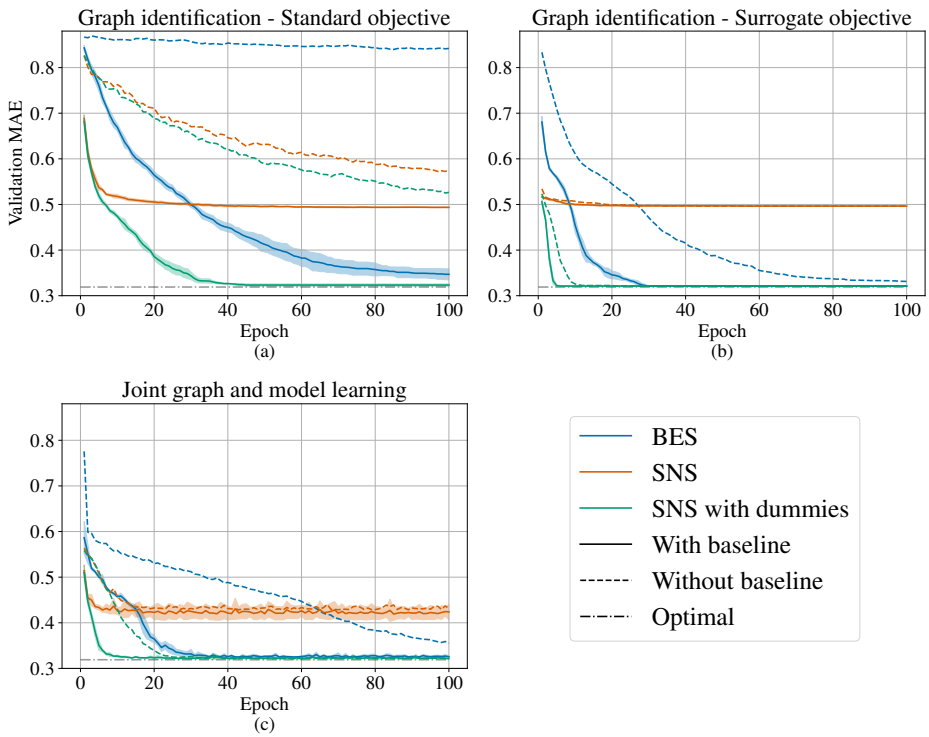


Figure 7.2. Experiments on GPVAR. All the curves show the validation MAE after each training epoch.

Joint training Finally, Figure 7.2c 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. $\boldsymbol{A}^{\mu}$.

7.7.2.2 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 7.2. In particular, we repeated the experiment on graph identification

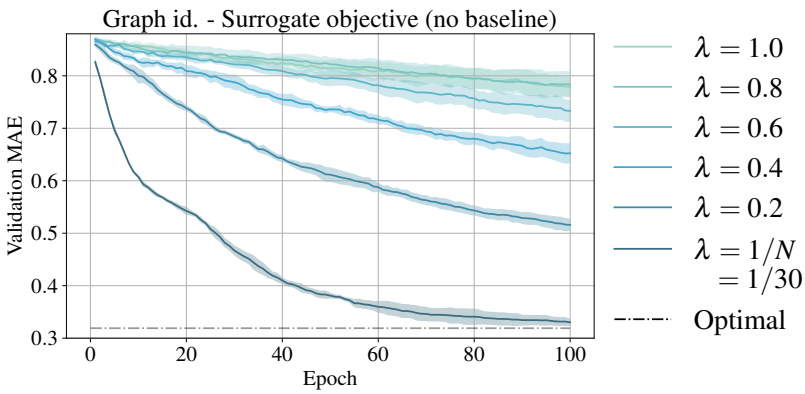


Figure 7.3. Sensitivity analysis on λ for the surrogate objective.

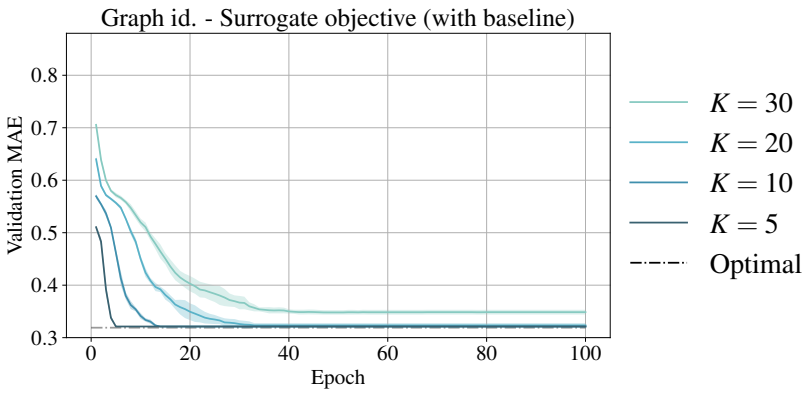


Figure 7.4. Sensitivity analysis on K for SNS.

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 7.3, 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 a number of dummy nodes equal to $K - 1$. Results in Figure 7.4 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

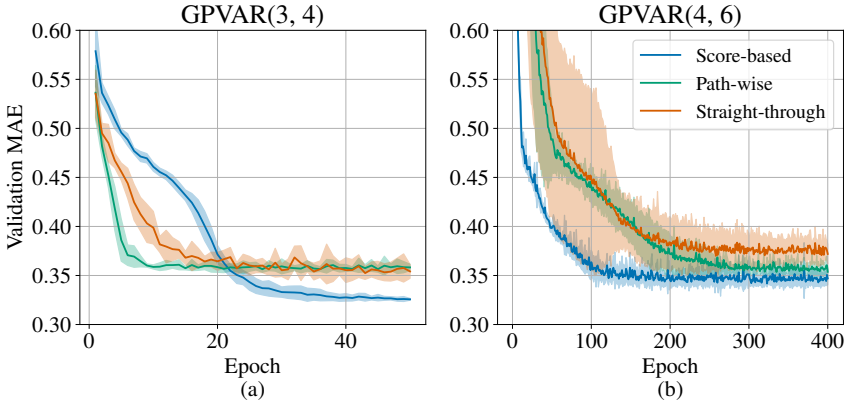


Figure 7.5. Comparison of different estimators on the joint training settings in GPVAR.

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.

7.7.2.3 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 subsubsection 7.7.2.1. 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 subsubsection 7.7.2.1 – and a more difficult scenario corresponding to filters up to orders 4 and 6, respectively. The results of the experiment are shown in Figure 7.5. In the simpler setting (Figure 7.5a), 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 7.5b), 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.

7.7.3 Real-world datasets

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

7.7.3.1 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 model with a simple architecture consisting of a GRU [Chung et al., 2014] encoder followed by 2 isotropic 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 7.1. 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 7.1, 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.

Table 7.1. AQI experiment.

	Tested on	
Trained on	Beijing	Tianjin
Beijing	$9.43_{\pm 0.03}$	$10.62_{\pm 0.05}$
Tianjin	$9.55_{\pm 0.06}$	$10.56_{\pm 0.03}$
Baseline	$10.21_{\pm 0.01}$	$11.25_{\pm 0.04}$

Table 7.2. Results on the traffic datasets.

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

7.7.3.2 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 the reference paper [Cini et al., 2023d]. Note that GTS was considered the state of the art for methods based on path-wise estimators [Zügner et al., 2021]. Results in Table 7.2 show the MAE 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 significantly 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 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.

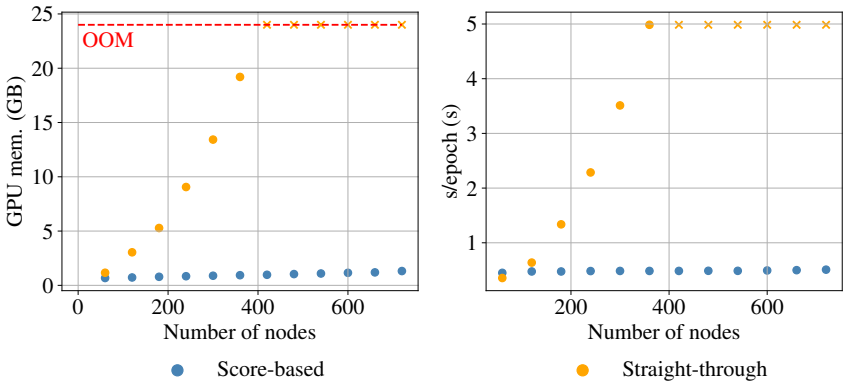


Figure 7.6. Computational scalability of the proposed estimator against the straight-through method.

7.7.4 Scalability

To assess the scalability of the proposed method, we consider a T&S model consisting of a GRU with gates implemented by an anisotropic MP operator. In particular, we consider a simple MP scheme s.t.

$$\boldsymbol{z}_t^{i,(l)} = \sum_{j \in \mathcal{N}(i)} \mathrm{MLP}\left(\boldsymbol{z}_t^{i,(l-1)}, \boldsymbol{z}_t^{j,(l-1)}\right). \tag{7.57}$$

The resulting model has a space and time complexity that scales as $\mathcal{O}(LT|\mathcal{E}|)$. By considering the same controlled environment of the experiments in Section 7.7.2 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 a straight-through estimator. 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 $\mathcal{O}(LT|\mathcal{E}|)$.

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 Section 7.7.4 show measured GPU usage and latency for the above settings. The computational advantages of the sparse MP operations enabled by our method are evident.

7.8 Conclusions and future directions

We proposed a methodological framework for learning graph distributions from correlated time series. 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 developed variance reduction techniques for our method to obtain accurate estimates of the gradient at training time. 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. In some sense, graph learning can be seen as a regularization of attention-based architectures [Vaswani et al., 2017], where, rather than relying on attention scores between each pair of nodes, the learned graph is used to route information only between certain nodes, thus providing localized node representations typical of graph-based processing. As such, we argue that the associated methodologies constitute a relevant aspect of modern approaches to correlated time series forecasting.

Future directions Possible directions for future research include the assessment of the proposed method w.r.t. the 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. The study of methods to calibrate the uncertainty over the existence of each edge is also an interesting direction which has already seen follow-up work [Manenti et al., 2024]. Beyond learning binary adjacency matrices, learning edge attributes [Kipf and Welling, 2017] and higher-order relationships (e.g., as in [Battiloro et al., 2024]) are relevant directions to explore further. 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 chapter.

Chapter 8

Computational scalability

This chapter focuses on the computational scalability of the introduced framework **(Challenge 4)** and introduces a scalable graph-based forecasting architecture. As highlighted in Chapter 3, training STGNNs can incur high computational costs: designing the forecasting architecture having these potential constraints in mind is crucial. Section 8.1 provides further details on the computation scalability of standard STGNNs architectures and discusses methods to deal with the issue. Preliminary concepts toward introducing the proposed scalable forecasting architecture are given in Section 8.2. Section 8.3 presents, then, the *Scalable Graph Predictor* (SGP) architecture, our proposed approach. Section 8.4 provides additional discussion on the related work. Empirical results (Section 8.5) show that the proposed method can match the state of the art in forecasting accuracy while being drastically more scalable w.r.t. both sequence length and graph size. Finally, Section 8.6 discusses future directions.

Reference papers The content of the chapter is partly based on material from the following papers.

- Andrea Cini, Ivan Marisca, Filippo Maria Bianchi, and Cesare Alippi. Scalable Spatiotemporal Graph Neural Networks. *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, 2023a

8.1 Dealing with large time series collection

Scalability concerns can emerge from both the number of target time series as well as their length. Notably, dealing with thousands of time series acquired at high sampling rates over long periods of time is rather common, e.g., in transportation systems and smart power grids [Cini et al., 2023a; Liu et al., 2023c]. This results in a large amount of data that must be processed at once. While the research to improve the scalability of models for static graph signals has been very prolific [Hamilton et al., 2017; Chiang et al., 2019; Zeng et al., 2020; Frasca et al., 2020], little attention has been paid to the additional challenges encountered when dealing with discrete-time dynamical graphs which corresponds to the setting we are dealing with. When designing and/or implementing an STGNN, the scalability issue, then, must be taken into account.

8.1.1 Computational scalability in STGNNs

As mentioned in Section 3.3, a generic T&S model performs L stacked MP operations for each time step resulting in a time and space complexity scaling with $\mathcal{O}\big(W(N+L|\mathcal{E}_{\max}|)\big)$, or $\mathcal{O}\big(WL|\mathcal{E}_{\max}|\big)$ assuming $N \ll |\mathcal{E}_{\max}|$. STT models are characterized by an analogous computation complexity, as the decoupled processing generally does not bring any advantage in this direction. Conversely, TTS models, by encoding the time series ahead of any MP operation, scale with $\mathcal{O}\big(WN + L|\mathcal{E}_t|\big)$, which, again assuming $N \ll |\mathcal{E}_t|$, is a notable improvement. However, even models following this paradigm can struggle whenever either N, W, or $|\mathcal{E}|$ are large, and appropriate computational resources can quickly become unaffordable. This issue is particularly relevant at training time when processing batches of such high-dimensional data concurrently is needed to fit STGNN' parameters on the available data.

Graph subsampling and precomputed representations An often viable solution is to subsample the data fed to the model. In particular, the computational burden can be reduced at training time by extracting subgraphs from the full-time series collection [Hamilton et al., 2017; Chiang et al., 2019; Zeng et al., 2020] by, e.g., considering the K-th order neighborhood of a subset of nodes. Such approaches have been exploited, mostly adapted from methods developed in the context of static graph processing, and have indeed been applied to scale graph-based time series forecasting to large networks [Wu et al., 2020; Gandhi et al., 2021; Rong et al., 2020]. Subsampling methods, then, allow for capping the number of nodes/edges to be processed for each

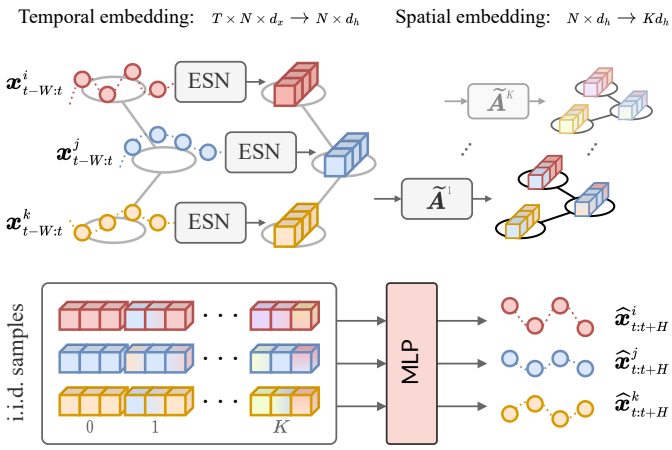


Figure 8.1. Overview of SGP forecasting framework. Light-grey boxes denote training-free components. At first, an echo state network – with shared parameters across nodes – encodes multi-scale temporal dynamics. Then, K graph shift operators are used to propagate spatial information. The resulting $K+1$ representations are concatenated and fed to an MLP to predict the next H node observations.

sample based on the available computational resources. The drawback of these approaches is that such a subsampling might break long-range spatiotemporal dependencies (n.b., data are not i.i.d.) and may result in a noisy and biased learning signal. Similar arguments can be made w.r.t. using small batch sizes and short input windows. To go beyond the limitations of subsampling methods, in the following, we introduce a novel approach to building scalable forecasting architectures based on *precomputing spatiotemporal representations* of the input ahead of training [Cini et al., 2023a]. Our approach relies on randomized deep echo state networks [Gallicchio et al., 2017; Bianchi et al., 2020c] and powers graph shift operators to extract spatiotemporal representations unsupervised. Figure 8.1 provides an overview of the approach. Since the encoding scheme requires neither training nor supervision, the representation of each node and time step can be computed as a preprocessing step. Representations can then be sampled uniformly across time and space (as if they were i.i.d.) to efficiently train a decoder for mapping them to predictions; this makes the computational cost of a training step independent of both sequence length and graph size.

8.2 Preliminaries

We consider the reference settings introduced in Section 3.1 and relational representations encoded by a static weighted adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$. Extensions that consider (dynamic) edge attributes are nonetheless possible (see Section 8.6). The following, then, provides some preliminary concepts on echo state network (ESN) architectures, a class of randomized RNNs already briefly introduced in Chapter 2 (see Section 2.5.1).

8.2.1 Echo state networks

Echo state networks (ESNs) [Jaeger, 2001; Lukoševičius and Jaeger, 2009] are a class of randomized architectures [Gallicchio and Scardapane, 2020] that consist of recurrent neural networks with random weights [Lukoševičius and Jaeger, 2009] that encode the history of input signals into a high-dimensional state representation to be used as input to a (trainable) readout layer. The main idea is to feed an input signal into a high-dimensional, randomized, and non-linear reservoir, whose internal state can be used as an embedding of the input dynamics. An echo state network is driven by the following state update equation:

$$\boldsymbol{h}_t = \sigma\left(\boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{b}\right), \tag{8.1}$$

where $\boldsymbol{x}_t$ indicates a generic input to the system, $\boldsymbol{W}_x \in \mathbb{R}^{d_h \times d_x}$ and $\boldsymbol{W}_h \in \mathbb{R}^{d_h \times d_h}$ are the random matrices defining the connectivity pattern in the reservoir, $\boldsymbol{b} \in \mathbb{R}^{d_h}$ is a randomly initialized bias, $\boldsymbol{h}_t$ indicates the reservoir state, and σ is a nonlinear activation function (usually tanh). If the random matrices are defined properly, the reservoir can extract a rich pool of dynamics characterizing the system underlying the input time series $\boldsymbol{x}_t$ and, thus, the reservoir states become informative embeddings of $\boldsymbol{x}_{t-T:t}$ [Lukoševičius and Jaeger, 2009]. Thanks to the non-linearity of the reservoir, embeddings are commonly processed with a linear readout that is optimized with a least squares procedure to perform classification, clustering, or time series forecasting [Bianchi et al., 2020c].

8.3 Scalable spatiotemporal GNNs

In this section, we introduce the *Scalable Graph Predictor* (SGP) [Cini et al., 2023a], a scalable graph-based forecasting architecture based on the idea of precomputing representations ahead of training [Frasca et al., 2020]. Figure 8.1 provides an overview of the proposed approach. SGP is based on a hybrid

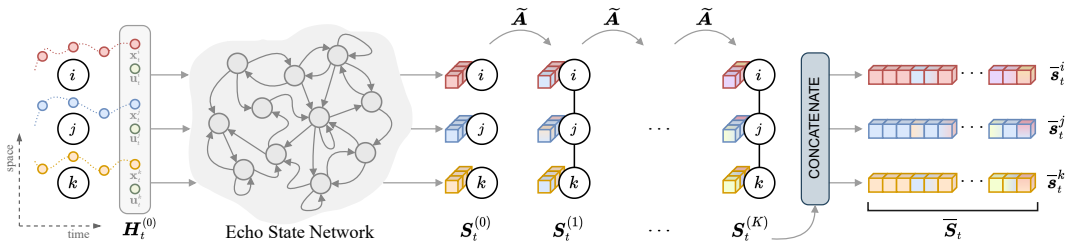


Figure 8.2. Overview of the SGP encoder. Input time series are fed into a randomized network with recurrent connections and embedded into a hierarchical vector representation. A graph shift operator is used to propagate and aggregate spatial information of different order which is then concatenated to obtain a final embedding.

encoder-decoder architecture. The encoder first constructs representations of the time series observed at each node by using a reservoir (a deep ESN Gallicchio et al. [2017]) that accounts for dynamics at different time scales. Representations are further processed to account for spatial dynamics described by the graph structure. In particular, we use incremental powers of the graph adjacency matrix to propagate and aggregate information along the spatial dimension. Each power of the propagation matrix accounts for different scales of spatial dynamics. The final embedding is then built by concatenating representations obtained w.r.t. each propagation step, thus resulting in a rich encoding of both spatial and temporal features. The computation of these representations can be done ahead of training (thus avoiding GPU memory bottlenecks) and is performed only once. Indeed, the encoding scheme does not need any training. Embeddings, once computed, can be uniformly sampled over time and space while training a nonlinear readout to obtain forecasts. The readout (i.e., a decoder) can be simply implemented as an MLP mapping encodings to multi-step predictions. To further enhance scalability, however, the embedding structure can be exploited to reduce the number of parameters and learn filters localized in time and space. As discussed in Section 8.3.2, this is done by learning separate weight matrices for each spatiotemporal scale. The following subsections describe each component of the architecture in detail.

8.3.1 Scalable spatiotemporal representation

We consider as temporal encoders deep ESNs [Gallicchio et al., 2017] with leaky integrator neurons [Jaeger et al., 2007]. In particular, we consider networks where the signal associated with each node is encoded by a stack of L randomized

recurrent layers s.t.

$$\begin{aligned}
\boldsymbol{h}_t^{i,(0)} &= \left[\boldsymbol{x}_t^i \| \boldsymbol{u}_t^i\right], \\
\hat{\boldsymbol{h}}_t^{i,(l)} &= \tanh\left(\boldsymbol{W}_u^{(l)} \boldsymbol{h}_t^{i,(l-1)} + \boldsymbol{W}_h^{(l)} \boldsymbol{h}_{t-1}^{i,(l)} + \boldsymbol{b}^{(l)}\right), \\
\boldsymbol{h}_t^{i,(l)} &= (1-\gamma_l)\boldsymbol{h}_{t-1}^{i,(l)} + \gamma_l \hat{\boldsymbol{h}}_t^{i,(l)}, \qquad l = 1, \dots, L
\end{aligned} \tag{8.2}$$

where $\gamma_l \in (0, 1]$ is a discount factor associated with l-th layer, $\boldsymbol{W}_u^{(l)} \in \mathbb{R}^{d_{h^l} \times d_{h^{l-1}}}$, $\boldsymbol{W}_h \in \mathbb{R}^{d_{h^l} \times d_{h^l}}$, $\boldsymbol{b} \in \mathbb{R}^{d_{h^l}}$ are random weight matrices, $\boldsymbol{h}_t^{i,(l)}$ indicates the hidden state of the system w.r.t. the i-th node at the l-th layer, and $\|$ indicates node-wise concatenation. Note that, to avoid ambiguity, the index of the layers is indicated between parentheses in this chapter. As shown by Equation 8.2, deep ESNs are a hierarchical stack of reservoir layers that, e.g., by changing the discount factor at each layer, extract a rich pool of multi-scale temporal dynamics [Gallicchio et al., 2017][1]. Given a deep ESN encoder, the input is represented by the concatenation of the states from each layer, i.e., we obtain node-level temporal encodings $\overline{\boldsymbol{h}}_t^i$ for each i-th node and time step t as

$$\overline{\boldsymbol{h}}_t^i = \left(\boldsymbol{h}_t^{i,(0)} \| \boldsymbol{h}_t^{i,(1)} \| \dots \| \boldsymbol{h}_t^{i,(L)}\right). \tag{8.3}$$

Similarly, we indicate as $\overline{\boldsymbol{H}}_t$ the encoding for the whole time series collection at the corresponding time t. Figure 8.2 provisides a visual representation of the processing by showing, on the left-end side, how node-level temporal embeddings are extracted from the input sequences; to simplify the drawing, the figure shows an sigle-layer ESN.

Spatial propagation The next step is to propagate information among time series. As discussed at the beginning of the section, we use powers of a graph shift operator $\widetilde{\boldsymbol{A}}$ to propagate and aggregate node representations at different scales. By using a notation similar to Equation 8.3, we obtain spatiotemporal encodings as

$$\begin{aligned}
\boldsymbol{S}_t^{(0)} &= \overline{\boldsymbol{H}}_t = \left(\boldsymbol{H}_t^{(0)} \| \boldsymbol{H}_t^{(1)} \| \dots \| \boldsymbol{H}_t^{(L)}\right), \\
\boldsymbol{S}_t^{(k)} &= \widetilde{\boldsymbol{A}} \boldsymbol{S}_t^{(k-1)} = \left(\widetilde{\boldsymbol{A}}^k \boldsymbol{H}_t^{(0)} \| \widetilde{\boldsymbol{A}}^k \boldsymbol{H}_t^{(1)} \| \dots \| \widetilde{\boldsymbol{A}}^k \boldsymbol{H}_t^{(L)}\right), \\
\overline{\boldsymbol{S}}_t &= \left(\boldsymbol{S}_t^{(0)} \| \boldsymbol{S}_t^{(1)} \| \dots \| \boldsymbol{S}_t^{(K)}\right),
\end{aligned} \tag{8.4}$$

[1] We refer to [Gallicchio et al., 2018] for more details on the properties and stability of deep ESNs.

where $\widetilde{\boldsymbol{A}}$ indicates a generic graph shift operator matching the sparsity pattern of the graph adjacency matrix. In practice, by indicating with $\boldsymbol{D}$ the graph degree matrix, we use $\widetilde{\boldsymbol{A}} = \boldsymbol{D}^{-1}\boldsymbol{A}$ in the case of a directed graph and the symmetrically normalized adjacency $\widetilde{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ in the undirected case. Furthermore, for directed graphs we optionally increase the number of representations to $2K+1$ to account for bidirectional dynamics, i.e., we repeat the encoding process w.r.t. the transpose adjacency matrix similarly to [Li et al., 2018]. Intuitively, each propagation step $\widetilde{\boldsymbol{A}}\boldsymbol{S}_t^{(k-1)}$ propagates and aggregates – properly weighted – features between nodes connected by paths of length k in the graph. As shown in Equation 8.4, features corresponding to each order k can be computed recursively with K sparse matrix-matrix multiplications (Figure 8.2). Alternatively, each matrix $\widetilde{\boldsymbol{A}}^k$ can be precomputed and the computation of the different blocks of matrix $\overline{\boldsymbol{S}}_t$ can be distributed in a parallel fashion as suggested by Figure 8.1. Even in the case of extremely large time series collections, features $\overline{\boldsymbol{S}}_t$ can be computed offline by exploiting distributed computing as they do not need to be loaded on GPU memory. The processing carried out in Equation 8.4 can be equivalently viewed within the MP framework as

$$\overline{\boldsymbol{s}}_t^i = \left(\overline{\boldsymbol{h}}_t^i \,\Big\|\, \underset{j\in\mathcal{N}(i)}{\textsc{Aggr}} \left\{ \overline{\boldsymbol{h}}_t^j \right\} \,\Big\|\, \underset{j\in\mathcal{N}^{(2)}(i)}{\textsc{Aggr}} \left\{ \overline{\boldsymbol{h}}_t^j \right\} \,\Big\|\, \dots \,\Big\|\, \underset{j\in\mathcal{N}^{(K)}(i)}{\textsc{Aggr}} \left\{ \overline{\boldsymbol{h}}_t^j \right\} \right) \tag{8.5}$$

where $\mathcal{N}^{(k)}(i)$ denotes the neighborhood of the i-th node w.r.t. adjacency matrix $\widetilde{\boldsymbol{A}}^k$ and the aggregation is performed by summing each representation multiplied by the associated edge weight.

8.3.2 Multi-scale decoder

The role of the decoder is that of selecting and weighing (possibly redundant) features from the pool extracted by the encoder and mapping them to the desired output. Representations $\overline{\boldsymbol{S}}_t$ can be fed into an MLP that performs node-wise predictions. Since the representations are large vectors, a naïve implementation of the MLP results in many parameters that hinder scalability. Therefore, we replace the first MLP layer with a more efficient implementation that exploits the structure of the embeddings.

Localized filters As we described in Section 8.3.1, $\overline{\boldsymbol{S}}_t$ is the concatenation of the representations corresponding to different spatial propagation steps which, in turn, are obtained from the concatenation of multi-scale temporal features. To exploit this structure, we design the first layer of the decoder with a sparse

connectivity pattern to learn representations $\overline{\boldsymbol{Z}}_t$ s.t.

$$\boldsymbol{Z}_t^{(k)} = \sigma\left(\widetilde{\boldsymbol{A}}^k \boldsymbol{H}_t^{(0)} \boldsymbol{\Theta}_k^{(0)} \| \ldots \| \widetilde{\boldsymbol{A}}^k \boldsymbol{H}_t^{(L)} \boldsymbol{\Theta}_k^{(L)}\right) \tag{8.6}$$

$$= \sigma\left(\boldsymbol{S}_t^{(k)} \begin{bmatrix} \boldsymbol{\Theta}_k^{(0)} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \boldsymbol{\Theta}_k^{(L)} \end{bmatrix}\right), \tag{8.7}$$

$$\overline{\boldsymbol{Z}}_t = \left(\boldsymbol{Z}_t^{(0)} \| \boldsymbol{Z}_t^{(1)} \| \ldots \| \boldsymbol{Z}_t^{(K)}\right), \tag{8.8}$$

where $\boldsymbol{\Theta}_k^{(l)} \in \mathbb{R}^{d_{h^l} \times d_z}$ are the learnable parameters and σ is an activation function. In practice, representations $\overline{\boldsymbol{Z}}_t$ can be efficiently computed by exploiting grouped 1-d convolutions (e.g., see Krizhevsky et al. 2012) to parallelize computation on GPUs. In particular, if we indicate the 1-d grouped convolution operator with g groups and kernel size r as $\star_{r,g}$, and the collection of the decoder parameters $\boldsymbol{\Theta}_k^{(l)}$ as $\boldsymbol{\Theta}$ we can compute $\overline{\boldsymbol{Z}}_t$ as

$$\overline{\boldsymbol{Z}}_t = \sigma\left(\boldsymbol{\Theta} \star_{1,g} \overline{\boldsymbol{S}}_t\right), \tag{8.9}$$

with $g = L(K+1)$ in the case of undirected graphs and $g = L(2K+1)$ for the directed case. Besides reducing the number of parameters by a factor of $L(K+1)$, this architecture localizes filters $\boldsymbol{\Theta}_k^{(L)}$ w.r.t. the dynamics of spatial order k and temporal scale l. In fact, as highlighted in Equation 8.6–8.8, representation $\overline{\boldsymbol{Z}}_t$ can be seen as a concatenation of the results of $L(K+1)$ graph convolutions of different order. Finally, the obtained representations are fed into an MLP that predicts the H-step-ahead observations as

$$\widehat{\boldsymbol{x}}_{t:t+H}^i = \text{MLP}\left(\overline{\boldsymbol{z}}_t^i, \boldsymbol{v}^i\right), \tag{8.10}$$

where the static node-level attributes $\boldsymbol{v}^i$ can also be augmented by concatenating learnable node embeddings $\boldsymbol{q}^i$ (see Chapter 5).

8.3.3 Training and sampling

The main improvement introduced by the proposed approach in terms of scalability concerns the training procedure. Representations $\overline{\boldsymbol{S}}_t$ account for both temporal and spatial dependencies among observations over the sensor network. Consequently, each sample $\overline{\boldsymbol{s}}_t^i$ can be processed independently since spatiotemporal information has already been propagated. This allows for training the decoder with SGD by uniformly and independently sampling mini-batches of data points $\overline{\boldsymbol{s}}_t^i$. This is the key property that makes the training procedure extremely scalable and drastically reduces training computational requirements w.r.t. standard STGNN.

8.4 Related work

SGP falls within the category of TTS predictors, i.e., forecasting models models where the temporal information is encoded before being propagated along the spatial dimension. As already mentioned, research on scalable graph-based methods for time series forecasting has been relatively limited. Practitioners have mostly relied on methods developed in the context of static graphs which include node-centric, GraphSAGE-like, approaches [Hamilton et al., 2017] or subgraph sampling methods, such as ClusterGCN [Chiang et al., 2019] or GraphSAINT [Zeng et al., 2020]. Wu et al. [2020]; Gandhi et al. [2021]; Wu et al. [2021b] are examples of such approaches. Among scalable GNNs for static graphs, SIGN [Frasca et al., 2020] is the approach most related to our method. Similarly to SGP, SIGN performs spatial propagation as a preprocessing step by using different shift operators to aggregate across different graph neighborhoods, which are then fed to an MLP. However, SIGN is limited to static graphs and propagates raw node-level attributes. Finally, analogously to our approach, DynGESN [Micheli and Tortorella, 2022] processes dynamical graphs with a recurrent randomized architecture. However, the architecture in DynGESN is completely randomized, while ours is hybrid as it combines randomized components in the encoder with trainable parameters in the decoder.

8.5 Empirical results

We empirically evaluate our approach in two different scenarios. In the first, we compare the performance of our forecasting architecture against state-of-the-art methods on the already introduced traffic forecasting benchmarks. In the second, we evaluate the scalability of the proposed method on large-scale spatiotemporal time series datasets by considering two additional benchmarks for load forecasting and PV production prediction. Further details on datasets, baselines, and experimental settings can be found in the *arxiv* version of the reference paper [2].

Datasets Besides traffic datasets (**METR-LA** and **PEMS-BAY**), we consider two larger datasets from for assessing the scalability of the different approaches. The first dataset is a larger version of CER-E, simply indicated as **CER**; differently from the version introduced in Chapter 4, we do not limit the analysis to load profiles coming from enterprises, but consider the full

[2] https://arxiv.org/abs/2209.06520

Table 8.1. Additional information on the considered datasets.

Dataset	# steps	# nodes	# edges
METR-LA	34272	207	1515
PEMS-BAY	52116	325	2369
PV-US (100nn)	8868	5016	417,199
CER (100nn)	8868	6435	639,369
PV-US	8868	5016	3,710,008
CER	8868	6435	3,186,369

network consisting of 6435 smart meters measuring energy consumption every 30 minutes at both residential and commercial/industrial premises. The second large-scale dataset is obtained from the synthetic **PV-US**[3] dataset [Hummon et al., 2012], consisting of the simulated energy production of 5016 PV farms scattered over the United States for the year 2006, aggregated in half an hour intervals. Since the model does not have access to weather information, PV production at neighboring farms is instrumental in obtaining good predictions. CER-E and PV-US are at least an order of magnitude larger than the datasets typically used for benchmarking spatiotemporal time series forecasting models. Weighted adjacency matrices are obtained, similarly to datasets introduced in Chapter 4, by applying a thresholded Gaussian kernel Shuman et al. [2013] to the similarity matrices obtained by considering the geographic distance among the sensors (PV-US) and the average weekly correntropy among the time series (CER). Additional details on the datasets are reported on Table 8.1.

Baselines We consider models analogous to those discussed in Chapter 4:

LSTM a single global univariate LSTM [Hochreiter and Schmidhuber, 1997];

FC-LSTM an LSTM processing input sequences as if they were a single high-dimensional multivariate time series;

DCRNN the GCRNN introduced by [Li et al., 2018]);

GraphWaveNet the graph-based convolutional model introduced by [Wu et al., 2019].

Additionally, we include two more baselines:

[3] https://www.nrel.gov/grid/solar-power-data.html

GatedGN (GGN) : a state-of-the-art TTS model introduced in [Satorras et al., 2022] for which we consider two different configurations. The first one (**FC**) uses attention over the full node set to perform spatial propagation, while the second one (**UG**) constrains the attention to edges of the underlying graph.

DynGESN : the echo state network for dynamical graphs proposed in [Micheli and Tortorella, 2022].

For all the baselines, we use, whenever appropriate, the configuration found in the original papers or in their open-source implementation; in all the other cases we tune hyperparameters on the holdout validation set.

8.5.1 Experimental setup

To run the experiments, we considered the following scenarios and setups.

Traffic datasets For traffic datasets, we replicate the setup used in previous works. In particular, each model is trained to predict the 12-step-ahead observations. In SGP, the input time series are first encoded by the spatiotemporal encoder, and then the decoder is trained by sampling mini-batches along the temporal dimension, i.e., by sampling B sequences $\mathcal{G}_{t-W:t}$ of observations.

Scalability benchmarks For the large-scale datasets, we focus on assessing the scalability of the different architectures rather than maximizing forecasting accuracy. In particular, for both datasets, we consider the first 6 months of data (4 for months for training, 1 month for validation, and 1 month for testing). The models are trained to predict the next $\{00{:}30, 07{:}30, 11{:}00\}$ hours. We repeat the experiment in two different settings to test the scalability of the different architectures w.r.t. the number of edges. In the first setting, we extract the graph by sparsifying the graph adjacency matrix imposing a maximum of 100 neighbors for each node, while, in the second case, we do not constrain the density of the adjacency matrix. Table 8.1 reports some details needed to appreciate the difference in sparsity levels. To assess the performance in terms of scalability, we fix a maximum GPU memory budget of 12 GB and select the batch size accordingly; if a batch size of 1 does not fit in 12 GB, we uniformly subsample edges of the graph to reduce the memory consumption. Differently from the other baselines, in SGP we first preprocess the data to obtain spatiotemporal embeddings and then train the decoder by uniformly

sampling the node representations. We train each model for 1 hour, then restore the weights corresponding to the minimum training error and evaluate the forecasts on the test set. The choice of not running validation at each epoch was dictated by the fact that for some of the baselines running a validation epoch would take a large portion of the 1 hour budget.

Additional details The time required to encode the datasets with SGP's encoder ranges from tens of seconds to $\approx$ 4 minutes on an AMD EPYC 7513 processor with 32 parallel processes. To ensure reproducibility, the time constraint is not imposed as a hard time out; conversely, we measure the time required for the update step of each model on an NVIDIA RTX A5000 GPU and fix the maximum number of updates accordingly. For SGP, the time required to compute node embeddings was considered as part of the training time and the number of updates was appropriately reduced to make the comparison fair. For all the baselines, we keep the same architecture used in the traffic experiment. For SGP we use the same hyperparameters for the decoder, but we reduce the dimension of the embedding (the value of K) so that a preprocessed dataset can fit in a maximum of $\approx$ 80 GB of storage. To account for the different temporal scales, we increase the window size for all baselines and increase the number of layers in the ESN (while keeping the final size of $\overline{\boldsymbol{H}}_t$ similar). For additional details, we refer to [Cini et al., 2023a].

8.5.2 Results

Results for the traffic benchmarks are reported in Table 8.2; while the outcomes of the scalability experiments are shown in Table 8.3. We consider MAE and MAPE as evaluation metrics.

Traffic experiment The purpose of the first experiment is to demonstrate that the proposed method achieves performance comparable to that of the state of the art. In this regard, results in Table 8.2 show that SGP is among the best-performing forecasting architectures in all the considered scenarios. The full-attention baseline is the strongest competitor but, has time and memory complexities that scale quadratically with the number of nodes. DynGESN, the fully randomized architecture, despite being very fast to train, obtains reasonable performance in short-range predictions but falls short over longer forecasting horizons in the considered scenarios. Note that the performance of DCRNN and GraphWaveNet are slightly different here w.r.t. Chapter 4 due to a change in the exogenous variables being used. In light of these results, it is worth

Table 8.2. Results on benchmark traffic datasets (averaged over 3 independent runs). We report MAE and MAPE averaged over a one-hour (12 steps) forecasting horizon. We also show MAE for $H \in \{15, 30, 60\}$ minutes time horizons. Bold numbers are within a standard deviation from the best average.

Models	METR-LA					PEMS-BAY				
	15 m	30 m	60 m	Average		15 m	30 m	60 m	Average	
	MAE	MAE	MAE	MAE	MAPE	MAE	MAE	MAE	MAE	MAPE
LSTM	$2.99_{\pm.00}$	$3.58_{\pm.00}$	$4.43_{\pm.01}$	$3.58_{\pm.00}$	$1.19_{\pm.05}$	$1.39_{\pm.00}$	$1.83_{\pm.01}$	$2.35_{\pm.01}$	$1.79_{\pm.00}$	$4.16_{\pm.05}$
FC-LSTM	$3.33_{\pm.01}$	$3.43_{\pm.01}$	$3.67_{\pm.01}$	$3.46_{\pm.01}$	$1.15_{\pm.09}$	$2.22_{\pm.01}$	$2.25_{\pm.01}$	$2.34_{\pm.02}$	$2.26_{\pm.01}$	$5.33_{\pm.04}$
DynGESN	$3.27_{\pm.00}$	$3.99_{\pm.00}$	$5.00_{\pm.00}$	$3.98_{\pm.00}$	$11.11_{\pm.01}$	$1.57_{\pm.00}$	$2.13_{\pm.01}$	$2.81_{\pm.02}$	$2.09_{\pm.01}$	$4.74_{\pm.01}$
DCRNN	$2.82_{\pm.00}$	$3.23_{\pm.01}$	$3.74_{\pm.01}$	$3.20_{\pm.00}$	$8.88_{\pm.05}$	$1.36_{\pm.00}$	$1.71_{\pm.00}$	$2.08_{\pm.01}$	$1.66_{\pm.00}$	$3.76_{\pm.01}$
GWNet	$2.72_{\pm.01}$	$3.10_{\pm.02}$	$3.54_{\pm.03}$	$3.06_{\pm.02}$	$8.40_{\pm.03}$	$1.31_{\pm.00}$	$1.64_{\pm.01}$	$1.94_{\pm.01}$	$1.58_{\pm.00}$	$3.58_{\pm.02}$
FC-GGN	$2.72_{\pm.01}$	$\mathbf{3.05}_{\pm.01}$	$\mathbf{3.44}_{\pm.01}$	$3.01_{\pm.00}$	$\mathbf{8.27}_{\pm.00}$	$1.32_{\pm.00}$	$1.63_{\pm.01}$	$\mathbf{1.89}_{\pm.01}$	$1.56_{\pm.01}$	$3.51_{\pm.03}$
UG-GGN	$2.72_{\pm.00}$	$3.10_{\pm.00}$	$3.54_{\pm.01}$	$3.06_{\pm.00}$	$8.40_{\pm.04}$	$1.33_{\pm.00}$	$1.67_{\pm.01}$	$1.99_{\pm.01}$	$1.61_{\pm.01}$	$3.59_{\pm.03}$
SGP	$\mathbf{2.69}_{\pm.00}$	$\mathbf{3.05}_{\pm.00}$	$\mathbf{3.45}_{\pm.00}$	$\mathbf{3.00}_{\pm.00}$	$\mathbf{8.27}_{\pm.02}$	$\mathbf{1.30}_{\pm.00}$	$\mathbf{1.60}_{\pm.00}$	$\mathbf{1.88}_{\pm.00}$	$\mathbf{1.54}_{\pm.00}$	$\mathbf{3.44}_{\pm.01}$
Ablations										
–No-Graph	$2.84_{\pm.00}$	$3.26_{\pm.00}$	$3.74_{\pm.00}$	$3.22_{\pm.00}$	$9.20_{\pm.01}$	$1.34_{\pm.00}$	$1.68_{\pm.00}$	$2.02_{\pm.00}$	$1.62_{\pm.00}$	$3.67_{\pm.01}$
–FC-Dec.	$2.76_{\pm.01}$	$3.13_{\pm.01}$	$3.52_{\pm.02}$	$3.08_{\pm.01}$	$8.63_{\pm.11}$	$1.35_{\pm.01}$	$1.67_{\pm.01}$	$1.96_{\pm.01}$	$1.61_{\pm.01}$	$3.61_{\pm.04}$
–GC-Dec.	$2.77_{\pm.00}$	$3.17_{\pm.00}$	$3.63_{\pm.00}$	$3.12_{\pm.00}$	$8.74_{\pm.01}$	$1.32_{\pm.00}$	$1.65_{\pm.00}$	$1.97_{\pm.00}$	$1.59_{\pm.00}$	$3.60_{\pm.01}$

commenting on the efficiency of SGP compared to the baselines. Approaches like DCRNN and GraphWaveNet, perform graph convolutions whose time and space of complexity is $\mathcal{O}(W(N + |\mathcal{E}|))$, being $|\mathcal{E}|$ the number of edges, L the number of layers (8 in Graph Wavenet), and W the window size. In SGP, this complexity is completely amortized by the preprocessing step. Similarly, GatedGN, while being a TTS model, can incur in scalability issues (as shown in the next experiment). The bottom of Table 8.2 reports results for the ablation of key elements of the proposed architecture: **No-Graph** indicates a variant of the model that does not incorporate the spatial propagation step; **FC-Dec.** consider the case where the sparse weight matrix in Equation 8.7 is replaced by a fully-connected one; **GC-Dec.** indicates a variant where the spatial propagation is limited to the neighbors of order $K = 1$ and, thus, the decoder behaves similarly to a single-layer graph convolutional network. Results clearly show the advantages of the proposed design.

Large-scale experiment Table 8.3 reports the results of the scalability experiment where we considered only the STGNNs trained by SGD. We excluded the full-attention baseline (FC-GatedGN) as its $\mathcal{O}(N^2)$ complexity prevented

Table 8.3. Results on large-scale datasets (averaged over at least 3 independent runs). We report MAE over H-step-ahead predictions, $H = \{30\text{m}, 7\text{h}30\text{m}, 11\text{h}\}$, together with timings and memory consumption. $*$ indicates that subsampling was needed to comply with the memory constraints. Bold numbers are within a standard deviation from the best average.

		Models	Prediction error (MAE)			Resource utilization		
			30 m	7 h 30 m	11 h	Batch/s	Memory	Batch
PV-US	100-NN	DCRNN	$1.39_{\pm.09}$	$3.34_{\pm.22}$	$\mathbf{3.54_{\pm.48}}$	$2.04_{\pm.01}$	9.63 GB	2
		GWNet	$1.45_{\pm.13}$	$5.09_{\pm.63}$	$5.26_{\pm1.34}$	$2.01_{\pm.02}$	11.64 GB	2
		UG-GGN	$1.33_{\pm.08}$	$\mathbf{2.94_{\pm.05}}$	$\mathbf{3.12_{\pm.14}}$	$8.41_{\pm.09}$	11.46 GB	5
		SGP	$\mathbf{1.09_{\pm.01}}$	$\mathbf{3.14_{\pm.21}}$	$\mathbf{3.16_{\pm.19}}$	$\mathbf{116.58_{\pm8.74}}$	**2.21 GB**	4096
	Full	DCRNN	$1.59_{\pm.17}$	$4.10_{\pm.27}$	$4.93_{\pm.60}$	$1.37_{\pm.00}$	11.59 GB	1^*
		GWNet	$1.65_{\pm.23}$	$6.93_{\pm.58}$	$7.93_{\pm.17}$	$.77_{\pm.00}$	11.35 GB	2
		UG-GGN	$1.61_{\pm.06}$	$3.25_{\pm.04}$	$\mathbf{3.04_{\pm.05}}$	$8.83_{\pm.10}$	11.14 GB	1^*
		SGP	$\mathbf{1.09_{\pm.00}}$	$\mathbf{3.06_{\pm.11}}$	$\mathbf{3.13_{\pm.13}}$	$\mathbf{118.64_{\pm8.35}}$	**2.21 GB**	4096
CER	100-NN	DCRNN	$0.22_{\pm.00}$	$\mathbf{0.28_{\pm.00}}$	$0.29_{\pm.00}$	$1.43_{\pm.02}$	11.10 GB	2
		GWNet	$0.23_{\pm.00}$	$0.36_{\pm.01}$	$0.36_{\pm.01}$	$2.41_{\pm.03}$	8.39 GB	1
		UG-GGN	$0.22_{\pm.00}$	$\mathbf{0.28_{\pm.00}}$	$\mathbf{0.28_{\pm.00}}$	$8.21_{\pm.08}$	11.70 GB	4
		SGP	$\mathbf{0.21_{\pm.00}}$	$0.30_{\pm.00}$	$0.31_{\pm.01}$	$\mathbf{117.32_{\pm8.36}}$	**2.21 GB**	4096
	Full	DCRNN	$0.23_{\pm.00}$	$0.29_{\pm.00}$	$\mathbf{0.29_{\pm.00}}$	$1.13_{\pm.01}$	11.10 GB	1^*
		GWNet	$0.25_{\pm.01}$	$0.38_{\pm.03}$	$0.37_{\pm.01}$	$1.26_{\pm.01}$	8.58 GB	1
		UG-GGN	$0.22_{\pm.00}$	$\mathbf{0.28_{\pm.00}}$	$\mathbf{0.29_{\pm.00}}$	$8.77_{\pm.10}$	11.14 GB	1^*
		SGP	$\mathbf{0.21_{\pm.00}}$	$0.30_{\pm.00}$	$0.31_{\pm.01}$	$\mathbf{115.85_{\pm1.60}}$	**2.21 GB**	4096

scaling to the larger datasets; however, we considered the UG version where attention is restrained to each node's neighborhood. Several comments need to be made here. First of all, batch size has a different meaning for our model and the other baselines. In our case, each sample corresponds to a single preprocessed representation (corresponding to a single observation); for the other methods, a sample corresponds to a window of observations $\mathcal{G}_{t-W:t}$ where edges of the graph are eventually subsampled if the memory constraints could not be met otherwise. In both cases, the loss is computed w.r.t. all the observations in the batch. The results clearly show that SGP can be trained efficiently also in resource-constrained settings, with contained GPU memory usage. In particular,

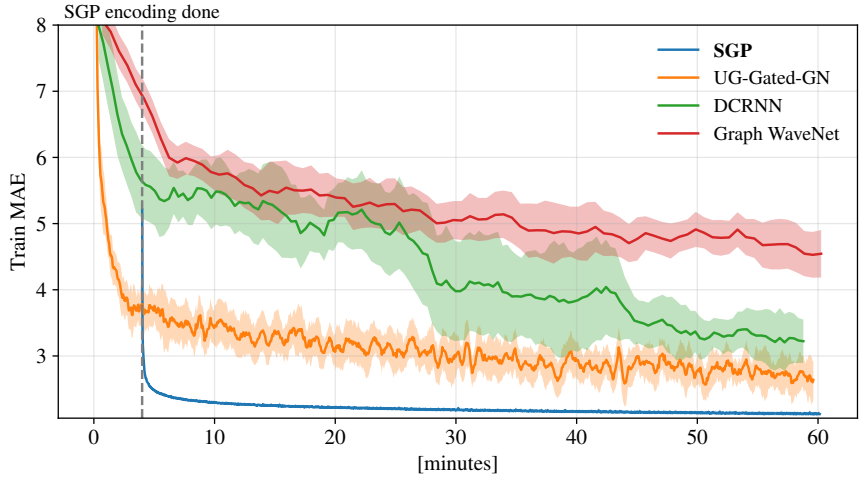


Figure 8.3. Training curves on PV-US. The plot shows the average ± the standard deviation of 3 independent runs. The plotted curves are smoothed with a running average of 8 steps.

the update frequency (batch/s) is up to 2 order of magnitude higher. Notably, resource utilization at training time remains constant, while almost all the baselines require edge subsampling to meet resource constraints. Figure 8.3 shows learning curves for the PV-US dataset, further highlighting the vastly superior efficiency, scalability, and learning stability of SGP. Finally, results concerning forecasting accuracy show that performance is competitive with the state of the art in all the considered scenarios.

8.6 Discussion and future directions

In this chapter, we addressed the problem of scalability in STGNNs, mainly by introducing SGP, a scalable architecture for graph-based time series forecasting. Our approach is competitive with the state of the art in popular medium-sized benchmark datasets, while greatly improving scalability in large sensor networks. While in SGP sampling largely reduces GPU memory usage compared to the other methods, the entire processed sequence can take up a large portion of system memory, depending on the reservoir size. Nevertheless, the preprocessing can be distributed, and the preprocessed data stored on disk and loaded in batches during training, as customary for large datasets. Furthermore, having separate encoding and decoding can be less effective in certain

scenarios and more reliant on hyperparameter selection compared to end-to-end approaches. Nonetheless, we believe that SGP has a wide application potential, and constitutes an important piece for future research on scalable graph-based forecasting architectures.

Future directions Future work can explore scalable architectures with tighter integration of the spatial and temporal encoding components and assess performance on even larger benchmarks. One could also consider extensions of SGP-like architectures to settings where both the edge attributes and the underlying graph are dynamic. This could be done, for example, following the TTS approach introduced in Gao and Ribeiro [2022] which consists of processing padded sequences of edge attributes with a temporal edge encoder to obtain a static graph representation. Dimensionality reduction techniques could be considered to reduce the size of the precomputed representations [Bianchi et al., 2020c]. Finally, other graph diffusion mechanisms could be considered for the spatial propagation step, e.g., by considering operators inspired by recent work on graph convolutions as gradient flows [Di Giovanni et al., 2023].

Chapter 9

Graph-based hierarchical forecasting

We have shown that the models introduced so far can achieve remarkable results on several scenarios. Addressing the challenges identified within the thesis has in fact led to the development of a flexible and mature framework for correlated time series forecasting. Although the methodology currently appears perfectly adequate for modeling pairwise dependencies at a fixed spatiotemporal resolution, it falls short in capturing the input dynamics at multiple spatiotemporal scales. This chapter presents a first step toward graph-based methods enabling the processing of input time series at different level of aggregation along the spatial dimension. To do so, we exploit ideas for hierarchical time series forecasting [Hyndman et al., 2011] to hierarchically group time series into clusters. Each level of the resulting hierarchy, will then correspond to a specific level of aggregation. In the following, we propose a novel and comprehensive graph-based framework for hierarchical time series clustering and forecasting. Our approach unifies hierarchical time series processing, graph pooling operators, and graph-based neural forecasting methods and results in a learning architecture for multi-step ahead forecasting operating at different levels of spatial resolution. Hierarchical and relational dependencies are embedded as inductive biases into the processing by exploiting neural message passing and graph pooling operators [Grattarola et al., 2022]. The proposed methodology, named *Hierarchical Graph Predictor* (HiGP) can propagate representations along the hierarchical structure and ensure the coherency of predictions w.r.t. aggregation constraints. In particular, we focus on settings where the hierarchical structure is not given but learned directly from data.

Section 9.1 discusses hierarchical time series processing, graph pooling

methods, and related works. Section 9.2 introduces the problem settings and provides preliminary concepts on hierarchical time series. Section 9.3 presents the proposed methods, first by assuming the existence of a predefined hierarchy and then providing a method to learn clusters directly from data. Section 9.4 empirically validate the proposed framework. Section 9.5, then, discusses the main outcomes of the research and indicates possible future directions.

Reference papers The content of the chapter is partly based on material from the following papers.

- Andrea Cini, Danilo Mandic, and Cesare Alippi. Graph-based Time Series Clustering for End-to-End Hierarchical Forecasting. *International Conference on Machine Learning*, 2024

9.1 Hierarchical time series and graph clustering

In most applications, collections of related time series can be organized and aggregated within a hierarchical structure. One practical example is forecasting energy consumption profiles which can be aggregated at the level of individual households as well as at city, regional, and national scales [Taieb et al., 2021]. Similar arguments can be made for forecasting photovoltaic production [Yang et al., 2017b], financial time series [Athanasopoulos et al., 2020], and the influx of tourists [Athanasopoulos et al., 2009], to name a few relevant application domains. By exploiting aggregation constraints, forecasts at different levels can be combined to obtain predictions at different levels of resolution. Similarly, coherency constraints can be used to regularize forecasts obtained for the different levels by considering *forecast reconciliation* (FR) methods [Hyndman et al., 2011; Wickramasuriya et al., 2019; Panagiotelis et al., 2023]. Said differently, constraining forecasts at different levels to "add up" *can* positively impact forecasting accuracy. Based on similar ideas, cluster-based aggregate forecasting methods learn to predict aggregates of clustered time series as an intermediate step for obtaining forecasts for the total aggregate [Alzate and Sinn, 2013; Fahiman et al., 2017; Cini et al., 2020]. The idea underlying both approaches is that combining multiple forecasts reduces variance, an observation dating back to Bates and Granger [1969]. In particular, FR is a special case of forecast combinations [Hollyman et al., 2021]. We argue that hierarchical representations can complement the relational inductive biases used so far to forecast groups of correlated time series.

Graph pooling and clustering The combination of GDL methods and hierarchical representations have been considered in the context of static graph by introducing graph pooling operators [Ying et al., 2018; Grattarola et al., 2022; Bacciu et al., 2023], i.e., operators that corsen the input graph often by grouping subsets of nodes Bianchi et al. [2020a]. Graph pooling methods enable GNN architectures to learn how to cluster nodes and obtain hierarchical, higher-order, graph representations that can be tailored to the task at hand [Bianchi and Lachi, 2023]. Yet the application of learnable graph pooling operators and the combination of hierarchical and relational constraints are underexplored in graph-based forecasting. HiGP fills this void by unifying graph-based and hierarchical representations for time series forecasting.

9.1.1 Related work

Hierarchical forecasting Hierarchical forecasting is a widely studied problem in time series analysis [Hyndman and Athanasopoulos, 2018; Hyndman et al., 2011]. The standard approach consists of obtaining (possibly independent) forecasts for (a subset of) the time series in the hierarchy in a first stage and then, in a separate step, reconciling and combining them to obtain (possibly coherent) predictions for the full hierarchy [Hyndman et al., 2011; Ben Taieb and Koo, 2019; Wickramasuriya et al., 2019]. In particular, MinT [Wickramasuriya et al., 2019] allows for obtaining optimal reconciled forecasts given a set of unbiased H-step-ahead predictions and the covariance matrix of the associated residuals. Analogous reconciliation methods have also been developed for probabilistic forecasts [Wickramasuriya, 2023; Taieb et al., 2017; Corani et al., 2021]. End-to-end methods have been instead proposed in the context of deep learning for time series forecasting by exploiting the hierarchical structure either as a hard [Rangapuram et al., 2021; Zhou et al., 2023; Das et al., 2023] or soft constraint [Paria et al., 2021; Han et al., 2021]. Notably, Rangapuram et al. [2021] incorporate the reconciliation step within the neural architecture as a differentiable convex optimization layer [Agrawal et al., 2019] and obtain probabilistic forecasts by MC sampling. None of these methods consider relational dependencies among and within the levels of the hierarchical structure.

Hierarchical graph-based architectures Graph pooling operators have been widely studied in GNN models for i.i.d. data [Grattarola et al., 2022; Bianchi and Lachi, 2023], but their application to time series data has received limited attention. Dense trainable pooling methods [Ying et al., 2018; Bianchi et al., 2020a; Hansen and Bianchi, 2023] learn soft cluster assignment regularized by considering the graph structure. Sparse approaches, instead, produce hard cluster assignments usually learned by exploiting both the graph structure and a learned ranking on the nodes [Bacciu et al., 2023; Gao and Ji, 2019]. Finally, non-trainable methods exploit a clustering of the nodes performed independently from the trained model [Bianchi et al., 2020b; Dhillon et al., 2007]. Pyramidal graph-based architectures have also been exploited in reservoir computing [Bianchi et al., 2022]. Rangapuram et al. [2023] have used GNNs to propagate representation in temporal hierarchies. Concerning STGNNs, hierarchical representations have been exploited in specific domains such as traffic analytics [Yu et al., 2019; Guo et al., 2021a; Hermes et al., 2022], air quality monitoring [Chen et al., 2021b], financial time series [Arya et al., 2023], and pandemic forecasting [Ma et al., 2022]. In particular, Yu et al.

[2019] propose a spatiotemporal graph U-network [Gao and Ji, 2019] where representations are pooled and then un-pooled to obtain a hierarchical processing of the time series. However, most of the above methods rely on fixed cluster assignments; furthermore, none of them directly address the hierarchical time series forecasting problem by optimizing predictions at each level of the hierarchy to learn cluster assignments and taking into account coherency constraints.

9.2 Preliminaries

This section introduces preliminary concepts and provides the problem settings.

9.2.1 Operational settings

We consider the framework introduced in Chapter 3, but focus on univariate time series. We also assume that simple scalar weights constitute edge attributes. In particular, we consider a N univariate observations $\boldsymbol{X}_t \in \mathbb{R}^{N\times 1}$ at each time step t, with associated exogenous variables $\boldsymbol{U}_t \in \mathbb{R}^{N\times d_u}$; to simplify the notation we assume static attributes $\boldsymbol{V}$ to be concatenated to exogenous variables $\boldsymbol{U}_t \in \mathbb{R}^{N\times d_u}$ at each time step. We also assume pairwise relationships among time series to be encoded by a static weighted adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ (and associated edge weights $\mathcal{E}$); extensions beyond these settings are relatively straightforward. As usual, we focus on the multi-step time series forecasting problem and point predictors.

Reference model As a reference model family $\mathcal{F}(\,\cdot\,;\boldsymbol{\theta})$, we consider a TTS model (see Section 3.3.4) paired with local learnable node embeddings $\boldsymbol{Q} \in \mathbb{R}^{N\times d_q}$ (Chapter 5) where the input time series are processed by a temporal encoder followed by a stack of MP layers such that

$$\begin{aligned} \boldsymbol{h}_t^{i,0} &= \textsc{SeqEnc}\left(\boldsymbol{x}^i_{t-W:t}, \boldsymbol{u}^i_{t-W:t}, \boldsymbol{q}^i\right), \\ \boldsymbol{H}_t^{l+1} &= \mathrm{MP}_l(\boldsymbol{H}_t^l, \mathcal{E}). \end{aligned} \tag{9.1}$$

Note that in this chapter we used subscripts to denote the layer while superscripts refer to the level of the hierarchy (see the next sections). Predictions can then be obtained by using any decoder, e.g., an MLP as

$$\hat{\boldsymbol{x}}^i_{t:t+H} = \mathrm{MLP}\left(\boldsymbol{h}_t^{i,L}\right). \tag{9.2}$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

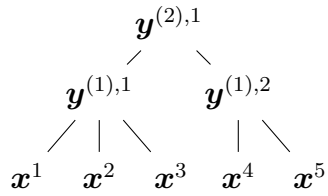


Figure 9.1. Example of hierarchical time series from [Hyndman and Athanasopoulos, 2018].

9.2.2 Hierarchical time series

In the hierarchical setting, the set of raw time series is augmented by considering additional sequences obtained by progressively aggregating those at the level below, thus building a pyramidal structure. In particular, *bottom* observations (raw time series) are denoted as $\boldsymbol{Y}_t^{(0)} = \boldsymbol{X}_t$, while $\boldsymbol{Y}_t^{(k)} \in \mathbb{R}^{N_k \times 1}$, with $k > 0$, indicates values of N_k series obtained by aggregating (e.g., summing up) a partition of $\boldsymbol{Y}_t^{(k-1)}$. The full collection of both raw and aggregated observations is denoted by matrix $\boldsymbol{Y}_t \in \mathbb{R}^{N_y \times 1}$, with $N_y = \sum_{k=0}^{K-1} N_k$, obtained by stacking the $\boldsymbol{Y}_t^{(k)}$ matrices vertically in decreasing order w.r.t. index k. In general, the level of the hierarchy is denoted as a superscript between parentheses. The aggregation constraints can be encoded in an aggregation matrix $\boldsymbol{C} \in \{0,1\}^{(N_y - N) \times N}$ such that the i-th aggregate time series can be obtained as $\boldsymbol{y}_t^{i+N} = \sum_{j=1}^{N} c_{ij} \boldsymbol{x}_t^j$, i.e., by summing the bottom-level observations given the hierarchical constraints[1]. Given the above, the following relationships hold:

$$\boldsymbol{Y}_t = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{I} \end{bmatrix} \boldsymbol{X}_t, \qquad \boldsymbol{M}\boldsymbol{Y}_t = \begin{bmatrix} \boldsymbol{I} \mid -\boldsymbol{C} \end{bmatrix} \boldsymbol{Y}_t = \boldsymbol{0}, \tag{9.3}$$

where $\boldsymbol{I}$ indicates an identity matrix of appropriate dimensions and $|$ the matrix concatenation operator. Figure 9.1 provides an example of a time series hierarchy with the associated aggregation matrix. A forecast $\widehat{\boldsymbol{Y}}_t$ is said to be *coherent* if the equality constraints in Equation 9.3 holds, i.e., if $\boldsymbol{M}\widehat{\boldsymbol{Y}}_t = \boldsymbol{0}$. As discussed in the following, learning to forecast time series at different resolutions can act as an effective regularization mechanism, even when the hierarchical structure is not predefined.

[1]Note that superscript $i + N$ does not refer to the level of the hierarchy but to the i-th element of the entire flattened collection $\boldsymbol{Y}_t$.

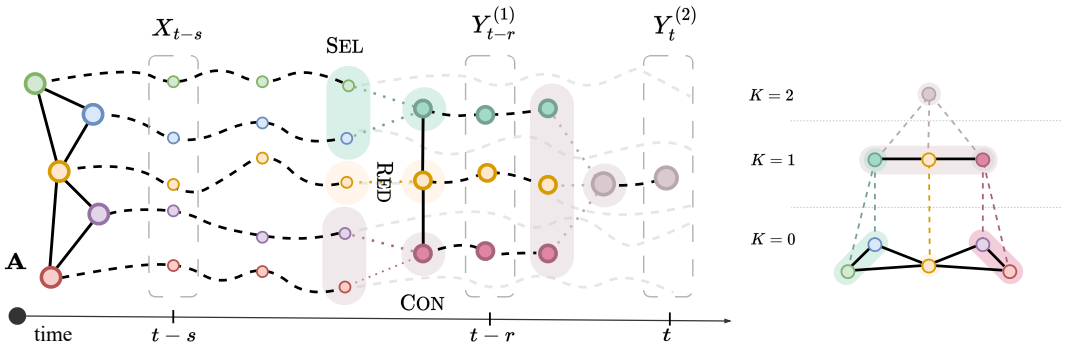


Figure 9.2. Time series with a hierarchical relational structure. **(Left)** Graphical representation of hierarchical time series with graph-side information; SRC operators allow for modeling relationships among the time series in the hierarchy. **(Right)** Pyramidal graph encompassing both hierarchical and relational dependencies; each pair of levels constitutes a bipartite graph.

9.3 Graph-based hierarchical clustering and forecasting

This section presents our approach to graph-based hierarchical time series forecasting. We start by discussing how to incorporate the hierarchical structure of the problem into a graph-based neural architecture (Section 9.3.1); then, we focus on our target setting and show how the hierarchical structure can be directly learned from data by exploiting trainable graph pooling operators (Section 9.3.2). Finally, we introduce an appropriate forecasting reconciliation mechanism to obtain forecasts coherent w.r.t. the learned hierarchy (Section 9.3.3).

9.3.1 Graph-based hierarchical forecasting

Embedding the hierarchical structure into the processing requires defining proper operators. In particular, we aim at designing a *pyramidal* processing architecture where each layer corresponds to a level of the time series hierarchy and has its own topology, related to those at the adjacent layers by the hierarchical structure. To obtain such processing, operators have to be specified to control how information is propagated among and within the levels of the hierarchy; we exploit the connection to graph pooling for defining such operators within the *select, reduce, connect* (SRC) framework [Grattarola et al., 2022]. In particular, we use SRC building blocks as a high-level formalization of the operators required to perform clustering, aggregation, and graph rewiring at each level

of the hierarchy. The three operators are defined as follows, by indicating as $\boldsymbol{H}_t^{(k)} \in \mathbb{R}^{N_k \times d_h}$ a feature matrix corresponding to representations at the k-th level of the hierarchy.

Select The selection operator $\textsc{Sel}(\,\cdot\,)$ outputs a mapping from input nodes into supernodes (i.e., clusters) given by the aggregation constraints at each level. The mapping can be encoded in a selection matrix $\textsc{Sel}(\boldsymbol{H}_t^{(k)}, \dots) = \boldsymbol{S}_k \in \{0,1\}^{N_{k-1} \times N_k}$ where s_{ij} is equal to 1 if and only if the i-th time series at level $k-1$ is mapped to the j-th aggregate at the k-th level. If the hierarchy is predefined, then the selection mechanism is given; conversely, learning a selection matrix is the key challenge for designing an end-to-end architecture and will be discussed in Section 9.3.2.

Reduce (and Lift) The reduction function $\textsc{Red}(\,\cdot\,)$ aggregates node features and propagates information from the k-th level to the adjacent upper level in the hierarchy. Reduction can be obtained by summation, i.e., $\textsc{Red}(\boldsymbol{H}_t^{(k-1)}, \boldsymbol{S}^{(k)}) \doteq \boldsymbol{S}^{(k)^T} \boldsymbol{H}_t^{(k-1)}$, but other choices are possible. In practice, reduction is used in HiGP to propagate information along the pyramidal structure by aggregating node representations and implementing an inter-level MP mechanism (see Equation 9.6). Similarly, we define the *lift* operator as $\textsc{Lift}(\boldsymbol{H}_t^{(k+1)}, \boldsymbol{S}^{(k)}) \doteq \boldsymbol{S}^{(k)} \boldsymbol{H}_t^{(k+1)}$, i.e., as an upsampling the pooled graph to the original size obtained by mapping each supernode back to the aggregated nodes.

Connect The connect operator $\textsc{Con}(\,\cdot\,)$ defines how the topology of the input graph is rewired after each aggregation step. There are several possible choices; we consider the rewiring where each pair of supernodes is connected by an edge with a weight obtained by summing weights of the edges from one subset to the other, i.e., $\textsc{Con}(\boldsymbol{S}^{(k)}, \boldsymbol{A}^{(k-1)}) \doteq \boldsymbol{S}^{(k)^T} \boldsymbol{A}^{(k-1)} \boldsymbol{S}^{(k)}$, where $\boldsymbol{A}^{(k)}$ indicates the adjacency matrix w.r.t. k-th level.

These operators can be used to design neural processing architectures to match the inductive biases coming from the hierarchical structure. Figure 9.2 provides a graphical illustration of how these operators can be used to implement a hierarchical processing architecture. In particular, the figure shows subsequent applications of the selection, reduction and connection operators allow for operating on a progressively coarser graph structure accounting for higher-order dependencies. By exploiting the introduced operators, we can move from the

reference architecture in Equation 9.1 to a hierarchical TTS model operating as

$$\boldsymbol{h}_t^{(k),i,0} = \textsc{SeqEnc}^{(k)}\left(\boldsymbol{y}_{t-W:t}^{(k),i}, \boldsymbol{u}_{t-W:t}^{(k),i}, \boldsymbol{q}^{(k),i}\right), \qquad \text{Temporal enc.} \quad (9.4)$$

$$\boldsymbol{Z}_t^{(k),l} = \mathrm{MP}_l^{(k)}\left(\boldsymbol{H}_t^{(k),l}, \mathcal{E}^{(k)}\right), \qquad \text{Intra-level prop.} \quad (9.5)$$

$$\boldsymbol{H}_t^{(k),l+1} = \mathrm{UP}_l^{(k)}\left(\boldsymbol{Z}_t^{(k),l}, \underbrace{\boldsymbol{S}^{(k)^T}\boldsymbol{Z}_t^{(k-1),l}}_{\textsc{Red}^{(k)}}, \underbrace{\boldsymbol{S}^{(k)}\boldsymbol{Z}_t^{(k+1),l}}_{\textsc{Lift}^{(k)}}\right). \qquad \text{Inter-level prop.} \quad (9.6)$$

Equation 9.4 to 9.6 need further consideration to be fully appreciated. Matrix $\boldsymbol{H}_t^{(k),l}$ indicates here representations w.r.t. the t-th time step obtained at the l-th MP layer for time series at the k-th level of the hierarchy (note the distinction between layers of MP and levels of the hierarchy). Compared to the model in Equation 9.1, the hierarchical constraints add further structure to the processing. As shown in Equation 9.4, each time series is at first encoded along the temporal dimension by an encoder which can be either shared or different for each aggregation level. Then, representations are processed by a stack of layers propagating information within and among levels. As shown in Equation 9.6, the representations are updated at each step by an update function $\mathrm{UP}_l^{(k)}(\,\cdot\,)$ (e.g., an MLP) taking as an input (1) the output $\boldsymbol{Z}_t^{(k),l}$ of an MP layer w.r.t. the graph topology at the k-th level (Equation 9.5), (2) aggregated features from the level $k-1$ and (3) the features corresponding to each node's supernode obtained by lifting $\boldsymbol{H}_t^{(k+1),l}$. Learnable parameters may optionally be shared among the different levels of the hierarchy. Final predictions, as in the reference architecture, can be obtained by using an arbitrary readout, i.e., a standard MLP as

$$\hat{\boldsymbol{y}}_{t:t+H}^{(k),i} = \mathrm{MLP}^{(k)}\left(\boldsymbol{h}_t^{(k),i,L}\right) \quad (9.7)$$

and by training the model to minimize the forecasting error w.r.t. all the time series and time steps, i.e.,

$$\mathcal{L}(\widehat{\boldsymbol{Y}}_{t:t+H}, \boldsymbol{Y}_{t:t+H}) \doteq \sum_{k=0}^{K-1} \ell\left(\widehat{\boldsymbol{Y}}_{t:t+H}^{(k)}, \boldsymbol{Y}_{t:t+H}^{(k)}\right). \quad (9.8)$$

Note that the model is trained to make predictions for each level of the hierarchy at once. Representation at the different levels can capture patterns at different spatial scales, less apparent at fine-grained resolutions. Indeed, the aggregation and pooling operators increase the receptive field of each filter at each level of the hierarchy. Discussion on how to further regularize predictions given the hierarchical structure is postponed to Section 9.3.3.

9.3.2 End-to-end clustering and forecasting

Learning a hierarchy and, consequently, a cluster-based forecasting architecture translates into learning a (differentiable) parametrization of the selection operator. For this task, we provide a general probabilistic framework, based on modeling cluster assignments as realizations of a parametrized categorical distribution. Then, we briefly discuss the applicability of standard graph pooling methods from the literature at the end of the section.

End-to-end clustering Similarly to popular dense trainable graph pooling operators [Bianchi et al., 2020a; Ying et al., 2018], we parametrize the selection operator with a score matrix $\boldsymbol{\Phi} \in \mathbb{R}^{N_{k-1} \times N_k}$, assigning a score ϕ_{ij} to each node-cluster pair. However, differently from previous works, we interpret such scores as (unnormalized) log-probabilities, such that

$$\begin{aligned} \boldsymbol{\Phi}^{(k)} &= \mathcal{F}_{\boldsymbol{\psi}} \left(\boldsymbol{Y}_{t-W:t}^{(k-1)}, \boldsymbol{A}^{(k-1)}, \boldsymbol{Q}^{(k-1)} \right), \\ \boldsymbol{S}^{(k)} &\sim P(\boldsymbol{S}_{ij}^{(k)} = 1) = \frac{e^{\phi_{ij}^{(k)}/\tau}}{\sum_j e^{\phi_{ij}^{(k)}/\tau}}, \end{aligned} \tag{9.9}$$

where τ is a temperature hyperparameter, while $\mathcal{F}_{\boldsymbol{\psi}}(\,\cdot\,)$ indicates a generic trainable function with trainable parameters $\boldsymbol{\psi}$. The conditioning on the input window $\boldsymbol{Y}_{t-W:t}^{(k-1)}$ can be dropped to obtain static cluster assignments; furthermore, depending on the dimensionality of the problem, the score matrix might also be parametrized directly as $\boldsymbol{\Phi} = \boldsymbol{\psi}$. Node embeddings and aggregates for the k-th level are then obtained through the reduction operator as $\boldsymbol{Q}^{(k)} = \boldsymbol{S}^{(k)^T} \boldsymbol{Q}^{(k-1)}$ and $\boldsymbol{Y}_t^{(k)} = \boldsymbol{S}^{(k)^T} \boldsymbol{Y}_t^{(k-1)}$, respectively. To differentiate through the sampling of $\boldsymbol{S}^{(k)}$ we use the Gumbel softmax reparametrization trick [Jang et al., 2017; Maddison et al., 2017] followed by a discretization step to obtain hard cluster assignments via the straight-through gradient estimator [Bengio et al., 2013]. In practice, τ is set to 1 at the beginning of training and is exponentially decayed towards 0 at each training step. The above discretization step avoids soft cluster assignments that could lead to degenerate solutions given the loss in Equation 9.8. Uniform soft assignments are indeed likely to minimize the variance of the aggregate time series and thus the prediction error at levels $k > 0$.

Graph-based regularization To take the graph structure into account when learning the assignments, we exploit the min-cut regularization introduced

by Bianchi et al. [2020a], i.e., we add to the loss the term

$$\mathcal{L}^c\left(\boldsymbol{S}_\mu^{(k)}, \boldsymbol{A}^{(k-1)}\right) \doteq \tag{9.10}$$
$$-\frac{\mathrm{Tr}\left(\boldsymbol{S}_\mu^{(k)^T}\widetilde{\boldsymbol{A}}^{(k-1)}\boldsymbol{S}_\mu^{(k)}\right)}{\mathrm{Tr}\left(\boldsymbol{S}_\mu^{(k)^T}\widetilde{\boldsymbol{D}}^{(k-1)}\boldsymbol{S}_\mu^{(k)}\right)} + \left\| \frac{\boldsymbol{S}_\mu^{(k)^T}\boldsymbol{S}_\mu^{(k)}}{\left\|\boldsymbol{S}_\mu^{(k)^T}\boldsymbol{S}_\mu^{(k)}\right\|_2} - \frac{\boldsymbol{I}}{\sqrt{N_k}} \right\|_2$$

where $\widetilde{\boldsymbol{D}}^{(k-1)}$ is the degree matrix of $\widetilde{\boldsymbol{A}}^{(k-1)} \doteq \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}^{(k-1)}\boldsymbol{D}^{-\frac{1}{2}}$ (i.e., of the symmetrically normalized adjacency matrix) and $\boldsymbol{S}_\mu^{(k)} = \mathrm{softmax}\left(\boldsymbol{\Phi}^{(k)}\right)$. The first term in the equation is a continuous relaxation of the min-cut problem [Dhillon et al., 2004] incentivizing the formation of clusters that pool together connected components of the graph; the second term helps in preventing degenerate solutions by favoring orthogonal cluster assignments [Bianchi et al., 2020a].

Training procedure The training objective identified in Equation 9.8 entails that the cluster assignments are learned to minimize the forecasting error w.r.t. both the bottom time series and aggregates. As a result, time series are clustered s.t. aggregates at all levels are easier to predict, thus providing a meaningful self-supervised learning signal. Intuitively, a signal will be easier to predict if characterized *low intra-cluster variance*. At the same time, different levels in the hierarchy will benefit from reading information from diverse supernodes, thus favoring a *high inter-cluster variance*.

Alternative pooling operators Besides the clustering method described here, HiGP is compatible with any graph pooling approach from the literature (see Grattarola et al. 2022). In particular, one might be interested in exploiting non-trainable graph pooling operators that obtain cluster assignments based on the graph topology only. The latter option becomes particularly attractive when obtaining predictions w.r.t. particular sub-graphs, or localized within specific connected components of the graph topology, is relevant for the downstream application. We discussed a selection of appealing methods from the literature in Section 9.1.1 and refer to Grattarola et al. [2022] for an in-depth discussion.

9.3.3 Forecast reconciliation

As mentioned in Section 9.1, FR allows for obtaining coherent forecast w.r.t. the hierarchical constraints (Equation 9.3). Furthermore, FR can often have a positive impact on forecasting accuracy as reconciled forecasts are obtained as a

combination of the predictions made at the different levels [Hollyman et al., 2021]. We follow Rangapuram et al. [2021] and embed a (differentiable) reconciliation step within the architecture as a projection onto the subspace of coherent forecasts.

Forecast reconciliation Given (trainable) selection matrices $\boldsymbol{S}^{(1)}, \dots, \boldsymbol{S}^{(K)}$ for each level of the hierarchy, the $\boldsymbol{M}$ matrix (see Equation 9.3) can be obtained as

$$\begin{aligned} \boldsymbol{M} &= \left[\boldsymbol{I} \,\middle|\, -\boldsymbol{C} \right] = \\ &= \left[\boldsymbol{I} \,\middle|\, -\left[\prod_{k=1}^{K} \boldsymbol{S}^{(k)} \mid \prod_{k=1}^{K-1} \boldsymbol{S}^{(k-i)} \mid \dots \mid \boldsymbol{S}^{(1)} \right]^T \right]. \end{aligned} \tag{9.11}$$

Then, raw predictions $\widehat{\boldsymbol{Y}}_t$ can be mapped into reconciled (coherent) forecasts $\overline{\boldsymbol{Y}}_t$ through a projection onto the space of coherent forecasts (i.e., the null space of $\boldsymbol{M}$). The projection matrix can be computed as

$$\boldsymbol{P} \doteq \boldsymbol{I} - \boldsymbol{M}^T \left(\boldsymbol{M}\boldsymbol{M}^T \right)^{-1} \boldsymbol{M}, \qquad \overline{\boldsymbol{Y}}_t = \boldsymbol{P}\widehat{\boldsymbol{Y}}_t, \tag{9.12}$$

where $\boldsymbol{P}$ is obtained by solving the constrained optimization problem $\min_{\boldsymbol{Z}} \|\boldsymbol{Z} - \widehat{\boldsymbol{Y}}_t\|_2$ s.t. $\boldsymbol{M}\boldsymbol{Z} = \boldsymbol{0}$. Model parameters are then learned by minimizing the loss $\mathcal{L}_f \doteq \mathcal{L}(\widehat{\boldsymbol{Y}}, \boldsymbol{Y}) + \mathcal{L}(\overline{\boldsymbol{Y}}, \boldsymbol{Y}) + \lambda \mathcal{L}(\overline{\boldsymbol{Y}}, \widehat{\boldsymbol{Y}})$ where we omitted the time indices. Note that minimizing the regularization term $\mathcal{L}(\overline{\boldsymbol{Y}}, \widehat{\boldsymbol{Y}})$ is equivalent to minimizing the distance between $\widehat{\boldsymbol{Y}}_{t:t+H}$ and the space of coherent forecasts. Unfortunately, computing the inverse of $\boldsymbol{M}\boldsymbol{M}^T$ incurs the cost $\mathcal{O}(N_y^3)$ in space and $\mathcal{O}(N_y^2)$ in time, which can be prohibitive for large time series collections. However, the solution is still practical for up to a few thousand nodes (most practical applications), and the regularization term, computed as $\mathcal{L}^{reg}(\widehat{\boldsymbol{Y}}, \lambda) \doteq \lambda \|\boldsymbol{M}\widehat{\boldsymbol{Y}}\|_2$, can be used in the other cases as the only regularization. The above FR method can be seamlessly integrated into our end-to-end forecasting framework, however, many possible alternatives could be considered here. The design of ad-hoc reconciliation methods for graph-based predictors is a promising research direction for future works (see Section 9.5).

9.4 Empirical results

HiGP is validated over several settings considering the forecasting benchmarks introduced in Chapter 4. Note that in these datasets, there is no predefined hierarchical structure to start with. In particular, we focus on validating of the

proposed end-to-end clustering and forecasting architecture against relevant baselines and state-of-the-art architectures. We then provide a qualitative analysis of the learned time series clusters on datasets coming from sensor networks. Additional details and results are provided in Appendix I and the reference paper [Cini et al., 2024].

9.4.1 End-to-end hierarchical clustering and forecasting

The empirical evaluation was set up by considering the benchmark datasets introduced in Chapter 4 (**METR-LA**, **PEMS-BAY**, **AQI**, and **CER-E**) and the following baselines.

Baselines To carry out meaningful comparisons we consider the reference TTS architecture (see Equation 9.1) obtained by stacking an node-wise temporal encoder implemented by an RNN, two GNN layers, and an MLP readout as

$$\mathrm{RNN}[d_h] - \mathrm{MP}[d_h] - \mathrm{MP}[d_h] - \mathrm{FC}[d_h] - \textsc{Lin}[H]$$

where MP indicates a generic message-passing block, FC indicates a dense fully connected layer, and $\textsc{Lin}(H)$ is a linear layer with an output size corresponding to the forecasting horizon. The number of neurons in each layer is indicated as d_h. Learnable node embeddings (Chapter 5) are concatenated to the input before both the recurrent encoder and after the MP layers. We compare the performance of different MP schemes commonly used in state-of-the-art graph-based forecasting architectures. In particular, the considered alternatives include the standard graph convolution (**GConv-TTS**, Kipf and Welling 2017), the bidirectional diffusion convolution operator (**Diff-TTS**, Li et al. 2018), a more advanced gated MP scheme (**Gated-TTS**, Cini et al. 2023c), and a hierarchical Graph U-Net (**GUNet-TTS**, Gao and Ji 2019). We use a standard **GRU** [Cho et al., 2014] as sequence encoder for all the baselines. Finally, we denote by **FC-RNN** the baseline which considers the input sequences as a single multivariate time series and by **RNN** the global univariate model. The number of neurons d_h is selected for each dataset on the validation set (more details in Appendix I), while the other hyperparameters are kept fixed among baselines (see Cini et al. [2024]). The **HiGP-TTS** model is implemented following the above template and Equation 9.4–9.6. Notably, the only architectural differences w.r.t. the baselines is the addition of a hierarchical propagation step after each MP layer and the addition of separate readouts for each level of the hierarchy. HiGP is trained end-to-end as to minimize the forecasting error

Table 9.1. Forecasting performance on benchmark datasets (5 runs). Best result in **bold**, second best underlined.

Models	METR-LA		PEMS-BAY		CER-E		AQI	
	MAE	MRE (%)	MAE	MRE (%)	MAE	MRE (%)	MAE	MRE (%)
RNN	$3.543_{\pm.005}$	$6.134_{\pm.008}$	$1.773_{\pm.001}$	$2.839_{\pm.001}$	$4.57_{\pm.00}$	$21.65_{\pm.01}$	$14.00_{\pm.03}$	$21.84_{\pm.05}$
FC-RNN	$3.566_{\pm.018}$	$6.174_{\pm.031}$	$2.305_{\pm.006}$	$3.690_{\pm.009}$	$7.13_{\pm.02}$	$33.77_{\pm.11}$	$18.33_{\pm.11}$	$28.59_{\pm.18}$
GConv-TTS	$3.071_{\pm.008}$	$5.317_{\pm.015}$	$1.584_{\pm.006}$	$2.536_{\pm.009}$	$4.12_{\pm.02}$	$19.50_{\pm.08}$	$12.30_{\pm.02}$	$19.20_{\pm.03}$
Diff-TTS	$3.012_{\pm.005}$	$5.214_{\pm.008}$	$1.569_{\pm.004}$	$2.512_{\pm.006}$	$4.11_{\pm.02}$	$19.47_{\pm.11}$	$12.24_{\pm.04}$	$19.10_{\pm.05}$
Gated-TTS	$3.027_{\pm.008}$	$5.240_{\pm.013}$	$1.582_{\pm.006}$	$2.533_{\pm.009}$	$4.13_{\pm.01}$	$19.54_{\pm.06}$	$12.07_{\pm.02}$	$18.83_{\pm.03}$
GUNet-TTS	$3.057_{\pm.016}$	$5.292_{\pm.028}$	$1.575_{\pm.006}$	$2.522_{\pm.010}$	$4.08_{\pm.02}$	$19.32_{\pm.10}$	$12.25_{\pm.03}$	$19.11_{\pm.05}$
HiGP-TTS (C)	$3.034_{\pm.008}$	$5.253_{\pm.013}$	$1.567_{\pm0.005}$	$2.508_{\pm0.008}$	$4.11_{\pm.07}$	$19.45_{\pm.34}$	$12.13_{\pm.02}$	$18.92_{\pm.04}$
HiGP-TTS (D)	$3.009_{\pm.005}$	$5.209_{\pm.008}$	$\mathbf{1.566}_{\pm.005}$	$\mathbf{2.506}_{\pm.008}$	$4.12_{\pm.06}$	$19.49_{\pm.30}$	$12.10_{\pm.01}$	$18.88_{\pm.02}$
HiGP-TTS (G)	$\mathbf{3.007}_{\pm.009}$	$\mathbf{5.205}_{\pm.016}$	$1.568_{\pm.008}$	$2.510_{\pm.013}$	$\mathbf{4.05}_{\pm.01}$	$\mathbf{19.20}_{\pm.03}$	$\mathbf{12.02}_{\pm.04}$	$\mathbf{18.75}_{\pm.06}$

w.r.t. the aggregates corresponding to the learned clusters. For this experiment, we use a static learnable hierarchical structure with 3 levels consisting of raw time series at the bottom, 20 supernodes in the middle level, and the total aggregate as the single time series at the top level. Selection matrices are learned directly by parametrizing the associated log-probabilities with tables of trainable parameters.

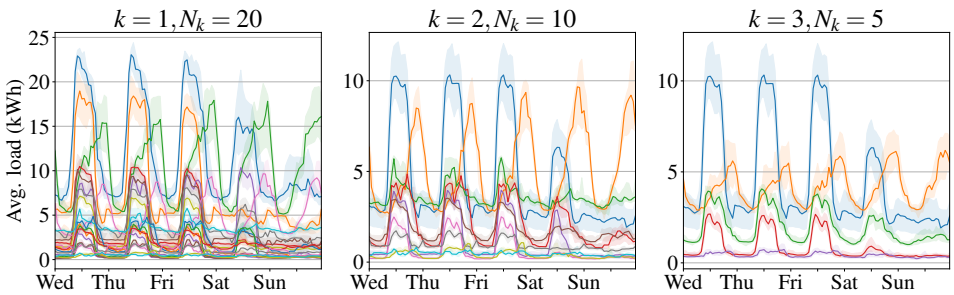
Results Table 4.3 show the results of the extensive empirical evaluation. We report HiGP forecasting accuracy w.r.t. 3 different MP schemes; in particular, (C), (D), and (G) indicate respectively the standard graph convolution, the diffusion convolution operator and the gated MP operator cited above. HiGP variants are among the best-performing methods in all the considered settings. Notably, hierarchical forecasting does not only act as self-supervision to learn cluster assignments but also provides a positive inductive bias that results – on average – in improved forecasting accuracy w.r.t. the flat architectures. Conversely, the GUNet baseline provides a comparison with a standard hierarchical MP architecture which, in this case, underperforms.

Comparison against the state of the art Next, we perform an additional experiment by taking advantage of the popularity of METR-LA and PEMS-BAY as traffic forecasting benchmarks and compare HiGP against specialized state-of-the-art architectures. We consider the following baselines: 1) **DCRNN** [Li et al., 2018], i.e., a recurrent introduced in Chapter 4; 2) Graph

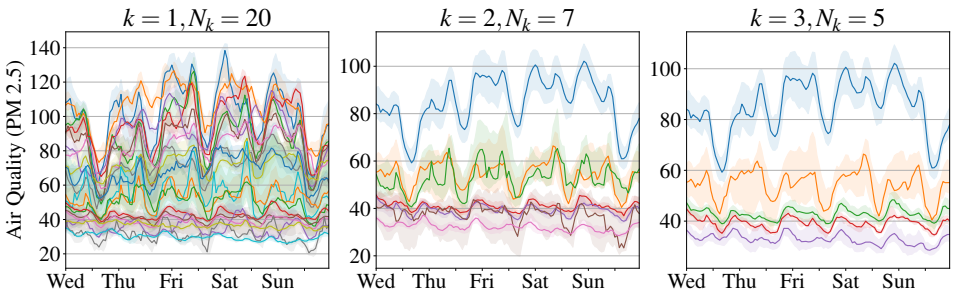
Table 9.2. Results on traffic datasets (5 runs). Best results in **bold**, second best underlined.

	Models	MAE		
		15 min.	30 min.	60 min.
METR-LA	DCRNN	$2.82_{\pm.00}$	$3.23_{\pm.01}$	$3.74_{\pm.01}$
	GWNet	$2.72_{\pm.01}$	$3.10_{\pm.02}$	$3.54_{\pm.03}$
	Gated-GN	$2.72_{\pm.01}$	$3.05_{\pm.01}$	$3.44_{\pm.01}$
	SGP	$2.69_{\pm.00}$	$3.05_{\pm.00}$	$3.45_{\pm.00}$
	HiGP (T)	$\mathbf{2.68}_{\pm.01}$	$\mathbf{3.02}_{\pm.01}$	$\mathbf{3.40}_{\pm.01}$
	No rel. prop.	$2.80_{\pm.01}$	$3.14_{\pm.01}$	$3.47_{\pm.02}$
	No hier. prop.	$\mathbf{2.68}_{\pm.01}$	$3.03_{\pm.02}$	$3.43_{\pm.02}$
PEMS-BAY	DCRNN	$1.36_{\pm.00}$	$1.71_{\pm.00}$	$2.08_{\pm.01}$
	GWNet	$1.31_{\pm.00}$	$1.64_{\pm.01}$	$1.94_{\pm.01}$
	Gated-GN	$1.32_{\pm.00}$	$1.63_{\pm.01}$	$1.89_{\pm.01}$
	SGP	$\mathbf{1.30}_{\pm.00}$	$\mathbf{1.60}_{\pm.00}$	$1.88_{\pm.00}$
	HiGP (T)	$1.31_{\pm.00}$	$1.61_{\pm.00}$	$\mathbf{1.87}_{\pm.00}$
	No rel. prop.	$1.32_{\pm.00}$	$1.63_{\pm.00}$	$1.88_{\pm.01}$
	No hier. prop.	$1.31_{\pm.00}$	$1.63_{\pm.00}$	$1.89_{\pm.00}$

WaveNet (**GWNet**), i.e., the already mentioned popular T&S graph convolutional model introduced by Wu et al. [2019]; 3) **Gated-GN** [Satorras et al., 2022] the a gated MP architecture operating on a fully connected graph (see also Section 8.5); 4) **SGP** [Cini et al., 2023a], i.e., the scalable architecture exploiting a randomized spatiotemporal encoder as presented Chapter 8. In this context, we tuned the HiGP architecture by simply adding residual connections and using a deeper MLP decoder; the tuned architecture is denoted as HiGP (T). The simulation results for multi-step-ahead forecasting in the traffic datasets, provided in Table 9.2, show that HiGP can achieve state-of-the-art forecasting accuracy. Additionally, the same table reports an ablation study of the proposed architecture. In particular, we consider two variants of the model: the first is characterized by the removal of all the MP layers, while the second does not perform any propagation of the learned representations through the learned hierarchy. Results show that both aspects have a significant impact on forecasting accuracy.



(a) CER-E dataset.



(b) AQI dataset.

Figure 9.3. Hierarchical cluster assignments learned by HiGP on 2 benchmark datasets. The models have been trained with a 5-level hierarchy. The figure shows, from left to right, the median for the clusters corresponding to levels from 1 to 3. The shaded areas correspond to 0.6 and 0.4 quantiles.

9.4.2 Cluster analysis

We analyze clusters extracted by HiGP on the CER-E and AQI datasets. Ideally, we would like to cluster customers w.r.t. their consumption patterns in the first case, and to partition air quality monitoring stations w.r.t. their dynamics and spatial location in the second. As discussed in Section 9.3.2, HiGP learns the cluster assignments by minimizing the forecasting error at each level of the hierarchy end-to-end. This form of self-supervision rewards, then, the formation of clusters resulting in aggregates that are easy to predict. At the same time, clusters are formed by taking the graph structure into account (Equation 9.10). We configure HiGP to learn 3 hierarchical cluster assignments and show the result of the procedure in Figure 9.3. In both scenarios, HiGP extracts meaningful clusters with aggregates exhibiting different patterns. Notably, each level corresponds to progressively smoother dynamics.

9.5 Discussion and future directions

We introduced the *Hierarchical Graph Predictor*, a methodological framework unifying relational and hierarchical inductive biases in deep learning architectures for time series forecasting. HiGP has been designed to learn hard cluster assignments end-to-end, by taking the graph structure into account and minimizing the forecasting error w.r.t. the resulting aggregates and bottom-level time series. Performance on relevant benchmarks supports the validity of the approach which, as we show, can also learn meaningful hierarchical cluster assignments.

Future directions As we discussed, we believe that methods like HiGP, able to operate at different scales, have the potential of enabling new powerful graph-based time series processing architectures. There are many possible extensions to the model presented here, which can be seen as a starting point for several specific studies and research directions. Future works might focus more on the clustering aspect and investigate additional auxiliary objectives to provide more supervision to the procedure. Alternative reconciliation strategies should be assessed as well, together with their impact on the learned cluster assignments and forecasting accuracy. Future research could also apply HiGP-like methods to settings where the hierarchical constraints are predefined. Finally, extensions of the framework to multivariate, heterogenous, and irregularly sampled time series [Marisca et al., 2024] would make the approach applicable to additional relevant and practical application domains. We refer to Chapter 10 for additional discussion on how methods operating at adaptive spatiotemporal resolution might constitute a relevant topic for future research in the field.

Chapter 10

Conclusion

In this research, we introduced a novel framework under which graph deep learning forecasting models can be designed, understood, and deployed to tackle real-world applications Cini et al. [2023b]. In particular, in Chapter 2 and 3, we proposed a comprehensive methodology for correlated time series forecasting based on graph representations and graph neural networks [Cini et al., 2023b]. In specifying the framework, we discussed available design choices and their implications, providing guidelines to the practitioner. Chapter 4, then, introduced benchmark datasets and empirically evaluated a selection of reference architectures.

Chapter 5 introduced and discussed hybrid global and local forecasting models [Cini et al., 2023c]. In particular, we identified learnable node embeddings as a methodology to effectively and efficiently design such architectures. We then discussed how to structure the learning of such embeddings and the impact of regularizations in transfer learning scenarios. Our work in this direction provides necessary and key methodologies for the understanding and design of effective graph-based predictors.

We pioneered several GDL methodologies for missing data imputation and tackled the processing of irregular time series and sparse observations [Cini et al., 2022b; Marisca et al., 2022; De Felice et al., 2024]. In Chapter 6, we discussed the problem and introduced GRIN and SPIN, novel approaches to time series imputation exploiting graph representations and relational inductive biases. Compared to state-of-the-art baselines, our approach offers higher flexibility and achieves better reconstruction accuracy on a wide range of relevant benchmarks.

In Chapter 7, we proposed a comprehensive framework for learning graph distributions from correlated time series [Cini et al., 2023d]. In particular, we introduced variance-reduced score-based gradient estimators allowing for

keeping message-passing computations sparse throughout the training and inference phases. Empirical results validate the effectiveness of the methodology in controlled environments and benchmark data.

Chapter 8 introduced SGP, a scalable architecture for graph-based time series forecasting based on training-free randomized encoder [Cini et al., 2023a]. SGP is competitive against the state of the art, while greatly improving scalability to processing data coming from large sensor networks.

Chapter 9 introduced HiGP, a methodology unifying relational and hierarchical inductive biases in deep learning architectures for time series forecasting [Cini et al., 2024]. HiGP clusters and forecasts input time series at different levels of spatial resolution.

The next section presents relevant topics for future research in the field.

10.1 Future directions

We identify a selection of promising future research directions which, we believe, have the potential to widen the applicability of graph-based forecasting models.

Spatial and temporal hierarchies HiGP is one of the first examples of a model that can operate on aggregate representations and multiple spatial resolutions (see Chapter 9). Future research could focus on taking advantage of the spatiotemporal structure of the data to process observations at different scales both in time [Athanasopoulos et al., 2017] and in space [Hyndman et al., 2011]. There have been several deep learning methods for hierarchical time series forecasting [Rangapuram et al., 2021, 2023; Challu et al., 2023; Zhou et al., 2023; Han et al., 2021], but only a few (e.g, [Marisca et al., 2024]) exploit this framework in the context of grpah-based representations. Future works should investigate methodologies to process spatiotemporal time series in an integrated and hierarchical fashion across both time and space while accounting for the coherency of the associated forecasts.

Continuous space-time models Continuous time (and/or space) models based on differential equations have become popular in deep learning [Lu et al., 2021; Chamberlain et al., 2021; Kovachki et al., 2023; Gravina et al., 2023]. Approaches operating in continuous time are particularly appealing when dealing with irregularly sampled data [Shukla and Marlin, 2020; Chen et al., 2018a; Kidger et al., 2020]. Rubanova et al. [2019] pioneered research of these methods in time-series applications. Graph-based approaches have

been recently adopting analogous approaches to modeling dynamic relational data [Huang et al., 2021; Fang et al., 2021; Choi et al., 2022; Jin et al., 2022; Liu et al., 2023d; Luo et al., 2023; Gravina et al., 2024a]. Particularly related to our work, Gravina et al. [2024b] propose a continuous time model exploiting GNNs to learn ODEs for modeling irregularly sampled correlated time series. Future works should address the design of unifying frameworks for continuous space-time modeling. In particular, models that process space and time continuously in an integrated fashion should be further explored.

Probabilistic forecasts and uncertainty quantification While we focused on point predictions, deep learning methods have been widely applied to probabilistic forecasting [Salinas et al., 2020; Wen et al., 2017; Rangapuram et al., 2018; Gasthaus et al., 2019; de Bézenac et al., 2020]. One can in principle exploit (most of) such methodologies to make STGNNs output probabilistic predictions. In this regard, Wu et al. [2021a] carry out a study of several standard uncertainty estimation techniques in the context of spatiotemporal forecasting. However, while specialized probabilistic STGNN architectures exist (e.g, Pal et al. [2021]; Chen et al. [2021a]), the topic is underexplored [Jin et al., 2023b]. To fill this void, future research should further explore the use of relational inductive biases to obtain calibrated probabilistic forecasts and uncertainty estimates.

10.2 Final remarks

This thesis offers a foundation for future research to build on and plenty of resources for practitioners to design effective and scalable graph-based forecasting architecture. The main outcome of the thesis is a set of graph deep learning methods aimed at enriching modern time series forecasting practices. Empirical results on benchmark data show the remarkable potential of the technology.

The methodology is sound and the foundations of the framework are now solid. Research carried out so far offers the support needed to bridge the gap with the real world and enables the design of forecasting architectures targeting practical, high-impact, applications.

Appendix A

Torch Spatiotemporal

Figure A.1. Torch Spatiotemporal logo.

In the context of the thesis, we developed Torch Spatiotemporal (TSL) [Cini and Marisca, 2022], a library for prototyping graph deep learning models for processing time series collections. We provide an overview of its main functionalities and refer to the official documentation[1] for more details. Figure A.1 shows the logo of the library.

Data handling and preprocessing TSL offers utilities for handling and preprocessing collections of time series. In particular, we implemented ad-hoc modules to allow for easy loading and batching of the data together with the associated relational information. TSL adopts and implements the graph-based framework and representation introduced in Section 3.1.

Model prototyping and training TSL builds upon PyTorch [Paszke et al., 2019] and PyTorch Geometric (PyG) [Fey and Lenssen, 2019] and provides utilities for enabling fast prototyping of STGNNs following as described in

[1] https://torch-spatiotemporal.readthedocs.io/en/latest/

Chapter 3, allowing for easily implementing custom STMP lyers. Additionally, TSL relies on the PyTorch Lightning framework [Falcon and The PyTorch Lightning team, 2019] for training and inference pipelines.

Benchmarks and baselines TSL includes several benchmark detasets and baselines from the state of the art (e.g., see Chapter 4). This allows to accelerate research on the topic dramatically, as it enables the researcher to quickly go from prototyping a model to comparing it against validated implementations of reference architectures on plenty of datasets.

A.1 Related work

PyG [Fey and Lenssen, 2019] is the most widely used library for developing GNNs. As the name suggests, PyG is based on PyTorch [Paszke et al., 2019] and offers utilities to process both static and temporal relational data as well. Specialized libraries that implement models from the temporal graph learning literature also exist [Rozemberczki et al., 2021]. For what concerns deep learning for time series processing, the PyTorch ecosystem offers several options such as GluonTS [Alexandrov et al., 2020], PyTorch Forecasting[2], Neural Forecast [Olivares et al., 2022] and BasicTS [Liang et al., 2022a].

[2] https://github.com/jdb78/pytorch-forecasting

Appendix B

Performance metrics

In this appendix we report formulas to compute the metrics used throughout the thesis w.r.t. collection of (possibly multivariate) time series. For additional discussion on mrtrics to evaluate point forecasts we refer to Hyndman and Koehler [2006] and Gneiting [2011].

Mean absolute error (MAE) The mean absolute error is an scale-dependent metric computed as

$$\mathrm{MAE}\left(\widehat{\boldsymbol{X}}_{t:t+T}, \boldsymbol{X}_{t:t+T}\right) \doteq \frac{1}{T\,N} \sum_{\tau=0}^{T-1} \sum_{i=1}^{N} \left\| \hat{\boldsymbol{x}}^i_{t+\tau} - \boldsymbol{x}^i_{t+\tau} \right\|_1. \tag{B.1}$$

Prediction minimizing the MAE are point forecasts of the median.

Mean squared error (MSE) The mean squared error is an scale-dependent metric computed as

$$\mathrm{MSE}\left(\widehat{\boldsymbol{X}}_{t:t+T}, \boldsymbol{X}_{t:t+T}\right) \doteq \frac{1}{T\,N} \sum_{\tau=0}^{T-1} \sum_{i=1}^{N} \left\| \hat{\boldsymbol{x}}^i_{t+\tau} - \boldsymbol{x}^i_{t+\tau} \right\|_2^2. \tag{B.2}$$

Prediction minimizing the MSE are point forecasts of the mean.

Mean absolute percentage error (MAPE) The mean absolute percentage error is a scale-independent metric computed as

$$\mathrm{MAPE}\left(\widehat{\boldsymbol{X}}_{t:t+T}, \boldsymbol{X}_{t:t+T}\right) \doteq \frac{100}{T\,N} \sum_{\tau=0}^{T-1} \sum_{i=1}^{N} \left\| \frac{\hat{\boldsymbol{x}}^i_{t+\tau} - \boldsymbol{x}^i_{t+\tau}}{\boldsymbol{x}^i_{t+\tau}} \right\|_1. \tag{B.3}$$

While the MAPE as the advantage of being scale-independent, its calculation can be problematic if any value within the considered time series is (close to) 0.

Mean relative error (MRE) The mean relative error (also known as weighted average percentage error – WAPE) is a scale-independent metric computed as

$$\mathrm{MRE}\left(\widehat{\boldsymbol{X}}_{t:t+T}, \boldsymbol{X}_{t:t+T}\right) \doteq 100 \frac{\sum_{i=1}^{N} \sum_{\tau=0}^{T-1} \|\hat{\boldsymbol{x}}_{t+\tau}^{i} - \boldsymbol{x}_{t+\tau}^{i}\|_1}{\sum_{i=1}^{N} \sum_{\tau=0}^{T-1} \|\boldsymbol{x}_{t+\tau}^{i}\|_1}. \tag{B.4}$$

The advantage of MRE compared to MAPE is that it is less likely to result in an undefined behavior if values of the target variables are close to 0.

Appendix C

Experimental setup

Benchmarks and baselines have been developed in Python [Van Rossum and Drake, 2009] by relying on TSL [Cini and Marisca, 2022] and the following open-source libraries.

- PyTorch [Paszke et al., 2019];
- PyTorch Lightning [Falcon and The PyTorch Lightning team, 2019];
- PyTorch Geometric [Fey and Lenssen, 2019]
- numpy [Harris et al., 2020];
- scikit-learn [Pedregosa et al., 2011];
- Neptune [neptune.ai, 2021].

In addition to making TSL available for anyone, the code to reproduce the experiments carried out in the thesis has been open-sourced and the associated links can be found in the reference papers.

Appendix D

Appendix to Chapter 4

D.1 Additional details on the experimental setup

We provide additional details on the experimental settings for the results presented in Chapter 4. In particular, we report a detailed description of the reference architectures and additional details on the selected hyperparameters.

D.1.1 Reference architectures

The reference STGNNs architectures follow the template given in Section 3.3.3. For all the baselines, the Encoder is implemented a simple linear layer, while we use an MLP with a single hidden layer for the Decoder. TTS architectures consists in a GRU followed by 2 layers of message passing. **RNN+IMP** use the isotropic MP operator defined in Equation 3.4, while **RNN+AMP** relies on the anisotropic operator of Equation 3.5–3.6. In T&S models, instead, we use a GRU where gates are implemented by using MP layers (see Equation 3.12–3.15) We indicate as **GCRNN-IMP** and **GCRNN-AMP** the models equipped with isotropic and anisotropic MP operators, respectively. Finally, for the standard RNN baselines, we follow the same template (Equation 3.8–3.10) but use a single GRU cell instead of the STMP blocks. For **LocalRNNs**, we train a RNN for each each time series in the collection. In the **FC-RNN** architecture, all the sequences are concatenated along the feature dimension and treated as if they were a single multivariate time series.

D.1.2 Hyperparameters

The number of neurons in each layer for all the reference architectures is set to 64 (reduced to 32 when exceeding the available memory capacity) and the embedding size is set to 32 for all the reference architectures in all the benchmark datasets. Analogous hyperparameters were used for the RNN baselines. For GPVAR instead, we use 16 and 8 as hidden and embedding sizes, respectively. For GPVAR experiments we use a batch size of 128 and train with early stopping for a maximum of 200 epochs with the Adam optimizer [Kingma and Ba, 2015] and a learning rate of 0.01 halved every 50 epochs. For the baselines from the literature, we use the hyperparameters used in the original papers whenever possible.

Appendix E

Appendix to Chapter 5

In Chapter 5, we use the same architectures and hyperparameters adopted in Chapter 4 and detailed in Section D.1. We refer to [Cini et al., 2023c] for additional details.

E.1 Transfer learning experiment

In the transfer learning experiments, we train the models with similar settings of experiments in Table 5.3. The maximum number of training epochs is decreased to 150, while the number of batches per epoch is increased to 500. We then assume that 2 weeks of readings from the target dataset are available for fine-tuning, and use the first week for training and the second as the validation set. Then, we test fine-tuned models on the immediately following week (Table 5.5). For the global model, we either tune all the parameters or none of them (zero-shot setting). For fine-tuning, we increase the maximum number of epochs to 2000 without limiting the batches processed per epoch and fixing the learning rate to 0.001. At the end of every training epoch, we compute the MAE on the validation set and stop training if it has not decreased in the last 100 epochs, restoring the model weights corresponding to the best-performing model. For the global-local models with variational regularization on the embedding space, during training, we set $\beta = 0.05$ and initialize the distribution parameters as

$$\boldsymbol{\mu}_i \sim \mathcal{U}\left(-0.01, 0.01\right), \qquad \boldsymbol{\sigma}_i = \mathbf{0.2}.$$

For fine-tuning instead, we initialize the new embedding table $\boldsymbol{Q}' \in \mathbb{R}^{N' \times d_v}$ as

$$\boldsymbol{Q}' \sim \mathcal{U}\left(-\Delta, \Delta\right),$$

where $\Delta = \frac{1}{\sqrt{d_q}}$ and remove the regularization loss. For the clustering regularization, instead, we use $K = 10$ clusters and sum the clustering loss multiplied by $\lambda = 0.5$ to the forecasting objective. We initialize embedding, centroid, and cluster assignment matrices as

$$\boldsymbol{Q} \sim \mathcal{U}(-\Delta, \Delta) \quad \boldsymbol{C} \sim \mathcal{U}(-\Delta, \Delta) \quad \boldsymbol{S} \sim \mathcal{U}(0, 1)$$

respectively. For fine-tuning, we fix the centroid table $\boldsymbol{C}$ and initialize the new embedding table $\boldsymbol{Q}' \in \mathbb{R}^{N' \times d_v}$ and cluster assignment matrix $\boldsymbol{S}' \in \mathbb{R}^{N' \times K}$ following an analogous procedure. Finally, we increase the weight of the regularization to $\lambda = 10$.

E.1.1 Additional results

Tables E.1 to E.4 show additional results for the transfer learning experiments in all the target datasets. In particular, each table shows results for the reference architectures w.r.t. different training set sizes (from 1 day to 2 weeks) and considers the settings where embeddings are fed to both encoder and decoder or decoder only. We report results on the test data corresponding to the week after the validation set but also on the original test split used in the literature. In the last columns of the table, we also show the performance that one would have obtained 100 epochs after the minimum in the validation error curve; the purpose of showing these results is to hint at the performance that one would have obtained without holding out 1 week of data for validation. The results indeed suggest that fine-tuning the full global model is more prone to overfitting.

Table E.1. Forecasting error (MAE) on PEMS03 in the transfer learning setting (5 runs average).

Model		Testing on 1 subsequent week				Testing on the standard split				Testing 100 epochs after validation min.			
RNN+IMP		2 weeks	1 week	3 days	1 day	2 weeks	1 week	3 days	1 day	2 weeks	1 week	3 days	1 day
Global		$14.86_{\pm 0.02}$	$15.30_{\pm 0.03}$	$16.26_{\pm 0.08}$	$16.65_{\pm 0.07}$	$16.11_{\pm 0.05}$	$16.36_{\pm 0.05}$	$16.95_{\pm 0.04}$	$17.39_{\pm 0.12}$	$16.30_{\pm 0.10}$	$16.58_{\pm 0.07}$	$17.62_{\pm 0.23}$	$18.33_{\pm 0.32}$
Enc.+Dec.	Embeddings	$14.53_{\pm 0.02}$	$14.64_{\pm 0.05}$	$15.87_{\pm 0.08}$	$16.78_{\pm 0.12}$	$16.03_{\pm 0.05}$	$16.12_{\pm 0.05}$	$17.18_{\pm 0.16}$	$17.82_{\pm 0.15}$	$16.09_{\pm 0.06}$	$16.16_{\pm 0.05}$	$17.28_{\pm 0.18}$	$17.94_{\pm 0.17}$
	– Variational	$14.50_{\pm 0.04}$	$14.56_{\pm 0.03}$	$15.40_{\pm 0.06}$	$15.65_{\pm 0.11}$	$15.69_{\pm 0.10}$	$15.70_{\pm 0.12}$	$16.33_{\pm 0.06}$	$16.52_{\pm 0.10}$	$15.70_{\pm 0.10}$	$15.70_{\pm 0.12}$	$16.35_{\pm 0.07}$	$16.54_{\pm 0.10}$
	– Clustering	$14.58_{\pm 0.02}$	$14.60_{\pm 0.02}$	$15.67_{\pm 0.08}$	$16.53_{\pm 0.13}$	$15.65_{\pm 0.07}$	$15.71_{\pm 0.08}$	$16.76_{\pm 0.15}$	$17.76_{\pm 0.15}$	$15.70_{\pm 0.05}$	$15.74_{\pm 0.07}$	$16.80_{\pm 0.14}$	$17.78_{\pm 0.14}$
Dec.	Embeddings	$14.79_{\pm 0.02}$	$14.84_{\pm 0.03}$	$15.49_{\pm 0.03}$	$16.01_{\pm 0.08}$	$16.06_{\pm 0.05}$	$16.12_{\pm 0.07}$	$16.74_{\pm 0.04}$	$17.25_{\pm 0.07}$	$16.08_{\pm 0.05}$	$16.13_{\pm 0.07}$	$16.75_{\pm 0.04}$	$17.29_{\pm 0.06}$
	– Variational	$15.33_{\pm 0.03}$	$15.39_{\pm 0.02}$	$15.83_{\pm 0.04}$	$16.03_{\pm 0.04}$	$16.15_{\pm 0.02}$	$16.20_{\pm 0.02}$	$16.60_{\pm 0.04}$	$16.75_{\pm 0.06}$	$16.15_{\pm 0.02}$	$16.20_{\pm 0.02}$	$16.60_{\pm 0.04}$	$16.76_{\pm 0.06}$
	– Clustering	$14.96_{\pm 0.06}$	$15.09_{\pm 0.06}$	$15.88_{\pm 0.07}$	$15.81_{\pm 0.03}$	$16.25_{\pm 0.08}$	$16.29_{\pm 0.06}$	$16.72_{\pm 0.08}$	$16.87_{\pm 0.07}$	$16.28_{\pm 0.07}$	$16.19_{\pm 0.05}$	$16.91_{\pm 0.10}$	$16.93_{\pm 0.07}$

Table E.2. Forecasting error (MAE) on PEMS04 in the transfer learning setting (5 runs average).

Model		Testing on 1 subsequent week.				Testing on the standard split				Testing 100 epochs after validation min.			
RNN+IMP		2 weeks	1 week	3 days	1 day	2 weeks	1 week	3 days	1 day	2 weeks	1 week	3 days	1 day
Global		$20.86_{\pm 0.03}$	$21.59_{\pm 0.11}$	$21.84_{\pm 0.06}$	$22.26_{\pm 0.10}$	$20.79_{\pm 0.02}$	$21.68_{\pm 0.10}$	$22.10_{\pm 0.10}$	$22.59_{\pm 0.11}$	$20.89_{\pm 0.05}$	$21.97_{\pm 0.13}$	$22.88_{\pm 0.17}$	$23.85_{\pm 0.22}$
Enc.+Dec.	Embeddings	$19.96_{\pm 0.08}$	$20.27_{\pm 0.11}$	$21.03_{\pm 0.14}$	$21.99_{\pm 0.13}$	$19.87_{\pm 0.07}$	$20.27_{\pm 0.07}$	$21.20_{\pm 0.15}$	$22.38_{\pm 0.14}$	$19.88_{\pm 0.07}$	$20.29_{\pm 0.07}$	$21.28_{\pm 0.14}$	$22.46_{\pm 0.14}$
	– Variational	$19.94_{\pm 0.08}$	$20.19_{\pm 0.05}$	$20.71_{\pm 0.12}$	$21.20_{\pm 0.15}$	$19.92_{\pm 0.06}$	$20.23_{\pm 0.05}$	$20.82_{\pm 0.08}$	$21.46_{\pm 0.13}$	$19.92_{\pm 0.06}$	$20.23_{\pm 0.05}$	$20.82_{\pm 0.08}$	$21.47_{\pm 0.12}$
	– Clustering	$19.69_{\pm 0.06}$	$19.91_{\pm 0.11}$	$20.48_{\pm 0.09}$	$21.91_{\pm 0.21}$	$19.70_{\pm 0.06}$	$19.96_{\pm 0.11}$	$20.62_{\pm 0.09}$	$22.28_{\pm 0.21}$	$19.72_{\pm 0.07}$	$19.97_{\pm 0.10}$	$20.65_{\pm 0.08}$	$22.29_{\pm 0.21}$
Dec.	Embeddings	$20.10_{\pm 0.06}$	$20.27_{\pm 0.04}$	$20.87_{\pm 0.08}$	$21.44_{\pm 0.09}$	$20.18_{\pm 0.07}$	$20.39_{\pm 0.05}$	$21.01_{\pm 0.09}$	$21.70_{\pm 0.08}$	$20.19_{\pm 0.07}$	$20.40_{\pm 0.05}$	$21.03_{\pm 0.08}$	$21.74_{\pm 0.08}$
	– Variational	$20.79_{\pm 0.06}$	$20.94_{\pm 0.05}$	$21.23_{\pm 0.07}$	$21.51_{\pm 0.06}$	$20.94_{\pm 0.06}$	$21.10_{\pm 0.05}$	$21.40_{\pm 0.08}$	$21.76_{\pm 0.05}$	$20.94_{\pm 0.06}$	$21.10_{\pm 0.05}$	$21.40_{\pm 0.08}$	$21.77_{\pm 0.05}$
	– Clustering	$20.19_{\pm 0.09}$	$20.45_{\pm 0.10}$	$20.63_{\pm 0.06}$	$21.03_{\pm 0.05}$	$20.27_{\pm 0.09}$	$20.56_{\pm 0.10}$	$20.81_{\pm 0.06}$	$21.28_{\pm 0.06}$	$20.75_{\pm 0.05}$	$20.78_{\pm 0.05}$	$20.89_{\pm 0.05}$	$21.35_{\pm 0.05}$

Table E.3. Forecasting error (MAE) on PEMS07 in the transfer learning setting (5 runs average).

Model		Testing on 1 subsequent week				Testing on the standard split				Testing 100 epochs after validation min.			
RNN+IMP		2 weeks	1 week	3 days	1 day	2 weeks	1 week	3 days	1 day	2 weeks	1 week	3 days	1 day
Global		$22.87_{\pm 0.05}$	$23.82_{\pm 0.03}$	$24.52_{\pm 0.06}$	$25.40_{\pm 0.06}$	$22.64_{\pm 0.04}$	$23.58_{\pm 0.02}$	$24.20_{\pm 0.06}$	$25.04_{\pm 0.06}$	$22.72_{\pm 0.05}$	$23.72_{\pm 0.02}$	$24.45_{\pm 0.11}$	$25.48_{\pm 0.05}$
Enc.+Dec.	Embeddings	$21.68_{\pm 0.07}$	$22.23_{\pm 0.08}$	$23.54_{\pm 0.19}$	$26.11_{\pm 0.61}$	$22.10_{\pm 0.09}$	$22.77_{\pm 0.05}$	$24.17_{\pm 0.24}$	$26.79_{\pm 0.63}$	$22.11_{\pm 0.09}$	$22.78_{\pm 0.05}$	$24.21_{\pm 0.22}$	$26.82_{\pm 0.63}$
	– Variational	$22.05_{\pm 0.05}$	$22.43_{\pm 0.02}$	$23.23_{\pm 0.08}$	$24.40_{\pm 0.13}$	$22.18_{\pm 0.04}$	$22.59_{\pm 0.04}$	$23.44_{\pm 0.11}$	$24.62_{\pm 0.13}$	$22.18_{\pm 0.04}$	$22.59_{\pm 0.04}$	$23.44_{\pm 0.10}$	$24.62_{\pm 0.13}$
	– Clustering	$21.75_{\pm 0.05}$	$22.16_{\pm 0.07}$	$23.36_{\pm 0.20}$	$26.44_{\pm 0.26}$	$22.03_{\pm 0.08}$	$22.52_{\pm 0.10}$	$23.85_{\pm 0.24}$	$27.12_{\pm 0.27}$	$22.03_{\pm 0.09}$	$22.55_{\pm 0.11}$	$23.85_{\pm 0.24}$	$27.13_{\pm 0.28}$
Dec.	Embeddings	$22.50_{\pm 0.14}$	$22.83_{\pm 0.13}$	$23.59_{\pm 0.12}$	$24.89_{\pm 0.19}$	$22.68_{\pm 0.12}$	$23.13_{\pm 0.10}$	$24.04_{\pm 0.09}$	$25.41_{\pm 0.20}$	$22.69_{\pm 0.12}$	$23.14_{\pm 0.10}$	$24.04_{\pm 0.09}$	$25.43_{\pm 0.20}$
	– Variational	$24.32_{\pm 0.16}$	$24.60_{\pm 0.16}$	$25.12_{\pm 0.17}$	$25.50_{\pm 0.15}$	$24.25_{\pm 0.14}$	$24.60_{\pm 0.13}$	$25.16_{\pm 0.13}$	$25.56_{\pm 0.12}$	$24.25_{\pm 0.14}$	$24.60_{\pm 0.13}$	$25.16_{\pm 0.13}$	$25.57_{\pm 0.12}$
	– Clustering	$23.02_{\pm 0.09}$	$23.53_{\pm 0.09}$	$24.42_{\pm 0.15}$	$24.87_{\pm 0.13}$	$23.18_{\pm 0.09}$	$23.77_{\pm 0.08}$	$24.66_{\pm 0.12}$	$25.24_{\pm 0.09}$	$23.91_{\pm 0.16}$	$24.10_{\pm 0.15}$	$24.73_{\pm 0.10}$	$25.27_{\pm 0.09}$

Table E.4. Forecasting error (MAE) on PEMS08 in the transfer learning setting (5 runs average).

Model		Testing on 1 subsequent week				Testing on the standard split				Testing 100 epochs after validation min.			
RNN+IMP		2 weeks	1 week	3 days	1 day	2 weeks	1 week	3 days	1 day	2 weeks	1 week	3 days	1 day
Global		$15.51_{\pm 0.03}$	$15.90_{\pm 0.07}$	$16.87_{\pm 0.05}$	$17.59_{\pm 0.04}$	$15.36_{\pm 0.03}$	$15.71_{\pm 0.06}$	$16.71_{\pm 0.06}$	$17.41_{\pm 0.03}$	$15.47_{\pm 0.06}$	$15.87_{\pm 0.04}$	$17.58_{\pm 0.16}$	$18.46_{\pm 0.12}$
Enc.+Dec.	Embeddings	$15.45_{\pm 0.08}$	$15.45_{\pm 0.06}$	$16.34_{\pm 0.07}$	$17.15_{\pm 0.08}$	$15.34_{\pm 0.07}$	$15.32_{\pm 0.04}$	$16.27_{\pm 0.07}$	$17.11_{\pm 0.08}$	$15.32_{\pm 0.09}$	$15.30_{\pm 0.03}$	$16.27_{\pm 0.05}$	$17.13_{\pm 0.08}$
	– Variational	$15.34_{\pm 0.04}$	$15.41_{\pm 0.06}$	$15.83_{\pm 0.07}$	$16.32_{\pm 0.11}$	$15.21_{\pm 0.03}$	$15.27_{\pm 0.06}$	$15.70_{\pm 0.06}$	$16.19_{\pm 0.12}$	$15.21_{\pm 0.03}$	$15.27_{\pm 0.05}$	$15.69_{\pm 0.06}$	$16.19_{\pm 0.12}$
	– Clustering	$15.41_{\pm 0.06}$	$15.41_{\pm 0.06}$	$15.96_{\pm 0.04}$	$16.99_{\pm 0.07}$	$15.27_{\pm 0.07}$	$15.27_{\pm 0.07}$	$15.90_{\pm 0.05}$	$16.99_{\pm 0.08}$	$15.30_{\pm 0.08}$	$15.28_{\pm 0.07}$	$15.90_{\pm 0.05}$	$17.02_{\pm 0.10}$
Dec.	Embeddings	$15.72_{\pm 0.06}$	$15.74_{\pm 0.06}$	$16.41_{\pm 0.07}$	$16.97_{\pm 0.08}$	$15.61_{\pm 0.06}$	$15.61_{\pm 0.06}$	$16.33_{\pm 0.08}$	$16.90_{\pm 0.09}$	$15.61_{\pm 0.06}$	$15.61_{\pm 0.06}$	$16.35_{\pm 0.08}$	$16.92_{\pm 0.10}$
	– Variational	$16.31_{\pm 0.10}$	$16.33_{\pm 0.15}$	$16.53_{\pm 0.15}$	$16.74_{\pm 0.12}$	$16.13_{\pm 0.09}$	$16.14_{\pm 0.14}$	$16.34_{\pm 0.13}$	$16.55_{\pm 0.11}$	$16.13_{\pm 0.09}$	$16.14_{\pm 0.13}$	$16.35_{\pm 0.14}$	$16.56_{\pm 0.11}$
	– Clustering	$15.81_{\pm 0.11}$	$15.92_{\pm 0.14}$	$16.11_{\pm 0.08}$	$16.55_{\pm 0.10}$	$15.70_{\pm 0.10}$	$15.77_{\pm 0.11}$	$15.97_{\pm 0.08}$	$16.43_{\pm 0.10}$	$15.98_{\pm 0.06}$	$15.90_{\pm 0.06}$	$16.01_{\pm 0.08}$	$16.45_{\pm 0.11}$

Appendix F

Appendix to Chapter 6

F.1 Additional details on the experimental setup

We provide additional details on the experimental settings for the results presented in Chapter 6.

F.1.1 Hyperparameters

For BRITS, we use the same network hyperparameters of Cao et al. [2018] for the AQI-36 dataset. To account for the larger input dimension, for the other datasets we increase the number of neurons to 128 for AQI and METR-LA and 256 for PEMS-BAY and CER-E. The number of neurons was tuned on the validation sets. For rGAIN we use the same number of units in the cells of the bidirectional RNN used by BRITS, but we concatenate a random noise vector (sampled from a uniform distribution) of dimension $z = 4$ to the input vector in order to model the sampling of the data generating process. To obtain predictions, we average out the outputs of $k = 5$ forward passes. We trained all the above baselines by sampling at random batches of 32 elements for a maximum of 300 epochs of 160 batches each and using early stopping with a patience of 40 epochs. All methods are trained using a cosine learning rate scheduler with initial value of 0.001, decayed over the 300 training epochs. For VAR we used an autoregressive order of 5 steps and trained the model with SGD. Here we used a batch size to 64 and a learning rate of 0.0005. The order was selected with a small search in the range $[2, 12]$.

For GRIN we use the same training configurations of the other deep learning baselines and the set a hidden dimension of 64 for all the layers. We did the same for MPGRU. For all the deep learning methods we set the imputation

window size to $W = 24$ for all the datasets except AQI-36 for which we use $W = 36$ steps, in line with Cao et al. [2018].

For SPIN, we use $L = 4$ layers, masking connections w.r.t. the missing observation for the first $\eta = 3$ layers. For each layer, we set the hidden size as $d_h = 32$ and use ReLU activation functions. For SPIN-H, we use similar hyperparameters, but increase the number of layers to 5 and introduce $K = 4$ attention hubs for node with $d_z = 128$. We use learning rate $lr = 0.0008$ and a cosine scheduler with a warm-up of 12 steps and (partial) restarts every 100 epochs. Hyperparameters for SPIN have been selected with a small search on the validation set; we expect far better performance to be achievable with further tuning. We train SPIN with 300 mini-batches of 8 random samples for a maximum of 300 epoch with early stopping.

We refer to [Cini et al., 2022b; Marisca et al., 2022] for additional details, results and ablation studies.

Appendix G

Appendix to Chapter 7

G.1 Deferred proofs

The following provides proofs for Lemma 1 and Lemma 2.

G.1.1 Proof of Lemma 1

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

$$\mathfrak{F}_H(\boldsymbol{A}') = \mathfrak{F}_F(\boldsymbol{A}') \doteq \mathbb{E}_{\boldsymbol{A}\sim \boldsymbol{p}_\psi}\left[\|\boldsymbol{A}-\boldsymbol{A}'\|_F^2\right] \tag{G.1}$$

w.r.t. the Frobenius norm, therefore

$$\min_{\boldsymbol{A}'\in\mathcal{A}} \mathfrak{F}_H(\boldsymbol{A}') = \min_{\boldsymbol{A}'\in\mathcal{A}} \mathfrak{F}_F(\boldsymbol{A}'). \tag{G.2}$$

Now, note that

$$\begin{aligned}
\mathfrak{F}_F(\boldsymbol{A}') &= \mathbb{E}_{\boldsymbol{p}_\psi}\left[\|\boldsymbol{A}-\boldsymbol{A}'\|_F^2\right] = \mathbb{E}_{\boldsymbol{p}_\psi}\left[\|\boldsymbol{A}+\boldsymbol{\mu}-\boldsymbol{\mu}-\boldsymbol{A}'\|_F^2\right] && \text{(G.3)}\\
&= \mathbb{E}_{\boldsymbol{p}_\psi}\left[\|\boldsymbol{A}-\boldsymbol{\mu}\|_F^2\right] + 2\mathbb{E}_{\boldsymbol{p}_\psi}\left[\langle \boldsymbol{A}-\boldsymbol{\mu}, \boldsymbol{\mu}-\boldsymbol{A}'\rangle_F\right] + \mathbb{E}_{\boldsymbol{p}_\psi}\left[\|\boldsymbol{\mu}-\boldsymbol{A}'\|_F^2\right] && \text{(G.4)}\\
&= \mathbb{E}_{\boldsymbol{p}_\psi}\left[\|\boldsymbol{A}-\boldsymbol{\mu}\|_F^2\right] + 2\underbrace{\langle \mathbb{E}_{\boldsymbol{p}_\psi}[\boldsymbol{A}]-\boldsymbol{\mu}, \boldsymbol{\mu}-\boldsymbol{A}'\rangle_F}_{=0} + \|\boldsymbol{\mu}-\boldsymbol{A}'\|_F^2. && \text{(G.5)}
\end{aligned}$$

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

$$\|\boldsymbol{\mu}-\boldsymbol{A}'\|_F^2 = \sum_{i,j=1}^{N} (\boldsymbol{\mu}_{ij} - \boldsymbol{A}'_{ij})^2. \tag{G.6}$$

G.1.2 Proof of Lemma 2

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

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

$$\mathbb{P}(G_{\phi_i} \geq g) \geq \mathbb{P}(G_{\phi_j} \geq g) \tag{G.7}$$

$$\iff e^{-e^{-(g-\phi_i)}} = \mathrm{cdf}_{\phi_i}(g) \leq \mathrm{cdf}_{\phi_j}(g) = e^{-e^{-(g-\phi_j)}} \tag{G.8}$$

$$\iff \left(e^{-e^{-g}}\right)^{e^{\phi_i}} \leq \left(e^{-e^{-g}}\right)^{e^{\phi_i}}. \tag{G.9}$$

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

$$\mathbb{P}(G_{\phi_i} \geq g) \geq \mathbb{P}(G_{\phi_j} \geq g) \iff e^{\phi_i} \geq e^{\phi_j} \iff \phi_i \geq \phi_j. \tag{G.10}$$

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

$$\mathbb{P}(\boldsymbol{A}_{n,i} = 1) = \mathbb{P}(G_{\phi_i} \in \text{top-K}\{G_{\phi_l} : l \in \mathcal{S}\}) = \mathbb{P}(G_{\phi_i} \geq \overline{G}) \tag{G.11}$$

$$= \int \mathbb{P}(G_{\phi_i} \geq g)\mathrm{pdf}_{\overline{G}}(g)\, dg \tag{G.12}$$

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

$$\mathbb{P}(\boldsymbol{A}_{n,i} = 1) \geq \mathbb{P}(\boldsymbol{A}_{n,j} = 1) \overset{\text{(Eq. G.12)}}{\iff} \mathbb{P}(G_{\phi_i} \geq g) \geq \mathbb{P}(G_{\phi_j} \geq g) \overset{\text{(Eq. G.10)}}{\iff} \phi_i \geq \phi_j, \tag{G.13}$$

concluding the proof.

G.2 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 equation 7.30. The derivations

provided here follow from the results of Kool et al. [2020].

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

which corresponds to the desired rewriting.

G.3 Additional details on the experimental setup

We provide additional details on the experimental settings for the results presented in Chapter 7.

G.3.1 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-G 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.

G.3.2 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:

$$\boldsymbol{Z}^{(l)} = \sigma\left(\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{Z}^{(l-1)}\boldsymbol{W} + \boldsymbol{V}\boldsymbol{Z}^{(l-1)}\right) \tag{G.14}$$

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.

G.3.3 Traffic experiment

As reported in Chapter 7, 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. 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 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.

Appendix H

Appendix to Chapter 8

H.1 Additional details on the experimental setup

We provide additional details on the experimental settings for the results presented in Chapter 8. In particular, being the chapter focused on the topic of computation scalability, we provide details on the hardware platform as well. We refer to [Cini et al., 2023a] for further details.

H.1.1 Hardware platform

Experiments were run on a server equipped with two AMD EPYC 7513 processors and four NVIDIA RTX A5000. Reproducibility of the scalability experiments was ensured by taking timings for the update step of each model and setting the number of updates performed by each model accordingly (more details in Sec. H.1.4).

H.1.2 Datasets

For all datasets, the only exogenous variable we consider is the encoding of the time of the day with two sinusoidal functions. As PV-US has been used only in this context through the thesis, we report here further details on the benchmark.

PV-US The PV-US[1] dataset [Hummon et al., 2012] consists of a collection of simulated energy production by 5016 PV farms for the year 2006. In the raw datasets, samples are generated every 5 minute, we aggregate observations

[1] https://www.nrel.gov/grid/solar-power-data.html

at 30 minutes intervals by taking their mean. A (small) subset of this dataset (often referred to as "Solar Energy"[2]) with only the 137 PV plants in Alabama state has been used as a multivariate time series forecasting benchmark [Lai et al., 2018]. To obtain an adjacency matrix, we consider the virtual position of the farms in terms of geographic coordinates, and we apply a Gaussian kernel over the pairwise Haversine distances. Similarly to the CER dataset, we set the window size of the baselines to 36 steps. The code to download and preprocess the data has been open-sourced together with code to reproduce the observed empirical results.

H.1.3 Additional details on the SGP architecture

We implemented the deep ESN encoder following the design principles assessed in previous works [Gallicchio et al., 2018; Lukoševičius, 2012]. In particular, we decrease the discount factor λ progressively at each layer by subtracting 0.1 from its initial value. We also randomly set 30% of the weights of the networks to 0 to obtain a sparse reservoir. We use *tanh* as nonlinear activation function. The recurrent weights are normalized so that the spectral radius of the corresponding matrix is lower than one [Jaeger, 2001].

For the spatial encoding, we compute the embeddings at the different spatial scales iteratively. Additionally, we also concatenate to $\overline{\boldsymbol{S}}_t$ the graph-wise average of the temporal embedding $\overline{\boldsymbol{H}}_t$ to act as a sort of global attribute.

The MLP decoder is implemented as standard feed-forward network with parametrized residual connections between layers [Srivastava et al., 2015], SiLU activation function [Hendrycks and Gimpel, 2016] and optional Dropout [Srivastava et al., 2014] regularization.

H.1.4 Training and evaluation procedure

Traffic As previously mentioned, for the traffic datasets we used the same training settings of previous and kept the same hyperparameters for all baselines whenever possible. For SGP we selected hyperparameters by performing a small search on the validation set. In particular, for METR-LA we used a deep ESN with 3 layers of 32 units each, an initial decay factor of 0.9, and a spectral radius of 0.9. For PEMS-BAY, instead, we used an encoder with a single layer of 128 units, a decay rate of 0.8, and a spectral radius of 0.9. For both datasets, we set $K = 4$ and used the bidirectional encoding scheme. In the decoder, for the first layer we used 32 units for each group in METR-LA and 96 PEMS-BAY,

[2] https://github.com/laiguokun/multivariate-time-series-data

followed by 2 fully connected layers of 256 units each with a dropout rate of 0.3. The model is trained with early stopping for a maximum of 200 epochs of 300 batch each with the Adam optimizer and a multi-step learning rate scheduler.

Large-scale benchmarks In Table 8.3, we report the time required for a single model update (in terms of batches per second) and GPU memory usage for every considered method. To ensure a fair assessment, we record the time interval between the beginning of the inference step and the end weights' update for 150 batches and exclude the first 5 and last 5 measurements (that may have overheads). We exclude from the computation the overhead introduced – for every batch – by the edge subsampling strategy adopted for the scalability of the baselines.

To measure the GPU memory required, we exploit NVIDIA System Management Interface[3], which provides near real-time GPU usage monitoring.

All the experiments designed to measure time and memory requirements have been run on the same machine on a dedicated reserved GPU. We kept the models mostly unchanged w.r.t. the traffic experiment. However, we increased the window size to 36 for the baselines and updated the configuration of the reservoir for SGP to account for the different time scales. In particular, we increased the number of reservoir layers to 8 and 6 in PV-US and CER, respectively, and reduced the number of units accordingly. The difference in the number of layers between the two datasets is motivated by the choice of keeping the size of the preprocessed sequences similar. For the same reason, we also set $K = 2$ and use the unidirectional encoding to limit the amount of required storage to a maximum ≈ 80 GB for each dataset.

Baselines The LSTM and FC-LSTM baselines are implemented as single-layer LSTM with 128 units followed by an MLP with one hidden layer of 256 units and dropout rate of 0.1. For DCRNN, as reported in [Li et al., 2018], we set the number of units in the hidden state to 64 and the order of the diffusion convolution to $K = 2$; compared to the original mode, we use a feed-forward readout instead of a recurrent one to enable scalability on the larger benchmarks. For GraphWaveNet and Gated-GN we use the same hyperparameters and learning rate schedulers reported in the relative papers. We implemented all the baselines in PyTorch and PyTorch Geometric (for graph-based methods) following the open-source implementations provided by the authors. To improve memory and computation efficiency in MP layers, we

[3] https://developer.nvidia.com/nvidia-system-management-interface

use sparse matrix-matrix multiplications instead of scatter-gather operations whenever possible. We fix the maximum number of training epochs to 300 to allow all the models to reach convergence, and stop the training if the MAE computed on the validation set does not decrease for 50 epochs. We evaluate the models using the weights corresponding to the minimum validation MAE. For DynGESN we set the hyperparameters of the reservoir to the same ones used for SGP and increase the number of units to approximately match the dimensions of the final representation extracted by our method. We trained the readout with Ridge regression by selecting the weight of the L2-regularization term on the validation set.

Appendix I

Appendix to Chapter 9

I.1 Additional details on the experimental setup

We provide additional details on the experimental settings for the results presented in Chapter 9. We refer to [Cini et al., 2024] for additional details.

I.1.1 Reference architecture

As discussed in Section 9.4, the main empirical results of the chapter (Table 9.1), were obtained by considering, for all the baselines, a template TTS architecture which can be schematically described as follows:

$$\boldsymbol{h}_t^{i,0} = \mathrm{GRU}\left(\boldsymbol{x}_{t-W:t}^i, \boldsymbol{u}_{t-W:t}^i, \boldsymbol{q}^i\right), \tag{I.1}$$

$$\boldsymbol{H}_t^1 = \mathrm{MP}_1\left(\boldsymbol{H}_t^0, \mathcal{E}\right), \tag{I.2}$$

$$\boldsymbol{H}_t^2 = \mathrm{MP}_2\left(\boldsymbol{H}_t^1, \mathcal{E}\right), \tag{I.3}$$

$$\hat{\boldsymbol{x}}_{t+h}^i = \boldsymbol{W}_h \xi\left(\boldsymbol{W}_{fc}\left[\boldsymbol{h}_t^{i,2} | \boldsymbol{q}_i\right] + \boldsymbol{b}_{fc}\right) + \boldsymbol{b}_h, \qquad h = 0, 1, \ldots, H-1, \tag{I.4}$$

with $\xi(\,\cdot\,)$ being the ELU activation function [Clevert et al., 2016], $\boldsymbol{W}_h \in \mathbb{R}^{1\times d_h}$, $\boldsymbol{W}_h \in \mathbb{R}^{d_h\times d_h}$, $\boldsymbol{b}_h \in \mathbb{R}$, $\boldsymbol{b}_{fc} \in \mathbb{R}^{d_h}$ denoting learnable weights. For HiGP, the template was modified to account for the hierarchical structure as discussed in Section 9.3.1. Similarly, for the GUNet baselines the template was modified to take into account the pooling and lifting operations. For the tuned version of HiGP we simply added skip connections and used a deeper readout.

I.1.2 Hyperparameters and training details

We trained each model with early stopping on the validation set and a batch size of 64 samples for a maximum of 200 epochs each of 300 batches maximum.

We used the Adam optimizer with an initial learning rate of 0.003 reduced by a factor $\gamma = 0.25$ every 50 epochs. The number of neurons d_h in the layers of each model was set to 64 or 32 based on the validation error on each dataset. For HiGP, the regularization coefficient λ was tuned and set to 0.25 based on the validation error on the METR-LA dataset and simply rescaled for the other datasets to take into account the different magnitude of the input. As discussed in Section 9.4, we used a 3-level hierarchy with 20 super-nodes in the middle level and a single super-node (the total aggregate) at the top level. Intra-level spatial propagation was performed only at the base level. For the Diff-TTS baseline, the order of the diffusion convolution was set to $k = 2$, while the pooling factor for the GUNet was set to $p = 0.1$. For what concerns the experimental results in Table 9.2 we use the same settings used in Chapter 8. Hyperparameters for HiGP (T) were obtained by tuning the model on the validation set of both datasets separately.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, 32, 2019.

Kareem Ahmed, Zhe Zeng, Mathias Niepert, and Guy Van den Broeck. SIMPLE: A Gradient Estimator for k-Subset Sampling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=GPJVuyX4p_h.

Juan Lopez Alcaraz and Nils Strodthoff. Diffusion-based Time Series Imputation and Forecasting with Structured State Space Models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=hHiIbk7ApW.

Ferran Alet, Erica Weng, Tomás Lozano-Pérez, and Leslie P Kaelbling. Neural relational inference with fast modular meta-learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C. Maddix,

Syama Rangapuram, David Salinas, Jasper Schulz, Lorenzo Stella, Ali Caner TÃ¼rkmen, and Yuyang Wang. GluonTS: Probabilistic and Neural Time Series Modeling in Python. *Journal of Machine Learning Research*, 21(116): 1–6, 2020. URL http://jmlr.org/papers/v21/19-820.html.

Carlos Alzate and Mathieu Sinn. Improved electricity load forecasting via kernel spectral clustering of smart meters. In *2013 IEEE 13th International Conference on Data Mining*, pages 943–948. IEEE, 2013.

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. Chronos: Learning the Language of Time Series, 2024.

Arie Arya, Yao Lei Xu, Ljubisa Stankovic, and Danilo Mandic. Hierarchical Graph Learning for Stock Market Prediction Via a Domain-Aware Graph Pooling Operator. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

George Athanasopoulos, Roman A Ahmed, and Rob J Hyndman. Hierarchical forecasts for Australian domestic tourism. *International Journal of Forecasting*, 25(1):146–166, 2009.

George Athanasopoulos, Rob J Hyndman, Nikolaos Kourentzes, and Fotios Petropoulos. Forecasting with temporal hierarchies. *European Journal of Operational Research*, 262(1):60–74, 2017.

George Athanasopoulos, Puwasala Gamakumara, Anastasios Panagiotelis, Rob J Hyndman, and Mohamed Affan. Hierarchical forecasting. *Macroeconomic forecasting in the era of big data: Theory and practice*, pages 689–719, 2020.

James Atwood and Don Towsley. Diffusion-convolutional neural networks. *Advances in Neural Information Processing Systems*, 29, 2016.

Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221, 2020.

Davide Bacciu, Alessio Conte, and Francesco Landolfi. Generalizing Downsampling from Regular Data to Graphs. In *Thirty-Seventh AAAI Conference on Artificial Intelligence*, 2023.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations*, 2015.

Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting. *Advances in Neural Information Processing Systems*, 33, 2020.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Kasun Bandara, Christoph Bergmeir, and Slawek Smyl. Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. *Expert systems with applications*, 140:112896, 2020.

Federico Barbero, Ameya Velingker, Amin Saberi, Michael M. Bronstein, and Francesco Di Giovanni. Locality-Aware Graph Rewiring in GNNs. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4Ua4hKiAJX.

Claudio DT Barros, Matheus RF Mendonça, Alex B Vieira, and Artur Ziviani. A survey on embedding dynamic graphs. *ACM Computing Surveys (CSUR)*, 55(1):1–37, 2021.

John M Bates and Clive WJ Granger. The combination of forecasts. *Journal of the operational research society*, 20(4):451–468, 1969.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Claudio Battiloro, Indro Spinelli, Lev Telyatnikov, Michael M. Bronstein, Simone Scardapane, and Paolo Di Lorenzo. From Latent Graph to Latent Topology Inference: Differentiable Cell Complex Module. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=0JsRZEGZ7L.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: Extended Long Short-Term Memory. *arXiv preprint arXiv:2405.04517*, 2024.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in Neural Information Processing Systems*, 14, 2001.

Souhaib Ben Taieb and Bonsoo Koo. Regularized regression for hierarchical forecasting without unbiasedness conditions. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1337–1347, 2019.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in Neural Information Processing Systems*, 28, 2015.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Yuyang Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, François-Xavier Aubet, Laurent Callot, and Tim Januschowski. Deep Learning for Time Series Forecasting: Tutorial and Literature Survey. *ACM Comput. Surv.*, 55(6), dec 2022. ISSN 0360-0300. doi: 10.1145/3533382. URL https://doi.org/10.1145/3533382.

Lorenzo Beretta and Alessandro Santaniello. Nearest neighbor imputation algorithms: a critical evaluation. *BMC medical informatics and decision making*, 16(3):197–208, 2016.

Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 359–370, 1994.

Filippo Maria Bianchi and Veronica Lachi. The expressive power of pooling in graph neural networks. *Advances in Neural Information Processing Systems*, 2023.

Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *International conference on machine learning*, pages 874–883. PMLR, 2020a.

Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Hierarchical representation learning in graph neural networks with node decimation pooling. *IEEE Transactions on Neural Networks and Learning Systems*, 33(5):2195–2207, 2020b.

Filippo Maria Bianchi, Simone Scardapane, Sigurd Løkse, and Robert Jenssen. Reservoir computing approaches for representation and classification of multivariate time series. *IEEE Transactions on Neural Networks and Learning Systems*, 32(5):2169–2179, 2020c.

Filippo Maria Bianchi, Claudio Gallicchio, and Alessio Micheli. Pyramidal reservoir graph neural network. *Neurocomputing*, 470:389–404, 2022.

Marin Biloš, Kashif Rasul, Anderson Schneider, Yuriy Nevmyvaka, and Stephan Günnemann. Modeling temporal data as continuous functions with stochastic process diffusion. In *International Conference on Machine Learning*, pages 2452–2470. PMLR, 2023.

Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah 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.

Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. Holden-Day, San Francisco, 1970.

Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and deep locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

Luca Butera, Andrea Cini, Alberto Ferrante, and Cesare Alippi. Object-Centric Relational Representations for Image Generation. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. Brits: Bidirectional recurrent imputation for time series. *Advances in Neural Information Processing Systems*, 31, 2018.

Cristian Challu, Kin G Olivares, Boris N Oreshkin, Federico Garza Ramirez, Max Mergenthaler Canseco, and Artur Dubrawski. N-HiTS: Neural hierarchical interpolation for time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6989–6997, 2023.

Ben Chamberlain, James Rowbottom, Maria I. Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. GRAND: Graph Neural Diffusion. In *Proceedings of the 38th International Conference on Machine Learning*, pages 1407–1418. PMLR, July 2021.

Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):1–12, 2018.

Chao Chen, Karl Petty, Alexander Skabardonis, Pravin Varaiya, and Zhanfeng Jia. Freeway performance measurement system: mining loop detector data. *Transportation Research Record*, 1748(1):96–102, 2001.

Hongjie Chen, Ryan A Rossi, Kanak Mahadik, Sungchul Kim, and Hoda Eldardiry. Graph Deep Factors for Forecasting with Applications to Cloud Resource Allocation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 106–116, 2021a.

Ling Chen, Jiahui Xu, Binqing Wu, Yuntao Qian, Zhenhong Du, Yansheng Li, and Yongjun Zhang. Group-aware graph neural network for nationwide city air quality forecasting. *arXiv preprint arXiv:2108.12238*, 2021b.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018a.

Sheng Chen, Stephen A Billings, and PM Grant. Non-linear system identification using neural networks. *International journal of control*, 51(6):1191–1214, 1990.

Siyuan Chen, Jiahai Wang, and Guoqing Li. Neural relational inference with efficient message passing mechanisms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7055–7063, 2021c.

Yakun Chen, Zihao Li, Chao Yang, Xianzhi Wang, Guodong Long, and Guandong Xu. Adaptive graph recurrent network for multivariate time series imputation. In *International Conference on Neural Information Processing*, pages 64–73. Springer, 2022.

Yingmei Chen, Zhongyu Wei, and Xuanjing Huang. Incorporating corporation relationship via graph convolutional neural networks for stock price prediction. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1655–1658, 2018b.

Yitian Chen, Yanfei Kang, Yixiong Chen, and Zizhuo Wang. Probabilistic forecasting with temporal convolutional neural network. *Neurocomputing*, 399:491–501, 2020.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266, 2019.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Jeongwhan Choi, Hwangyong Choi, Jeehyun Hwang, and Noseong Park. Graph neural controlled differential equations for traffic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6367–6374, 2022.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Andrzej Cichocki and Anh-Huy Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 92(3):708–721, 2009.

Andrea Cini and Ivan Marisca. Torch Spatiotemporal, 2022. URL https://github.com/TorchSpatiotemporal/tsl.

Andrea Cini, Slobodan Lukovic, and Cesare Alippi. Cluster-based aggregate load forecasting with deep neural networks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

Andrea Cini, Carlo D'Eramo, Jan Peters, and Cesare Alippi. Deep reinforcement learning with weighted Q-Learning. *The Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2022a.

Andrea Cini, Ivan Marisca, and Cesare Alippi. Filling the G_ap_s: Multivariate Time Series Imputation by Graph Neural Networks. In *International Conference on Learning Representations*, 2022b.

Andrea Cini, Ivan Marisca, Filippo Maria Bianchi, and Cesare Alippi. Scalable Spatiotemporal Graph Neural Networks. *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, 2023a.

Andrea Cini, Ivan Marisca, Daniele Zambon, and Cesare Alippi. Graph Deep Learning for Time Series Forecasting. *arXiv preprint arXiv:2310.15978*, 2023b.

Andrea Cini, Ivan Marisca, Daniele Zambon, and Cesare Alippi. Taming Local Effects in Graph-based Spatiotemporal Forecasting. *Advances in Neural Information Processing Systems*, 2023c.

Andrea Cini, Daniele Zambon, and Cesare Alippi. Sparse graph learning from spatiotemporal time series. *Journal of Machine Learning Research*, 24(242): 1–36, 2023d.

Andrea Cini, Danilo Mandic, and Cesare Alippi. Graph-based Time Series Clustering for End-to-End Hierarchical Forecasting. *International Conference on Machine Learning*, 2024.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *International Conference on Learning Representations*, 2016.

Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 2006. ISSN 1063-5203. doi: https://doi.org/10.1016/j.acha.2006.04.006.

Commission for Energy Regulation. CER Smart Metering Project - Electricity Customer Behaviour Trial, 2009-2010 [dataset]. *Irish Social Science Data Archive. SN: 0012-00*, 2016. URL https://www.ucd.ie/issda/data/commissionforenergyregulationcer.

Giorgio Corani, Dario Azzimonti, João PSC Augusto, and Marco Zaffalon. Probabilistic reconciliation of hierarchical forecast via Bayes' rule. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part III*, pages 211–226. Springer, 2021.

Gonçalo Correia, Vlad Niculae, Wilker Aziz, and André Martins. Efficient marginalization of discrete and structured latent variables via sparsity. *Advances in Neural Information Processing Systems*, 33:11789–11802, 2020.

Abhimanyu Das, Weihao Kong, Biswajit Paria, and Rajat Sen. Dirichlet proportions model for hierarchically coherent probabilistic forecasting. In *Uncertainty in Artificial Intelligence*, pages 518–528. PMLR, 2023.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting, 2024.

Emmanuel de Bézenac, Syama Sundar Rangapuram, Konstantinos Benidis, Michael Bohlke-Schneider, Richard Kurle, Lorenzo Stella, Hilaf Hasson, Patrick Gallinari, and Tim Januschowski. Normalizing kalman filters for multivariate time series analysis. *Advances in Neural Information Processing Systems*, 33:2995–3007, 2020.

Giovanni De Felice, Andrea Cini, Daniele Zambon, Vladimir Gusev, and Cesare Alippi. Graph-based Virtual Sensing from Sparse and Partial Multivariate Observations. In *International Conference on Learning Representations*, 2024.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems*, 29:3844–3852, 2016.

Ailin Deng and Bryan 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.

Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 551–556, 2004.

Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.

Francesco Di Giovanni, Lorenzo Giusti, Federico Barbero, Giulia Luise, Pietro Lio, and Michael M Bronstein. On over-squashing in message passing neural networks: The impact of width, depth, and topology. In *International Conference on Machine Learning*, pages 7865–7885. PMLR, 2023.

Francesco Di Giovanni, James Rowbottom, Benjamin Paul Chamberlain, Thomas Markovich, and Michael M. Bronstein. Understanding convolution on graphs via energies. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=v5ew3FPTgb.

Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.

Xiaowen Dong, Dorina Thanou, Pascal Frossard, and Pierre Vandergheynst. Learning Laplacian matrix in smooth graph signal representations. *IEEE Transactions on Signal Processing*, 64(23):6160–6173, 2016.

Xiaowen Dong, Dorina Thanou, Michael Rabbat, and Pascal Frossard. Learning graphs from data: A signal representation perspective. *IEEE Signal Processing Magazine*, 36(3):44–63, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Wenjie Du, David Cote, and Yan Liu. SAITS: Self-Attention-based Imputation for Time Series. *Expert Systems with Applications*, 219:119619, 2023. ISSN 0957-4174. doi: 10.1016/j.eswa.2023.119619.

James Durbin and Siem Jan Koopman. *Time series analysis by state space methods*. Oxford university press, 2012.

James Durbin and Geoffrey S Watson. Testing for Serial Correlation in Least Squares Regression: I. *Biometrika*, 37(3/4):409–428, 1950.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph Neural Networks with Learnable Structural and Positional Representations. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=wTTjnvGphYj.

Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking Graph Neural Networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023. URL http://jmlr.org/papers/v24/22-0567.html.

Carlo D'Eramo, Andrea Cini, Alessandro Nuara, Matteo Pirotta, Cesare Alippi, Jan Peters, and Marcello Restelli. Gaussian Approximation for Bias Reduction in Q-Learning. *Journal of Machine Learning Research*, 22:1–51, 2021.

Simone Eandi, Andrea Cini, Slobodan Lukovic, and Cesare Alippi. Spatio-Temporal Graph Neural Networks for Aggregate Load Forecasting. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

Nikolaos A Efkarpidis, Stefano Imoscopi, Martin Geidl, Andrea Cini, Slobodan Lukovic, Cesare Alippi, and Ingo Herbst. Peak shaving in distribution networks using stationary energy storage systems: A Swiss case study. *Sustainable Energy, Grids and Networks*, 34:101018, 2023.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Federico Errica, Henrik Christiansen, Viktor Zaverkin, Takashi Maruyama, Mathias Niepert, and Francesco Alesiani. Adaptive Message Passing: A General Framework to Mitigate Oversmoothing, Oversquashing, and Underreaching. *arXiv preprint arXiv:2312.16560*, 2023.

Fateme Fahiman, Sarah M Erfani, Sutharshan Rajasegarar, Marimuthu Palaniswami, and Christopher Leckie. Improving load forecasting based on deep learning and K-shape clustering. In *2017 international joint conference on neural networks (IJCNN)*, pages 4134–4141. IEEE, 2017.

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

Zheng Fang, Qingqing Long, Guojie Song, and Kunqing Xie. Spatial-Temporal Graph ODE Networks for Traffic Flow Forecasting. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pages 364–373, New York, NY, USA, August 2021. Association for Computing Machinery. ISBN 978-1-4503-8332-5. doi: 10.1145/3447548.3467430.

Lorenzo Ferretti, Andrea Cini, Georgios Zacharopoulos, Cesare Alippi, and Laura Pozzi. Graph neural networks for high-level synthesis design space exploration. *ACM Transactions on Design Automation of Electronic Systems*, 28(2):1–20, 2022.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Jakob Foerster, Gregory Farquhar, Maruan Al-Shedivat, Tim Rocktäschel, Eric Xing, and Shimon Whiteson. Dice: The infinitely differentiable Monte Carlo estimator. In *International Conference on Machine Learning*, pages 1529–1538. PMLR, 2018.

Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *International Conference on Machine Learning*, pages 1972–1982. PMLR, 2019.

Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

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

Cornelius Fritz, Emilio Dorigatti, and David Rügamer. Combining graph neural networks and spatio-temporal disease models to improve the prediction of weekly COVID-19 cases in Germany. *Scientific Reports*, 12(1):3930, 2022.

Daniel Y Fu, Elliot L Epstein, Eric Nguyen, Armin W Thomas, Michael Zhang, Tri Dao, Atri Rudra, and Christopher Ré. Simple hardware-efficient long convolutions for sequence modeling. In *International Conference on Machine Learning*, pages 10373–10391. PMLR, 2023.

Claudio Gallicchio and Simone Scardapane. Deep Randomized Neural Networks. In *Recent Trends in Learning From Data: Tutorials from the INNS Big Data and Deep Learning Conference (INNSBDDL2019)*, pages 43–68. Springer, 2020.

Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, 268:87–99, 2017.

Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. Design of deep echo state networks. *Neural Networks*, 108:33–47, 2018.

Ankit Gandhi, Sivaramakrishnan Kaveri, Vineet Chaoji, et al. Spatio-Temporal Multi-graph Networks for Demand Forecasting in Online Marketplaces. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 187–203. Springer, 2021.

Hongyang Gao and Shuiwang Ji. Graph U-Nets. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2083–2092. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/gao19a.html.

Jianfei Gao and Bruno Ribeiro. On the Equivalence Between Temporal and Static Equivariant Graph Representations. In *International Conference on Machine Learning*, pages 7052–7076. PMLR, 2022.

Azul Garza and Max Mergenthaler-Canseco. TimeGPT-1. *arXiv preprint arXiv:2310.03589*, 2023.

Alberto Gasparin, Slobodan Lukovic, and Cesare Alippi. Deep learning for time series forecasting: The electric load case. *CAAI Transactions on Intelligence Technology*, 7(1):1–25, 2022.

Jan Gasthaus, Konstantinos Benidis, Yuyang Wang, Syama Sundar Rangapuram, David Salinas, Valentin Flunkert, and Tim Januschowski. Probabilistic forecasting with spline quantile function RNNs. In *The 22nd international conference on artificial intelligence and statistics*, pages 1901–1910. PMLR, 2019.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR, 2017.

Zoubin Ghahramani and Michael Jordan. Supervised learning from incomplete data via an EM approach. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1994.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017.

Paul Glasserman and Yu-Chi Ho. *Gradient estimation via perturbation analysis*, volume 116. Springer Science & Business Media, 1991.

Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

Tilmann Gneiting. Making and evaluating point forecasts. *Journal of the American Statistical Association*, 106(494):746–762, 2011.

Dong Gong, Frederic Z Zhang, Javen Qinfeng Shi, and Anton Van Den Hengel. Memory-augmented dynamic neural relational inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11843–11852, 2021.

Colin Graber and Alexander G. Schwing. Dynamic Neural Relational Inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Clive WJ Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica: journal of the Econometric Society*, pages 424–438, 1969.

Francesco Grassi, Andreas Loukas, Nathanaël Perraudin, and Benjamin Ricaud. A time-vertex signal processing framework: Scalable processing and meaningful representations for time-series on graphs. *IEEE Transactions on Signal Processing*, 66(3):817–829, 2017.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the Void: Optimizing control variates for black-box gradient estimation. In *International Conference on Learning Representations*, 2018.

Daniele Grattarola, Daniele Zambon, Filippo Bianchi, and Cesare Alippi. Understanding Pooling in Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–11, 2022. doi: 10.1109/TNNLS.2022.3190922.

Alessio Gravina and Davide Bacciu. Deep learning for dynamic graphs: models and benchmarks. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

Alessio Gravina, Davide Bacciu, and Claudio Gallicchio. Anti-Symmetric DGN: a stable architecture for Deep Graph Networks. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=J3Y7cgZOOS.

Alessio Gravina, Giulio Lovisotto, Claudio Gallicchio, Davide Bacciu, and Claas Grohnfeldt. Long Range Propagation on Continuous-Time Dynamic Graphs. *International Conference on Machine Learning*, 2024a.

Alessio Gravina, Daniele Zambon, Davide Bacciu, and Cesare Alippi. Temporal Graph ODEs for Irregularly-Sampled Time Series. *International Joint Conference on Artificial Intelligence*, 2024b.

Jake Grigsby, Zhe Wang, and Yanjun Qi. Long-Range Transformers for Dynamic Spatiotemporal Forecasting, 2021.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Karan Goel, and Christopher Re. Efficiently Modeling Long Sequences with Structured State Spaces. In *International Conference on Learning Representations*, 2022a. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Albert Gu, Ankit Gupta, Karan Goel, and Christopher Ré. On the Parameterization and Initialization of Diagonal State Space Models. *Advances in Neural Information Processing Systems*, 35, 2022b.

Kan Guo, Yongli Hu, Yanfeng Sun, Sean Qian, Junbin Gao, and Baocai Yin. Hierarchical graph convolution network for traffic forecasting. In *Proceedings of the 35th AAAI conference on artificial intelligence*, volume 35, pages 151–159, 2021a.

Shengnan Guo, Youfang Lin, Huaiyu Wan, Xiucheng Li, and Gao Cong. Learning dynamics and heterogeneity of spatial-temporal graph data for traffic forecasting. *IEEE Transactions on Knowledge and Data Engineering*, 2021b.

Ankit Gupta, Albert Gu, and Jonathan Berant. Diagonal state spaces are as effective as structured state spaces. *Advances in Neural Information Processing Systems*, 35:22982–22994, 2022.

Benjamin Gutteridge, Xiaowen Dong, Michael M Bronstein, and Francesco Di Giovanni. DRew: Dynamically Rewired Message Passing with Delay . In *International Conference on Machine Learning*, pages 12252–12267. PMLR, 2023.

James D Hamilton. *Time Series Analysis*. Princeton University pPress, 2020.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30, 2017.

Xing Han, Sambarta Dasgupta, and Joydeep Ghosh. Simultaneously reconciled quantile forecasting of hierarchically related time series. In *International Conference on Artificial Intelligence and Statistics*, pages 190–198. PMLR, 2021.

Jonas Berg Hansen and Filippo Maria Bianchi. Total variation graph neural networks. In *International Conference on Machine Learning*, pages 12445–12468. PMLR, 2023.

Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

Andrew C Harvey et al. Forecasting, Structural Time Series Models and the Kalman Filter. *Cambridge Books*, 1990.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units. *arXiv preprint arXiv:1606.08415*, 2016.

Luca Hermes, Barbara Hammer, Andrew Melnik, Riza Velioglu, Markus Vieth, and Malte Schilling. A Graph-based U-Net Model for Predicting Traffic in unseen Cities. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1):388–427, 2021.

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

Ross Hollyman, Fotios Petropoulos, and Michael E Tipping. Understanding forecast reconciliation. *European Journal of Operational Research*, 294(1): 149–160, 2021.

Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. Temporal graph benchmark for machine learning on temporal graphs. *arXiv preprint arXiv:2307.01026*, 2023.

Zijie Huang, Yizhou Sun, and Wei Wang. Learning continuous system dynamics from irregularly-sampled partial observations. *Advances in Neural Information Processing Systems*, 33:16177–16187, 2020.

Zijie Huang, Yizhou Sun, and Wei Wang. Coupled Graph ODE for Learning Interacting System Dynamics. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pages 705–715, New York, NY, USA, August 2021. Association for Computing Machinery. ISBN 978-1-4503-8332-5. doi: 10.1145/3447548.3467385.

Iris AM Huijben, Wouter Kool, Max B Paulus, and Ruud JG 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*, 45(2):1353–1371, 2022.

Marissa Hummon, Eduardo Ibanez, Gregory Brinkman, and Debra Lew. Subhour solar data for power system modeling from static spatial variability analysis. Technical report, National Renewable Energy Lab.(NREL), Golden, CO (United States), 2012.

Rob Hyndman, Anne B Koehler, J Keith Ord, and Ralph D Snyder. *Forecasting with exponential smoothing: the state space approach*. Springer Science & Business Media, 2008.

Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

Rob J Hyndman, Anne B Koehler, Ralph D Snyder, and Simone Grose. A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of Forecasting*, 18(3):439–454, 2002.

Rob J Hyndman, Roman A Ahmed, George Athanasopoulos, and Han Lin Shang. Optimal combination forecasts for hierarchical time series. *Computational statistics & data analysis*, 55(9):2579–2589, 2011.

Ditsuhi Iskandaryan, Francisco Ramos, and Sergio Trilles. Graph Neural Network for Air Quality Prediction: A Case Study in Madrid. *IEEE Access*, 11:2729–2742, 2023.

Elvin Isufi, Andreas Loukas, Nathanael Perraudin, and Geert Leus. Forecasting time series with VARMA recursions on graphs. *IEEE Transactions on Signal Processing*, 67(18):4870–4885, 2019.

Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34): 13, 2001.

Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. Optimization and Applications of Echo State Networks with Leaky-Integrator Neurons. *Neural Networks*, 20(3):335–352, 2007.

Hosagrahar V Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M Patel, Raghu Ramakrishnan, and Cyrus Shahabi. Big data and its technical challenges. *Communications of the ACM*, 57(7): 86–94, 2014.

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

Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

Tim Januschowski, Jan Gasthaus, Yuyang Wang, David Salinas, Valentin Flunkert, Michael Bohlke-Schneider, and Laurent Callot. Criteria for classifying forecasting methods. *International Journal of Forecasting*, 36(1):167–177, 2020.

Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications*, 207:117921, 2022.

Guangyin Jin, Yuxuan Liang, Yuchen Fang, Jincai Huang, Junbo Zhang, and Yu Zheng. Spatio-temporal graph neural networks for predictive learning in urban computing: A survey. *arXiv preprint arXiv:2303.14483*, 2023a.

Ming Jin, Yu Zheng, Yuan-Fang Li, Siheng Chen, Bin Yang, and Shirui Pan. Multivariate time series forecasting with dynamic graph neural ODEs. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

Ming Jin, Huan Yee Koh, Qingsong Wen, Daniele Zambon, Cesare Alippi, Geoffrey I Webb, Irwin King, and Shirui Pan. A Survey on Graph Neural Networks for Time Series: Forecasting, Classification, Imputation, and Anomaly Detection. *arXiv preprint arXiv:2307.03759*, 2023b.

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

Vassilis Kalofolias, Xavier Bresson, Michael Bronstein, and Pierre Vandergheynst. Matrix completion on graphs. *arXiv preprint arXiv:1408.1717*, 2014.

Amol Kapoor, Xue Ben, Luyang Liu, Bryan Perozzi, Matt Barnes, Martin Blais, and Shawn O'Banion. Examining covid-19 forecasting using spatio-temporal graph neural networks. *arXiv preprint arXiv:2007.03113*, 2020.

Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019.

Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation Learning for Dynamic Graphs: A Survey. *J. Mach. Learn. Res.*, 21(70):1–73, 2020.

Anees Kazi, Luca Cosmo, Seyed-Ahmad Ahmadi, Nassir Navab, and Michael Bronstein. Differentiable graph module (dgm) for graph convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural Controlled Differential Equations for Irregular Time Series. In *Advances in Neural Information Processing Systems*, volume 33, pages 6696–6707. Curran Associates, Inc., 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *3nd International Conference on Learning Representations, ICLR 2015*, 2015.

Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*, 2013.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pages 2688–2697. PMLR, 2018.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Roger Koenker and Kevin F Hallock. Quantile regression. *Journal of economic perspectives*, 15(4):143–156, 2001.

Wouter Kool, Herke Van Hoof, and Max 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.

Wouter Kool, Herke van Hoof, and Max 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.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. *Journal of Machine Learning Research*, 24(89):1–97, 2023. ISSN 1533-7928.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.

Manuel Kunz, Stefan Birr, Mones Raslan, Lei Ma, Zhen Li, Adele Gouttes, Mateusz Koren, Tofigh Naghibi, Johannes Stephan, Mariia Bulycheva, et al. Deep Learning based Forecasting: a case study from the online fashion industry. *arXiv preprint arXiv:2305.14406*, 2023.

Sanmukh R. Kuppannagari, Yao Fu, Chung Ming Chueng, and Viktor K. Prasanna. Spatio-Temporal Missing Data Imputation for Smart Power Grids. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, e-Energy '21, page 458–465, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383332. doi: 10.1145/3447555.3466586.

Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pages 95–104, 2018.

Yann LeCun and Yoshua Bengio. *Convolutional Networks for Images, Speech, and Time Series*, page 255–258. MIT Press, Cambridge, MA, USA, 1998. ISBN 0262511029.

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

Geert Leus, Antonio G Marques, José MF Moura, Antonio Ortega, and David I Shuman. Graph Signal Processing: History, development, impact, and outlook. *IEEE Signal Processing Magazine*, 40(4):49–60, 2023.

Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in Neural Information Processing Systems*, 32:5243–5253, 2019.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations*, 2018.

Yuhong Li, Tianle Cai, Yi Zhang, Deming Chen, and Debadeepta Dey. What Makes Convolutional Models Great on Long Sequence Modeling? In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=TGJSPbRpJX-.

Yubo Liang, Zezhi Shao, Fei Wang, Zhao Zhang, Tao Sun, and Yongjun Xu. BasicTS: An Open Source Fair Multivariate Time Series Prediction Benchmark. In *International Symposium on Benchmarking, Measuring and Optimization*, pages 87–101. Springer, 2022a.

Yuebing Liang, Zhan Zhao, and Lijun Sun. Memory-augmented dynamic graph convolution networks for traffic data imputation with diverse missing patterns. *Transportation Research Part C: Emerging Technologies*, 143:103826, 2022b.

Yuxuan Liang, Haomin Wen, Yuqi Nie, Yushan Jiang, Ming Jin, Dongjin Song, Shirui Pan, and Qingsong Wen. Foundation models for time series analysis: A tutorial and survey. *arXiv preprint arXiv:2403.14735*, 2024.

Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.

Tsungnan Lin, B.G. Horne, P. Tino, and C.L. Giles. Learning long-term dependencies in NARX recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338, 1996. doi: 10.1109/72.548162.

Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019.

Hangchen Liu, Zheng Dong, Renhe Jiang, Jiewen Deng, Jinliang Deng, Quanjun Chen, and Xuan Song. Spatio-temporal adaptive embedding makes vanilla transformer sota for traffic forecasting. In *Proceedings of the 32nd ACM international conference on information and knowledge management*, pages 4125–4129, 2023a.

Mingzhe Liu, Han Huang, Hao Feng, Leilei Sun, Bowen Du, and Yanjie Fu. PriSTI: A Conditional Diffusion Framework for Spatiotemporal Imputation, February 2023b.

Weifeng Liu, Puskal P Pokharel, and Jose C Principe. Correntropy: Properties and applications in non-Gaussian signal processing. *IEEE Transactions on Signal Processing*, 55(11):5286–5298, 2007.

Xu Liu, Yutong Xia, Yuxuan Liang, Junfeng Hu, Yiwei Wang, Lei Bai, Chao Huang, Zhenguang Liu, Bryan Hooi, and Roger Zimmermann. LargeST: A Benchmark Dataset for Large-Scale Traffic Forecasting. *arXiv preprint arXiv:2306.08259*, 2023c.

Yijing Liu, Qinxian Liu, Jian-Wei Zhang, Haozhe Feng, Zhongwei Wang, Zihan Zhou, and Wei Chen. Multivariate Time-Series Forecasting with Temporal Polynomial Graph Neural Networks. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=pMumil2EJh.

Yukai Liu, Rose Yu, Stephan Zheng, Eric Zhan, and Yisong Yue. NAOMI: Non-autoregressive multiresolution sequence imputation. *Advances in Neural Information Processing Systems*, 32, 2019.

Zibo Liu, Parshin Shojaee, and Chandan K. Reddy. Graph-based Multi-ODE Neural Networks for Spatio-Temporal Traffic Forecasting. *Transactions on Machine Learning Research*, 2023d. ISSN 2835-8856.

Greta M Ljung and George EP Box. On a Measure of Lack of Fit in Time Series Models. *Biometrika*, 65(2):297–303, 1978.

Antonio Longa, Veronica Lachi, Gabriele Santin, Monica Bianchini, Bruno Lepri, Pietro Lio, Franco Scarselli, and Andrea Passerini. Graph Neural Networks for Temporal Graphs: State of the Art, Open Challenges, and Opportunities. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=pHCdMat0gI.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning Nonlinear Operators via DeepONet Based on the Universal Approximation Theorem of Operators. *Nature Machine Intelligence*, 3(3): 218–229, March 2021. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5.

Mantas Lukoševičius. A practical guide to applying echo state networks. In *Neural networks: Tricks of the trade*, pages 659–686. Springer, 2012.

Mantas Lukoševičius and Herbert Jaeger. Reservoir Computing Approaches to Recurrent Neural Network Training. *Computer Science Review*, 3(3):127–149, 2009.

Xiao Luo, Jingyang Yuan, Zijie Huang, Huiyu Jiang, Yifang Qin, Wei Ju, Ming Zhang, and Yizhou Sun. HOPE: High-order Graph ODE For Modeling Interacting Dynamics. In *Proceedings of the 40th International Conference on Machine Learning*, pages 23124–23139. PMLR, July 2023.

Yonghong Luo, Xiangrui Cai, Ying Zhang, Jun Xu, and Yuan Xiaojie. Multivariate Time Series Imputation with Generative Adversarial Networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Yonghong Luo, Ying Zhang, Xiangrui Cai, and Xiaojie Yuan. E^2GAN: End-to-End Generative Adversarial Network for Multivariate Time Series Imputation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 3094–3100. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

Jiawei Ma, Zheng Shou, Alireza Zareian, Hassan Mansour, Anthony Vetro, and Shih-Fu Chang. CDSA: cross-dimensional self-attention for multivariate, geo-tagged time series imputation. *arXiv preprint arXiv:1905.09904*, 2019.

Yihong Ma, Patrick Gerard, Yijun Tian, Zhichun Guo, and Nitesh V Chawla. Hierarchical Spatio-Temporal Graph Neural Networks for Pandemic Forecasting. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 1481–1490, 2022.

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

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The M4 Competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74, 2020.

Danilo P Mandic and Jonathon Chambers. *Recurrent neural networks for prediction: learning algorithms, architectures and stability.* John Wiley & Sons, Inc., 2001.

Alessandro Manenti, Daniele Zambon, and Cesare Alippi. Learning Latent Graph Structures and their Uncertainty. *arXiv preprint arXiv:2405.19933*, 2024.

Ivan Marisca, Andrea Cini, and Cesare Alippi. Learning to Reconstruct Missing Data from Spatiotemporal Graphs with Sparse Observations. In *Advances in Neural Information Processing Systems*, 2022.

Ivan Marisca, Cesare Alippi, and Filippo Maria Bianchi. Graph-based Forecasting with Missing Data through Spatiotemporal Downsampling. *arXiv preprint arXiv:2402.10634*, 2024.

Tommaso Marzi, Arshjot Khehra, Andrea Cini, and Cesare Alippi. Feudal Graph Reinforcement Learning. *arXiv preprint arXiv:2304.05099*, 2023.

Daiki Matsunaga, Toyotaro Suzumura, and Toshihiro Takahashi. Exploring graph neural networks for stock market predictions with rolling window analysis. *arXiv preprint arXiv:1909.10660*, 2019.

Nicholas Metropolis and Stanislaw Ulam. The Monte Carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

Xiaoye Miao, Yangyang Wu, Jun Wang, Yunjun Gao, Xudong Mao, and Jianwei Yin. Generative Semi-supervised Learning for Multivariate Time Series Imputation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8983–8991, 2021.

Alessio Micheli and Domenico Tortorella. Discrete-time dynamic graph echo state networks. *Neurocomputing*, 496:85–95, 2022.

Pasquale Minervini, Luca Franceschi, and Mathias 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.

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

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Rearning. In *International Conference on Machine Learning*, pages 1928–1937. PMLR, 2016.

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte Carlo Gradient Estimation in Machine Learning. *J. Mach. Learn. Res.*, 21 (132):1–62, 2020.

Pablo Montero-Manso and Rob J Hyndman. Principles and algorithms for forecasting groups of time series: Locality and globality. *International Journal of Forecasting*, 37(4):1632–1653, 2021.

Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks. *Advances in Neural Information Processing Systems*, 30, 2017.

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

Vlad Niculae, Caio F Corro, Nikita Nangia, Tsvetomila Mihaylova, and André FT Martins. Discrete latent structure in neural networks. *arXiv preprint arXiv:2301.07473*, 2023.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Jbdc0vTOcol.

Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit MLE: backpropagating through discrete exponential family distributions. *Advances in Neural Information Processing Systems*, 34:14567–14579, 2021.

Hoang Nt and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Kin G. Olivares, Cristian Challú, Federico Garza, Max Mergenthaler Canseco, and Artur Dubrawski. NeuralForecast: User friendly state-of-the-art neural forecasting models. PyCon Salt Lake City, Utah, US 2022, 2022. URL https://github.com/Nixtla/neuralforecast.

Shayegan Omidshafiei, Daniel Hennes, Marta Garnelo, Zhe Wang, Adria Recasens, Eugene Tarassov, Yi Yang, Romuald Elie, Jerome T Connor, Paul Muller, et al. Multiagent off-screen behavior prediction in football. *Scientific reports*, 12(1):8638, 2022.

Kenta Oono and Taiji Suzuki. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In *International Conference on Learning Representations*, 2019.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1ecqn4YwB.

Boris N. Oreshkin, Arezou Amini, Lucy Coyle, and Mark J. Coates. FC-GAGA: Fully Connected Gated Graph Architecture for Spatio-Temporal Traffic Forecasting. In *AAAI*, 2021.

Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, 2018.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pages 26670–26698. PMLR, 2023.

Benjamin Paassen, Daniele Grattarola, Daniele Zambon, Cesare Alippi, and Barbara Eva Hammer. Graph edit networks. In *International Conference on Learning Representations*, 2020.

Soumyasundar Pal, Liheng Ma, Yingxue Zhang, and Mark Coates. RNN with Particle Flow for Probabilistic Spatio-temporal Forecasting. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8336–8348. PMLR, 18–24 Jul 2021.

Anastasios Panagiotelis, Puwasala Gamakumara, George Athanasopoulos, and Rob J Hyndman. Probabilistic forecast reconciliation: Properties, evaluation and score optimisation. *European Journal of Operational Research*, 306(2): 693–706, 2023.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *The Journal of Machine Learning Research*, 22(1): 2617–2680, 2021.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5363–5370, 2020.

Biswajit Paria, Rajat Sen, Amr Ahmed, and Abhimanyu Das. Hierarchically regularized deep forecasting. *arXiv preprint arXiv:2106.07630*, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Max Paulus, Dami Choi, Daniel Tarlow, Andreas Krause, and Chris J Maddison. Gradient estimation with stochastic softmax tricks. *Advances in Neural Information Processing Systems*, 33:5691–5704, 2020.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Fotios Petropoulos, Daniele Apiletti, Vassilios Assimakopoulos, Mohamed Zied Babai, Devon K Barrow, Souhaib Ben Taieb, Christoph Bergmeir, Ricardo J Bessa, Jakub Bijak, John E Boylan, et al. Forecasting: theory and practice. *International Journal of Forecasting*, 2022.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena Hierarchy: Towards Larger Convolutional Language Models. In *International Conference on Machine Learning*, pages 28043–28078. PMLR, 2023.

Chendi Qian, Andrei Manolache, Kareem Ahmed, Zhe Zeng, Guy Van den Broeck, Mathias Niepert, and Christopher Morris. Probabilistically Rewired Message-Passing Neural Networks. In *International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=Tj6Wcx7gVk.

Chendi Qian, Andrei Manolache, Christopher Morris, and Mathias Niepert. Probabilistic Graph Rewiring via Virtual Nodes. *arXiv preprint arXiv:2405.17311*, 2024b.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35: 14501–14515, 2022.

Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. *Advances in Neural Information Processing Systems*, 31: 7785–7794, 2018.

Syama Sundar Rangapuram, Lucien D Werner, Konstantinos Benidis, Pedro Mercado, Jan Gasthaus, and Tim Januschowski. End-to-end learning of coherent probabilistic forecasts for hierarchical time series. In *International Conference on Machine Learning*, pages 8832–8843. PMLR, 2021.

Syama Sundar Rangapuram, Shubham Kapoor, Rajbir Singh Nirwan, Pedro Mercado, Tim Januschowski, Yuyang Wang, and Michael Bohlke-Schneider. Coherent probabilistic forecasting of temporal hierarchies. In *International Conference on Artificial Intelligence and Statistics*, pages 9362–9376. PMLR, 2023.

Nikhil Rao, Hsiang-Fu Yu, Pradeep Ravikumar, and Inderjit S Dhillon. Collaborative Filtering with Graph Information: Consistency and Scalable Methods. In *Advances in neural information processing systems*, volume 2, page 7. Citeseer, 2015.

Kashif Rasul, Calvin Seward, Ingmar Schuster, and Roland Vollgraf. Autoregressive denoising diffusion models for multivariate probabilistic time

series forecasting. In *International Conference on Machine Learning*, pages 8857–8868. PMLR, 2021a.

Kashif Rasul, Abdul-Saboor Sheikh, Ingmar Schuster, Urs M Bergmann, and Roland Vollgraf. Multivariate Probabilistic Time Series Forecasting via Conditioned Normalizing Flows. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=WiGQBFuVRv.

Anirudh Ravula, Chris Alberti, Joshua Ainslie, Li Yang, Philip Minh Pham, Qifan Wang, Santiago Ontanon, Sumit Kumar Sanghai, Vaclav Cvicek, and Zach Fisher. ETC: Encoding long and structured inputs in Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7008–7024, 2017.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*, 2020.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

Emanuele Rossi, Henry Kenlay, Maria I Gorinova, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael Bronstein. On the Unreasonable Effectiveness of Feature propagation in Learning on Graphs with Missing Node Features. *arXiv preprint arXiv:2111.12128*, 2021.

Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, , Guzman Lopez, Nicolas Collignon, and Rik Sarkar. PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, page 4564–4573, 2021.

Yulia Rubanova, Ricky T. Q. Chen, and David K Duvenaud. Latent Ordinary Differential Equations for Irregularly-Sampled Time Series. In *Advances in*

Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019.

Donald B Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976.

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

Alex Rubinsteyn and Sergey Feldman. fancyimpute: An Imputation Library for Python. 2016. URL https://github.com/iskandr/fancyimpute.

Luana Ruiz, Luiz Chamon, and Alejandro Ribeiro. Graphon neural networks and the transferability of graph neural networks. *Advances in Neural Information Processing Systems*, 33:1702–1712, 2020.

T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

Mohammad Sabbaqi and Elvin Isufi. Graph-time convolutional neural networks: Architecture and theoretical analysis. *arXiv preprint arXiv:2206.15174*, 2022.

David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

Victor Garcia Satorras, Syama Sundar Rangapuram, and Tim Januschowski. Multivariate time series forecasting with latent graph inference. *arXiv preprint arXiv:2203.03423*, 2022.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *Advances in Neural Information Processing Systems*, 28, 2015.

Rajat Sen, Hsiang-Fu Yu, and Inderjit S Dhillon. Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. *Advances in Neural Information Processing Systems*, 32, 2019.

Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer, 2018.

Chao Shang and Jie Chen. Discrete Graph Structure Learning for Forecasting Multiple Time Series. In *Proceedings of International Conference on Learning Representations*, 2021.

Zezhi Shao, Zhao Zhang, Fei Wang, Wei Wei, and Yongjun Xu. Spatial-Temporal Identity: A Simple yet Effective Baseline for Multivariate Time Series Forecasting. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, page 4454–4458, 2022.

Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, Rif A. Saurous, Yannis Agiomvrgiannakis, and Yonghui Wu. Natural TTS Synthesis by Conditioning Wavenet on MEL Spectrogram Predictions. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018. doi: 10.1109/ICASSP.2018.8461368.

Jiaxin Shi, Ke Alexander Wang, and Emily Fox. Sequence Modeling with Multiresolution Convolutional Memory. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 31312–31327. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/shi23f.html.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *Advances in Neural Information Processing Systems*, 28, 2015.

Satya Narayan Shukla and Benjamin Marlin. Multi-Time Attention Networks for Irregularly Sampled Time Series. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=4c0J6lwQ4_.

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

David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access*, 9:79143–79168, 2021.

Slawek Smyl. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36 (1):75–85, 2020.

Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3): 714–735, 1997.

Indro Spinelli, Simone Scardapane, and Aurelio Uncini. Missing data imputation with adversarially-trained graph convolutional networks. *Neural Networks*, 129:249–260, 2020.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. *Advances in Neural Information Processing Systems*, 28, 2015.

Ljubiša Stanković, Danilo Mandic, Miloš Daković, Miloš Brajović, Bruno Scalzo, Shengxi Li, and Anthony G Constantinides. Data Analytics on Graphs Part II: Signals on Graphs. *Foundations and Trends® in Machine Learning*, 13, 2020.

Michael L Stein. *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media, 1999.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy Gradient methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems*, 12, 1999.

Souhaib Ben Taieb, James W Taylor, and Rob J Hyndman. Coherent probabilistic forecasts for hierarchical time series. In *International Conference on Machine Learning*, pages 3348–3357. PMLR, 2017.

Souhaib Ben Taieb, James W Taylor, and Rob J Hyndman. Hierarchical probabilistic forecasting of electricity demand with smart meter data. *Journal of the American Statistical Association*, 116(533):27–43, 2021.

Siyi Tang, Jared A Dunnmon, Qu Liangqiong, Khaled K Saab, Tina Baykaner, Christopher Lee-Messer, and Daniel L Rubin. Modeling multivariate biosignals with graph neural networks and structured state space models. In *Conference on Health, Inference, and Learning*, pages 50–71. PMLR, 2023.

Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. CSDI: Conditional score-based diffusion models for probabilistic time series imputation. *Advances in Neural Information Processing Systems*, 34:24804–24816, 2021.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022.

Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000. doi: 10.1126/science.290.5500.2319.

Peter Tino, M Cernansky, and Lubica Benusková. Markovian architectural bias of recurrent neural networks. *IEEE Transactions on Neural Networks*, 15(1): 6–15, 2004.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=7UmjRGzp-A.

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B Altman. Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17(6):520–525, 2001.

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

Emile van Krieken, Jakub Mikolaj Tomczak, and Annette Ten Teije. Storchastic: A Framework for General Stochastic Automatic Differentiation. In *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=KAFyFabsK88.

Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, 2018.

Dingsu Wang, Yuchen Yan, Ruizhong Qiu, Yada Zhu, Kaiyu Guan, Andrew Margenot, and Hanghang Tong. Networked time series imputation via position-aware graph enhanced variational autoencoders. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2256–2268, 2023.

Xiyuan Wang and Muhan Zhang. How powerful are spectral graph neural networks. In *International Conference on Machine Learning*, pages 23341–23362. PMLR, 2022.

Yuyang Wang, Alex Smola, Danielle Maddix, Jan Gasthaus, Dean Foster, and Tim Januschowski. Deep factors for forecasting. In *International conference on machine learning*, pages 6607–6617. PMLR, 2019.

Ezra Webb, Ben Day, Helena Andres-Terre, and Pietro Lió. Factorised neural relational inference for multi-interaction systems. *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data*, 2019.

Théophane Weber, Nicolas Heess, Lars Buesing, and David Silver. Credit assignment techniques in stochastic computation graphs. In *International*

Conference on Artificial Intelligence and Statistics, pages 2650–2660. PMLR, 2019.

Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053*, 2017.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Ian R White, Patrick Royston, and Angela M Wood. Multiple imputation using chained equations: issues and guidance for practice. *Statistics in medicine*, 30(4):377–399, 2011.

Shanika L Wickramasuriya. Probabilistic forecast reconciliation under the Gaussian framework. *Journal of Business & Economic Statistics*, pages 1–14, 2023.

Shanika L Wickramasuriya, George Athanasopoulos, and Rob J Hyndman. Optimal forecast reconciliation for hierarchical and grouped time series through trace minimization. *Journal of the American Statistical Association*, 114 (526):804–819, 2019.

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

Andrew J Wren, Pasquale Minervini, Luca Franceschi, and Valentina Zantedeschi. Learning Discrete Directed Acyclic Graphs via Backpropagation. *arXiv preprint arXiv:2210.15353*, 2022.

Dongxia Wu, Liyao Gao, Matteo Chinazzi, Xinyue Xiong, Alessandro Vespignani, Yi-An Ma, and Rose Yu. Quantifying uncertainty in deep spatiotemporal forecasting. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1841–1851, 2021a.

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ju_Uqw384Oq.

Yuankai Wu, Dingyi Zhuang, Aurelie Labbe, and Lijun Sun. Inductive Graph Neural Networks for Spatiotemporal Kriging. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4478–4485, 2021b.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 1907–1913, 2019.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi 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.

Zonghan Wu, Da Zheng, Shirui Pan, Quan Gan, Guodong Long, and George Karypis. TraverseNet: Unifying Space and Time in Message Passing for Traffic Forecasting. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Jingyun Xiao, Ran Liu, and Eva L Dyer. GAFormer: Enhancing Timeseries Transformers Through Group-Aware Embeddings. In *International Conference on Learning Representations*, 2024.

Sang Michael Xie and Stefano Ermon. Reparameterizable subset sampling via continuous relaxations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 3919–3925, 2019.

Qianxiong Xu, Cheng Long, Ziyue Li, Sijie Ruan, Rui Zhao, and Zhishuai Li. KITS: Inductive Spatio-Temporal Kriging with Increment Training Strategy. *arXiv preprint arXiv:2311.02565*, 2023.

Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. Towards K-means-friendly spaces: Simultaneous deep learning and clustering. In *International Conference on Machine Learning*, pages 3861–3870. PMLR, 2017a.

Dazhi Yang, Hao Quan, Vahid R Disfani, and Licheng Liu. Reconciling solar forecasts: Geographical hierarchy. *Solar Energy*, 146:276–286, 2017b.

Jiexia Ye, Juanjuan Zhao, Kejiang Ye, and Chengzhong Xu. How to build a graph-based deep learning architecture in traffic domain: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(5):3904–3924, 2020.

Yongchao Ye, Shiyao Zhang, and James JQ Yu. Spatial-temporal traffic data imputation via graph attention convolutional network. In *International Conference on Artificial Neural Networks*, pages 241–252. Springer, 2021.

Xiuwen Yi, Yu Zheng, Junbo Zhang, and Tianrui Li. ST-MVL: filling missing values in geo-sensory time series data. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, 2016.

Izzet B Yildiz, Herbert Jaeger, and Stefan J Kiebel. Re-visiting the echo state property. *Neural networks*, 35:1–9, 2012.

Xueyan Yin, Feifan Li, Yanming Shen, Heng Qi, and Baocai Yin. NodeTrans: A Graph Transfer Learning Approach for Traffic Prediction. *arXiv preprint arXiv:2207.01301*, 2022.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

Jinsung Yoon, William R Zame, and Mihaela van der Schaar. Multi-directional recurrent neural networks: A novel method for estimating missing data. In *Time Series Workshop at the 34th International Conference on Machine*, pages 1–5, 2017.

Jinsung Yoon, James Jordon, and Mihaela Schaar. Gain: Missing data imputation using generative adversarial nets. In *International Conference on Machine Learning*, pages 5689–5698. PMLR, 2018.

Jiaxuan You, Xiaobai Ma, Daisy Yi Ding, Mykel Kochenderfer, and Jure Leskovec. Handling Missing Data with Graph Representation Learning. *Advances in Neural Information Processing Systems*, 2020.

Bing Yu, Haoteng Yin, and Zhanxing 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.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. ST-UNet: A spatio-temporal U-network for graph-structured time series modeling. *arXiv preprint arXiv:1903.05631*, 2019.

Hsiang-Fu Yu, Nikhil Rao, and Inderjit S Dhillon. Temporal regularized matrix factorization for high-dimensional time series prediction. *Advances in Neural Information Processing Systems*, 29, 2016.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in Neural Information Processing Systems*, 30, 2017.

Daniele Zambon. Anomaly and Change Detection in Sequences of Graphs. *PhD thesis*, 2022.

Daniele Zambon and Cesare Alippi. AZ-whiteness Test: A Test for Signal Uncorrelation on Spatio-Temporal Graphs. In *Advances in Neural Information Processing Systems*, 2022.

Daniele Zambon and Cesare Alippi. Where and How to Improve Graph-based Spatio-temporal Predictors, 2023. URL http://arxiv.org/abs/2302.01701.

Daniele Zambon, Cesare Alippi, and Lorenzo Livi. Concept Drift and Anomaly Detection in Graph Streams. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2018. doi: 10.1109/TNNLS.2018.2804443.

Daniele Zambon, Andrea Cini, Lorenzo Livi, and Cesare Alippi. Graph state-space models. *arXiv preprint arXiv:2301.01741*, 2023.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 11121–11128, 2023.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *International Conference on Learning Representations*, 2020.

Guoqiang Zhang, B Eddy Patuwo, and Michael Y Hu. Forecasting with artificial neural networks: The state of the art. *International journal of forecasting*, 14 (1):35–62, 1998.

Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit Yan Yeung. GaAN: Gated Attention Networks for Learning on Large and Spatiotemporal Graphs. In *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, 2018.

Michael Zhang, Khaled Kamal Saab, Michael Poli, Tri Dao, Karan Goel, and Christopher Re. Effectively Modeling Time Series with Simple Discrete State Spaces. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=2EpjkjzdCAa.

Xiang Zhang, Marko Zeman, Theodoros Tsiligkaridis, and Marinka Zitnik. Graph-Guided Network For Irregularly Sampled Multivariate Time Series. In *International Conference on Learning Representations, ICLR*, 2022.

Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE transactions on intelligent transportation systems*, 21(9): 3848–3858, 2019.

Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. GMAN: A graph multi-attention network for traffic prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1234–1241, 2020.

Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, Jianzhong Qi, Chaochao Chen, and Longbiao Chen. INCREASE: Inductive Graph Representation Learning for Spatio-Temporal Kriging. In *Proceedings of the ACM Web Conference 2023*, pages 673–683, 2023.

Yu Zheng, Xiuwen Yi, Ming Li, Ruiyuan Li, Zhangqing Shan, Eric Chang, and Tianrui Li. Forecasting fine-grained air quality based on big data. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 2267–2276, 2015.

Weida Zhong, Qiuling Suo, Xiaowei Jia, Aidong Zhang, and Lu Su. Heterogeneous spatio-temporal graph convolution network for traffic forecasting with missing values. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 707–717. IEEE, 2021.

Dengyong Zhou, Olivier Bousquet, Thomas Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. *Advances in Neural Information Processing Systems*, 16, 2003.

Fan Zhou, Chen Pan, Lintao Ma, Yu Liu, Shiyu Wang, James Zhang, Xinxin Zhu, Xuanwei Hu, Yunhua Hu, Yangfei Zheng, et al. SLOTH: structured learning and task-based optimization for time series forecasting on hierarchies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 11417–11425, 2023.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence

time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021.

Daniel Zügner, François-Xavier Aubet, Victor Garcia Satorras, Tim Januschowski, Stephan Günnemann, and Jan Gasthaus. A study of joint graph inference and forecasting. *arXiv preprint arXiv:2109.04979*, 2021.