Towards Robust Deep Learning Methods for Time Series Data and their Applications

Chrysoula Kosma

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2023IPPAX140

Thèse de doctorat

Towards Robust Deep Learning Methods for Time Series Data and their Applications

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à École Polytechnique

École doctorale n°626 Dénomination École Doctorale IP Paris (ED IP Paris)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 27/11/2023, par

CHRYSOULA KOSMA

Composition du Jury :

Catuscia Palamidessi Director of Research at INRIA, École Polytechnique (LIX)	Président
Laurent Oudre Professor, ENS Paris-Saclay University (Centre Borelli)	Rapporteur
Germain Forestier Professor, University of Haute-Alsace (IRIMAS)	Rapporteur
Vasileios Megalooikonomou Professor, University of Patras (CEID)	Examinateur
Themis Palpanas Professor, Université Paris Cité, French University Institute (IUF)	Examinateur
Yang Yang Associate Professor, Zhejiang University	Examinateur
Fragkiskos Malliaros Assistant Professor, CentraleSupélec, Paris-Saclay University	Examinateur
Michalis Vazirgiannis Professor, École Polytechnique (LIX)	Directeur de thèse
Jean-Marc Steyaert Emeritus Professor, École Polytechnique (LIX)	Co-directeur & Invité

I was taught that the way of progress was neither swift nor easy.

— Marie Curie

Dedicated to my mother, Sofia, the most impactful teacher I have ever had and the strongest woman I know.

ABSTRACT

Time series data is abundant in various fields including finance, energy, and social networks. Extracting knowledge from time series, to perform subsequent tasks such as forecasting, interpolation, and classification, has therefore become essential in several scientific and industrial domains. Deep Learning (DL) models, i. e., neural networks, have emerged as powerful tools in various data applications since they provide flexible methods for knowledge extraction and can easily learn from multiple sources of data. However, DL models typically require large datasets for improved generalization and often lack interpretability, thus their industrial application remains limited. Specifically for time series modeling, the adaptability of neural networks to the continuous-time dynamics of such data remains an open question. Additionally, neural networks' origins in discrete sequential tasks, like text analysis, limit their understanding of complex time-dependent series properties. To bridge this gap, systematically assessing the effectiveness of DL architectures under diverse time series characteristics and distortions is crucial. Time series distortions encompass extrinsic factors, such as noisy observations and irregular sampling. Additionally, intrinsic features, such as strong inter-variable and inter-temporal correlations and prior knowledge (e. g., imposed constraints) of the underlying dynamics are prominent among different time series datasets. However, both extrinsic distortions and intrinsic characteristics pose significant challenges to the robust deep learning modeling of time series. The current study critically evaluates the performance of existing DL models across these challenges and proposes new modeling approaches.

In terms of time series regression tasks, such as forecasting, standard evaluation metrics often fall short of capturing the nuanced statistical characteristics of the series. Simultaneously, the robustness of training loss functions while training for time series tasks often proves insufficient, particularly when dealing with abrupt changes and data noise. Additionally, conventional sequential architectures like RNNs and CNNs, designed to handle fixed time intervals between successive time steps, may struggle with the complexity of irregular time series datasets. Moreover, the prevalence of strong temporal and inter-variable correlations in multivariate time series datasets necessitates the development of sophisticated networks capable of adeptly handling variable dependencies across both time and variables. As a response to these challenges, our contributions encompass the formulation of robust loss functions and metrics tailored for forecasting, the adaptation of traditional neural architectures, e. g., CNNs, to accommodate irregular sampling, and the creation of algorithms adept at capturing rich embeddings of intricate temporal and inter-variable dependencies. Furthermore, our study extends to practical applications in spatio-temporal data and dynamic systems, integrating physics-informed models to harness prior knowledge.

In conclusion, this thesis delivers a comprehensive exploration of cutting-edge deep learning approaches for time series modeling, shedding light on their limitations while introducing novel contributions. The extensive discussion section outlines potential directions for future work, including the exploration of generative modeling for time series. We envision that

the topics explored, coupled with our proposed methods, present promising avenues for significantly enhancing the robustness of neural and automatic modeling for time series data.

RÉSUMÉ

Les données de séries temporelles sont abondantes dans divers domaines, notamment la finance, l'énergie et les réseaux sociaux. L'extraction de connaissances à partir des séries temporelles, pour effectuer des tâches ultérieures telles que la prévision, l'interpolation et la classification, est donc devenue essentielle dans plusieurs domaines scientifiques et industriels. Les modèles d'apprentissage profond (deep learning - DL), c'est-à-dire les réseaux neuronaux, se sont révélés être des outils puissants dans diverses applications de données, car ils fournissent des méthodes flexibles pour l'extraction de connaissances et peuvent facilement apprendre à partir de sources de données multiples. Cependant, les modèles d'apprentissage profond nécessitent généralement de grands ensembles de données pour une meilleure généralisation et manquent souvent d'interprétabilité, de sorte que leur application industrielle reste limitée. En ce qui concerne plus particulièrement la modélisation des séries temporelles, l'adaptabilité des réseaux neuronaux à la dynamique à temps continu de ces données reste une question ouverte. En outre, l'origine des réseaux neuronaux dans des tâches séquentielles discrètes, comme l'analyse de texte, limite leur compréhension des propriétés complexes des séries dépendant du temps. Pour combler cette lacune, il est essentiel d'évaluer systématiquement l'efficacité des architectures DL en fonction de diverses caractéristiques et distorsions des séries temporelles. Les distorsions des séries temporelles englobent des facteurs extrinsèques, tels que des observations bruyantes et un échantillonnage irrégulier. En outre, les caractéristiques intrinsèques, telles que les fortes corrélations inter-variables et inter-temporelles et la connaissance préalable (contraintes imposées) de la dynamique sous-jacente, sont prédominantes parmi les différents ensembles de données de séries temporelles. Cependant, les distorsions extrinsèques et les caractéristiques intrinsèques posent des défis importants à la modélisation robuste des séries temporelles par l'apprentissage profond. L'étude actuelle évalue de manière critique la performance des modèles d'apprentissage profond existants face à ces défis et propose également de nouvelles approches de modélisation.

En ce qui concerne les tâches de régression de séries temporelles, telles que la prévision, les mesures d'évaluation standard ne parviennent souvent pas à capturer les caractéristiques statistiques nuancées des séries. Simultanément, la robustesse des fonctions de perte d'apprentissage lors de l'apprentissage pour les tâches de séries temporelles s'avère souvent insuffisante, en particulier lorsqu'il s'agit de changements brusques et de bruit de données. En outre, les architectures séquentielles conventionnelles telles que les RNN et les CNN, conçues pour traiter des intervalles de temps fixes entre des étapes temporelles successives, peuvent se heurter à la complexité des ensembles de données de séries temporelles irrégulières. En outre, la prévalence de fortes corrélations temporelles et inter-variables dans les ensembles de données de séries temporelles multivariées nécessite le développement de réseaux sophistiqués capables de gérer habilement les dépendances variables à la fois dans le temps et entre les variables. En réponse à ces défis, nos contributions englobent la formulation de fonctions de perte robustes et de mesures adaptées à la prévision, l'adaptation des architectures neuronales

traditionnelles pour tenir compte de l'échantillonnage irrégulier, par example CNNs, et la création d'algorithmes capables de capturer de riches enchâssements de dépendances temporelles et inter-variables complexes. En outre, notre étude s'étend aux applications pratiques des données spatio-temporelles et des systèmes dynamiques, en intégrant des modèles informés par la physique pour exploiter les connaissances antérieures.

En conclusion, cette thèse propose une exploration complète des approches d'apprentissage profond de pointe pour la modélisation des séries temporelles, mettant en lumière leurs limites tout en introduisant de nouvelles contributions. La section de discussion détaillée décrit les directions potentielles pour les travaux futurs, y compris l'exploration de la modélisation générative pour les séries temporelles. Nous pensons que les sujets explorés, associés aux méthodes que nous proposons, présentent des voies prometteuses pour améliorer de manière significative la robustesse de la modélisation neuronale et automatique des données de séries temporelles.

PUBLICATIONS

In terms of this doctorate thesis, the following contributions have been provided:

[1] Chrysoula Kosma, Giannis Nikolentzos, George Panagopoulos, Jean-Marc Steyaert, and Michalis Vazirgiannis. "Neural Ordinary Differential Equations for Modeling Epidemic Spreading." *Transactions on Machine Learning Research* (2023).

[2] Chrysoula Kosma, Giannis Nikolentzos, and Michalis Vazirgiannis. "Time-Parameterized Convolutional Neural Networks for Irregularly Sampled Time Series." *arXiv preprint arXiv:2308.03210* (2023). (submitted for publication).

[3] Chrysoula Kosma, Giannis Nikolentzos, Nancy Xu, and Michalis Vazirgiannis. "Time Series Forecasting Models Copy the Past: How to Mitigate." In: *International Conference on Artificial Neural Networks*. Springer. 2022, pp. 366–378.

[4] Nancy Xu, Chrysoula Kosma, and Michalis Vazirgiannis. "TimeGNN: Temporal Dynamic Graph Learning for Time Series Forecasting." In: *Proceedings of the 12th International Conference on Complex Networks and Their Applications*. 2023. (to appear).

ACKNOWLEDGMENTS

This thesis would not have been possible without the contributions and encouragement of several individuals, that made my PhD journey a wonderful experience. I want to express my deep appreciation and gratitude to my supervisor, Prof. Michalis Vaziriannis, for their unwavering support, guidance, and mentorship throughout the research process and the chance he gave me to pursue my doctorate degree along with his wonderful team. His encouragement and trust have not only allowed me to work independently but have also nurtured my creativity, for which I am truly appreciative. I would also like to express my sincere appreciation to my co-supervisor, Prof. Jean-Marc Steyaert whose contribution has been crucial in enhancing my research endeavors through insightful ideas and engaging discussions. I am also thankful to the members of the jury, and especially the reviewers, for the time they devoted to studying my manuscript and for their valuable feedback and fruitful suggestions. I would also like to thank my main collaborator and friend, Prof. Giannis Nikolentzos. His exemplary dedication to our research field and his constant push for improvement, have been a great source of inspiration for my resilience as a researcher and his advice has significantly boosted my research skills. Finally, I feel honoured and grateful for the IP Paris scholarship for theses in AI, provided by ANR, which gave significant financial support for my doctoral studies.

My dear friends are also a huge part of this effort. I would like to thank those who were physically present, inspired me and supported me in challenging times but also helped me have fun on a daily basis, specifically Michalis, Giannis, Panagopoulos, Iakovos, Dasoulas, Kapsalis, Kyriakos, Lena, Sofia and Manousakis. I am also grateful for the friends and great researchers that I met and collaborated with within the research group, Nancy, Johannes, Ashraf, Virgile, Christos, Yanzhu, Hadi and the whole DaSciM team. I would also like to extend my heartfelt gratitude to my long-term friends Marilena, Mary and Grigoris who have always been there for me, despite the physical distance that separated us. Their friendship and presence in my life have been a true blessing. Finally, I deeply thank my boyfriend, Manolis, whose support and presence have been a constant source of strength throughout this journey. His encouragement and belief in me, as well as his mindset, have inspired me to reach new heights personally.

My greatest gratitude goes to the three most important people in my life who stand by my side: my parents, Sofia and Christos, and my sister, Katerina. My parents have always been my greatest role models, exemplifying professional integrity, great kindness, humility, and unconditional love and trust in every significant moment of my life. My sister has been my greatest source of joy and companionship, always taking care of me, and I believe she is my biggest fan, just as I am hers.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

DL Deep Learning

MSE Mean Squared Error

MAE Mean Absolute Error

SMAPE Symmetric Mean Absolute Percentage error

DTW Dynamic Time Warping

TDI Temporal Distortion Index

NLP Natural Language Processing

ODEs Ordinary Differential Equations

RNNs Recurrent Neural Networks

LSTMs Long Short-Term Memory Networks

GRUs Gated Recurrent Units Networks

CNNs Convolutional Neural Networks

TCN Temporal Convolutional Network

GNNs Graph Neural Networks

MPNN Message Passing Neural Network

AE Autoencoder

GANs Generative Adversarial Networks

TPCNN Time-Parameterized Convolutional Neural Network

TimeGNN Time Graph Neural Network

SIR Susceptible-Infectious-Recovered

GN-ODE Graph Neural ODE Network

Part I

SETTING THE STAGE: INTRODUCTION AND BACKGROUND

1 INTRODUCTION

1.1 MODERN DEEP LEARNING CHALLENGES FOR TIME SERIES DATA

The extraction of valuable insights from time series data can be useful in various practical situations and challenging real-world applications. In particular, predicting the future values of observed time series [65], interpolating missing points [196] and classifying multivariate [115], fully or partially observed inputs, are important in almost all scientific and engineering domains, including economics, business intelligence, meteorology, telecommunication and energy [10, 201, 239, 308]. At the same time, extracting embeddings that capture both inter-variable and inter-temporal dependencies in time series is crucial [82]. The need for learning large amounts of historical data and capturing non-linear patterns has drawn attention to Deep Learning (DL) models. However, similar to the the pitfalls of discrete statistical models [52], deep neural network architectures for time series mostly derive from networks for other sequential tasks, such as text that is discrete, thus they have limited intuition on the continuous time-dependent properties of the series [47]. Since DL architectures make fewer structural assumptions compared to statistical approaches, they often require large datasets to make accurate predictions. Additionally, in the presence of input noise and perturbations, such as irregular sampling, missing observations, high autocorrelations and others, they often fail or show unstable behavior in capturing the complex statistical properties of the underlying continuous-time dynamics. Enhancing the robustness of DL architectures when handling specific types of data, has been a research area of paramount importance in the recent years [303] and is also very prominent to the problem of effective time series modeling.

Distortions in time series data refer to any factors or anomalies that disrupt the idealized, regular, and continuous nature of the time series [83]. Those typically include irregularities, errors, or artifacts that are introduced during data collection, such as irregularly sampled or missing observations, noise and outliers, repeating patterns (e. g., cyclical) that are not associated with the underlying process and distributional shifts of the data over time [254]. On the other hand, inherent characteristics of the data, such as high correlations among variables in different timestamps, as well as, prior knowledge about the evolution of the series (e. g., constraints and approximations on governing equations) [128, 168] constitute aspects that necessitate the development of complex models. Such peculiarities, while they do not directly account for extrinsic distortions can pose challenges to robust and continuous modeling of time series data.

In the next sections, we highlight some key challenging characteristics of time series data that motivate our contributions to the development of robust deep learning approaches for time series modeling. Following this discussion, we outline the topics that will be covered in this study.

1.1.1 *Noisy Observations*

CHALLENGES. Noise in time series and data, in general, refers to random variations or errors that can be present in measurements or observations. These variations can result from various sources, including measurement inaccuracies, or other factors that introduce randomness into the data. For instance, time series may sometimes miss key correlated variables or features that are critical for making accurate predictions, which can lead to increased noise on the available data. Noise and outliers are therefore considered a form of distortion in data. In terms of preprocessing, techniques such as noise reduction, smoothing, or filtering are commonly employed to mitigate the impact of noise and enhance the reliability of the data for further analysis or modeling for subsequent tasks.

MOTIVATION. The robustness of DL approaches under noisy inputs is crucial and has been a long-standing challenge while training with deep neural networks [6]. Robust models should effectively filter out or minimize the impact of noise, ensuring that it does not unduly influence their predictions. At the same time, robust models should maintain stable and consistent performance across different datasets with varying levels of noise, instead of overfitting the training data. In terms of time series, robust deep learning modeling incorporates the loss function design [17, 57, 97, 98] as well as the neural architecture module design to implicitly or explicitly handle noisy observations.

1.1.2 *Irregular Sampling*

CHALLENGES. Time series data are often collected with the assumption of regular sampling, where data points are observed at consistent time intervals. In many real-world scenarios, time series data may not be collected at fixed intervals, leading to irregular sampling. Irregular sampling can be considered a form of distortion in time series data, particularly when the irregularity deviates significantly from the expected or desired regular sampling frequency. This can result in data points that are not uniformly spaced in time and varying time gaps between observations, creating discontinuities in the time series. Additionally, for multivariate inputs, observations might be missing due to irregular sampling, which can result in gaps in the input series. These gaps can hinder the continuous analysis and modeling of the data if not treated properly. However, several time series models, from statistical to deep learning ones, assume fixed intervals between observation times, thus such kind of time series data necessitates important modifications in existing approaches to enable continuous modeling.

MOTIVATION. Instead of ignoring or discarding time series segments with missing data [164], robust DL models should provide reasonable estimates for the missing values based on the available information. Robustness also implies that the model can handle different lengths of missing data segments. Some segments might include just a few missing values, while others might form larger gaps in the input series. The models should be capa-

ble of accommodating these variations, e. g., by introducing continuous-time deep learning approaches and minimizing preprocessing steps that hinder end-to-end modeling of the underlying dynamics [31, 132, 214, 234].

1.1.3 *Strong Correlations (Inter-Variable and Inter-Temporal)*

CHALLENGES. Many datasets contain multiple variables or features, which can be correlated with each other, meaning that they exhibit some degree of statistical association. For example, in a healthcare dataset, variables like blood pressure, heart rate, and cholesterol levels might be correlated and are overall related to the patient's health condition. Such dependencies can be referred to as inter-variable. On the other hand, inter-temporal correlations refer to the dependencies or relationships that exist between observations at different time steps [133]. For example, in financial data, stock prices today may be correlated with stock prices from the previous days due to trends or market dynamics. High correlations between variables or strong temporal dependencies in time series data are not distortions in the traditional sense. Instead, they represent inherent characteristics of the data. These relationships exist naturally and provide valuable information for time series analysis and modeling [147, 231]. While such dependencies are not distortions, they present challenges when building DL architectures, as they often require more complex modeling techniques to be captured effectively.

MOTIVATION. Therefore, robust deep learning models should be able to account for inter-variable correlations and make accurate predictions while considering how changes in one variable might affect others. At the same time, they should be able to capture temporal dependencies on the time series evolution which is crucial for generating accurate forecasts or detecting anomalies.

1.1.4 *Prior Knowledge on the Underlying Dynamics*

CHALLENGES. In many real-world applications, data must adhere to certain physical constraints [242]. For example, in a physics experiment, measurements must satisfy conservation laws (e. g., conservation of energy or mass). Dynamic systems constitute prominent examples of time series data in which the state of the system evolves over time according to specific rules or equations and are present in different fields, e. g., mechanics, biology, economics and others [5, 108]. Constraints or laws of dynamics in data can not be considered distortions. In contrast, they represent the inherent rules and principles that govern how data behaves in a specific context and are fundamental to understanding the true nature of the data and the underlying system it represents. However, failing to effectively incorporate such prior knowledge about the temporal process into the modeling approach, can lead to significant errors [255].

MOTIVATION. To conclude, specific knowledge and constraints about the data, which is particularly common in the case of dynamical systems, can be a significant aspect of robust DL modeling for time series [255]. In this case, robustness can be evaluated based on the model's ability to generalize from the training data to unseen data while maintaining fidelity to the underlying physical or dynamic principles.

1.2 THESIS OUTLINE

The challenging properties of time series data, e. g., noise, irregular sampling, strong correlations of variable periodicities and prior knowledge on constraints on the underlying dynamics, as presented in Section 1.1.1, Section 1.1.2, Section 1.1.3, Section 1.1.4, pose significant barriers on DL models' robustness and therefore performance and generalization capabilities. While several studies focus on improving models' performance in various tasks and domains of applications, there is yet little understanding of the general requirements of end-to-end continuous and robust modeling of time series and dynamical systems using deep neural networks. At the same time, there is room for improvements in existing approaches, in terms of computational resources or complexity, as well as methods that have not been adequately explored or extended to enable robust modeling. Therefore, our study consists of validating and developing neural network architectures, that are able to handle continuous-time dynamics and noise and perturbations of large real-world time series, by extending standard deep learning approaches on various tasks.

We next present the organization of the presentation of the studies conducted in terms of this thesis, as well as the theoretical background information provided for understanding the topics discussed in each chapter.

INTRODUCTION AND BACKGROUND. In Chapter 1, we presented some of the most dominant challenges on the robustness of modern deep learning approaches for real-world time series data. In this chapter, we also provide the thesis organization and an outline of the main chapters and topics that are discussed. In Chapter 2, we provide some key definitions and notations for time series data and an overview of traditional methods and recent advances in machine and deep learning approaches. Those are important for understanding the background of the existing methods in this field and contain information about modules and properties to which we will refer in the main chapters that follow.

INCREASING ROBUSTNESS VIA LOSS FUNCTIONS. The largely dominant loss functions to train and evaluate deep models in time series point forecasting are L_p norms, i. e., Mean Squared Error (MSE) and its variants (e. g., Mean Absolute Error (MAE), Symmetric Mean Absolute Percentage error (SMAPE)). Even though MSE and related loss functions enjoy some nice properties, when dealing with real-world data with multiple co-occurring patterns, abrupt changes and noisy components, these functions might become sensitive to noise. In Chapter 3, we aim to address the issue of 'mimicking' in time series forecasting, particularly in noisy or

incomplete datasets (i. e., missing correlated variables). This can be achieved by introducing a robust loss function that extends the standard MSE loss with a regularization term.

In Section 3.1, we provide an introduction to the notations and background related to forecasting techniques and loss functions, setting the foundation for the analysis of our contribution in the next section. Section 3.2 formalizes the phenomenon of 'mimicking', by providing a quantifiable definition to help practitioners identify models that replicate past values instead of making genuine predictions. Synthetic experiments are conducted to assess the relationship between low MSE values and forecast quality. We also emphasize the importance of improving forecasting quality with standard loss functions and introduce a versatile regularization term applicable to various neural network architectures. Lastly, the proposed loss function is validated on common deep neural networks and real-world datasets, with an emphasis on evaluating change point accuracy in the predicted series.

However, apart from improving the optimization process of the models, several studies have focused on incorporating time information in the learnable hidden states of the models, knowing that this information is essential in terms of another extrinsic distortion in the data, called irregular sampling.

HANDLING IRREGULAR SAMPLING IN TIME SERIES. In Chapter 4, we focus on robust modeling of irregularly sampled multivariate time series data, a common challenge in various application domains. These time series often exhibit sparse, non-aligned, and incomplete observations across different variables, presenting unique challenges for modeling and analysis. Traditional sequential neural network architectures, including Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs), assume regularly spaced observations, which can limit their effectiveness in handling irregular time intervals.

Section 4.1 presents the most dominant continuous-time modeling techniques, giving an overview of their key components and the architectural requirements to handle irregularly-sampled time series. While RNN variants have been proposed to address irregular sampling, CNNs have not received as much attention in this context. In response to this gap, in Section 4.2, we introduce the Time-Parameterized Convolutional Neural Network (TPCNN), an extension of vanilla 1-D CNNs to handle time series of variable time intervals. The proposed model utilizes kernels that are explicitly initialized with temporal information. These kernels can capture the underlying dynamics of continuous-time inputs, making them well-suited for handling irregularly sampled data. To assess TPCNN's performance, we conduct a series of experiments involving real-world multivariate time series datasets, focusing on tasks like interpolation and classification. The outcomes of our experiments not only showcase TPCNN's competitiveness and efficiency when compared to existing methods but also emphasize its distinctive feature: interpretability. TPCNN enables us to gain deeper insights into the input time series by employing adaptable time functions.

Designing architectures that can capture intricate inter-variable and inter-temporal correlations to extract valuable discrete embeddings, is an additional novel direction to boost the robustness of deep neural networks for time series. When such correlations are not explicitly

extracted due to their variable periodicities can act like additional noise to the input, leading to poor optimization of the models.

EXTRACTING DISCRETE EMBEDDINGS FROM CONTINUOUS TIME SERIES VIA GRAPH LEARNING. Time series are continuous in nature, but at the same time, they might encounter several distortions (e. g., noise, irregular timestamps) potentially due to their collection process. Contrary to other types of sequential data, e. g., text that is discrete, it is hard to introduce algorithms for discretizing time series and capture their information, similarly to the context or meaning of words, in descriptive embeddings. Discretization algorithms for time series could pave the way for summarizing their information in task-specific representations, but necessitate the construction of efficient methodologies that capture substantially inter-variable and inter-temporal correlations, while not violating the series evolution of dynamics. This idea is extensively presented in Chapter 5.

Graph representation learning algorithms that operate on spatio-temporal data can boost our creativity towards this direction. We present an overview of such algorithms and existing works on time series in Section 5.1. Building upon the motivation for graph-based embeddings that effectively capture correlations in complex time series data, we recognize that the existing approaches, though promising, often encounter challenges related to computational intensity and scalability. We, therefore, propose the Time Graph Neural Network (TimeGNN) framework in Section 5.2 with the objective of harnessing the power of graph neural networks while addressing these challenges. TimeGNN introduces a dynamic temporal graph representation that retains the capacity to capture intricate inter-variable and inter-temporal correlations in a remarkably efficient manner. The proposed model not only accelerates the inference process, with speeds surpassing state-of-the-art methods but also upholds good forecasting performance in several benchmark datasets. At the same time, TimeGNN bridges the gap between the need for robust, correlation-sensitive embeddings and the imperative of computational efficiency in handling large-scale time series datasets.

It is crucial to acknowledge that the concept of robustness, especially when dealing with time series data, encompasses more than just addressing distortions and capturing inherent correlations. Specifically, situations where data adheres to specific rules or equations governing the temporal system but isn't accurately accounted for in the model should also be considered.

APPLICATIONS ON DYNAMICAL SYSTEMS. In Chapter 6, we delve into the integration of prior knowledge, particularly for modeling dynamical systems, that constitute a characteristic example of such data. Dynamical systems follow specific physical constraints or laws of dynamics, which far from being distortions, are inherent to the data. Those are, in essence, guiding principles that can enhance the accuracy and robustness of our forecasting models. Leveraging this prior knowledge enables us to develop models that align with the underlying physics or dynamics governing the time series, resulting in more reliable predictions.

In Section 6.1, we present an overview of the intersection of physics-based knowledge and DL, exploring how existing approaches extend predictive models, particularly when applied to dynamic systems with inherent physical constraints. Following this, in Section 6.2, we

focus on a specific application for dynamical systems, i. e., spreading processes on networks of contacts under the constraint of epidemic compartmental models. In this study, we focus on the Susceptible-Infectious-Recovered (SIR) epidemic model on networks. While this model is theoretically approximated by a system of ordinary differential equations, the mathematical derivation of the system's solution is infeasible in practice due to its computational complexity. We tackle this challenge by devising a streamlined approximation system, enabling us to harness recent advancements in neural ordinary differential equations. Our proposed neural architecture proves adept at predicting epidemic courses on networks, demonstrated through extensive experimentation with various datasets and settings. The results affirm the potential of interpretable and robust neural networks in advancing the field of epidemic spreading, providing both improved predictive capabilities and computational efficiency, even on large networks where conventional methods falter.

DISCUSSION. In Chapter 7, we present a comprehensive analysis of the topics presented in this thesis, focusing on their interactions with intricate challenges and opportunities in the domain of deep learning for time series, where the interpretability, generalizability, and robustness of models are of paramount importance. Our aim is to dissect the broader applicability of the methods and their practical implications. Furthermore, we explore future research directions stemming from these findings. Through this discussion, we shed light on the intricate performance of deep learning models in the face of real-world time series complexities, offering insights that can guide both researchers and practitioners in leveraging these models effectively or exploring new directions in the field.

Finally, Appendix A presents additional results and specifications on the employed DL architectures for some of the contributions we introduce in the chapters of Part ii.

2 BACKGROUND: LEARNING FROM TIME SERIES DATA

2.1 TIME SERIES DATA

Time series data, a collection of sequentially recorded observations over time, can be found in various scientific and engineering fields and encompass a wide range of applications. For instance, in finance, stock prices exhibit daily fluctuations, while economic indicators, e. g., inflation rates, are regularly tracked over the years [75]. In climate science, weather data such as temperature, precipitation, and wind speed measurements are collected at specific locations [230]. In healthcare, patients' vital signs, such as heart rate and blood pressure, are continuously monitored [30]. Additionally, traffic data [101, 156], social media metrics [76], energy production or consumption patterns [237], and environmental factors, e. g., air quality measurements [104], form examples of time series data. This versatile data type can be therefore significant for forecasting, pattern recognition, and trend analysis in a wide range of application domains.

2.1.1 *Fundamentals*

Definition 1 (Time Series)**.** *A time series is a sequence of data points collected, observed, or recorded at successive time intervals. Let* $\mathbf{x}_{1:T} = (x_1, \ldots, x_T)$ *a sequence of* T *data points,* t *the time index and* x_t *represents the value of the series at time step* t*. For multivariate time series, the notation above can be extended to include multiple variables and additional covariates. For the multivariate case, let* $\mathcal{X} = \{\mathbf{x}_{i,1:T_i}\}_{i=1}^{N}$ *a set of* N *univariate time series, where* $\mathbf{x}_{i,1:T_i} = (x_{i,1}, \ldots, x_{i,T_i})$ *the* i*-th time series. The set of time series might be associated with additional variables or covariates that can be used to explain or predict the behavior of each time series from the multivariate collection.*

For the definitions of Chapter 2, for simplicity we refer to the notation of univariate time series, i. e., x_t, but (DL) models can directly be extended to deal with multivariate inputs.

TIME SERIES MAIN COMPONENTS. The main components of a time series are often identified to correspond to the following patterns [180]:

- **Trend.** It represents the long-term movement or pattern in the data. It captures whether the data tends to increase, decrease, or remain relatively constant over time. Trends can be linear, exponential, or follow other patterns.
- **Seasonality.** This component pertains to the consistent and recurring variations or patterns in the data that happen at specific intervals. An illustration of seasonality can be observed in retail sales, which tend to surge during the holiday season.

- **Cyclic Patterns.** Contrary to the predictable and recurring nature of the seasonal component, cyclic patterns do not adhere to precise and predetermined time intervals. They denote longer-term oscillations or fluctuations that are less consistent than seasonal trends. These cycles can endure for multiple years and may lack a predetermined length.
- **Irregular or Residual Component.** Residuals capture the stochastic fluctuations or random noise in the data that cannot be ascribed to the aforementioned components and are very prominent in real-world data.

A time series x_t can be directly decomposed into the aforementioned main components, as follows:

$$x_t = T_t + S_t + R_t$$

that constitutes an additive decomposition model, where T_t captures the trend and cyclic patterns, S_t represents the seasonal patterns and R_t is the remaining irregular component. Similarly, a multiplicative decomposition model or more sophisticated techniques can be employed based on the complexity and general evolution of the series [49, 59].

STATIONARITY. Stationarity is a key notion in the field of time series analysis. Stationary time series are those in which statistical characteristics, including mean, variance and autocorrelation, exhibit no change during their whole length [180]. In other words, a stationary time series has consistent behavior throughout time. There are two main types of stationarity:

- **Strict Stationarity.** In this case, the joint probability distribution of any set of observations in the time series is the same at different time points. This means that the statistical properties do not change with time.
- **Weak Stationarity (or Covariance Stationarity).** In this case, the time series exhibits a consistent mean and variance across time, and the autocovariance (or autocorrelation) between any two observations is solely determined by the time lag between them, rather than the specific moment at which they occur.

Certain statistical modeling approaches presuppose stationarity in the input series, which makes these properties critical [145]. A typical transformation is based on differencing the original time series at lagged time points. For instance, first-order differencing, i. e., at lag 1, $\Delta x_t = x_t - x_{t-1}$, is often used to achieve stationarity prior to employing a statistical model.

AUTOCORRELATION. Autocorrelation captures the degree of linear association between a time series and a lagged version of itself [18]. It quantifies how a data point at a particular time is related to data points at previous times. Mathematically, the autocorrelation function (ACF) at lag k for a time series x_t is defined as:

$$\mathrm{ACF}(k) = \frac{\mathrm{Cov}(x_t, x_{t-k})}{\sqrt{\mathrm{Var}(x_t) \cdot \mathrm{Var}(x_{t-k})}}$$

where the covariance $\mathrm{Cov}(\cdot,\cdot)$ measures the degree to which two random variables change together and variance $\mathrm{Var}(\cdot)$ measures the spread or dispersion of a random variable. If

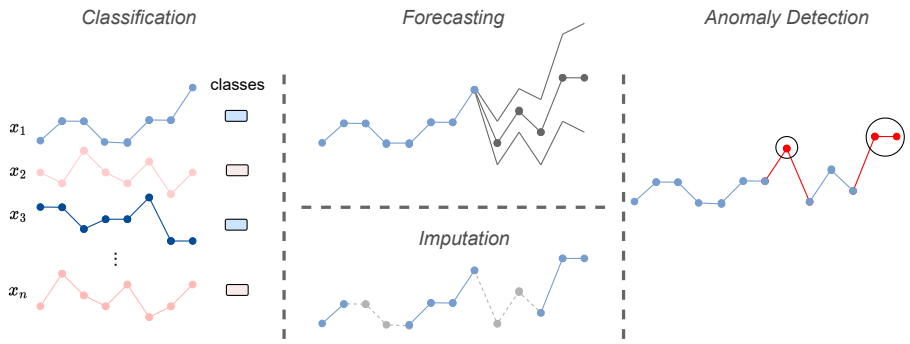


Figure 2.1: Time series main tasks. Examples for a given univariate time series.

ACF(k) is close to 1 or -1, this indicates a strong positive or negative autocorrelation respectively, meaning that values at time t and $t-k$ are highly correlated or follow an inverse relationship. If ACF(k) is close to 0 there is no or little relationship between values at different lags. Autocorrelation is a valuable tool in time series analysis for identifying patterns within the data and understanding how past observations can influence future values in the series. Similarly, partial autocorrelation (PACF(k)) measures the correlation between two observations at different lags while removing the influence of all the intermediate observations between them. In other words, it measures the direct relationship between the time series and its lagged observations.

2.1.2 *Time Series Tasks*

Common time series tasks include forecasting, classification, imputation and anomaly detection. These tasks are fundamental in various domains, including finance, healthcare, environmental monitoring, and more. We provide some relevant definitions below:

Forecasting. Time series forecasting involves predicting future values of a time-ordered sequence of data points based on historical observations. For some given time series data, which can be univariate or multivariate, the goal is to predict the value of one or multiple time series respectively for a specified horizon h (i. e., the number of future time steps to predict) in the future.

Classification. Time series classification aims to assign a class label or category to each time series data sample. For some univariate or multivariate time series data and a set of possible class labels, the goal is to map the one or multiple time series to a class label.

Imputation. Time series imputation involves filling in missing or incomplete values within a time-ordered sequence of data points. For one or multiple time series some values may be missing, denoted by a relevant indicator (e. g., zero value). The task is to estimate these

missing values to reconstruct the complete time series data.

Anomaly Detection. Time series anomaly detection identifies data points or sequences that deviate significantly from normal patterns in a time series dataset. Anomalies represent time points or sub-sequences where the data behaves unusually compared to the expected behavior.

Additional tasks that are typical in classic time series data mining, outside the scope of this study, include clustering, motif discovery and others [83]. A visualization of the main time series tasks described above is provided in Figure 2.1.

2.2 MODELING TIME SERIES DATA

Different mathematical models can be employed to analyze and model time series data for various applications and relevant tasks. We next provide an overview of the most prominent statistical, machine and deep learning methods for tackling time series data that will be often referred to in the next chapters. In terms of the applications of this study and the proposed contributions, we mainly focus on time series prediction, classification and imputation methods. More details on time series prediction with modern deep learning approaches are provided in Chapter 3 and Chapter 5, whereas further approaches for imputation and classification of irregularly sampled data are provided in Chapter 4.

2.2.1 *Statistical and Machine Learning Methods*

METHODS FOR FORECASTING TIME SERIES. A traditional statistical technique, the Autoregressive Integrated Moving Average (ARIMA) model combines autoregressive (AR), differencing (I), and moving average (MA) parts to represent a time series. Its application assumes stationarity of the time series data [18, 20]. It is often represented using the mathematical notation ARIMA(p, d, q). The order of the autoregressive (AR) component, denoted by the p variable, determines the number of lagged observations incorporated into the model. The d variable represents the degree of differencing, which denotes the number of times the time series data needs to be differenced to achieve stationarity. Finally, variable q denotes the rank of the moving average (MA) component in the prediction equation, which specifies the number of lagged forecast errors.

Definition 2 (ARIMA). *We first denote the backshift operator $B(\cdot)$ such that $B(x_t) = x_{t-1}, B^2(x_t) = x_{t-2}, \ldots, B^d(x_t) = x_{t-d}$. Let $y_t = \nabla^d x_t = (1-B)^d x_t$, based on the degree of differencing d. Let also e_t represent the white noise error term in a distribution with zero average and contact variance σ_e^2. Then for p,q parameters, the value of the time series at t, x_t, is derived from:*

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + \theta_q e_{t-q},$$
$$\Rightarrow \phi_p(B)(1-B)^d x_t = \theta_q(B) e_t,$$

where $\phi_p(B) = 1 - \phi_1 B - \phi_2 B^2 - ... - \phi_p B^p$ *and* $\theta_q(B) = 1 + \theta_1 B + \theta_2 B^2 + ... + \theta_q B^q$. *We denote as* $\phi_1, \phi_2, \ldots, \phi_p$ *the autoregressive coefficients and as* $\theta_1, \theta_2, \ldots, \theta_q$ *the moving average coefficients.*

The values of p, d, q can be defined using the autocorrelation plots (ACF and PACF) or information criteria, e. g., the Akaike Information Criterion (AIC) for estimating the goodness of fit based on the number of parameters and the log-likelihood of the model [180]. The next step involves estimating the coefficients outlined in the above equation, by Maximum Likelihood Estimation (MLE), which is similar to Least Square Estimation used in regression equations [106].

The seasonal ARIMA (SARIMA) method extends the ARIMA model to handle seasonality in time series by combining a non-seasonal and a seasonal ARIMA component at multiples of the specified season. This model is appropriate in cases where seasonal variations are not successfully handled by differencing I(d). For multivariate inputs, the autoregressive (AR) part can be extended to form the Vector Autoregressive (VAR) statistical models [309], which formulate each variable as a linear combination of past values of itself and past values of all other variables in the system. Additionally, classic statistical methods for forecasting are based on exponential smoothing [90], such as Single Exponential Smoothing (SES), Double Exponential Smoothing (Holt's), and Triple Exponential Smoothing (Holt-Winters) methods. Such methods focus on capturing patterns in data by assigning exponentially decreasing weights to past observations. The forecasting process in exponential smoothing models involves updating the level of the series and, if applicable, trend and seasonality components at each time step based on historical data, while progressively reducing the influence of previous observations.

State-space models (SSM) [81] constitute a category of statistical models applied in the analysis and modeling of time series data. They prove especially beneficial when handling intricate systems and data with substantial noise. SSMs use state variables to describe the underlying hidden states and observations of a system as well as the probabilistic relationships between them in a system (e. g., by a set of differential equations). Additionally, gradient-boosting models, e. g., XGBoost and LightGBM [35, 173], are often employed for time series forecasting by treating time lags as features. Finally, the more recent Prophet model [246] for univariate time series is an additive regression method that decomposes the input into trend, seasonality, and holiday components and is suitable for data including missing values and multiple seasonal patterns.

METHODS FOR CLASSIFYING TIME SERIES. Time series classification employs supervised machine learning techniques to extract patterns from various labeled instances of time series data, and then predict or assign class labels based on the samples' similar features. We can identify four categories of different machine learning approaches for the classification task [7], applying to both univariate and multivariate data with some modifications:

Distance-based methods measure the similarity or dissimilarity of two time series, typically by converting the data into vectors and employing a distance measure [260], e. g., L_p

norms (Manhattan, Euclidean distance) between those points in the vector space. Dynamic time warping (DTW) offers an alternative solution for comparing two series, that do not perfectly align by considering slight shifts or distortions in the series to be similar, showing improved performance [74]. DTW is calculated using dynamic programming techniques. Distance measurements can be effectively combined with distance-based algorithms, such as the k-nearest neighbors (k-NN) method.

Shapelet-based methods identify distinctive patterns within time series, called shapelets, that are subsequences of the time series that are representative of a specific class [278]. The most dominant method among shapelet-based is the shapelet transform (ST) [109], which ranks potential shapelets based on their information gain. The resulting selection of k number of indicative shapelets is employed to transform the time series into k respective features based on their distance measurements from the initial time series. Consequently, this transformation constitutes datasets suitable for standard vector-based classification algorithms, such as the tree-based Rotation Forest [210].

Dictionary-based methods follow the concept of dictionaries used in natural language processing. Instead of words, these dictionaries capture patterns or shapes within time series data. These patterns, often referred to as shapelets or kernels, are essential elements that help in understanding and classifying time series. The idea is to create a dictionary of such patterns that can effectively represent the underlying characteristics of different classes or categories within time series datasets. Algorithms like Symbolic Fourier Approximation (SFA) [222], BOSS [221] and Bag-of-Patterns (BoP) [161] leverage these dictionaries to transform raw time series data into a structured format suitable for machine learning models. Similarly, the ROCKET method [67] employs convolutional kernels to extract feature maps that express if a pattern captured by the kernel is present in the series.

Interval-based methods, such as the Time Series Forest (TSF) [69], segment time series data into intervals, similar to the Bag-of-Patterns approach. Each interval is used to train an individual machine learning classifier based on summary (e. g., mean, standard deviation, slope measurements) or spectral features (e. g., Fourier, autocorrelation features) [163]. An ensemble of classifiers collectively assigns the final class label to each sample, based on the most frequent class predicted by the individual classifiers.

2.2.2 *Sequential Deep Learning Models*

Temporal learning or sequential models are a class of machine learning models specifically designed to handle sequential data, where the order and respective values of data points are crucial for analyzing and modeling for subsequent prediction tasks. These models are widely used in various fields, including natural language processing, time series forecasting, speech recognition, and others [61, 186]. We next present the most widely used sequential neural network components that are often incorporated into time series model architectures.

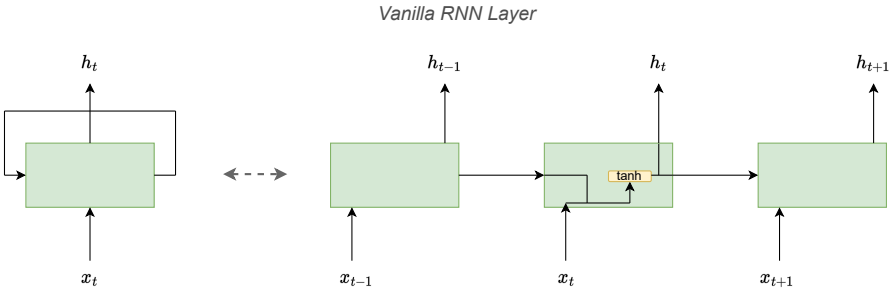


Figure 2.2: RNN visualization. (Left) Rolled and (Right) Unrolled.

RECURRENT ARCHITECTURES. Recurrent Neural Networks (RNNs) were introduced to handle and process data incorporating sequential relations. Unlike traditional feedforward neural networks [176], RNNs have recurrent connections (i. e., self-loops), allowing them to maintain a hidden state representing information from previous time steps in the sequence. This recurrent architecture enables RNNs to capture temporal dependencies and relationships within sequential data.

Definition 3 (Recurrent Neural Networks (RNNs))**.** *RNNs are a type of neural network designed to model sequences. They maintain a hidden state that allows them to capture dependencies between previous and current inputs in a sequence. Let* $\mathbf{x}_{1:T} = (x_1, \ldots, x_T)$ *represent the input sequence of length* T*, where* x_t *is the input value at time step* t*. Then,* $\mathbf{h}_t \in \mathbb{R}^d$ *forms the hidden state of the RNN at time step* t *and captures information about the sequence up to that point. The update of the hidden state in standard RNNs is calculated as follows:*

$$\mathbf{h}_t = f(\mathbf{W}_x x_t + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{2.1}$$

where $f(\cdot)$ *is usually a non-linear activation function, such as the hyperbolic tangent (tanh) or rectified linear unit (ReLU),* $\mathbf{W}_x$ *and* $\mathbf{W}_h$ *are weight matrices and* $\mathbf{b}_h$ *a bias vector that the model learns during training.*

A visualization of a vanilla RNN layer is provided in Figure 2.2, including both a more abstract and an unrolled version of the general architecture. However, traditional RNNs can struggle with capturing long-range dependencies due to the vanishing gradient problem [170]. To overcome this limitation two prominent RNN variants, i. e., Long Short-Term Memory Networks (LSTMs) [110] and Gated Recurrent Units Networks (GRUs) [72] networks have been proposed [289].

Definition 4 (Long Short-Term Memory Networks (LSTMs))**.** *LSTMs are a variant of RNNs that use gated cells to selectively update and remember information over long sequences, making them more suitable for tasks that require capturing long-term dependencies. The forget gate,* $\mathbf{f}_t$*, decides what information from the previous cell state* $\mathbf{C}_{t-1}$ *should be discarded or kept for the current time step. The input gate,* $\mathbf{i}_t$*, determines the new information to be added*

to the cell state. The output gate, $\mathbf{o}_t$*, regulates what information from the current cell state should be forwarded as the output of the LSTM cell. An LSTM cell can be defined as:*

$$\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_f),\\
\mathbf{i}_t &= \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_i),\\
\mathbf{o}_t &= \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_o),\\
\tilde{\mathbf{C}}_t &= tanh(\mathbf{W}_C[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_C),\\
\mathbf{C}_t &= \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t,\\
\mathbf{h}_t &= \mathbf{o}_t \odot tanh(\mathbf{C}_t),
\end{aligned} \tag{2.2}$$

where $\sigma(\cdot)$ *is the sigmoid activation function that squashes the gate output values between* 0 *and* 1*,* $tanh(\cdot)$ *the hyperbolic tangent function and* $\odot$ *denotes the element-wise (Hadamard) product.* $\tilde{\mathbf{C}}_t$ *represents the candidate for the cell state of the LSTM unit and* $\mathbf{C}_t$ *the final cell state value. The hidden state for each time step is computed based on the output state* $\mathbf{o}_t$ *and the cell state* $\mathbf{C}_t$*.*

Additionally, GRUs were introduced as an alternative to LSTMs to simplify architecture, reduce parameters, speed up training, and potentially improve performance in certain tasks. Their two-gate design, as opposed to LSTM's three gates, makes them computationally efficient and easier to train, particularly in scenarios with limited data or computational resources.

Definition 5 (Gated Recurrent Units Networks (GRUs))**.** *GRUs are a variant of RNNs, that similarly to LSTMs, use an update gate,* $\mathbf{u}_t$*, and a reset gate,* $\mathbf{r}_t$*. The update gate calculates the proportion of the previous memory cell content that should be transferred to the current memory cell, while the reset gate determines the amount of information from the previous state that should be disregarded. A GRU cell is defined as follows:*

$$\begin{aligned}
\mathbf{u}_t &= \sigma(\mathbf{W}_u x_t + \mathbf{U}_u \mathbf{C}_{t-1} + \mathbf{b}_u),\\
\mathbf{r}_t &= \sigma(\mathbf{W}_r x_t + \mathbf{U}_r \mathbf{C}_{t-1} + \mathbf{b}_r),\\
\tilde{\mathbf{C}}_t &= tanh(\mathbf{W}_C x_t + \mathbf{U}_C(\mathbf{r}_t \odot \mathbf{C}_{t-1}) + \mathbf{b}_C),\\
\mathbf{C}_t &= \mathbf{u}_t \odot \mathbf{C}_{t-1} + (1 - \mathbf{u}_t) \odot \tilde{\mathbf{C}}_t
\end{aligned} \tag{2.3}$$

where $\tilde{\mathbf{C}}_t$ *represents the candidate for the cell state of the current GRU unit and* $\mathbf{C}_t$ *the final cell state value.*

TEMPORAL CONVOLUTIONAL NEURAL NETWORKS (CNNS). A 1-D Convolutional Neural Network (CNN) is a neural network architecture specifically designed for analyzing one-dimensional data. While traditional CNNs are designed for two-dimensional data such as images, 1-D CNNs are tailored to handle sequential data, such as time series, audio signals, text, and other one-dimensional sequences [9, 96]. 1-D CNNs may offer advantages over traditional RNNs when handling sequential data. They excel in parallelization, are efficient

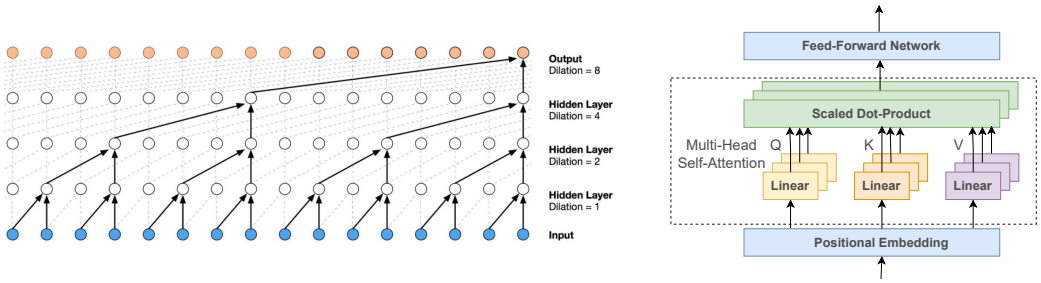


Figure 2.3: Sequential Models. (Left) Causal TCN derived from [186] and (Right) Transformer [248].

at capturing local patterns, exhibit translation invariance, require fewer parameters, and mitigate the vanishing gradient problem that can affect RNNs.

A Causal Temporal Convolutional Network (TCN) [186] is a convolutional network variant designed for sequential tasks involving very long input sequences. It preserves causality, by making predictions based only on only past and present data. This makes TCN ideal for tasks such as time series forecasting where respecting the temporal order is crucial. Additionally, causal TCNs can be employed to efficiently capture long-range dependencies in sequences.

Definition 6 (Causal Temporal Convolutional Neural Networks (Causal TCN))**.** *Causal TCN employs a causal 1-D convolution, where each output element depends only on past and current inputs within its receptive field. The convolution operation in a causal TCN can be expressed as:*

$$\mathbf{y}_t = (\mathbf{x}_{1:T} *_d f)(t) = \sum_{i=0}^{k-1} f(i) \cdot x_{t-d \cdot i} \tag{2.4}$$

where $f : \{0, \ldots, k-1\} \to \mathbb{R}$ is the filter (also called the kernel) of size k applied to the input and d is the dilation rate that defines the range of dependencies to capture. Also, $\mathbf{y}_t$ represents the value of the output sequence at time t, $f(i)$ is the i-th element of the filter and $\mathbf{x}_{t-d \cdot i}$ is the i-th past value of the input sequence adjusted by the dilation rate d.

Following the idea of Wavenet [186], one can incorporate hierarchical structures into the network architecture by stacking multiple layers of causal 1-D convolutions with increasing dilation factors at each level [286]. This approach allows for the extraction of increasingly abstract features from the input data, which can be useful for more complex tasks and remains computationally efficient despite the increased depth of the network. A visualization of stacked Causal TCN layers following Wavenet architecture is presented in Figure 2.3 (Left).

TEMPORAL SELF-ATTENTION NETWORKS. Temporal self-attention networks are neural architectures designed for processing sequences, emphasizing the order of elements, such as text and audio data. They employ self-attention mechanisms [248] to weigh the significance of elements within a sequence based on their temporal relationships. This

sequential processing captures long-range dependencies, which enables temporal self-attention networks to excel in applications where understanding the temporal context of data is crucial for accurate modeling and predictions.

Definition 7 (Self-Attention or Transformer-basedNetworks (AttN))**.** *Self-attention is a mechanism used in neural networks, particularly in Transformer-based models [248], to weigh and combine different parts of the input sequence, giving more attention to relevant elements. The mathematical notation for self-attention in a Transformer-like model can be represented as follows, employing a scaled-dot product mechanism:*

$$\begin{aligned} &\mathbf{Q} = \mathbf{x}_{1:T}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{x}_{1:T}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{x}_{1:T}\mathbf{W}_V, \\ &\mathit{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathit{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{\mathbf{d}_k}}\right)\mathbf{V} \end{aligned} \tag{2.5}$$

where $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$ *are weight matrices for learnable linear transformations on the input* $\mathbf{x}_{1:T}$, $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ *are the obtained representations that correspond to the queries, keys and values of self-attention respectively and* $\mathbf{d}_k$ *is the dimension of the key vectors which determines the scaling factor. Operation* $\mathbf{Q}\mathbf{K}^T$ *represents the dot product between queries and keys and the* $\mathit{Softmax}(\cdot)$ *function normalizes the dot product scores to produce attention scores that sum to* 1*. Dividing by* $\sqrt{\mathbf{d}_k}$ *prevents the dot products from becoming too large and stabilizes the training process.*

Since transformer-based architectures do not inherently consider the order of elements, similar to RNNs or CNNs, positional encoding is crucial to inject information about the positions or order of elements in a sequence into the model. The most common way to introduce positional encoding in the model is by adding sinusoidal functions of different frequencies to the input embeddings, as follows:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(\frac{pos}{1000^{2i/d_{model}}}\right), \\ PE_{(pos,2i+1)} &= \cos\left(\frac{pos}{1000^{2i/d_{model}}}\right) \end{aligned}$$

where $PE_{(pos,2i)}$, $PE_{(pos,2i+1)}$, *are the positional encoding values for position* pos *and dimension* $2i$, $2i+1$ *respectively,* pos *is the position of the element in the sequence,* i *the dimension of the positional encoding and* d_{model} *the dimension of the model's embedding. The use of sine and cosine functions with varying frequencies ensures that each dimension of the positional encoding captures different periodic patterns, allowing the model to differentiate between positions effectively. By adding this positional encoding to the input embeddings, the transformer model can differentiate between elements in different positions within the sequence, even if the elements have similar values.*

Multi-head self-attention is a crucial design choice of the transformer architecture, that leverages several self-attention mechanisms on the input data. This mechanism splits the

query, key, and value matrices into multiple heads, computes attention independently for each head, and then combines the results to create a rich representation of the input sequence. It allows the model to focus on different parts of the input simultaneously and capture various patterns and dependencies within the sequence. Transformer's main components, including positional embeddings, linear transformations, the scaled dot-product attention and a final feed-forward neural network are visualized in Figure 2.3 (Right).

HYBRID DEEP LEARNING MODELS. Hybrid models for time series tasks combine different modeling approaches or algorithms to improve the accuracy and robustness of predictions. Hybrid DL approaches mostly include variants based on RNNs, CNNs and Attention-based networks. Dominant combinations include convolutions combined with RNNs [147, 230], convolutional self-attention [254] and self-attention applied prior to RNNs [234].

TASK-SPECIFIC OVERVIEW. Based on the above definitions, different DL modules and their combinations show prominent results for task-specific applications.

In time series forecasting, DeepAR [216], a deep autoregressive model, is a neural network-based architecture that leverages the power of RNNs to capture complex temporal dependencies in time series data. DeepAR provides probabilistic forecasts instead of point predictions, by modeling the entire probability distribution of future values. This makes it well-suited for uncertainty quantification which is crucial in a wide range of applications, e. g., demand forecasting. More details about different loss functions for probabilistic and point estimate forecasting are provided in Section 3.1. Similarly, the multi-horizon quantile forecaster (MQRNN) [264] combines the flexibility of RNNs with quantile regression to model different quantiles simultaneously, extending the idea of the one-step ahead DeepAR model to direct multi-step ahead forecasting. Details about the difference between one- and multi-step ahead forecasting modules and the architectural design for the latter case of forecasting, are provided in Section 3.1. On the other hand, Deep State Space Models [206] combine the strengths of both recurrent neural networks and state space models (SSMs), specifically a linear Gaussian SSM. The strict structural assumption of observations following a Gaussian distribution has been tackled by extending SSMs with normalizing flows using Normalizing Kalman Filters [15]. More details on recent contributions in the state-of-the-art for time series forecasting with deep neural networks beyond RNNs (e. g., convolutional, attention modules and recently graph neural networks) are provided in Section 3.2 and Section 5.2.

In time series classification, CNNs have achieved significant performance and are often employed by stacking hierarchically several 1-D convolutional kernels [115]. Inspired by image recognition networks, Multi-scale CNNs (MCNN) [55] and Time LeNet (t-LeNet) [151] combine convolutional with pooling layers and employ data augmentation techniques, e. g., window slicing and window warping, to prevent overfitting. In the opposite direction, Fully Convolutional Networks (FCNs) [262], which are not based on pooling layers, replace the final prediction, i. e., fully-connected, layer with a global average pooling layer that summarizes the contribution of parts of the series to the output class [306]. Deep CNN architectures with residual connections, i. e., Residual Networks (ResNets), exploit skip or shortcut connections

to improve accuracy by avoiding unstable training in deep architectures [262]. Additionally, Multi-Channel Deep Convolutional Neural Networks (MCDCNN) [304, 305] offer an alternative for modeling multivariate time series, by applying convolutions independently on each channel of the multivariate input. On the other hand, instead of using a softmax output layer as the aforementioned approaches, the Time-CNN model is optimized using MSE loss in place of the standard cross-entropy for classification [299]. A single convolution is applied to all dimensions of the multivariate input and the final fully-connected layer is followed by a sigmoid activation function. Finally, inspired by AlexNet [143] for image classification, InceptionTime [116] extends the idea of ensembling multiple Inception modules with convolutional filters of various lengths for extracting information in different historical lengths. DL approaches for classifying time series which are sampled in non-fixed intervals and involve missing data, necessitate more sophisticated formalizations of standard modules. The most dominant methods for classification and imputation of irregular series are thoroughly described in Chapter 4.

2.3 TIME SERIES DISTORTIONS AND ROBUSTNESS

In time series analysis, modeling complex inherent patterns or extrinsic distortions of the data poses a significant challenge to modern machine and DL approaches. Natural irregularities or variations may be present in the data due to the underlying processes governing them. These are often considered intrinsic fluctuations in the time series and are essential to understanding its true behavior. On the other hand, distortions can also arise from errors or inaccuracies introduced during the data collection, processing, or modeling stages. These distortions are external to the inherent characteristics of the data and can lead to misinterpretations or deviations from the actual underlying patterns. Effectively capturing these challenging types of time series characteristics is critical for improving the model's performance.

2.3.1 *Definitions*

Time series data exhibit specific characteristics based on transformations or distortions applied to them. Following the definitions of existing literature [83] to refer to such transformations, some relevant notations are provided below.

Given a univariate time series $\mathbf{x}_{1:T} = (x_1, \ldots, x_T)$ of T data points, several transformations on the series can be applied so as to obtain a transformed series $\mathbf{x'}_{1:T} = (x'_1, \ldots, x'_T)$:

1. Amplitude Shifting. Time series $\mathbf{x'}_{1:T}$ can be extracted via a linear amplitude shift of the series, such that $x'_i = x_i + c$, where $c \in \mathbb{R}$ denotes an added constant value.

2. Uniform Amplification. Time series $\mathbf{x'}_{1:T}$ can be extracted by multiplying the amplitude of the series by a constant $c \in \mathbb{R}$, such that $x'_i = c \cdot x_i$.

3. Dynamic Amplification. Time series $\mathbf{x'}_{1:T}$ can be extracted by multiplying the amplitude of the series with a dynamic amplification non-zero function, such that $x'_i = h(i) \cdot x_i$.

4. Uniform Time Scaling. Time series $\mathbf{x'}_{1:T}$ can be obtained by applying a uniform change of the time scale of the series, such that $x'_i = x_{\lceil c \cdot i \rceil}$, where $c \in \mathbb{R}$.

5. Dynamic Time Scaling. Time series $\mathbf{x'}_{1:T}$ can be obtained by dynamically changing the time scale, $x'_i = x_{h(i)}$, where $h(i)$ a positive, strictly increasing function such that $h : \mathbb{N} \rightarrow [1, \ldots, n]$.

6. Noise and Outliers. Noise can be added to the original series such that $x'_i = x_i + \epsilon_i$, where ϵ_i is independent identically distributed white noise. Outliers can also be introduced at random positions, such that $x'_k = \epsilon_k$ for some points $\{k \mid k \in [1, \ldots, n]\}$.

Most time series learning tasks use similarity metrics between time series, e. g., distance metrics for time series classification and loss functions (e. g., mean squared error (MSE) and L_p norms) for optimizing neural networks in terms of forecasting and imputation. Based on the time series task, the employed model architecture or similarity measure must be robust to some or all combinations of the above transformations. This necessitates the definition of four general robustness properties in terms of time series data. Those properties can be defined as robustness with respect to scaling (i. e., modifications in amplitude), robustness with respect to warping (i. e., temporal distortions) as well as robustness in the presence of noise and outliers. Similarity measures need to respect several properties, in order to be scale robust, warp robust and noise and outlier robust [83].

2.3.2 *Prominent Challenges and Contributions*

However, except for the evaluation of similarity measures for time series with respect to the aforementioned distortions, there has been limited attention devoted to the formalization and empirical evaluation of the robustness of deep neural network architectures for different time series tasks.

In this study, we systematically assess the robustness of deep learning methodologies when confronted with diverse time series data characteristics or distortions, encompassing:

- The presence of *noisy observations* and training using conventional loss functions (e.g., L_p norms) and sequential deep learning architectures (e. g., recurrent, convolutional and attention-based). We propose a methodology to mitigate the effects of poor optimization by introducing a new loss variant for time series forecasting.

- *Irregularly sampled* and multivariate instances that contain *missing observations*. Such distortions pose a significant challenge to continuous modeling and to understand them we provide a thorough overview of continuous-time deep neural architectures as well as extend such efforts to basic modules not previously studied (i. e., convolutional).

- *Strong inter-variable and inter-temporal correlations.* We focus on formalizing algorithms for discretizing the time series in dependence networks, that capture such characteristics and extracting informative time series embeddings in the spectral domain.

- *Physical constraints or information* about the governing temporal dynamic process that produces the experimental data. We present the most prominent challenges while modeling spatio-temporal dynamical systems. We also introduce a physics-informed robust model and an experimental methodology to assess robustness while modeling spreading phenomena on networks.

The first two aforementioned challenges can be conceptualized as extrinsic distortions in the studied time series data and directly refer to the notations (i. e., 4-6, time scaling and noise) provided in Section 2.3.1. The last two challenges, while they do not directly refer to time series distortions, are inherent characteristics of the underlying data. Those inherent characteristics when are not properly accounted for in terms of modeling or analysis, can potentially lead to reduced performance introduced by the modeling process. Therefore, since strong correlations and physical knowledge of the data constitute common characteristics of different time series data, we include them in our theoretical and experimental analysis as part of the dominant challenges in robust DL modeling for time series.

We, therefore, analyze in Chapter 3, Chapter 4, Chapter 5 that follow, specific case studies on the robustness of standard deep neural network architectures applied to different tasks (e. g., forecasting, classification and imputation). We also propose and mathematically formalize some novel neural architectures that handle time series challenging properties and experimentally test their generalization performance in various real-world datasets. Finally, we present in chapter Chapter 6 some significant real-world applications incorporating time series data on networks, particularly focusing on predicting spreading processes using information for the underlying dynamical system's principles.

Part II

BREAKING GROUND: METHODOLOGIES, CONTRIBUTIONS AND FUTURE AVENUES

3 INCREASING ROBUSTNESS VIA LOSS FUNCTIONS

3.1 PRELIMINARIES: FORECASTING OUTPUTS AND LOSS FUNCTIONS

Time series forecasting constitutes the task of predicting the future values of single or multiple variables based on observed historical (lagged) values. In this section, we provide some key definitions and notations for different types of forecasting outputs, with respect to the employed loss function (Section 3.1.1) as well as the number of considered variables and the length of the forecasting horizon (Section 3.1.2). The design of the optimization steps for DL architectures, i. e., employed loss function and forecasting strategy (e. g., autoregressive or sequence-to-sequence), combined with the deep sequential neural architecture modules discussed in Chapter 2, enable the construction and evaluation of several forecasting architectures, that we present in Section 3.2 and Section 5.2.

3.1.1 *Types of Forecasting Outputs with Respect to the Loss Function*

Deep Neural Network architectures offer the versatility of adapting decoder blocks and output layers so as to approximate both discrete and continuous output targets. Predictions can be classified into two main categories, i. e., probabilistic outputs and point estimates [159].

PROBABILISTIC OUTPUTS. Except for the significance of estimating the expected value of a target variable in the future, obtaining the uncertainty of a model can be crucial for decision-making in several application domains, e. g., financial risk management. In order to model the uncertainty of predictions, deep neural networks can be parameterized to learn estimates of the probability distribution of the target values [92, 208], by employing a probability density function (PDF) or a quantile function. Gaussian distributions are widely used for forecasting with the networks producing parameter estimates (e. g., mean, variance) that can be used to model the target values [206, 216].

POINT ESTIMATES. However, there are situations in which it is beneficial to model particular values, i. e.,the expected value, middle value, or other specified quantiles, instead of learning the complete probability distribution. These models are commonly known as point forecast models. Different evaluation metrics may favor different summary statistics, and the choice should be made based on the specific requirements and goals of the forecasting task [138]. Determining the expected value of the target variable denotes a classification task for discrete targets and a regression task for continuous outputs. Mean squared error (MSE) and binary cross-entropy constitute the most common loss functions, for continuous and binary targets respectively. Variants of the above losses have also been introduced [114].

3.1.2 *Types of Forecasting with Respect to the Forecasting Horizon and Variables*

In the following sections, we provide some formal definitions of univariate and multivariate forecasting and multi-step ahead time series prediction, following the notations and categorization of techniques in relevant studies [13, 159].

3.1.2.1 *Univariate and Multivariate Forecasting*

Let $\mathcal{X} = \{\mathbf{x}_{i,1:T_i}\}_{i=1}^{N}$ a set of N univariate time series, where $\mathbf{x}_{i,1:T_i} = (x_{i,1}, \ldots, x_{i,T_i})$ the i-th time series and $x_{i,t}$ the value of the i-th time series at t. The input time series are correlated with a set of covariates, which may exhibit temporal variation or remain constant, denoted by $\mathcal{Z} = \{\mathbf{Z}_{i,1:T_i}\}_{i=1}^{N}$. We can denote by Φ the learnable parameters (e. g., weights and biases) of the employed neural network. In the general form, we aim to forecast the conditional distribution:

$$p(\mathbf{X}_{t+1:t+h}|\mathbf{X}_{1:t}, \mathbf{Z}_{1:t+h}; \theta) \tag{3.1}$$

where θ the parameters of the probabilistic model, for multivariate $\mathcal{X}$ (i. e., $N > 1$) and multi-step ahead forecasting (where h the horizon length) and $\mathbf{X}_{t+1:t+h}$ the sequence of values of all N time series in the time interval $[t+1, t+h]$.

UNIVARIATE MODELS. A separate model can be trained independently for N time series to model (i) the predictive distribution or (ii) the point estimate:

$$p(\mathbf{x}_{i,t+1:t+h}|\mathbf{x}_{i,1:t}, \mathbf{Z}_{i,1:t+h}; \theta_i), \quad \theta_i = \Psi(\mathbf{x}_{i,1:t}, \mathbf{Z}_{i,1:t+h}) \tag{3.2}$$

$$\hat{\mathbf{x}}_{i,t+1:t+h} = \Psi(\mathbf{x}_{i,1:t}, \mathbf{Z}_{i,1:t+h}) \tag{3.3}$$

for probabilistic (3.2) and point forecasts (3.3) respectively. In the above equations, Ψ is a generic function that maps input features to the parameters of the probabilistic model (i. e., case (i)) for the i-th time series.

A single, cross-learning model can also be trained using all data from the N time series and still predict a univariate output. This joint model provides (i) the predictive distribution or (ii) the point estimate:

$$p(\mathbf{x}_{i,t+1:t+h}|\mathbf{X}_{1:t}, \mathbf{Z}_{1:t+h}; \theta_i), \quad \theta_i = \Psi(\mathbf{x}_{i,1:t}, \mathbf{Z}_{1:t+h}, \Phi) \tag{3.4}$$

$$\hat{\mathbf{x}}_{i,t+1:t+h} = \Psi(\mathbf{x}_{i,1:t}, \mathbf{Z}_{i,1:t+h}, \Phi) \tag{3.5}$$

for probabilistic (3.4) and point forecasts (3.5) respectively. In the above equations, Ψ uses shared parameters Φ, although the parameters of the probabilistic model θ_i for each time series are different (i. e., in case(i)). Such a model exploits information while learning across the different time series, resulting in improved extracted features. After optimizing Ψ, the model forecasts each time series independently.

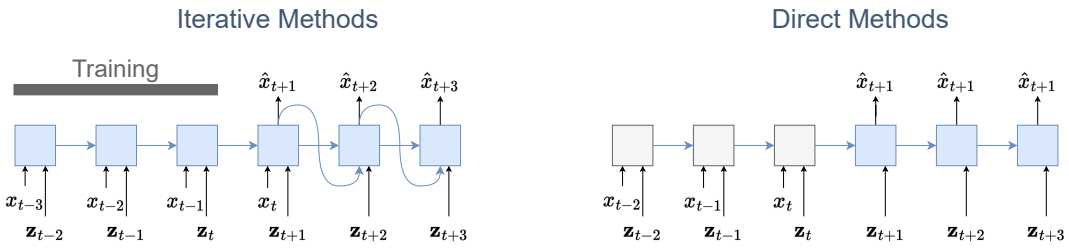


Figure 3.1: (Left) Iterative methods for multi-step forecasting. (Right) Direct methods for multi-step forecasting.

MULTIVARIATE MODELS. A single model is trained for N time series using all data to predict the multivariate target:

$$p(\mathbf{X}_{t+1:t+h}|\mathbf{X}_{1:t}, \mathbf{Z}_{1:t+h}; \theta), \quad \theta = \Psi(\mathbf{X}_{1:t}, \mathbf{Z}_{1:t+h}, \Phi) \tag{3.6}$$

$$\hat{\mathbf{X}}_{t+1:t+h} = \Psi(\mathbf{X}_{1:t}, \mathbf{Z}_{1:t+h}, \Phi) \tag{3.7}$$

for probabilistic (3.6) and point forecasts (3.7) respectively. This model captures the dependency among time series while forecasting.

3.1.2.2 *One-step and Multi-step Ahead Forecasting*

Practitioners are often interested in predicting variables at multiple future time steps. The provided notation in the previous equations covers the one-step ahead forecasting case, i. e., for $h = 1$, and for greater horizon values, i. e., $h > 1$, the multi-step ahead forecasting case. Methods for multi-step ahead forecasting using deep neural networks can be classified into iterative and direct ones [159], or autoregressive and Seq2Seq respectively [13].

ITERATIVE METHODS. Iterative techniques for forecasting multiple time steps ahead are implemented using autoregressive deep learning modules [160, 216, 261]. For the generation of multi-horizon forecasts, the iterative strategy extends conventional one-step ahead forecasting models, by progressively feeding target estimates to the subsequent time step. However, it is essential to note that the errors introduced recursively, can accumulate significantly, especially for large forecasting horizons h [159, 243].

DIRECT METHODS. The key idea behind direct forecasting techniques incorporates the employment of only the accessible lagged information for generating forecasts, which is often implemented by the popular sequence-to-sequence architectures. In this case, an encoder module is used to condense historical information and a decoder that combines them with known future inputs to generate forecasts. Alternatively, simpler models can be employed to produce a fixed-length vector output representing the multi-step forecast, which necessitates specifying a predefined maximum horizon length h [159].

Figure 3.1 provides a visualization of the two main strategies, i. e., iterative methods (Left) and direct methods (Right), for multi-step ahead prediction using neural network architectures, adapted from the relevant study [159].

While several sophisticated loss variants for forecasting have been proposed, MSE and L_p norms, in general, remain the principally employed loss functions and most evaluated point forecast metrics across different research works. Their computational efficiency, simplicity and the fact that they provide a direct measure of the real deviation from target values constitute them a straightforward choice for many practitioners, that are not necessarily experts in time series analysis. For this reason, in the study we present in the next section (Section 3.2), we aim to highlight the main challenges encountered when employing MSE for training and a methodology to tackle its poor performance.

3.2 TIME SERIES FORECASTING MODELS COPY THE PAST: HOW TO MITIGATE

Time series forecasting is at the core of important application domains posing significant challenges to machine learning algorithms. Recently neural network architectures have been widely applied to the problem of time series forecasting. Most of these models are trained by minimizing a loss function that measures predictions' deviation from the real values. Typical loss functions include mean squared error (MSE) and mean absolute error (MAE). In the presence of noise and uncertainty, neural network models tend to replicate the last observed value of the time series, thus limiting their applicability to real-world data. In this work, our goal is to formally define the aforementioned problem such that practitioners can identify whether their models replicate previous values instead of making predictions. Thus, we provide a formal definition of the above problem and we also give some examples of forecasts where the problem is observed [142]. We also propose a regularization term penalizing the replication of previously seen values. We evaluate the proposed regularization term both on synthetic and real-world datasets. Our results indicate that the regularization term mitigates to some extent the aforementioned problem and gives rise to more robust models.

3.2.1 *Introduction*

Time series are ubiquitous in several application domains including quantitative finance, seismology and meteorology, just to name a few. Due to this abundance of time series data, the problem of time series forecasting has recently emerged as a very important task with applications ranging from traffic forecasting to financial investment. Indeed, accurate forecasting is of great importance since it can improve future decisions which is the main objective in a number of scenarios. For example, traffic forecasting seeks to predict future web traffic to make decisions for better congestion control [101]. Moreover, forecasting the spread of COVID-19 is of paramount importance to governments and policymakers in order to impose measures to combat the spread of the disease [44].

With the advent of deep learning, deep neural networks have become the dominant approach to the problem of time series forecasting. For instance, models and layers such as Long Short-Term Memory [110], Gated Recurrent Units [45] and Temporal Convolution Networks [9] have proven to be very successful in temporal modeling. Specifically, these models have demonstrated great success in capturing complex nonlinear dependencies between variables and time, while they usually operate on raw time series data, thus requiring considerably less human effort than traditional approaches. However, as these architectures make fewer structural assumptions, they typically require larger training datasets to learn accurate models, while they also lack robustness and are very sensitive to noise and perturbations. A common problem in time series forecasting with deep neural networks is the one where the model just replicates the last observed value of the time series. This is quite common in the case of noisy datasets, and is a problem of paramount importance since most real-world datasets contain noise. In fact, this problem which we refer to as "mimicking" also depends on the nature of the employed loss function (e. g., MSE).

In this contribution, our goal is to formally define the aforementioned problem such that practitioners can identify whether their models replicate previous values instead of making predictions. Therefore, we provide a definition of "mimicking" in time series forecasting and a methodology to quantify the extent to which a model suffers from it. Furthermore, we present examples of forecasts where this phenomenon is clearly observed. The key technical contribution of this work is a carefully designed regularization term which can be added to the loss function and mitigate the drawbacks of "mimicking" which might occur in models trained by minimizing common loss functions. The proposed regularization term is evaluated on a range of different datasets. Our results suggest that the proposed term mitigates "mimicking" and reduces its impact on the model's performance. The main contributions of this study are summarized as follows:

- To the best of our knowledge, we are the first to formally define the problem of "mimicking" in time series forecasting.
- We designed a regularization term that, when added to the loss function, mitigates to some degree the effect of "mimicking". This term is general for all neural network architectures and does not make any assumptions.
- We specifically investigate and deal with the phenomenon of "mimicking" on three standard deep neural networks (LSTM, TCN and Transformer) which are some of the most widely used and effective models in time series forecasting and sequence modeling.
- The proposed regularization term improves the movement predictive performance of the vanilla models on 6 public time series benchmark datasets and one stock dataset. On average, it leads to absolute improvements of 3.33% in accuracy (considered for the three models), while MSE increases slightly.

3.2.2 *Related Work*

TIME SERIES FORECASTING. Before the advent of deep learning, the Auto-Regressive Integrated Moving Average (ARIMA) model [18] and exponential smoothing [111] were among the most popular and widely used methods for time series forecasting. However, these approaches have some drawbacks (e. g., ARIMA assumes stationarity, while most real-world time series are not stationary), and thus have been replaced recently with neural network architectures [159]. Different instances of recurrent neural networks such as Long Short Term Memory Networks (LSTMs) [110] and Gated Recurrent Units (GRUs) [45] have become the dominant approaches for time series forecasting mainly due to their ability to model complex patterns and long term dependencies, and to extract useful features from raw data. Besides recurrent neural networks, convolutional neural networks have also been recently investigated in the task of time series forecasting. The Temporal Convolution Network (TCN) [9] is perhaps the most prominent example from this family of models. Attention mechanisms have proven very successful in many tasks and have also been applied to the problem of time series forecasting [154, 199]. Different neural network components such as recurrent, convolutional and attention layers have been combined with autoregressive components to make predictions [147]. The potential of residual connections along with a very deep stack of fully-connected layers in the context of time series forecasting has also been explored recently [189]. Matrix factorization methods have achieved prominent results in the case of high-dimensional time series data [225, 288]. Some recent works have combined neural networks and state space models [206, 261]. Probabilistic forecasting, for predicting the distribution of possible future outcomes, has also recently started to receive increasing attention [38, 216].

LOSS FUNCTIONS. Besides the traditional functions (MSE, MAE, etc.), other measures that capture different time series properties have been proposed. However, in most cases, these evaluation metrics are not differentiable, thus they cannot be directly employed as loss functions. Examples of such measures include the dynamic time warping algorithm which captures the shape of the time series, and standard evaluation metrics of supervised learning algorithms (e. g., accuracy, f1-score) in the context of change point detection algorithms [4]. The need for measures alternatives to MSE has recently led to the development of new differentiable loss functions which capture different meaningful statistical properties of time series such as shape and time, including differentiable variants of Dynamic Time Warping (DTW) [17, 57]. These differentiable dynamic time warping terms can also be combined with terms that penalize temporal distortions (Temporal Distortion Index (TDI)) for more accurate temporal localization [97], while they have also been generalized to non-stationary time series [98] and binary series [209].

3.2.3 *The Phenomenon of "Mimicking" and How to Mitigate*

We first introduce some key notations for time series forecasting. Let $\mathbf{x}_{1:T} = (x_1, x_2, \ldots, x_T)$ be a univariate time series where $x_t \in \mathbb{R}$ denotes the value of the time series at time t. We denote sliding windows extracted from the whole series $\mathbf{x}_{1:T}$ as samples of length τ, such that $(x_t, x_{t+1}, \ldots, x_{t+\tau-1})$ for $t \in \{1, \ldots, T-\tau\}$. The goal of a forecasting model is to predict the future values $x_{t+\tau}$ for $t \in \{1, \ldots, T-\tau\}$. Let $\mathbf{y}_{1:n} = (y_1, y_2, \ldots, y_n) = (x_{\tau+1}, x_{\tau+2}, \ldots, x_{\tau+n})$ the target values of time series to predict (i. e., for $n = T - \tau$) and $\hat{\mathbf{y}}_{1:n} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n)$ denote the predictions of the forecasting model. Neural network models for time series forecasting are typically trained to minimize the MSE which is defined as the sum of squared distances between the target variables and predicted values, i. e., $\text{MSE} = {}^1\!/_n \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$. Similar metrics, that measure the difference between the forecast and the actual value per time step, such as MAE, are also employed in various applications.

3.2.3.1 *"Mimicking" in Time Series Forecasting*

Even though MSE and related functions enjoy some nice properties (e. g., MSE is convex on its input), when dealing with real-world data with multiple co-occurring patterns and noisy components, these functions might become sensitive to noise. This might result into the problem of predicting previously seen values (usually the last seen observation in the time series), rather than making predictions based on long-term extracted patterns. We next formalize the problem described above. The following analysis focuses on the MSE loss, but it also applies to other loss functions that are commonly employed in time series forecasting (e. g., MAE). To investigate whether the model just replicates the last observed value of the time series, we can examine if the MSE between the forecast in time step t and the real value in time step t is greater than the MSE between the forecast in time step t and the real value in time step $t-1$.

Definition 8 ("Mimicking" in Time Series)**.** *We say that the phenomenon of "mimicking" in time series forecasting occurs if the following inequality holds*

$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 > \sum_{i=1}^{n} (y_{i-1} - \hat{y}_i)^2 \tag{3.8}$$

We can quantify the amount of "mimicking" as follows (the larger the (positive) value of MIM, the larger its severity): MIM $= \sum_{i=1}^{n} \left[(y_i - \hat{y}_i)^2 - (y_{i-1} - \hat{y}_i)^2 \right]$.

To demonstrate that "mimicking" is related to the level of noise present in a dataset, we generated a synthetic dataset that corresponds to a sum of sinusoidal series with added random Gaussian noise (more details are given in Section 3.2.4). A linear term is also added to the above terms. Table 3.1 illustrates the MSE achieved by an LSTM and a TCN model along with the amount of "mimicking" as a function of the level of noise (i. e., increasing variance). We observe that the LSTM model is more prone to "mimicking" than the TCN model, while the greater the value of the variance of the Gaussian noise, the greater the

Table 3.1: MSE and "mimicking" as a function of the level of noise added to a synthetic dataset.

	σ	MSE ($\times 10^{-3}$)	MIM ($\times 10^{-3}$)
LSTM	0	6.191	0.048
	0.01	3.225	0.035
	0.1	9.765	0.052
	0.25	10.446	0.066
	0.5	13.357	0.069
TCN	0	0.024	-0.006
	0.01	0.007	-0.003
	0.1	0.046	0.002
	0.25	0.068	0.004
	0.5	0.182	0.008

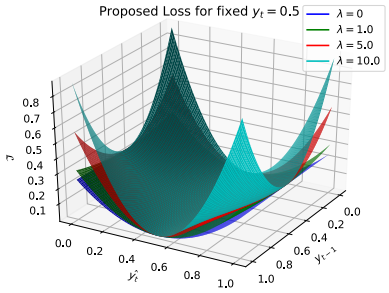


Figure 3.2: A visualization of the proposed loss $\mathcal{L}$ of Equation 3.14 for different values of λ, $\hat{y}_t$ and y_{t-1}.

impact of "mimicking" on the models' performance. We also need to mention that the TCN model does not suffer from "mimicking" for $\sigma = 0$ and $\sigma = 0.01$.

3.2.3.2 *Proposed Regularization Term*

To mitigate the effects of mimicking in time series forecasting, we begin our analysis from the definition provided above. Specifically, we would like the second term of inequality (3.8) to be greater or at least equal to the first term, i. e., we would like the following to hold

$$\sum_{i=1}^{n} \left[(y_i - \hat{y}_i)^2 - (y_{i-1} - \hat{y}_i)^2 \right] \leq 0 \tag{3.9}$$

By introducing the above term into the loss function, we directly punish "mimicking" to some extent. However, incorporating solely the above term into the loss function gives rise to an unbounded function. Indeed, in case $\sum_{i=1}^{n}(y_i - y_{i-1}) < 0$, setting $\hat{y}_i \to +\infty$ can drive the loss to negative infinity. Likewise, if $\sum_{i=1}^{n}(y_i - y_{i-1}) > 0$, setting $\hat{y}_i \to -\infty$ also leads to a loss function that is unbounded from below. Hence, since the loss is not bounded, there is no admissible estimator, and this will render the model to be of no practical use.

Note that a perfect model would achieve an MSE equal to 0, i. e., $\sum_{i=1}^{n}(y_i - \hat{y}_i) = 0$. In such a scenario, we would like the loss function to take its lowest value. If we replace the term that corresponds to the MSE in Equation 3.9 with 0, we obtain $0 - \sum_{i=1}^{n}(y_{i-1} - \hat{y}_i)^2 = -\sum_{i=1}^{n}(y_{i-1} - y_i)^2$ The equality is due to the fact that the model is perfect, i. e., $y_i = \hat{y}_i$ $\forall i \in 1, \ldots, n$ holds. We would like the above term to be the lower bound of the proposed loss function (since the model is perfect). Therefore, we have

$$-\sum_{i=1}^{n}(y_{i-1} - y_i)^2 \leq \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 - \sum_{i=1}^{n}(y_{i-1} - \hat{y}_i)^2 \tag{3.10}$$

By combining Equation 3.9 and Equation 3.10, we obtain the following inequality

$$
\begin{aligned}
&-\sum_{i=1}^{n}(y_{i-1}-y_i)^2 \leq \sum_{i=1}^{n}(y_i-\hat{y}_i)^2 - \sum_{i=1}^{n}(y_{i-1}-\hat{y}_i)^2 \leq 0 \\
\iff\ &0 \leq \sum_{i=1}^{n} 2(y_i-y_{i-1})(y_i-\hat{y}_i) \leq \sum_{i=1}^{n}(y_{i-1}-y_i)^2 \\
\iff\ &0 \leq \sum_{i=1}^{n}(y_i-y_{i-1})(y_i-\hat{y}_i) \leq \frac{1}{2}\sum_{i=1}^{n}(y_{i-1}-y_i)^2
\end{aligned}
\tag{3.11}
$$

Ideally, we would like the above inequality to hold. That would mean that the phenomenon of mimicking does not occur. However, the middle term is still not bounded, thus we cannot directly minimize that term. Note that all the terms are nonnegative. Hence, we can square all the sides of the inequality as follows

$$
0 \leq \sum_{i=1}^{n}\left[(y_i-y_{i-1})(y_i-\hat{y}_i)\right]^2 \leq \frac{1}{4}\sum_{i=1}^{n}(y_{i-1}-y_i)^4 \tag{3.12}
$$

Now, the middle term is nonnegative by construction, and we can thus safely minimize it. Interestingly, the above function is continuous and differentiable which are both desirable properties for loss functions. For instance, the first and second derivatives of the function are shown below

$$
\frac{d}{d\hat{y}_i} \sim \sum_{i=1}^{n} 2(y_{i-1}-y_i)^2(\hat{y}_i-y_i), \qquad \frac{d^2}{d\hat{y}_i^2} \sim \sum_{i=1}^{n} 2(y_{i-1}-y_i)^2 \tag{3.13}
$$

From the above, it is also clear that the second derivative of the function is nonnegative on its entire domain, thus the function is convex. However, we need to mention that even though the function is convex in $\hat{y}_i$, in case neural networks are employed (or other non-linear models), we have $\hat{y}_i = f(y_{i-1}, \ldots, y_{i-k}; \theta)$ and the function is not convex in θ.

Our proposed loss function for a sequence of n time steps is defined as

$$
\mathcal{L} = \sum_{i=1}^{n}(y_i-\hat{y}_i)^2 + \lambda \sum_{i=1}^{n}\left[(y_i-y_{i-1})(y_i-\hat{y}_i)\right]^2 \tag{3.14}
$$

where λ is a parameter which controls the importance of the regularization term, i. e., how much penalty needs to be imposed to alleviate "mimicking". The two factors $(y_i - y_{i-1})$, $(y_i - \hat{y}_i)$ that constitute the penalty term above can be interpreted as a discrete-time cross-correlation measure function between the difference of the series at i and $i-1$ and the predicted error at i. If we expand the term for a specific i, we derive $[(y_i - y_{i-1})y_i - (y_i - y_{i-1})\hat{y}_i)]^2$. The closer the prediction $\hat{y}_i$ is to y_{i-1} and the farther y_i is from y_{i-1}, the larger the imposed penalty term will be. Figure 3.2 illustrates how the proposed loss function varies as a function of y_{t-1} and $\hat{y}_t$ for different values of λ (for y_t fixed to 0.5).

In some cases, the model might not replicate solely the last observed value of the time series x_τ, but also observations that occurred farther in the past, e. g., $x_{\tau-1}, x_{\tau-2}$, etc. We next generalize the proposed penalty term to account for such kind of scenarios. To prevent a neural network model from replicating the last K observations, we can use the following loss function

$$\mathcal{L} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{n}\sum_{k=1}^{K}\left[(y_i - y_{i-k})(y_i - \hat{y}_i)\right]^2 \tag{3.15}$$

The proposed loss of Equation 3.14 can also be generalized to the case of multi-step ahead forecasting. Specifically, it can be directly applied to iterative 1-step methods [159], while in the case of direct multi-horizon forecasting (vector output/Seq2Seq architectures), we need to consider vectors $\hat{\mathbf{y}}_i \in \mathbb{R}^h$ of length h which refer to the desired horizon.

3.2.4 *Experimental Evaluation*

3.2.4.1 *Datasets*

SYNTHETIC. This synthetic dataset corresponds to a sum of sinusoidal series with added random noise: $y(t) = \sin(t) + \sin\left(\frac{\pi}{2}t\right) + \sin\left(\frac{-3\pi}{2}t\right) + \epsilon(t)$, where $\epsilon(t)$ is a Gaussian distribution with mean μ and variance σ^2 ($\mu = 0$, $\sigma = 0.5$).

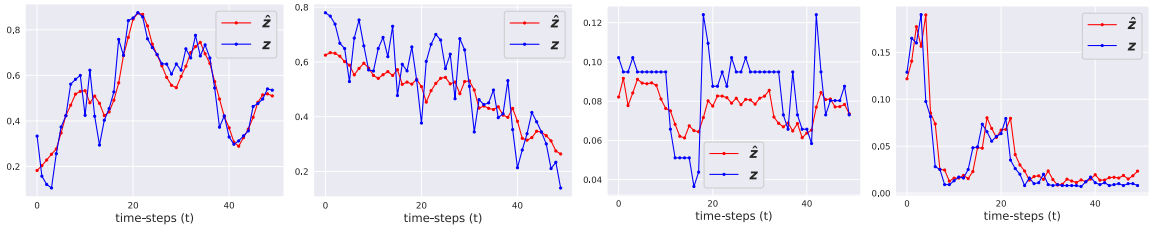
MONTHLY SUNSPOTS. This dataset describes a monthly count of the number of observed sunspots from 1749 to 1983, a total of $2,820$ observations.

ELECTRICITY. It contains electricity consumption measurements (kWh) from 321 clients, recorded every 15 minutes from 2012 to 2014. We utilize the first univariate series of length $26,304$.

BEIJING PM2.5. This hourly dataset contains the PM2.5 data of the US Embassy in Beijing. It is a multivariate dataset that consists of eight variables, including the PM2.5 concentration and a total number of $43,824$ observations. The task is to predict the future hourly concentration given the other variables.

SOLAR ENERGY. It contains the solar power production data from photovoltaic plants in Alabama in 2006. We utilize the first univariate series of length $52,560$.

EXCHANGE RATE. It includes the exchange rates of eight foreign countries (Australia, Britain, Canada, China, Japan, New Zealand, Singapore, and Switzerland) from 1990 to 2016. We utilize the first univariate series of length $7,588$.



(a) Synthetic-LSTM (b) Sunspots-LSTM (c) Electricity-TCN (d) PM2.5-TCN

Figure 3.3: 1-step predictions of models trained with MSE on part of the test set.

3.2.4.2 *Evaluation Metrics*

In order to evaluate the performance of our proposed loss function in the experiments that follow, we employ the following metrics:

MEAN SQUARED ERROR (MSE) AND SHIFTED MEAN SQUARED ERROR (S-MSE). MSE compares the predictions $\hat{y}_t$ against the targets y_t. Shifted MSE compares $\hat{y}_t$ against the last values y_{t-1}, i. e., s-MSE $= 1/n \sum_{i=1}^{n} (\hat{y}_i - y_{i-1})^2$.

ACCURACY (ACC) AND SHIFTED ACCURACY (S-ACC). To compute these two metrics, we turn the forecasting problem into a classification one. Let v be a n-dimensional vector such that its i-th element is defined as $v_i = \text{change}(y_i, y_{i-1})$ where $\text{change}(a, b) = \text{sign}(a - b)$. Let also $\hat{v}$ be a n-dimensional vector such that $\hat{v}_i = \text{change}(\hat{y}_i, \hat{y}_{i-1})$. Then, Acc is defined as the accuracy between the above two vectors. s-Acc is defined as the accuracy between the vector $\hat{v}$ and the vector v shifted by 1 step to the left. Predicting whether the value of a time series will increase or decrease is a task of high importance for many applications such as stock price prediction. Indeed, successful predictions would enable hedge funds or investors to lay a successful strategy for buying and selling stocks.

We should note here that MSE and accuracy are two metrics orthogonal to each other. A time series forecasting model ideally would achieve a low value of MSE and a high accuracy. Models that suffer from "mimicking" can yield low values of MSE, thus achieving solely a low MSE might not be a clear indicator of the model's predictive power. On the other hand, a model that yields solely high accuracy captures the shape and the change points of the time series, but the predicted values might significantly deviate from the actual values of the series.

3.2.4.3 *Experimental Setup*

We divide the sequence into multiple samples, where τ observations are given as input and the expected output is the actual value of the observation that follows each sample. We choose τ from $\{32, 64, 128, 256, 512\}$.

Table 3.2: 1-step prediction performance of the different models, and the baseline on the 6 considered datasets. All MSE/s-MSE results are in scale ($\times 10^{-3}$).

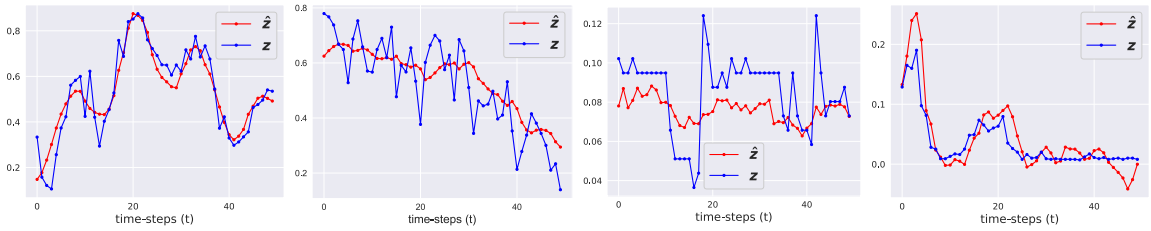
Methods		**Synthetic**		**Sunspots**		**Electricity**	
		MSE	**s-MSE**	**MSE**	**s-MSE**	**MSE**	**s-MSE**
Avg. Window	1	9.759	0.0	6.825	0.0	5.063	0.0
	3	12.398	4.524	6.881	3.03	4.766	2.15
	5	18.378	9.791	7.541	4.54	4.851	3.05
	7	22.896	15.382	8.642	5.91	5.066	3.62
	9	24.883	19.127	9.737	7.22	5.363	4.11
LSTM		4.742	4.778	5.739	3.572	3.594	1.49
TCN		3.784	5.650	6.043	3.485	3.582	2.22
Transf.		5.422	5.128	12.21	11.21	4.299	2.659
Methods		**Beijing PM2.5**		**Solar**		**Exchange Rate**	
		MSE	**s-MSE**	**MSE**	**s-MSE**	**MSE**	**s-MSE**
Avg. Window	1	0.489	0.0	1.803	0.0	0.248	0.0
	3	0.905	0.30	3.531	1.150	0.244	0.103
	5	1.358	0.72	5.654	2.918	0.292	0.171
	7	1.793	1.16	7.998	4.998	0.348	0.235
	9	2.192	1.57	10.485	7.291	0.407	0.298
LSTM		0.421	0.08	1.358	0.463	0.388	0.236
TCN		0.506	0.12	1.489	0.397	0.270	0.122
Transf.		0.560	0.18	1.802	0.391	3.811	3.747

We choose parameters as follows. For the LSTM model, we use a single LSTM layer. We use the hidden state of the last time step of the LSTM layer as the vector representation of the time series. The generated vector representations are then fed into a two-layer MLP with a ReLU activation function. For the TCN model, we adjust the parameters to capture the different history lengths τ that we test, from the equation $R_{field} = 2^{D-1} \cdot K_{size}$, for $K_{size} = 2$ and D the number of dilation layers. Each layer has dilation rate of $2^{N_l - 1}$, where $N_l = \{1, 2, ..., D\}$. We also implement a model consisting of two stacked encoders of the Transformer architecture followed by a fully-connected layer for the final prediction. The hidden-dimension size of the LSTM, TCN and Transformer layers is chosen from $\{32, 64, 128, 256\}$. For all the three models, we use the Adam optimizer with an initial learning rate of 10^{-3} and decay the learning rate by 0.1 every 10 epochs. We choose the batch size from $\{32, 64, 128\}$. We set the number of epochs to 100, and we retrieve the model that achieves the lowest validation loss. The regularization parameter λ is chosen from $\{0.1, 0.5, 1, 5, 10, 20, 50, 100, 200, 500, 800, 1000\}$.

We also implement a simple baseline method (Avg. Window) which, given the past n values of the time series, predicts the average value: $\hat{y}_t = {}^1/_n \sum_{i=1}^{n} y_{t-i}$.

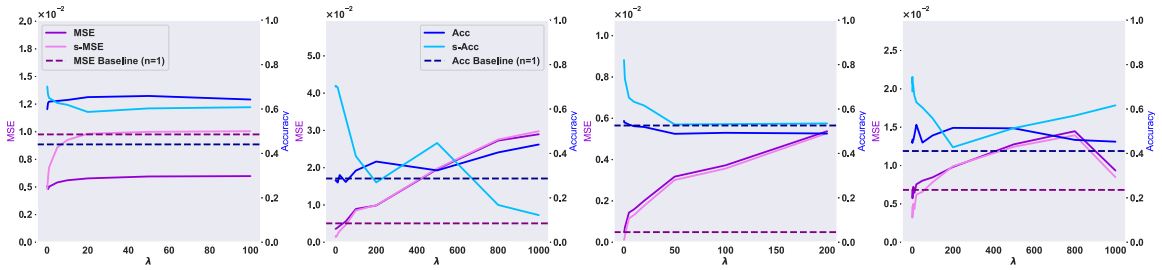
3.2.4.4 *Results*

EXAMPLES OF "MIMICKING". We next provide some examples of forecasts where the LSTM model and the TCN model just learn to replicate the last seen observations. Figure 3.3



(a) Synthetic-LSTM (b) Sunspots-LSTM (c) Electricity-TCN (d) PM2.5-TCN

Figure 3.4: Predictions as in Figure 3.3 but after training with the proposed loss.



(a) Synthetic-LSTM (b) Electricity-LSTM (c) PM2.5-LSTM (d) Sunspots-TCN

Figure 3.5: 1-step prediction performance of different models, trained to minimize the proposed loss, as a function of λ and the baseline (Avg. Window ($n = 1$)).

illustrates such examples for some of the considered datasets. The first 2 plots (i. e., (a) and (b)) correspond to predictions of the LSTM model, while the last 2 plots (i. e., (c) and (d)) to predictions of the TCN model. On the synthetic dataset, the LSTM model learns to infer quite accurately the future values of the time series. This is mainly due to the simplistic nature of that dataset. On the other hand, on the real-world datasets, the two models fail to generalize, replicating previously observed data. This is especially true for plots (b) and (d). Specifically, on the Beijing PM2.5 dataset, "mimicking" is observed to a very large extent, probably due to the complexity of the dataset.

Besides the above qualitative results, we also present some quantitative results in Table 3.2. We can see that on the 5 real-world datasets, the LSTM, TCN and Transformer models suffer from "mimicking" since s-MSE is smaller than MSE in all cases. Interestingly, s-MSE can even be an order of magnitude smaller than MSE (see LSTM on Beijing PM2.5 and all three models on Solar). On the other hand, on the synthetic dataset both the LSTM and the TCN model achieve a smaller MSE than s-MSE. Thus, on this dataset, the two models are more robust. Indeed, this dataset is less noisy, while its trend is more predictable than that of the real-world datasets. With regards to the baselines, in most cases, they also achieve low values of MSE (especially when $n = 1$). In fact, on the Beijing PM2.5 dataset, the Avg. Window ($n = 1$) outperforms the TCN model since it yields a smaller MSE than TCN. This

Table 3.3: 1-step prediction performance of the different models on the 6 considered datasets trained with MSE and with the proposed loss function. All MSE/s-MSE results are in scale ($\times 10^{-3}$). We mention in bold the maximum accuracy (Acc) and we underline the minimum shifted accuracy (s-Acc) achieved for each model.

Methods	**Synthetic**				**Sunspots**			
	MSE	**s-MSE**	**Acc**	**s-Acc**	**MSE**	**s-MSE**	**Acc**	**s-Acc**
LSTM	4.742	4.778	0.616	0.695	5.739	3.572	0.388	0.739
LSTM+reg.	5.747	9.767	**0.657**	0.586	8.857	8.033	**0.440**	0.628
TCN	3.784	5.650	0.677	0.636	6.043	3.485	0.439	0.725
TCN+reg.	4.491	9.335	**0.685**	0.569	7.554	6.207	**0.529**	0.631
Transf.	5.422	5.128	0.603	0.713	12.21	11.21	0.447	0.659
Transf.+reg.	6.467	8.780	**0.636**	0.610	22.05	21.99	**0.479**	0.488
Methods	**Electricity**				**Beijing PM2.5**			
	MSE	**s-MSE**	**Acc**	**s-Acc**	**MSE**	**s-MSE**	**Acc**	**s-Acc**
LSTM	3.594	1.490	0.278	0.702	0.421	0.080	0.547	0.846
LSTM+reg.	28.97	29.79	**0.440**	0.122	1.605	1.263	**0.553**	0.661
TCN	3.582	2.220	0.366	0.497	0.506	0.120	0.540	0.822
TCN+reg.	26.01	27.32	**0.442**	0.319	0.518	0.140	**0.542**	0.809
Transf.	4.299	2.659	0.350	0.509	0.560	0.180	**0.559**	0.791
Transf.+reg.	15.86	16.73	**0.417**	0.337	0.890	0.510	0.545	0.696
Methods	**Solar**				**Exchange Rate**			
	MSE	**s-MSE**	**Acc**	**s-Acc**	**MSE**	**s-MSE**	**Acc**	**s-Acc**
LSTM	1.358	0.463	**0.262**	0.352	0.388	0.236	0.427	0.685
LSTM+reg.	3.638	3.250	0.257	0.313	0.524	0.414	**0.436**	0.594
TCN	1.489	0.397	**0.269**	0.364	0.270	0.122	0.430	0.643
TCN+reg.	3.855	3.240	0.255	0.304	0.494	0.379	**0.443**	0.586
Transf.	1.802	0.391	**0.266**	0.363	3.811	3.747	0.454	0.444
Transf.+reg.	3.927	2.834	0.254	0.320	9.320	9.321	**0.471**	0.453

interesting result indicates that a simplistic baseline may outperform a sophisticated model on this dataset.

REGULARIZATION TERM. In this set of experiments, we train the models to minimize the loss function of Equation 3.14 and we report the 1-step forecasting results in Table 3.3. We also provide some examples of the predictions of the models in Figure 3.4. We observe that the proposed regularization term mitigates to some extent the effects of "mimicking", however, it does not eliminate it completely. In most cases, the models trained with the proposed loss function result into a slight increase in MSE compared to the vanilla models, but also into a larger increase in s-MSE. We also observe that even though the proposed function incurs a very small increase in MSE, it improves the generalization ability of the base models since they achieve higher accuracy in the task of predicting whether the value of the time series will increase or decrease. The increase in the achieved accuracy of the binary problem is in some cases significant. The proposed loss offers LSTM a relative increase of

Table 3.4: 5-step prediction performance of models on the Electricity dataset. All MSE/ s-MSE results are in scale ($\times 10^{-3}$).

Methods	MSE	s-MSE	Acc	s-Acc
Seq2Seq	5.495	0.522	0.375	0.605
Seq2Seq+reg.	5.410	0.477	**0.383**	0.547
LSTM	5.534	0.502	0.380	0.646
LSTM+reg.	5.561	0.328	**0.385**	0.614
TCN	5.546	0.621	0.398	0.626
TCN+reg.	5.541	0.745	**0.401**	0.610
Transf.	5.845	0.3819	0.373	0.659
Transf.+reg.	5.346	0.3850	**0.387**	0.592

Table 3.5: 1-step prediction performance of models on the stock price dataset.

Methods	Acc	F1
LSTM	0.552	0.398
LSTM+reg.	**0.570**	**0.520**
TCN	0.545	0.184
TCN+reg.	**0.586**	**0.360**

16.2% in accuracy and Transformer an increase of 6.7% on the Electricity dataset, while TCN's accuracy increases by 9.0% on Sunspots.

SENSITIVITY ANALYSIS. We next study how the performance of the proposed loss varies as a function of hyperparameter λ. We expect the effect of "mimicking" to be inversely proportional to λ. Figure 3.5 illustrates how the performance of the different models on 4 datasets varies with respect to λ. We observe that both MSE and s-MSE increase as the value of λ increases. This is not surprising since the objective of the regularization term is to make s-MSE as large as possible without hurting MSE much. In most cases, the increase of s-MSE is larger than that of MSE, which is the desired behavior. In many cases, large values of λ result into MSEs that are even greater that that of the baseline (Avg. Window $(n = 1)$). In terms of accuracy, we observe that in most cases, increasing the value of λ leads to a slight increase of Acc and a slight decrease of s-Acc.

MULTI-STEP AHEAD PREDICTIONS. We present in Table 3.4 results of the multi-step ahead forecasting experiments performed on Electricity. We employ a sequence-to-sequence model of LSTM encoder and decoder, as well as LSTM, TCN, and Transformer encoders followed by fully connected layers for direct predictions. In most cases, when trained to minimize the proposed loss function, the different models achieve slightly larger values of Acc and in some cases significantly smaller values of s-Acc. In terms of MSE, quite surprisingly in the case of all models except LSTM, MSE decreases when the proposed loss is employed.

CASE STUDY: PREDICTING STOCK PRICES TRENDS. We also experiment with a dataset recording high-frequency bids for the TSLA stocks. Due to the class imbalance, besides accuracy, we also report F1-scores in Table 3.5. The proposed term leads to slight improvements in accuracy, but significant ones in F1-score.

3.2.5 *Conclusion*

In this study, we deal with "mimicking" in time series forecasting. Our results indicate that the proposed regularization term partially mitigates this phenomenon, constituting a first approach towards this research direction. We plan to further study its properties along with potential improvements in the future. Also, investigating the exact conditions under which a model replicates the last observed values of the time series is on our agenda for future work.

4 HANDLING IRREGULAR SAMPLING IN TIME SERIES

4.1 PROMINENT DEEP LEARNING APPROACHES

Recurrent neural networks are the most prominent sequential neural network architectures for several application domains, especially when involving one-dimensional data, e. g., speech, text and time series. However, time series data may naturally arise in non-uniform time intervals, which fundamentally violates the hypothesis behind RNNs for operating on equidistant data observations. This is typical in different fields, such as electronic health records, astronomy and others [205, 233]. A straightforward solution to handle this time irregularity is to fill in missing data by employing an imputation method prior to applying a standard RNN. However, this approach remains heuristic and data-driven leading to potential information loss on the relevant input data, since knowledge about the underlying temporal dynamics is not explicitly taken into consideration for the model design.

We next provide some overview of key studies in this field for obtaining a continuous-time model, starting from the standard RNN architecture for irregular time series data, based on significant contributions in the field [34, 214]. We also provide an overview of the interpolation-prediction networks that can provide a flexible framework for interpolating or classifying irregular data with missing values [232, 234].

RECURRENT NEURAL NETWORKS (RNNS). Let $\mathbf{x}_{0:L} = (x_0, x_1, \ldots, x_L)$ and $\mathbf{t}_{0:L} = (t_0, t_1, \ldots, t_L)$ denote univariate time series observations of length $(L+1)$ and their associated timestamps. We refer to x_i as the i-th ordered value of the series and t_i as the corresponding timestamp. Without loss of generality, we assume a univariate series as input, but the notations that follow are simply extended for the multivariate case.

A naive way to employ RNNs for irregularly sampled time series incorporates the time interval between input observations, $\Delta t = t_i - t_{i-1}$, i. e., in the conventional function of hidden state update of standard RNNs, as presented by the authors in [214], such that:

$$h_i = \text{RNNCell}(h_{i-1}, \Delta t, x_i) \tag{4.1}$$

where h denotes the hidden states. It is then necessary to define a function that describes the hidden state variation between successive observations. Hidden states can naturally vary with exponential decay as proposed in several studies [26, 31, 181, 205]. Then the employed RNN can be described as:

$$h_i = \text{RNNCell}(h_{i-1} \cdot \exp\{-\tau \Delta t\}, x_i) \tag{4.2}$$

where τ denotes the decay rate. Unfortunately, this exponential decay of the hidden state has been shown experimentally to be as efficient in terms of performance as conventional RNNs [181].

NEURAL ORDINARY DIFFERENTIAL EQUATIONS (NEURAL ODES). Neural ODEs are a class of neural network architectures that build upon the concept of ordinary differential equations (Ordinary Differential Equations (ODEs)) to model continuous-time dynamics, as proposed in [34]. They provide a flexible framework for learning continuous transformations of data over time, which makes them particularly useful for modeling time series data, dynamical systems, and continuous processes. In a Neural ODE, the key idea is to parameterize the continuous dynamics of a system using neural networks [34]. Instead of specifying discrete layers and time steps as in traditional neural networks, a Neural ODE defines a continuous path of transformations. This path is governed by an ODE, typically represented as [34]:

$$\frac{dh(t)}{dt} = f_\theta(h(t), t) \tag{4.3}$$

where $h(t)$ represents the hidden state of the system at time t, f_θ is a neural network function with learnable parameters θ, which defines how the state changes over time, $\frac{dh(t)}{dt}$ represents the rate of change of the state.

Let also the initial value be $h(t_0) = h_0$. Then, the solution to the above ODE can be approximated using a solver [34]:

$$h_0, \ldots, h_L = \text{Solver}(f_\theta, h_0, (t_0, \ldots, t_L)) \tag{4.4}$$

Neural ODEs can be trained by learning the parameters θ to match observed data. They are especially powerful when dealing with irregularly sampled time series or when continuous-time modeling is more natural for the problem at hand. The adjoint sensitivity method is used in Neural ODEs to efficiently compute gradients during training [34]. It introduces adjoint variables to capture how changes in the loss function affect the intermediate states of the ODE solution. Unlike traditional backpropagation, it computes gradients by integrating these adjoint variables backwards in time, enabling efficient training of continuous-time models [198]. This method is particularly useful in scenarios where continuous-time modeling is beneficial and long-range horizons are involved, as it avoids the need to store and propagate gradients at every time step, making training more computationally efficient.

LATENT NEURAL ODES (ODE-RNN). RNNs with exponential decay rate on hidden states of Equation 4.2 can be considered to satisfy the ODE [214]:

$$\frac{dh(t)}{dt} = -\tau h, \text{ where } h(t_0) = h_0$$

The solution to the above equation is indeed $h_0 \cdot e^{-\tau \Delta t}$. This fixed approximation can be generalized by modeling the hidden state dynamics using a Neural ODE, as observed by the authors in [214].

Then an ODE-RNN hybrid model [214] can be defined as follows, using Equation 4.3 and Equation 4.4:

$$\begin{aligned} \tilde{h}_i &= \text{Solver}(f_\theta, h_{i-1}, (t_{i-1}, t_i)), \\ h_i &= \text{RNNCell}(\tilde{h}_i, x_i) \end{aligned} \tag{4.5}$$

Table 4.1: Comparison of the presented encoder-decoder architectures.

Model	Encoder	Decoder
RNN-VAE	RNN	RNN
Latent ODE (ODE)	ODE-RNN	ODE
Latent ODE (RNN)	RNN	ODE

Table 4.2: Hidden state h evolution of the presented autoregressive models.

Model	Hidden state h_i evolution
RNN	h_{i-1}
RNN-Decay	$h_{i-1} \cdot \exp\{-\tau \Delta t\}$
ODE-RNN	$\text{Solver}(f_\theta, h_{i-1}, (t_{i-1}, t_i))$

such that the hidden state between observations $\tilde{h}_i$ is considered to be the solution to an ODE and an RNN model is used to update the value of the hidden state h_i in the presence of the current observation x_i. Following [181], the proposed method in [214] describes the evolution of hidden state $\tilde{h}$ by the solution of a Neural ODE, rather than keeping it fixed, which constitutes an implicit and elegant parameterization of the temporal dynamics of the input series.

Autoregressive Modeling. The hybrid ODE-RNN of Equation 4.5 [214] can be employed for probabilistic autoregressive modeling, as standard RNNs, by learning the conditional distributions $p_\theta(x_i|x_{i-1}, \ldots, x_0)$ for each x_i based on the historical observations. However, autoregressive models, have experimentally shown to lead to decreased performance on missing input data, and accumulated errors over large forecasting horizons.

Latent-variable Modeling (Latent ODE). As in the standard Neural ODE model [34], latent-variable time series models can be conceptualized as encoder-decoder models, in which input series are transformed into a learnable fixed-size representation z, which is consequently processed by the decoder to generate the new sequence. More specifically, authors in [34] employ an RNN for encoding the input series and computing the approximate posterior distribution, which is then followed by an ODE-based decoder for generation for the latent embedding. To extend this architecture to the irregular sampling setting, authors of the hybrid ODE-RNN [214] propose an ODE-RNN encoder followed by an ODE decoder. The characteristics of the approximate posterior distribution (i. e., mean and standard deviation) are computed as a learnable function of the last hidden state of the encoder. The variational encoder-decoder framework is trained to maximize the evidence lower bound (ELBO) function. The key advantage of this approach is that it directly incorporates an uncertainty measure in the predictions, contrary to conventional RNNs and ODE-RNNs.

Table 4.1 and Table 4.2 provide a comparison of the key components of the previously explained sequence-to-sequence models for the irregular sampling setting, as well as the hidden state dynamics of the respective autoregressive parts of the architectures.

Poisson Process Likelihood Modeling. Observation times might be also significant for explicitly describing the evolution of the dynamics of particular time series data [31]. In the original Neural ODE paper [34], a Poisson process ODE-based model is also demonstrated to parameterize dynamics from observation times solely on a synthetic dataset. Following this line of work, authors of the latent ODE-RNN architecture [214] employed a Poisson process

to parameterize the intensity of events as a function of the latent representation z. Based on the starting and ending timestamp of an observation, the integral of the Poisson intensity function can be directly calculated and then latent states at all time steps can be derived using a solver on the parameterized ODE.

INTERPOLATION-BASED MODELS. For multivariate time series inputs, the irregular sampling setting might lead to a different number of observations across each variable as well as missing values in the channel dimension for a specific variable and a given timestamp. Thus, frameworks that jointly perform interpolation and prediction for multivariate time series can be a promising approach for solving subsequent tasks [89, 148, 155, 232].

Continuous-time interpolation-based models are described as follows:

$$\hat{x}(t) = \frac{\sum_i \kappa_\theta(t, t_j) x(t_i)}{\sum_i \kappa_\theta(t, t_i)} \tag{4.6}$$

where κ_θ denotes a similarity kernel, e. g., squared exponential.

Following this idea, authors in [232] propose a framework that consists of a radial basis function (RBF) network for interpolation against a set of reference points, followed by a deep neural network that performs the prediction. Instead of using a fixed similarity kernel, the key idea of the proposed model in [234] is to extend to a learnable similarity measure that can be optimized in a neural network architecture. This could improve the representational flexibility of the model compared to traditional interpolation-based approaches. Attention-based similarity [248] is a straightforward approach to form such a learnable kernel. We next present an example of the attention-based interpolation framework of [234].

Let $\mathcal{T} = \{(\mathbf{x}_{i,1:L_i}, \mathbf{t}_{i,1:L_i})\}_{i=1}^m$ be the collection of observations and time points and for the multivariate case of m time series. In the case of irregular sampling, each time series $i \in \{1, \ldots, m\}$ has a different length equal to L_i. We can access the values of the i-th channel and j-th timestamp by $x_{i,j}$ and the values of the i-th channel for all timestamps by $\mathbf{x}_i$. Let also a set of R reference time points $\tilde{\mathbf{t}} = (\tilde{t}_1, \ldots, \tilde{t}_R)$. The proposed multi-time attention module (mTAN) [234] uses the following kernel smoothing approach applied to the i-th dimension of the series:

$$\hat{x}_i^{(h)}\big(\tilde{t}, (\mathbf{x}_i, \mathbf{t}_i)\big) = \sum_{i=1}^{L_i} \text{softmax}\left(\frac{\phi_h(\tilde{t})\mathbf{w}\mathbf{v}^T \phi_h(t_{i,j})^T}{\sqrt{m}}\right) x_{i,j} \tag{4.7}$$

where $\mathbf{w}$,$\mathbf{v} \in \mathbb{R}^{d \times m}$ are learnable matrices and $\frac{1}{m}$ a scaling factor that normalizes the dot product in dimension m. The attention weights are defined based on a time attention mechanism using learnable time embeddings ϕ_h:

$$\phi_h(t)[i] = \begin{cases} \omega_{0h} \cdot t + \alpha_{0h}, & i = 0 \\ \sin\left(\omega_{ih} \cdot t + \alpha_{ih}\right), & 0 < i < d \end{cases} \tag{4.8}$$

where ω_{ih}, α_{ih} are learnable parameters. The time embedding consists of a linear function at the first dimension and is followed by periodic functions for the rest dimensions. The above

time embedding function transforms each continuous time point into H different $(d+1)$-dimensional vectors. Employing learnable time embeddings in the attention mechanism [129, 275], instead of the standard fixed positional encodings that capture discrete positions, provide flexible and continuous time representations.

Finally, mTAN given a set of query time points and a set of keys and values from the multivariate input time series, returns a P-dimensional embedding at time $\tilde{t}$:

$$\text{mTAN}\Big(\tilde{t}, (\mathbf{x}_i, \mathbf{t}_i)\Big)[p] = \sum_{h=1}^{H} \sum_{i=1}^{m} \hat{x}_i^{(h)}\big(\tilde{t}, (\mathbf{x}_i, \mathbf{t}_i)\big) \cdot U_{hip} \tag{4.9}$$

that is a linear combination of continuous-time functions $\hat{x}_i^{(h)}(\tilde{t}, (\mathbf{x}_i, \mathbf{t}_i))$ parameterized by the learnable weights U_{hip}. The introduced mTAN modules of Equation 4.9 in [234] are coupled with RNN components in both the encoder and decoder parts and form a latent variable sequence-to-sequence architecture. The whole framework is trained in a variational manner, similar to the aforementioned Latent ODE model.

CONVOLUTIONAL NETWORKS. The potential of Convolutional Neural Networks (CNNs) for addressing multivariate and irregularly sampled time series problems has not been as extensively explored as that of Recurrent Neural Networks (RNNs), which are employed as main building blocks in all the aforementioned architectures. CNNs are often favored over Recurrent Neural Networks for time series data and sequence modeling in general, due to their ability to efficiently capture localized features, parallelize processing across multiple channels, and provide translation-invariant and hierarchical feature learning. CNNs are not susceptible to the vanishing gradient problems of RNNs [170], offer regularization mechanisms, and are easier to implement. However, the choice between CNNs and RNNs depends on the specific characteristics of the data and the problem's requirements, with RNNs being more suitable when modeling sequential dependencies is crucial.

We next aim to provide a thorough experimental analysis of the state-of-the-art baselines in the irregular sampling case, as well as formalize and experimentally evaluate the use of convolutions for this particular type of time series data (Section 4.2).

4.2 TIME-PARAMETERIZED CNNS FOR IRREGULARLY SAMPLED TIME SERIES

Irregularly sampled multivariate time series are ubiquitous in several application domains, leading to sparse, not fully-observed and non-aligned observations across different variables. Standard sequential neural network architectures, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), consider regular spacing between observation times, posing significant challenges to irregular time series modeling. While most of the proposed architectures incorporate RNN variants to handle irregular time intervals, convolutional neural networks have not been adequately studied in the irregular sampling setting. In this study, we parameterize convolutional layers by employing time-explicitly initialized

kernels [141]. Such general functions of time enhance the learning process of continuous-time hidden dynamics and can be efficiently incorporated into convolutional kernel weights. We, thus, propose the time-parameterized convolutional neural network (TPCNN), which shares similar properties with vanilla convolutions but is carefully designed for irregularly sampled time series. We evaluate TPCNN on both interpolation and classification tasks involving real-world irregularly sampled multivariate time series datasets. Our experimental results indicate the competitive performance of the proposed TPCNN model which is also significantly more efficient than other state-of-the-art methods. At the same time, the proposed architecture allows the interpretability of the input series by leveraging the combination of learnable time functions that improve the network performance in subsequent tasks and expedite the inaugural application of convolutions in this field.

4.2.1 *Introduction*

Recently, there has been a growing interest in applying machine learning techniques to time series data. Besides time series forecasting, which has been extensively studied for decades [65], other tasks have also emerged recently such as time series classification [115] and generation [84].

Time series are constructed from real-world data and usually several of their observations are missing or are subject to noise. This is mainly due to irregular sampling and is common in different types of data including medical records, network traffic, and astronomical data. Unfortunately, the most successful machine learning models in sequential modeling, namely recurrent neural networks (RNNs) and convolutional neural networks (CNNs) cannot properly handle such irregularly sampled time series data. Indeed, those models treat observations successively and assume an equidistant sampling scheme. Thus, time series data that exhibits variable gaps between consecutive time points pose a significant challenge to such conventional deep learning architectures. A naive approach to deal with the above problem would be to drop some observations such that the distance between consecutive (remaining) observations is fixed. However, this would increase data sparsity, thus leading to poorly defined latent variables. A more prominent approach would be to first apply some imputation method to replace missing values with estimated values, and then to use the standard models which assume an equidistant sampling scheme. In fact, several recent approaches build on the above idea [31, 89]. However, this could potentially result in a loss of information and a violation of the underlying dynamics.

Recently, there has been an increasing interest in effectively capturing the continuous dynamics of real-world sparse and irregular multivariate time series. Most studies have extended RNNs to continuous-time hidden dynamics defined by ordinary differential equations (ODEs) [34, 214]. The effectiveness of Convolutional Neural Networks (CNNs) [152] as an alternative to recurrent architectures has been established, as long as the input dependencies that are essential fall within the memory horizon of the network. CNNs are based on parallel computations and thus are more efficient, contrary to the training instability and gradient problems of RNNs that employ back-propagation through time [265]. However, since discrete

convolutions learn independent weights for each time step in the kernel range, they do not directly capture the time irregularities. Efforts for the continuous implementation of convolutional kernels have targeted 3D data [224, 257] and recently, sequences [211]. The proposed continuous convolution for sequential data [211], CKConv, parameterizes the kernel values using a multi-layer perception (MLP) on the relative positions $\{\Delta_{\tau_i}\}$ of the observations, followed by a periodic activation function [238]. In contrast to [211] that take advantage of periodic activations, our layer can be constructed employing any predefined set of continuous functions and be followed by any activation, while using significantly fewer learnable parameters, since a single feed-forward layer is used for the parameterization of the convolutional kernel.

Following the above line of research, in this study, we develop a new model, so-called *Time-Parameterized Convolutional Neural Network* (TPCNN), which generalizes the standard CNN model to irregularly sampled time series. To achieve that, we replace the fixed kernels of CNNs with kernels whose values are parameterized both by time and by trainable variables. Thus, instead of keeping the kernel weights fixed over the whole time series length, we use different functions (e.g., linear, sinusoidal) to produce the kernels that will be convolved with each patch of the time series. Therefore, kernels can be seen as continuous functions of time, and the proposed TPCNN model can naturally learn continuous latent representations of irregular time series. Furthermore, the use of the aforementioned functions improves the explainability of the proposed model. We combine our time-parameterized convolutions with vanilla convolutions by stacking them in a deep encoder module. The proposed TPCNN model is evaluated in the tasks of time series classification and time series interpolation. Our experiments demonstrate that the proposed model performs comparably to state-of-the-art methods. The main contributions of the study are summarized as follows:

(i) Generalizing conventional, fixed convolutional kernels to time functions, that increase their representational power and still leverage properties of convolutions (e.g., locally aggregated information, fast training).

(ii) Enabling the application and proving the efficiency of deep stacked convolutions in the irregular sampling setting.

(iii) Achieving high-performance results in interpolation and classification of irregularly sampled benchmark datasets, which are comparable to other state-of-the-art methods.

4.2.2 *Related Work*

The long-standing challenge in multivariate irregular time series modeling has led to the development of various neural network architectures that explicitly handle such time-dependent peculiarity.

One strategy suggests dividing the timeline into equal intervals, filling in missing data, and then using a Recurrent Neural Network (RNN) on the imputed inputs. Using a weighted average between the empirical mean and the previous observation to perform imputation has also been proposed [31]. Alternative methods for imputation include the use of Gaussian

processes [89], or generative adversarial networks [171] prior to running the RNN on time-discretized inputs. The interpolation-prediction network [232] employs several semi-parametric interpolation layers for multivariate time series input with missing values, followed by a prediction network which is applied on the produced regularly spaced and fully observed representations. Multi-directional RNNs (M-RNN) combine past and future observations for each timestamp [281]. A differentiable set function method for classifying irregularly sampled is another line of work presented in [112].

An alternative strategy for handling irregularly sampled data involves architectures that directly model such temporal sequences. Various techniques, including adaptations of gated recurrent unit networks (GRUs) [46] and Long Short-term Memory networks (LSTMs) [110], have been introduced for this purpose. Among the several proposed modified GRU architectures [31], a prominent example takes as input observed values, indicators denoting missing data points, and the differences in time between observations. The LSTM architecture has been extended for handling the time irregularity of the data, by introducing a novel time gate in [182] that updates the memory state. The activation and deactivation of this gate are governed by distinct rhythmic oscillations, controlled by some learnable parameters. Another LSTM modification is presented in [197], where the proposed forget gate moderates the passing of memory from one time step to another. Another solution for handling irregularly sampled data is to incorporate the time gaps between observations directly into Recurrent Neural Networks (RNNs). One approach is to add the time gap Δ_t to the RNN input, which has been found to be susceptible to overfitting [181]. An alternative method is to introduce hidden states that decay over time, which has been proposed in several works as a viable solution [26, 31, 205].

Hidden states with an exponential decay can be employed to parameterize neural Hawkes processes and explicitly model observations via latent state changes at each observation event [177]. Many works focus on the continuous modeling of time series by learning a continuous-time neural representation with a latent state defined at all times. More specifically, a variational auto-encoder model, which utilizes a latent ordinary differential equation (ODE) method to approximate the hidden state dynamics via an ODE solver, has been presented [34]. Based on this approach, an ODE-RNN encoder that combines the neural ODE part of the encoder [34] with an RNN part for the hidden state update, has been proposed as an improved variation [214]. A continuous version of the GRU architecture was introduced to model the input series via continuous ODE dynamics describing the evolution of the probability distribution of the data [64]. Finally, an alternative to Neural ODEs, Neural Controlled Differential Equations represent the continuous-time analogue of an RNN, which benefits from memory-efficient adjoint-based backpropagation across observations [132].

Attention mechanisms combined with time encodings, as an alternative to positional ones [248], have been proposed [241, 244, 298]. By extending attention with learnable time embeddings [274], the recently proposed Multi-Time Attention Network [234] computes the similarity between observations at different time points using a learnable time embedding. This approach works similarly to kernel-based interpolation, but by leveraging a learnable time attention-based similarity kernel. Except for the optimization issues of RNNs, the conventional

dot-product self-attention mechanism matches queries with keys without considering the surrounding context. At the same time, space complexity grows quadratically with the input length, leading to memory constraints and potential performance limitations.

The use of implicit neural representations for creating continuous data representations by encoding the input in the weights of a neural network has recently gathered interest [193, 238]. Our approach can be conceptualized as an implicit representation of the convolutional kernels since they are parameterized as learnable and continuous functions of time. In this study, the proposed time-parameterized convolutional layer (TPC) introduces time-varying convolutional kernels, allowing for more efficient representational learning of the time dependencies among partially-observed variables. We leverage several continuous time functions for extracting learnable time embeddings of the time intervals across different variables. The proposed architecture is carefully designed for interpolation and classification tasks on irregularly sampled time series.

4.2.3 *The TPC Layer*

In this section, we define the mathematical properties of the employed Time-Parameterized layer (TPC) and analytically explain a proposed framework for tasks involving irregularly sampled, partially observed and multivariate time series.

4.2.3.1 *Preliminaries*

Convolution is a well-studied mathematical operation which has applications in many diverse scientific fields [19]. The convolution of two functions f and g, denoted by $f * g$, expresses how the shape of one is modified by the other.

CONTINUOUS CONVOLUTION. If the domains of functions f and g are continuous, convolution is defined as the integral of the product of the two functions after one is reflected and shifted. Formally, given $f\colon \mathbb{R}^D \to \mathbb{R}$ and $g\colon \mathbb{R}^D \to \mathbb{R}$, the continuous convolution operation is defined as:

$$(f * g)(\mathbf{x}) = \int_{-\infty}^{\infty} f(\mathbf{y}) g(\mathbf{x} - \mathbf{y}) dy$$

DISCRETE CONVOLUTION. In real world, signals are discrete and finite. For functions f, g, defined over the support domain of finite integer set $\mathbb{Z}^D$ and $\{-K, -K+1, ..., K-1, K\}^D$, respectively, the discrete equivalent of convolution is defined as:

$$(f * g)[n] = \sum_{k=-K}^{K} f[n-k] g[k] \tag{4.10}$$

Thus, the integral is replaced by a finite summation. Standard CNN models consist of layers that perform discrete convolutions that are defined over the discrete domain.

4.2.3.2 *Time-Parameterized 1D Convolutions*

We first introduce the key notations behind the employed time-parameterized convolutions for irregular and multivariate time series and analyze their fundamental properties.

IRREGULAR TIME SERIES AND STANDARD CNNS. Let $\{\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(N)}\}$ be a collection of multivariate time series where $\mathbf{X}^{(i)} \in \mathbb{R}^{m \times L}$ for all $i \in \{1, \ldots, N\}$. For irregular and multivariate time series, each sample $\mathbf{X}^{(i)}$ can have a different length L_i, but in this notation, we assume that samples, i. e., values and timestamps, are zero-padded to reach a maximum length L. Thus, each time series consists of m channels and has a length (i.e., number of observations) equal to L which corresponds to the observation times $\{t_1, t_2, \ldots, t_L\}$. Let also $d(\cdot, \cdot)$ denote a function that measures the distance (in time) between observations of a single channel of the collection of time series. The convolution operation of standard CNNs assumes that consecutive observations are equally spaced across all samples, and thus, the weights of the different kernels of standard CNNs are fixed across all chunks of the time series. In other words, the summation in the right part of Equation 4.10 is performed over the elements of the same set for all n. Formally, we have that $d\Big(\mathbf{X}^{(\imath)}_{i,j}, \mathbf{X}^{(\jmath)}_{i,j+1}\Big) = \tau$ holds for all $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, L-1\}$ and $\imath, \jmath \in \{1, \ldots, N\}$ where N is the number of samples. However, the above does not necessarily hold in the case of irregularly sampled time series data. Indeed, irregularly sampled time series can exhibit varying observation counts across distinct dimensions and also between different data instances. Thus, due to the assumptions it makes, the standard convolution operation of CNNs is not suitable for irregular time series data.

TIME-PARAMETERIZED CONVOLUTIONAL KERNELS. To deal with the irregularity of time series, we propose to use time-parameterized kernels. Thus, instead of a fixed kernel that slides over the patches of the time series, we use a parameterized kernel whose components are functions of time. The kernel is also parameterized by the weights of a neural network. We constraint the size of the kernel to be equal to $2z + 1$ where $z \in \mathbb{N}_0$ where $\mathbb{N}_0$ denotes the set of natural numbers together with zero. Then, the elements of the kernel are constructed by some function $g(\theta, \Delta t)$ where θ denotes some trainable parameters and Δt denotes the distance (in time) of the observation associated with some element of the kernel and the $z + 1$-th observation. Formally, the convolution is defined as follows:

$$(f * g)(t) = \sum_{i=1}^{2z+1} f(t_i) g(\theta, t - t_i) = \sum_{i=1}^{2z+1} f(t_i) g(\theta, \Delta t_i) \tag{4.11}$$

where $t_1, \ldots, t_{2z+1}$ are the timestamps associated with the observations of the patch the kernel is applied to.

The function $g(\theta, \Delta t)$ is quite general and can have different forms. In this work, we focus on interpretability and thus function $g(\theta, \Delta t) \colon \mathbb{R}^5 \to \mathbb{R}$ is defined as follows:

$$g\Big(\begin{bmatrix}\theta_1 & \theta_2 & \theta_3 & \theta_4 & \Delta t\end{bmatrix}^\top\Big) = \theta_1 \Big(\sigma\Big(h\big(\theta_3 \cdot \Delta t + \theta_4\big)\Big) + \theta_2\Big)$$

where $h\colon \mathbb{R} \to \mathbb{R}$ is a continuous function in $\mathbb{R}$ and $\sigma\colon \mathbb{R} \to \mathbb{R}$ denotes some activation function (i.e., sigmoid, ReLU, etc.). Thus, to construct each element of the kernel, function g takes as input four trainable parameters (i.e., $\theta_1, \theta_2, \theta_3$ and θ_4) and the time difference between the current observation and the center observation of the patch. Function h is chosen such that inductive bias is injected into the model. This can allow the model to capture patterns that commonly occur in time series data and also make its internal operations more interpretable. For example, a function $h(x) = c$ where c is some constant would not be a good candidate for extracting useful features from the time series. On the other hand, we employ more informative functions which can capture useful properties of time series such as trend and seasonality. In particular, we employ the following ten functions:

1. $h_1(x) = x$
2. $h_2(x) = \sin(x)$
3. $h_3(x) = \cos(x)$
4. $h_4(x) = \tan(x)$
5. $h_5(x) = \exp(x)$
6. $h_6(x) = x^2$
7. $h_7(x) = x^3$
8. $h_8(x) = \sinh(x)$
9. $h_9(x) = \cosh(x)$
10. $h_{10}(x) = \tanh(x)$

Most of the time, trend is a monotonic function, and therefore, functions h_1, h_6 and h_7 are chosen to detect trend in time series. Seasonality is a typical characteristic of time series in which the data experiences regular and predictable changes that recur over a defined cycle. Functions h_2, h_3, h_9 and h_{10} are responsible for extracting features that take seasonality into account.

The approach presented above generates kernels for univariate time series. In the case of multivariate time series, different parameters are learned for the different components of the time series. Therefore, the four parameters ($\theta_1, \theta_2, \theta_3$ and θ_4) are replaced by vectors of dimension m, i. e., $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \boldsymbol{\theta}_4 \in \mathbb{R}^m$. Thus, function $g(\boldsymbol{\theta}, \Delta t)\colon \mathbb{R}^{4m+1} \to \mathbb{R}^m$ is computed by applying function $h(\cdot)$ pointwise to m different elements. Note that Δt is still a scalar since observation times are identical across all components of the series.

4.2.3.3 *The Time-Parameterized Convolutional (TPC) Layer*

Given a sample $\mathbf{X}^{(i)}$, its corresponding observation times $\{t_1, t_2, \ldots, t_L\}$, and a time-parameterized function g, the kernel centered at the j-th observation (i. e., $\mathbf{X}^{(i)}_{:,j}$) is constructed as follows:

Patch	$\mathbf{X}^{(i)}_{:,j-K}$	$\ldots$	$\mathbf{X}^{(i)}_{:,j}$	$\ldots$	$\mathbf{X}^{(i)}_{:,j+K}$
Observation time	t_{j-K}	$\ldots$	t_j	$\ldots$	t_{j+K}
Difference in time	Δt_{j-K}	$\ldots$	0	$\ldots$	Δt_{j+K}
Kernel	$g(\boldsymbol{\theta}, \Delta t_{j-K})$	$\ldots$	$g(\boldsymbol{\theta}, 0)$	$\ldots$	$g(\boldsymbol{\theta}, \Delta t_{j+K})$

Note that $\mathbf{X}^{(i)}_{:,j}$ denotes the j-th column of matrix $\mathbf{X}^{(i)}$. Once we construct the kernel, the output of the convolution is computed as follows:

$$c = \sum_{l=1}^{m} g(\boldsymbol{\theta}, \Delta t_{j-K})_l \, \mathbf{X}^{(i)}_{l,j-K} + \ldots + \sum_{l=1}^{m} g(\boldsymbol{\theta}, 0)_l \, \mathbf{X}^{(i)}_{l,j} + \ldots + \sum_{l=1}^{m} g(\boldsymbol{\theta}, \Delta t_{j+K})_l \, \mathbf{X}^{(i)}_{l,j+K}$$

where $c \in \mathbb{R}$. In some cases, features of the multivariate time series might be missing. In such cases, the above operation would compute the sum of a smaller number of terms (since missing features are ignored). Thus, we also experimented with the mean function:

$$c = \frac{1}{\nu}\left(\sum_{l=1}^{m} g(\boldsymbol{\theta}, \Delta t_{j-K})_l \, \mathbf{X}^{(i)}_{l,j-K} + \ldots + \sum_{l=1}^{m} g(\boldsymbol{\theta}, 0)_l \, \mathbf{X}^{(i)}_{l,j} + \ldots + \sum_{l=1}^{m} g(\boldsymbol{\theta}, \Delta t_{j+K})_l \, \mathbf{X}^{(i)}_{l,j+K} \right) \tag{4.12}$$

where ν denotes the actual number of features (out of the $(2K+1)m$ features, those that are not missing).

Thus, the convolution between a sequence of observations and the kernel outputs a real number. We use zero padding and apply the kernel to all observations and, therefore we obtain a vector $\mathbf{c} \in \mathbb{R}^L$. Furthermore, similar to standard CNNs, not a single kernel, but instead a collection of kernels is generated and applied to the input. These kernels might correspond to different functions of the ones defined above (i. e., $h_1, \ldots, h_{10}$). Suppose that we use p different kernels in total (potentially of different functions). Then, the output of the TPC layer of the multivariate and irregularly sampled time series $\mathbf{X}^{(i)}$ is computed as:

$$TPC\big(\mathbf{X}^{(i)}, \mathbf{t}^{(i)}\big) = \|_{i=1}^{p} \mathbf{c}_i \in \mathbb{R}^{L \times p}$$

where $\|$ is the concatenation operator between vectors and $\mathbf{t}^{(i)}$ is a vector that stores the observation times of the time series.

4.2.3.4 *Properties of TPC Layer*

CONSTANT NUMBER OF PARAMETERS An interesting property of the TPC layer is that the number of parameters of each kernel is constant and equal to $4m$ regardless of the size of the kernel. This is because the kernel is dynamically generated based on the observation times and only $4m$ trainable parameters are involved. This is in contrast to standard convolutional layers where the number of parameters is equal to the size of the kernel plus the bias. Thus, the number of parameters of the TPC layer will be less than the number of parameters of a standard convolutional layer when the size of the kernels is greater than 4. This is likely to lead to less complex models and might significantly reduce overfitting.

TIME COMPLEXITY. The time complexity of the proposed TPC layer is approximately $\mathcal{O}(L\ell mp)$ for kernel size ℓ, similar to the vanilla 1D convolution. Since TPC relies on convolutions, that take advantage of parallel computations, it can be trained faster than recurrent neural network architectures. The complexity comparison becomes even more advantageous when compared with continuous-time models, such as neural ODEs that are significantly slower than RNNs [132].

LEARNING PROPERTIES. The proposed TCP layer introduces time-varying convolutional kernels as opposed to fixed kernels that are commonly employed in traditional convolutional neural networks (CNNs). In other words, the employed kernels do not remain fixed throughout the whole length of the input series. This particular trait of TPC does not explicitly force weight sharing between different subsequences of the time series during convolution. Weight sharing is, however, implicitly modeled via the learnable representations of time, that are used to initialize the kernel weights. This is based on the assumption that observations that are mapped to similar time embeddings will probably share similar values of weights in the convolutional operation. The proposed approach still maintains the ability to locally aggregate information by retaining the notion of fixed kernel size in the convolution operation. This allows for the output of the convolution to be locally aggregated, while still incorporating the benefits of time-varying convolutional kernels.

INVARIANCE PROPERTIES. If some patterns in the time series are identical, both in terms of the observations but also in terms of difference in time between the observations, then the TPC layer will produce the same output for those two patterns. For example, let $\mathbf{x}_i = (x_{i-K}, \ldots, x_i, \ldots, x_{i+K})$ and $\mathbf{x}_j = (x_{j-K}, \ldots, x_j, \ldots, x_{j+K})$ denote two sequences of values and $\mathbf{t}_i = (t_{i-K}, \ldots, t_i, \ldots, t_{i+K})$ and $\mathbf{t}_j = (t_{j-K}, \ldots, t_j, \ldots, t_{j+K})$ denote their respective observation times. If $\mathbf{x}_i = \mathbf{x}_j$ holds and $\Delta\mathbf{t}_i = \Delta\mathbf{t}_j$ also holds, where $\Delta\mathbf{t}_i = (t_{i-K} - t_i, \ldots, 0, \ldots, t_{i+K} - t_i)$ and $\Delta\mathbf{t}_j = (t_{j-K} - t_j, \ldots, 0, \ldots, t_{j+K} - t_j)$, then the kernels produced for these two sequences of values are identical and therefore, the layer produces the same output.

Furthermore, the different functions defined in the previous subsection make the kernels invariant to different transformations. For instance, in the above example, suppose that $\Delta\mathbf{t}_i \neq \Delta\mathbf{t}_j$, and that the k-th element of the second sequence is equal to $(k+1)2\pi$ times the corresponding element of the first sequence for $k \in \{0, 1, \ldots, 2K+1\}$. Then, the TPC layer equipped with the h_2 function (i. e., $\sin(\cdot)$ function) and with $\theta_3 = 1$ and $\theta_4 = 0$ would produce the same output for both patterns. Such a function can capture periodic temporal correlations.

4.2.3.5 *TPCNN Framework for Irregularly Sampled Time Series*

We will next discuss how the TPC layer can be integrated into neural network architectures for dealing with various tasks that involve irregular time series, such as interpolation and classification. Following previous work, we propose an encoder-decoder framework, so-called Time-Parameterized Convolutional Neural Network (TPCNN) framework. In what follows,

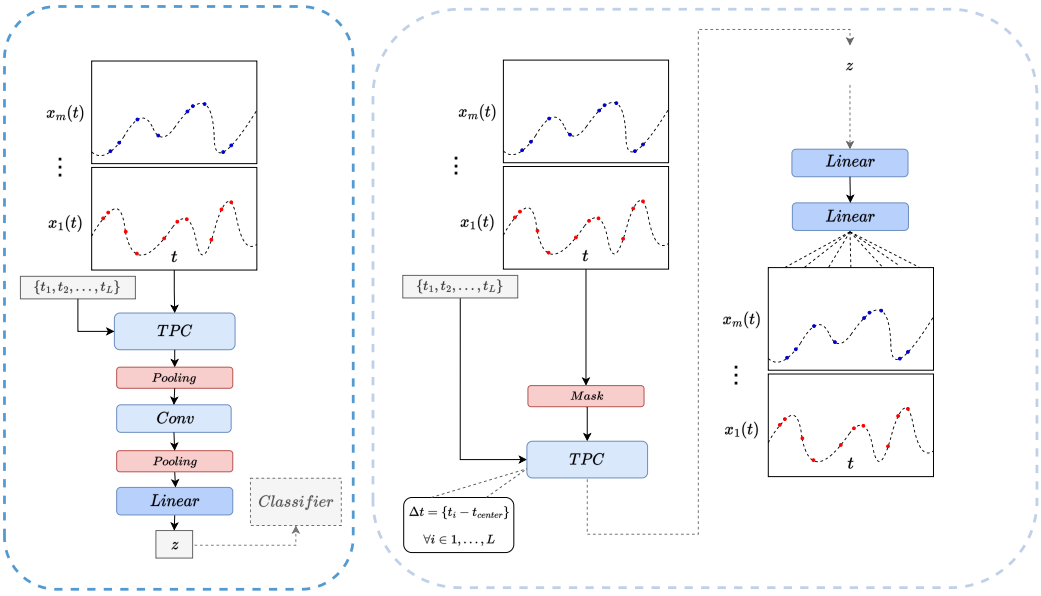


Figure 4.1: (Left) An encoder that consists of the proposed TPC layer, convolutions and pooling layer and produces a fixed-size latent representation $\boldsymbol{z}$. (Right) An encoder-decoder framework that reconstructs the series from the input using TPC and linear layers.

we give more details about the two main components of the proposed framework, namely its encoder and its decoder.

TPCNN ENCODER. This module is responsible for mapping the input time series into a latent vector which captures their overall shape and their specificities. The first layer of the encoder is an instance of the TPC layer introduced above. The TPC layer receives as input the irregular and multivariate series $\mathbf{X}^{(i)} \in \mathbb{R}^{m \times L}$ and the corresponding vector of observation times $\mathbf{t}^{(i)} = \{t_1, t_2, ..., t_L\}$. The output of TPC layer is then successively fed to vanilla convolution layers which can capture longer-time dependencies of the continuous latent representation of the time series. A pooling layer follows each convolution layer, including TPC. By down-sampling the output, such layers are expected to extract features that are good indicators of class membership or of the shape of the time series. Finally, a fully-connected layer is applied to the output of the last convolution layer to extract a low-dimensional representation $\mathbf{z}^{(i)} \in \mathbb{R}^d$ of the series.

TPCNN DECODER. This part of the architecture is responsible for reconstructing the multivariate input series from the latent vector that is produced by the encoder. The latent vector $\mathbf{z}$ that was produced by the encoder is first given as input to a fully-connected layer whose objective is to perform rescaling. The emerging vector is then passed onto another fully-connected layer which produces a matrix $\hat{\mathbf{X}}^{(i)}$ that matches the dimension of the input

time series. These reconstructed time series are then compared against the input series to evaluate the autoencoder's performance.

INTERPOLATION AND CLASSIFICATION SETTING. Note that some components of the TPCNN framework depend on the considered task, i. e., interpolation or classification. For instance, in the interpolation setting, each time a kernel of the TPC layer is applied to some subset of the input series, the observation that lies at the center of that subset is masked such that the model does not have direct access to it. On the other hand, such a masking is not performed in the case of the classification setting.

The reconstruction loss of a standard autoencoder is typically measured using the mean squared error (MSE) between the original input and the reconstructed output. For an input time series $\mathbf{X}^{(i)}$, the MSE loss is computed as:

$$\mathcal{L}_{interpolation} = \frac{1}{|O|} \sum_{j \in \mathcal{O}} \|\mathbf{X}^{(i)}_{:,j} - \hat{\mathbf{X}}^{(i)}_{:,j}\|_2^2$$

where $\mathcal{O}$ is a set that contains the indices of the observed values and $\hat{\mathbf{X}}^{(i)}$ denotes the reconstructed series produced by the decoder as a function of the latent vector $\mathbf{z}$.

The encoder-decoder framework of Figure 4.1 (Right) is combined with the MSE loss for the interpolation task. Additionally, as already discussed, masking is performed on the center element of each slice of the input series, and the rest of the observed values of the slice are used for interpolation.

In the case of classification, the latent representation $\mathbf{z}$ that is generated by the encoder and which preserves the information about the multivariate time series' dependencies, can be directly fed to a classifier module to make predictions. In the experiments that follow, we employ a 2-layer multi-layer perceptron (MLP) with ReLU activation function.

Thus, in the case of a classification problem with $|\mathcal{C}|$ classes, the output is computed as follows:

$$\mathbf{p} = softmax(MLP(\mathbf{z}))$$

Then, given a training set consisting of time series $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(N)}$, we use the negative log-likelihood of the correct labels as training loss:

$$\mathcal{L}_{classification} = -\sum_{i=1}^{N} \sum_{j=1}^{|\mathcal{C}|} \mathbf{y}_j^{(i)} \log \mathbf{p}_j^{(i)}$$

where $\mathbf{y}_j^{(i)}$ is equal to 1 if $\mathbf{X}^{(i)}$ belongs to the j-th class, and 0 otherwise.

The application of the TPCNN model to the above two scenarios is illustrated in Figure 4.1 (classification on the left and interpolation on the right).

4.2.4 *Experimental Evaluation*

In this section, we describe the experimental setup and methodology used to evaluate the performance of our proposed time-parameterized convolutional layer on various tasks involving irregular time series, including interpolation and classification.

4.2.4.1 *Datasets*

We evaluate the performance of the proposed architecture and the baselines on the following real-world datasets:

PHYSIONET: The PhysioNet Challenge 2012 dataset [236] comprises 8000 multivariate time series that correspond to records from the first 48 hours of a patient's admission to the intensive care unit (ICU). Measurements include 37 variables which can be missing at different steps and occur in irregular intervals. Half of the instances are labeled with 13.8% of instances being in the positive class (in-hospital mortality). For the interpolation experiments, we used all available instances and for the classification experiments, we used the labeled ones, following the work of [234]. We use the same experimental protocols and preprocessing steps as in [214].

MIMIC-III: The MIMIC-III dataset [124] consists of multivariate health records, that can have missing values, at the Beth Israel Deaconess Medical Center between 2001 and 2012. Based again on the preprocessing strategy of [214], we extract 53211 samples including 12 features. Given the first 48 hours of data, the task is to predict in-hospital mortality, with 8.1% of the data samples in the positive class.

HUMAN ACTIVITY: The human activity dataset contains time series data from five individuals performing various activities (such as walking, sitting, lying, standing, etc.), based on the 3D positions of tags attached to their belts, chest and ankles (12 features in total). Following the preprocessing procedures outlined by [214], a dataset of 6554 sequences and 50 time steps is extracted. The task for this dataset is to classify each time step in the series in one of the different activity classes.

4.2.4.2 *Experimental Setting*

We next explain the experimental setting we follow for interpolation and classification, similar to the work of [234]. In the case of interpolation, we study all instances (labeled and unlabeled) from the PhysioNet dataset. The dataset is partitioned into an 80% training set and a 20% test set, with a fraction (20%) of the training data serving as the validation set. The interpolation task is to predict based on a subset of available data points values for the unobserved points. This is executed using different percentages of observed steps, which vary between 50% and 90% of the total available steps. For this experiment, we perform five

different runs and report performance on the unobserved data using the mean squared error (MSE) metric.

We also use the labeled data from the PhysioNet, MIMIC-III and Human Activity datasets to conduct classification experiments. For the physiological data of PhysioNet and MIMIC-III, the classification task considers the entire time series, whereas, in the context of the human activity dataset, classification is performed for each time step in the series. We follow the same train, validation and test splitting procedure as described in the interpolation setting. For this experiment, we perform five different runs to provide the classification performance on the different datasets. For PhysioNet and MIMIC-III datasets, we report performance using the area under the ROC curve (AUC) score, due to class imbalance. For the Human Activity dataset, we asses the model performance using the accuracy metric. The validation set is used to select the best set of hyperparameters for our models via grid search.

4.2.4.3 *Baseline Models*

In this study, we conduct a thorough evaluation of several deep learning architectures as baseline models for performance comparison. These models are specifically designed to handle irregular time series and include variations of the Recurrent Neural Network (RNN), Attention modules and encoder-decoder architectures.

The specific models evaluated in this study include:

(i) Basic RNN variants including: *RNN-Impute*, *RNN-*Δ_t, *RNN-decay*, *GRU-D*. The *RNN-Impute* model employs a method to impute missing data points based on the weighted average between the last observation and the total mean of the variable [31]. In *RNN-*Δ_t the input to RNN is extended with a missing indicator for the variable and the time interval Δ_t since the last observed point. The *RNN-decay* is an RNN with hidden states that decay exponentially over time [31, 181], whereas *GRU-D* employs exponential decay on both hidden states and input [31].

(ii) Other RNN variants, such as *Phased-LSTM*, *IP-Nets*, *SeFT*, *RNN-VAE*. The *Phased-LSTM* model incorporates time irregularity through the use of a time gate that controls information from the hidden and cell states of the LSTM [182]. *IP-Nets* are Interpolation-Prediction Networks (IPN), which perform interpolation prior to prediction with an RNN on the transformed equally-spaced intervals [232]. The *SeFT* model employs learnable set functions for time series and combines the representations with an attention-based mechanism [112]. *RNN-VAE* is a standard variational RNN encoder-decoder.

(iii) ODE variants, such as *ODE-RNN*, *L-ODE-RNN*, *L-ODE-ODE*. In *ODE-RNN* neural ODEs are combined with an RNN and trained autoregressively [214]. The other variants correspond to encoder-decoder frameworks, with the former, *L-ODE-RNN*, employing an RNN encoder [34] and the latter, *L-ODE-ODE*, an ODE-RNN encoder [214].

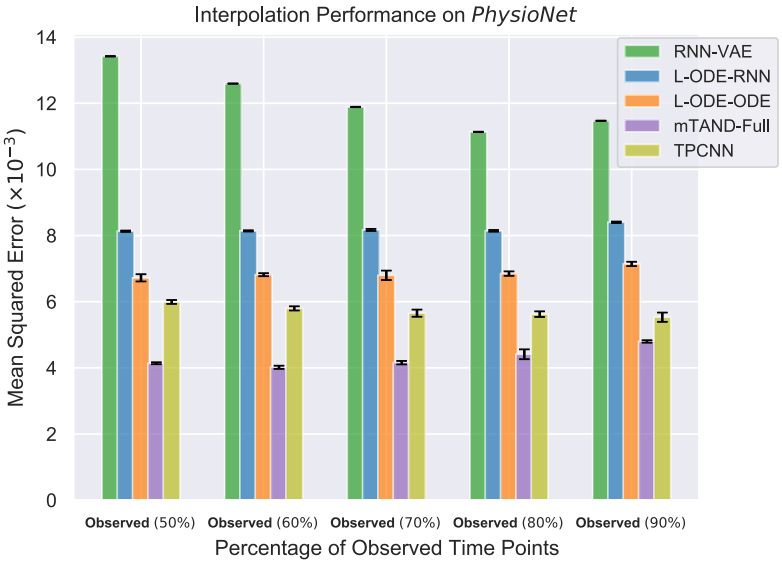


Figure 4.2: Performance for interpolation with different percentages of observed time points on *PhysioNet*.

(iv) Attention-based frameworks, including *mTAND*. The multi-time attention network, *mTAND*, interpolates missing data using a learnable attention similarity kernel between observations, which are accessed based on trainable temporal embeddings [234].

4.2.4.4 *Results*

INTERPOLATION OF MISSING DATA. In Figure 4.2 we present the results of the experimental setting designed for interpolation, as described in Section 4.2.4.2. For different percentages of observed values (i. e., ranging from 50% to 90%), we record the interpolation performance on the reconstructed irregularly sampled multivariate time series of the PhysioNet dataset using the MSE metric. We compare the proposed TPCNN model to different baseline methods designed for interpolation, including RNN-VAE, L-ODE-RNN, L-ODE-ODE and mTAND-Full (i. e., mTAND encoder-decoder framework for interpolation). We also perform tests for measuring the statistical significance of the studied methods, which leads to two distinct models that refer to the highest performances, with mTAND-Full being the best performing followed by our TPCNN model, which is the second best performing. The rest of the baselines show significantly worse performance compared to the proposed TPCNN, including the highly accurate in the irregular setting ODE-based method L-ODE-ODE. The performance of the proposed model ranges from $\sim 6.0 \times 10^{-3}$ to $\sim 5.5 \times 10^{-3}$ in terms of MSE, showing a slightly improved performance as the percentage of missing observations decreases. On the other hand, mTAND-Full ranges from $\sim 4.9 \times 10^{-3}$ to $\sim 4.1 \times 10^{-3}$ in terms of MSE, while it also shows a slightly degrading performance for a smaller percentage

Table 4.3: Performance for **per-sequence** classification on *PhysioNet* and *MIMIC-III* and **per-time-point** classification on *Human Activity* datasets. We mention in bold the best-performing method(s) and underline the second best-performing method(s) based on statistical significance tests.

Model	AUC		Accuracy
	PhysioNet	MIMIC-III	Human Activity
RNN-Impute	0.764 ± 0.016	0.8249 ± 0.0010	0.859 ± 0.004
RNN-Δ_t	0.787 ± 0.014	0.8364 ± 0.0011	0.857 ± 0.002
RNN-Decay	0.807 ± 0.003	0.8392 ± 0.0012	0.860 ± 0.005
RNN GRU-D	0.818 ± 0.008	0.8270 ± 0.0010	0.862 ± 0.005
Phased-LSTM	0.836 ± 0.003	0.8429 ± 0.0035	0.855 ± 0.005
IP-Nets	0.819 ± 0.006	0.8390 ± 0.0011	0.869 ± 0.007
SeFT	0.795 ± 0.015	0.8485 ± 0.0022	0.815 ± 0.002
RNN-VAE	0.515 ± 0.040	0.5175 ± 0.0312	0.343 ± 0.040
ODE-RNN	0.833 ± 0.009	**0.8561 ± 0.0051**	0.885 ± 0.008
L-ODE-RNN	0.781 ± 0.018	0.7734 ± 0.0030	0.838 ± 0.004
L-ODE-ODE	0.829 ± 0.004	**0.8559 ± 0.0041**	0.870 ± 0.028
mTAND-Full	**0.858 ± 0.004**	**0.8544 ± 0.0024**	**0.910 ± 0.002**
TPCNN (ours)	0.833 ± 0.001	0.8380 ± 0.0011	0.897 ± 0.004

of missing data. Finally, the simple RNN-VAE model is the worst-performing among the compared approaches.

CLASSIFICATION. We also report in Table 4.3 the results of the different baselines, as described in Section 4.2.4.3, and the proposed TPCNN model on classification for the labeled instances of PhysioNet, MIMIC-III and Human Activity datasets. For the first two imbalanced datasets, we use AUC as an evaluation metric and perform per-sequence binary classification, whereas, for the Human Activity dataset, we report accuracy for the task of per-time-point classification. For all datasets, we boldly mention the best-performing methods and underline the results for the second best-performing methods. Due to several non-statistically significant differences in performances, we have several methods being among the first or second best-performing. For PhysioNet and Human Activity datasets, our proposed TPCNN framework is the second-best method in terms of metrics, surpassed by the attention-based model mTAND-Full. More specifically, in PhysioNet the proposed model performs as well as the ODE variants (i. e., ODE-RNN, L-ODE-ODE) that are however significantly slow in terms of computational time, as mentioned in [234]. In Human Activity classification, TPCNN shows quite improved performance being $\sim 1\%$ worse than mTAND-Full. However, in the MIMIC-III classification, the proposed TPCNN model lies among the third-best-performing methods, being surpassed by several baselines. In this dataset, ODE-RNN, L-ODE-ODE and mTAND-Full methods achieve the highest AUC scores, followed by the SeFT model, which

Table 4.4: Memory and computational costs, in terms of size (number of parameters) and time per epoch (in minutes).

Model	**PhysioNet**	**MIMIC-III**	**Human Activity**
		Size (parameters)	
mTAND-Full	1.3M	1.4M	1.6M
TPCNN (ours)	350K	100K	300K
		Time per epoch (*min*)	
mTAND-Full	0.06	0.5	0.006
TPCNN (ours)	0.15	0.2	0.008

however performs significantly worse in classification experiments for the other two datasets. The significant performance advantage of mTAND-Full in this task can be attributed to its design which jointly performs interpolation and classification while directly attending only to observed time points. On the other hand, the proposed model handles missing data inside the convolutional kernel of the TPC layer by applying the mean aggregator of Equation 4.12. The aggregation neighborhood however is constrained by the kernel size and remains fixed throughout the series length. Extending the proposed architecture to incorporate size-varying kernels could further improve the learning capabilities of the TPC layer.

COMPUTATIONAL COST. In Table 4.4 we provide a comparison in terms of memory and computational costs between the proposed TPCNN and its main competitor mTAND-Full. We report the size, i. e., the number of parameters, and the time per epoch in minutes for the two methods and the three real-world datasets. Comparisons of mTAND and previous state-of-the-art models, among which the efficient ODE-based methods, as shown in [234] have demonstrated that the former is significantly faster (i. e., approximately 100 times) than ODE-based methods that make use of an ODE solver. As we can observe in Table 4.4, TPCNN is as fast as mTAND-Full in terms of time cost comparison. When it comes to the size of the model, the proposed TPCNN uses significantly fewer parameters compared to mTAND-Full, while maintaining competitive performance. More specifically, TPCNN uses approximately some hundred thousand parameters, i. e., $100 - 350$ thousand parameters, while mTAND-Full size scales to millions of parameters, i. e., approximately 1.5 million. This comparison highlights the high efficacy of convolutions in the irregular sampling setting, which allow the training of neural networks that are significantly smaller and fast compared to the baselines. Therefore, the proposed TPCNN can easily scale to larger datasets and remains efficient even when trained with fewer parameters.

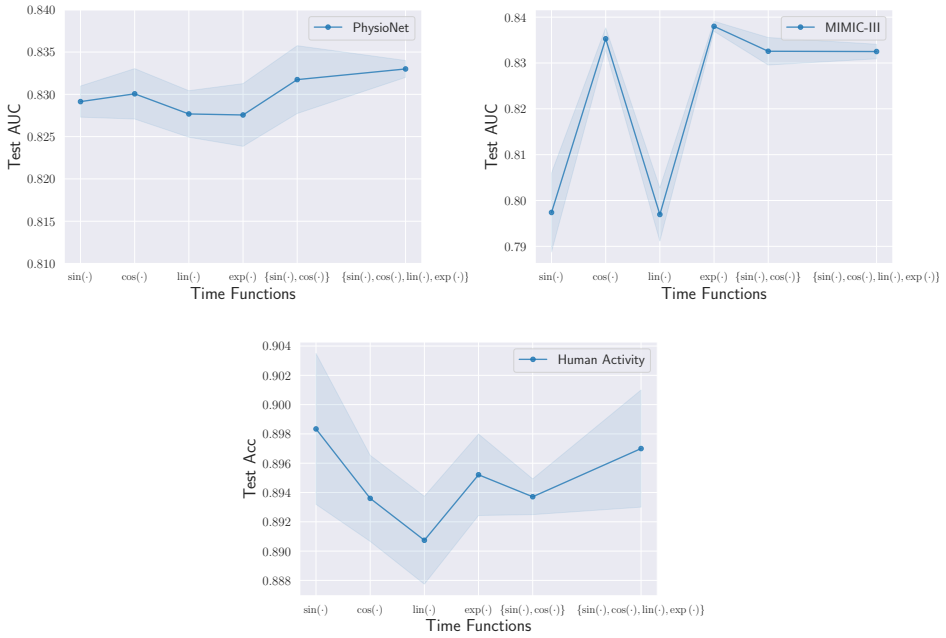


Figure 4.3: Ablation study on different time functions for the parameterization of convolutional kernels for each dataset. Each plot captures the performance (AUC or Accuracy) for each function or combination of functions on the test set.

ABLATION STUDY. We also present in Figure 4.3 an ablation study on different time functions employed for parameterizing the weights of the convolutional kernels. The performance metric (AUC or accuracy) on the test set is reported on the classification task of the real-world datasets given a different time function or combination of time functions. For all three datasets, we examine a subset of the functions described in Section 4.2.3.2. More specifically, we employ $h_1(x), h_2(x), h_3(x), h_5(x)$ (i. e., $\text{lin}(\cdot), \sin(\cdot), \cos(\cdot), \exp(\cdot)$) and their combination (e. g., $\{\sin(\cdot), \cos(\cdot)\}, \{\sin(\cdot), \cos(\cdot), \text{lin}(\cdot), \exp(\cdot)\}$). We observe that different functions may contribute more or less to the classification performance for the given dataset. In PhysioNet, while the linear function $\text{lin}(\cdot)$ and exponential function $\exp(\cdot)$ lead to the lowest AUC on the test set, when combined with $\sin(\cdot)$ and $\cos(\cdot)$ they achieve a performance improvement by $\sim 1\%$. Additionally, in MIMIC-III classification $\cos(\cdot)$ and $\exp(\cdot)$ functions show the highest AUC test, while $\sin(\cdot)$ and $\text{lin}(\cdot)$ (i. e., linear function) lead to a reduced performance by $\sim 4\%$. At the same, the combination of functions improves performance but does not surpass $\cos(\cdot)$ and $\exp(\cdot)$ when employed alone. Finally on the Human Activity dataset, $\cos(\cdot)$ function and the combination $\{\sin(\cdot), \cos(\cdot), \text{lin}(\cdot), \exp(\cdot)\}$, followed by the $\exp(\cdot)$ function achieve the highest test accuracy. The linear $\text{lin}(\cdot)$ function again, in this case, leads to the lowest accuracy score compared to the rest of the time functions. During training, we can observe that the linear time function followed by a standard non-linear activation (e. g., ReLU) when used for the parameterization of the convolutional kernel weights suffers

from slow convergence and consequently worse performance. On the other hand, periodic time functions and the exponential function seem to more efficiently describe the time dynamics and lead to smoother training when used for parameterizing convolutions. This experiment highlights the explainability aspects of the proposed TPCNN model since it allows us to determine which time functions better describe the considered time series. Furthermore, under certain conditions, the time series could be considered as a composition of such kind of functions.

4.2.5 *Conclusion*

In this work, we carefully designed and experimentally evaluated a novel time-parameterized convolutional neural network, which incorporates learnable time functions into the weights of convolutional kernels. The proposed method generalizes well in different tasks involving irregularly sampled multivariate time series while being computationally efficient and interpretable.

5 EXTRACTING DISCRETE EMBEDDINGS FROM CONTINUOUS TIME SERIES VIA GRAPH LEARNING

5.1 MODELING TIME SERIES WITH GRAPH NEURAL NETWORKS

Graph Neural Networks (GNNs) have emerged as a transformative paradigm in the field of machine learning, enabling the modeling of complex relationships and dependencies in data structured as graphs. These networks are designed to operate on data represented using nodes and edges, making them ideal for scenarios where relationships between entities are as critical as the entities themselves. GNNs have found applications across a wide spectrum of domains, ranging from social network analysis and recommendation systems to bioinformatics and chemistry [88, 91, 184]. Despite the emergence of various GNN variations in recent years, they all fundamentally adhere to a shared core concept [14, 54, 66, 103, 130, 249, 294]. Specifically, GNNs are grounded in a message passing mechanism, wherein each node refines its feature vector by aggregating information from its neighboring nodes, as outlined by [94]. Following p iterations of this message passing process, each node possesses a feature vector that captures the structural characteristics within its p-hop neighborhood. These feature vectors can serve as valuable resources for tasks on individual nodes. When confronted with tasks that pertain to the entire graph, GNNs employ a permutation-invariant readout function, such as summation over the feature vectors of all nodes within the graph, to compute a feature vector representing the entire graph.

In the next paragraphs, we provide some key definitions and notations on GNNs and present relevant ideas on how to perform graph learning on raw time series to extract underlying networks. We also present an overview of spatio-temporal modeling approaches for different time series tasks and their basic components in Section 5.1.4.

5.1.1 *Motivation*

The plethora of physical and virtual sensors has enabled the vast abundance of time series data in various scientific and engineering fields [83, 263]. Hence, techniques for time series analysis have become a crucial stage in modeling historical data to enable the extraction of insights into both past occurrences and future trends in various tasks, such as forecasting, classification, time series imputation and anomaly detection [16, 86]. Time series data may incorporate complex interactions across different variables, i. e., inter-variable relationships, as well as interactions across different time steps, i. e., inter-temporal relationships. Often, time series data arise in a spatio-temporal structure, with several variables within the series containing information about different spatial locations, thus data involve both temporal and spatial information [256]. Recently, graph learning techniques for mapping such relationships between time series variables and GNNs for feature extraction on the relevant graphs, have

been employed for several tasks involving data without necessarily an apriori underlying graph structure [122, 310].

5.1.2 *Preliminaries: Graph-Structured Data and Graph Neural Networks*

In this section, we provide some formal definitions for graph-structured data as well as some useful notation for graph neural networks.

Definition 9 (Static Graph with Node Attributes)**.** *A graph denoted as* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A}, \mathbf{X})$*, with* $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ *being the set of nodes,* $\mathcal{E} = \{(v_i, v_j) \in \mathcal{V} \times \mathcal{V} \mid \mathbf{A}_{ij} \neq 0\}$ *the set of edges (notation extends also for directed or weighted graphs),* $\mathbf{A} \in \mathbb{R}^{n \times n}$ *the adjacency matrix representing the network structure, and* $\mathbf{X} \in \mathbb{R}^{n \times d}$ *the nodes' feature matrix. Then,* $\mathbf{A}_{ij}$ *denotes the element corresponding to the* i*-th row and* j*-th column of matrix* $\mathbf{A}$*. Let also* $\boldsymbol{x}_v \in \mathbb{R}^d$ *the node features of the* i*-th node, i. e.,* v*, that is contained in the corresponding row of matrix* $\mathbf{X}$*.*

We next provide a more general definition involving a sequence of graphs, referring to spatial-temporal graphs, for the cases where the network structure dynamically evolves rather than being static.

Definition 10 (Discrete-Time Dynamic Graph)**.** *A discrete-time dynamic graph can be defined as the collection of graphs for* T *time steps, i. e.,* $\mathcal{G} = \{\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^T\}$*, where* $\mathcal{G}^t = (\mathcal{V}, \mathcal{E}^t, \mathbf{A}^t, \mathbf{X}^t)$ *is the graph with node attributes at* t*,* $\mathbf{A}^t$ *the respective adjacency matrix, and* $\mathbf{X}^t$ *the nodes' feature matrix.*

Graph Neural Networks (GNNs) are neural network models specifically created to process data that is structured as a graph. They enable learning complex relationships and patterns within graph data by aggregating information from neighboring nodes and edges. GNNs have garnered considerable interest across several domains owing to their capacity to effectively handle a wide range of applications that include structured data represented as graphs. Mathematically, GNNs may be described as a mapping function that transforms graph-structured input, consisting of nodes, edges, and features, into an output representation. These representations can be further processed to classify nodes or the whole graph and predict links for different data instances. Following, we define spatial GNNs, which contrary to spectral GNNs that are inspired by spectral graph theory, are more efficient in modeling local interactions and are computationally efficient [271].

Definition 11 (Graph Neural Networks)**.** *Let* $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ *denote an attributed graph and* $\mathbf{x}_v \in \mathbb{R}^d$ *the vector of features for node* v*. Graph neural networks (GNNs) perform operations on the graph to learn and propagate information. The two primary operations include (1) aggregation; where information is aggregated from nodes in each node's neighborhood and transformed, e. g., for node* v*, this operation involves aggregating information from its neighbors* u *based on the edge structure and (2) updating; where the aggregated information is used to update the node's feature representation (e. g., using neural network layers, non-linear*

activations, and other operations). For $p \in \{1, \ldots, P\}$ aggregation steps, the p-th GNN layer is defined by:

$$\mathbf{m}_v^{(p)} = AGGREGATE^{(p)}\left(\left\{\mathbf{h}_u^{(p-1)} : u \in \mathcal{N}(v)\right\}\right)$$
$$\mathbf{h}_v^{(p)} = COMBINE^{(p)}\left(\mathbf{h}_v^{(p-1)}, \mathbf{m}_v^{(p)}\right)$$

where $\mathbf{m}_v^{(p)}$, $\mathbf{h}_v^{(p)}$ denote the aggregated neighborhood (i. e., $\mathcal{N}(v)$) information and the node embedding for v at the p-th layer respectively. The input to the above GNN is $\mathbf{x}_v$ and the output is extracted from the last GNN layer, i. e., $\mathbf{h}_v^{(P)}$.

5.1.3 *Graph Construction for Time Series Data*

In the absence of an underlying graph structure in a given time series dataset, several approaches can be employed to infer a topology that can be subsequently processed by machine learning methods that extract knowledge from graphs. We next provide some key methods for graph construction from input data.

HEURISTIC-BASED APPROACHES. A straightforward way to extract a graph structure from the available data involves the use of heuristics that extract correlations among variables based on prior knowledge or their statistical properties. A few relevant heuristic-based graph extraction examples [122], are provided below:

- **Based on priors about correlation and proximity:** If some information on whether two time series are correlated is provided, a binary adjacency matrix $\mathbf{A}$ (directed or undirected) can be constructed, containing an edge between v_i, v_j that corresponds to the connectivity of series i, j. Relevant examples, in this case, refer to spatial data information about roads or other types of networks [93, 259]. When time series data can be described by geographical locations a simple way to define their adjacency graph $\mathbf{A}$ is to consider their proximity and more specifically the shortest distance, $d_{v_i v_j}$, between two adjacent nodes (v_i, v_j), such that $\mathbf{A}_{ij} = 1/d_{v_i v_j}$. Alternative measures, such as kernel functions can also be used to define proximity [28, 283].

- **Based on similarity measures:** Correlation graphs can also be constructed based on similarity measures between pairs of correlated time series. Such measures might include the cosine similarity measure [77], Pearson Correlation Coefficient [11], Dynamic Time Warping (DTW) [153] and others. For instance, the cosine similarity measure between nodes v_i, v_j can be computed as follows:

$$\mathbf{A}_{ij} = \frac{\mathbf{x}_{v_i}^{\intercal} \mathbf{x}_{v_j}}{\|\mathbf{x}_{v_i}\| \|\mathbf{x}_{v_j}\|}$$

  where $\|\cdot\|$ the Euclidean norm. The dependencies among pairs of time series or nodes can also be expressed in terms of binary graphs based on the Granger causality [3], as

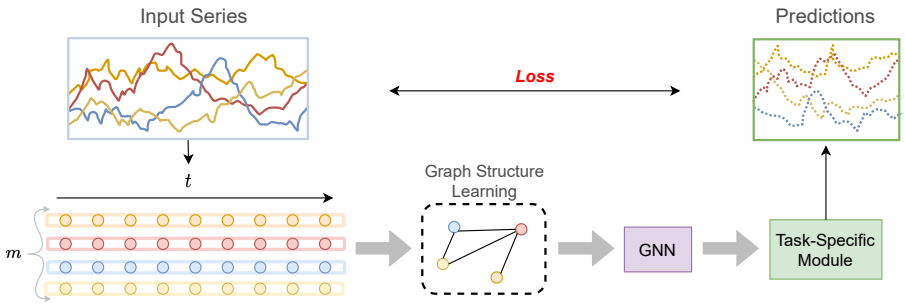


Figure 5.1: Time series graph structure learning framework.

well as by employing other dependency functions such as the transfer entropy [218] and the directed phase lag index [185].

The above heuristics are in their primary form suitable for creating correlation graphs among time series, such that each node corresponds to features extracted from an individual time series and the edges among the nodes capture the correlation or inter-variable relationship between pairs of series. Fewer works focus on creating graphs that capture inter-temporal dependencies. In terms of heuristic approaches, the visibility graph [146] and recurrence network [78] are some prominent examples, where edges connect time steps rather than variables in a univariate series.

DEEP LEARNING APPROACHES. Deep learning methods for graph construction from time series data focus on extracting a parameterized graph structure, preferably in a differentiable way, and jointly optimize it with the modules employed for performing downstream tasks (e. g., classification, forecasting). Such learning-based approaches give the opportunity to capture more complex, non-linear dependencies among time series data compared to heuristic ones. At the same time, the underlying graphs are more informative of the relevant task.

Graph learning modules aim to produce an adjacency matrix $\mathbf{A}$, where $\mathbf{A}_{ij}$ denotes the edge weight between nodes v_i, v_j. In many studies, there is a need for $\mathbf{A}$ to be sparse, such that few relations are considered important to be captured in the parameterized graph. A straightforward approach is to firstly represent each node v_i by a fixed-size vector $\mathbf{h}_{v_i}$ from a given time series i and calculate the pairwise similarity between any pair (v_i, v_j), based on different measures (e. g., dot product). These fixed-size representations can be obtained directly from the whole training time series [227] or dynamically for temporal windows of the series [135]. The former method leads to a global matrix for all time series making the number of parameters increase with the length of the input series . At the same time, the latter method is more flexible but computationally expensive in terms of handling several adjacency matrices for the different windowss [310]. A simpler approach computes the fixed-size representations $\mathbf{h}_{v_i}$ as node embeddings [68, 272]. The adjacency matrix is frequently obtained by considering the top-K scores per node and adding relevant edges [68, 272]. However, this method is not differentiable, which might significantly challenge end-to-end

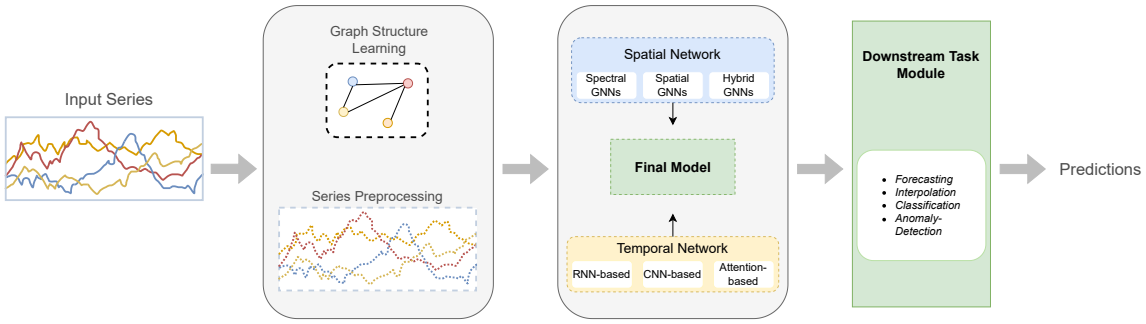


Figure 5.2: General time series joint graph structure learning and prediction framework for downstream tasks, including a taxonomy of the basic modules.

optimization of graph structure learning along with the other deep learning modules in the architecture. Another approach involves mapping the pairwise scores into the range $[0, 1]$ by applying an activation function e. g., sigmoid or softmax [135, 227]. Then a differentiable sampling-based approach can be applied so as to extract the adjacency matrix based on the edge probabilities. Gumbel softmax parameterization trick constitutes a sampling method widely used in several studies for this purpose [118, 172]. Similarly, attention-based methods can be employed for extracting a weighted adjacency matrix, based on the attention similarity score between the series [25, 220].

In the above methods, jointly learning a correlation graph, i. e., among variables in the multivariate input (m in number), introduces a computational complexity of $\mathcal{O}(m^2)$, thus the benefit of such an approach should be considered along with the potential performance improvement in the underlying task.

5.1.4 *Spatio-Temporal Modeling for the Time Series Inferred Graph Structure*

We next provide a general literature review of the basic deep learning approaches for graph-based feature learning performed jointly with neural network architectures for different downstream tasks, e. g., forecasting, classification and time series imputation and their taxonomy based on relevant surveys [120, 122]. Figure 5.1 visualizes the pipeline to incorporate graph-based extracted features to a task-specific module using a correlation graph (i. e., each node represents a single correlated time series of the data collection). Figure 5.2 shows the general framework along with the key modules (i. e., spatial and temporal) for jointly performing graph structure learning, graph-based feature extraction and temporal feature extraction for the downstream tasks involving time series [120, 122].

Spatial information from the parameterized graphs is encoded by exploiting the node-to-node relationships, while temporal information can be captured by modeling the node features' variation over time. Inter-variable and inter-temporal relations are depicted by spatial and temporal network parts respectively, as shown in Figure 5.2.

SPATIAL NETWORK. Dependencies between time series in a correlation graph can be captured using the main GNN architectures on static graphs, i. e., spectral, spatial and hybrid, which combine spectral and spatial modules [120, 122]. Spectral GNNs employ the graph shift operator (e. g., Laplacian) to encode frequency-based node interactions [219, 235]. Spatial GNNs, on the other hand, capitalize on each node's graph neighborhood to extract representations applying relevant functions. Mathematical details for spatial GNNs were provided in Definition 11.

TEMPORAL NETWORK. Temporal dependencies across time series data can be extracted by the temporal part of the architecture that can be combined with the spatial network that operates on the structure information. Methods can be classified based on whether they extract information on the time or frequency domain. Typical time-domain approaches include recurrent [156, 253, 258, 300], convolutional [179, 273, 283] and attention-based [302] architectures as well as their combination [121, 174], which are the most prominent for operating on raw sequential or temporal data.

The final model architecture for spatial-temporal data modeling is formed using a combination of spatial and temporal modules. More specifically, spatial learning networks and temporal learning networks are commonly arranged in a stacked manner, either sequentially or in parallel, as outlined by the authors in relevant surveys [120, 122]. In the first case, each distinct network might follow the other (i. e., temporal module before or after the spatial), in discrete [283, 300] or continuous architectures [87]. In the second case, modules can be repeatedly coupled in a discrete [156] or continuous way [123].

Time Series Forecasting. For capturing dependencies among variables for time series forecasting, various architectural frameworks have been explored. Spectral GNN-based approaches employ ChebConv [66] and novel modifications for spatial-temporal pattern capturing, e. g., STGCN [283]. Spatial GNN-based approaches use Message Passing Neural Network (MPNN) or graph diffusion to extract local relationships. Relevant models include DCRNN [156] and ST-MetaNet [190]. Hybrid approaches combine spectral and spatial GNNs. Auto-STGNN [252] uses neural architecture search for performance optimization, while SLCNN [295] combines global and local convolutions for extracting multi-scale spatial relations.

For capturing inter-temporal dependencies in time series, some methods, e. g., DCRNN [156], ST-MetaNet [190] and MRA-BGCN [37], utilize recurrent models such as GRUs to capture inter-temporal relations in the time domain. Convolutional Neural Networks (CNNs) also offer efficient ways to model inter-temporal dependencies. For instance, several works use convolutional networks with various modifications to extract temporal patterns [73, 283, 295]. Attention mechanisms, inspired by the Transformer model, are also gaining popularity for modeling temporal correlations. Prominent examples of such models include GMAN [302], ST-GRAT [192] and STAR [285], which employ self-attention layers to embed historical information and capture temporal dependencies. Additionally, hybrid approaches integrate

multiple techniques. For instance, some methods combine temporal attention with convolution [99, 100], while GRUs have also been combined with Transformers [259].

Finally, in terms of mixing modules for capturing spatial-temporal dependencies within time series data, architectures can be discrete and continuous, while the spatial and temporal modules can be stacked sequentially or in parallel [120, 122]. The most prominent discrete architecture, i. e., ST-UNet [284] model, incorporates graph diffusion or attention modules into RNN cells to extract both spatial and temporal dependencies. The method proposed in [37] forms another relevant example, by combining a multi-range attention mechanism with a graph convolutional RNN for leveraging multiple-range spatial-temporal information. Similarly, the STGNN* method integrates a position-wise attention mechanism in the GNN module for capturing spatial relations and combines GRU with transformer layers to extract multi-range temporal information [259]. Additionally, the presented model in [8] merges GRUs with a factorized variant of Graph Convolutional Network (GCN) on top of a graph structure learning module, while similarly, the authors in [287] propose a slightly modified graph convolutional recurrent network.

Time Series Classification. In univariate time series classification, the goal is to differentiate patterns between input series examples using class labels. Two main graph-based strategies have been employed for classification in the literature, i. e., series-as-graph and series-as-node. In the series-as-graph method, each univariate time series is transformed into a graph structure that captures distinct patterns. These graphs represent each series, with subsequences serving as nodes connected by edges that signify the evolution dependencies between them. A GNN then classifies these graphs, by modeling inter-temporal relations in order to distinguish patterns among different series classes. This approach was first introduced by Time2Graph [42] and later improved upon with Time2Graph+ [41], which incorporates shapelet graphs and GNNs for classification. On the other hand, in the series-as-node approach, each series sample refers to a separate node and edges between nodes are estimated based on pairwise distance measures. Thus, the time series classification task is essentially reframed as a node classification task. SimTSC [292] was the original study following this approach by connecting series nodes using DTW distances and applying GNNs to identify similar patterns among time series samples.

For multivariate time series classification, the idea is to model the inter-variable relationships in a relevant graph structure. Spatio-temporal architectures, as previously presented for forecasting, can then be employed and the final output layer is replaced so as to produce the class probabilities [80, 296].

Time Series Imputation. GNN-based time series imputation can be classified into in-sample and out-of-sample imputation. In-sample imputation is focused on filling missing values within provided time series data, offering both deterministic, single-estimate, and probabilistic, uncertainty-aware, imputations. Notable GNN-based methods for in-sample imputation include GACN [279], which combines Graph Attention Networks (GAT) and temporal convolutional layers for capturing both spatial and temporal dependencies in de-

scriptive embeddings that are then fed into reverse variants of such modules for imputation. Similarly, the SPIN method [175] utilizes sparse joint spatio-temporal and hierarchical attention modules to extract representations and also conditions the series' reconstruction on available observations. Additionally, GRIN [48] method leverages an RNN, which gates are implemented by a MPNN that performs imputation by aggregating information from neighboring nodes. Finally, DGCRIN [139] introduces a novel graph generator to model dynamic spatial correlations based on recurrent generated imputation data and historical information. Among probabilistic in-sample methods, a promising architecture, i. e., PriSTI [166], treats imputation by employing a denoising diffusion probabilistic generation model.

On the other hand, out-of-sample imputation is preferred when the purpose of the model is to predict missing values in different, but possibly related, time series. Recent out-of-sample imputation methods, e. g., IGNNK [269] and SATCN [270], focus on recovering signals for unobserved time series by leveraging subgraph sampling, and the integration of spatial GNNs networks with temporal convolutional networks.

Following the recent advances in time series joint graph structure learning and forecasting, in the study we present in the next section (Section 5.2), we aim to enhance the representational power of the underlying learnable graphs as well as provide a scalable but efficient end-to-end architecture for graph-based forecasting. Our proposed method aims to highlight a more natural mapping of time series to evolution graphs by constructing and modeling temporal dynamic graphs from input time series.

5.2 TIMEGNN: TEMPORAL DYNAMIC GRAPH LEARNING FOR TIME SERIES FORECASTING

Time series forecasting lies at the core of important real-world applications in many fields of science and engineering. The abundance of large time series datasets that consist of complex patterns and long-term dependencies has led to the development of various neural network architectures. Graph neural network approaches, which jointly learn a graph structure based on the correlation of raw values of multivariate time series while forecasting, have recently seen great success. However, such solutions are often costly to train and difficult to scale. In this study, we propose TimeGNN, a method that learns dynamic temporal graph representations that can capture the evolution of inter-series patterns along with the correlations of multiple series [276]. TimeGNN achieves inference times 4 to 80 times faster than other state-of-the-art graph-based methods while achieving comparable forecasting performance.

5.2.1 *Introduction*

Accurately predicting the future real values of series based on available historical records forms a coveted task over time in various scientific and industrial fields. There is a wide variety of methods employed for time series forecasting, ranging from statistical [18] to recent deep learning approaches [159]. However, there are several major challenges present.

Real-world time series data are often subject to noisy and irregular observations, missing values, repeated patterns of variable periodicities and very long-term dependencies.

While the time series are supposed to represent continuous phenomena, the data is usually collected using sensors. Thus, observations are determined by a sampling rate with potential information loss. Several continuous analogues of architectures, mainly based on RNNs and CNNs, that implicitly handle the time information have been proposed to address irregularly sampled missing data [211, 214]. The variable periodicities and long-term dependencies present in the data make models prone to shape and temporal distortions, overfitting and poor local minima while training with standard loss functions (e. g., MSE). Variants of DTW and MSE have been proposed to mitigate these phenomena and can increase the forecasting quality of deep neural networks [142, 150].

A novel perspective for boosting the robustness of neural networks for complex time series is to extract representative embeddings for patterns after transforming them to another representation domain, such as the spectral one. Spectral approaches have seen much use in the text domain. Graph-based text mining (i. e., Graph-of-Words) [213] can be used for capturing the relationships between the terms and building document-level representations.

It is natural, then, that such approaches might be suitable for more general sequence modeling. Capitalizing on the recent success of graph neural networks (GNNs) on graph-structured data, a new family of algorithms jointly learns a correlation graph between interrelated time series while simultaneously performing forecasting [25, 227, 272]. The nodes in the learnable graph structure represent each individual time series and the links between them express their temporal similarities. However, since such methods rely on series-to-series correlations, they do not explicitly represent the inter-series temporal dynamics evolution.

Some preliminary studies have proposed simple computational methods for mapping time series to temporal graphs where each node corresponds to a time step, such as the visibility graph [146] and the recurrence network [78]. In this work, we propose a novel neural network, TimeGNN, that extends these previous approaches by jointly learning dynamic temporal graphs for time series forecasting on raw data. TimeGNN (i) extracts temporal embeddings from sliding windows of the input series using dilated convolutions of different receptive sizes, (ii) constructs a learnable graph structure, which is forward and directed, based on the similarity of the embedding vectors in each window in a differentiable way, (iii) applies standard GNN architectures to learn embeddings for each node and produces forecasts based on the representation vector of the last time step.

We evaluate the proposed architecture on various real-world datasets and compare it against several deep learning benchmarks, including graph-based approaches. Our results indicate that TimeGNN is significantly less costly in both inference and training while achieving comparable forecasting performance.

5.2.2 *Related Work*

TIME SERIES FORECASTING MODELS. Time series forecasting has been a long-studied challenge in several application domains. As mentioned already, the most recent advances in

deep sequential time series modeling with application to forecasting, are built on RNNs, CNNs and Attention-based modules. Bridging CNNs and LSTMs to capture both short-term local dependency patterns among variables and long-term patterns, the Long- and Short-term Time-series network (LSTNet) [147] has been proposed. For univariate point forecasting, the recently proposed N-BEATS model [188] introduces a deep neural architecture based on a deep stack of fully-connected layers with basis expansion. Attention-based approaches have also been employed for time-series forecasting, including Transformer [248] and Informer [307]. Finally, for efficient long-term modeling, the most recent Autoformer architecture [268] introduces an auto-correlation mechanism in place of self-attention, which extracts and aggregates similar sub-series based on the series periodicity.

GRAPH NEURAL NETWORKS. Over the past few years, graph neural networks (GNNs) have been applied with great success to machine learning problems on graphs in various fields [95, 130]. The field of GNNs has been largely dominated by the so-called message passing neural networks (MPNNs) [94], where each node updates its feature vector by aggregating the feature vectors of its neighbors. In the case of time series data on arbitrary known graphs, e. g., in traffic forecasting, several architectures that combine sequential models with GNNs have been proposed [156, 226, 283, 300].

JOINT GRAPH STRUCTURE LEARNING AND FORECASTING. However, since spatial-temporal forecasting requires an apriori topology which does not apply in the case of most real-world time series datasets, graph structure learning has arisen as a viable solution. Recent models perform joint graph learning and forecasting for multivariate time series data using GNNs, intending to capture temporal patterns and exploit the interdependency among time series while predicting the series' future values. The most dominant algorithms include NRI [135], MTGNN [272] and GTS [227], in which the graph nodes represent the individual time series and their edges represent their temporal evolution. MTGNN obtains the graph adjacency from the as a degree-k structure from the pairwise scores of embeddings of each series in the multivariate collection, which might pose challenges to end-to-end learning. On the other hand, NRI and GTS employ the Gumbel softmax trick [118] to sample from the edge probabilities a discrete adjacency matrix in a differentiable way. Both models compute fixed-size representations of each node based on the time series, with the former dynamically producing the representations per individual window and the latter extracting global representations from the whole training series. MTGNN combines temporal convolution with graph convolution layers, and GTS uses a Diffusion Convolutional Recurrent Neural Network (DCRNN) [156], where the hidden representations of nodes are diffused using graph convolutions at each step.

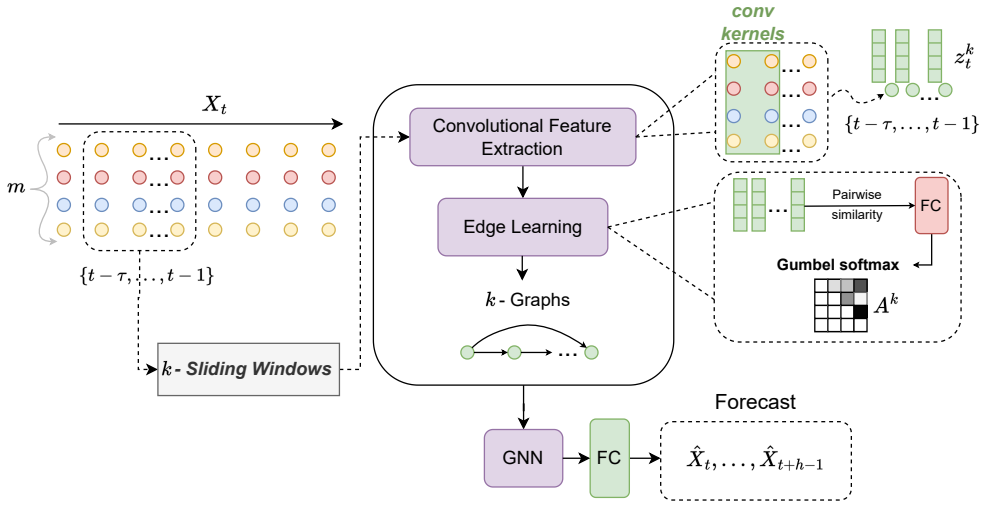


Figure 5.3: The proposed TimeGNN framework for graph learning from raw time series and forecasting based on embeddings learned on the parameterized graph structures.

5.2.3 *Method*

Let $\mathbf{X} \in \mathbb{R}^{m \times T}$ be a multivariate time series that consists of m channels and has a length equal to T. Then, $\mathbf{X}_{:,t} \in \mathbb{R}^m$ represents the observed values at time step t. Let also $\mathcal{G}$ denote the set of temporal dynamic graph structures that we want to infer.

Given the observed values of τ previous time steps of the time series, i. e., $\mathbf{X}_{:,t-\tau},\ldots,\mathbf{X}_{:,t-1}$, the goal is to forecast the next h time steps (e. g., $h=1$ for 1-step ahead), i. e., $\hat{\mathbf{X}}_{:,t}, \hat{\mathbf{X}}_{:,t+1},\ldots,\hat{\mathbf{X}}_{:,t+h-1}$. These values can be obtained by the forecasting model $\mathcal{F}$ with parameters Φ and the graphs $\mathcal{G}$ as follows:

$$\hat{\mathbf{X}}_{:,t}, \hat{\mathbf{X}}_{:,t+1},\ldots,\hat{\mathbf{X}}_{:,t+h-1} = \mathcal{F}(\mathbf{X}_{:,t-\tau},\ldots,\mathbf{X}_{:,t-1}; \mathcal{G}; \Phi) \tag{5.1}$$

5.2.3.1 *Time Series Feature Extraction*

Unlike previous methods which extract one feature vector per variable in the multivariate input, our method extracts one feature vector per time step in each window k of length τ. More specifically, temporal sub-patterns are learned using stacked dilated convolutions, similar to the main blocks of the inception architecture [162].

Given the sliding windows $\mathbf{S} = \{\mathbf{X}_{:,t-\tau+k-K}, \mathbf{X}_{:,t-\tau+1+k-K},\ldots,\mathbf{X}_{:,t+k-K-1}\}_{k=1}^{K}$, we perform the following convolutional operations to extract three feature maps $\mathbf{f}_0^k$, $\mathbf{f}_1^k$, $\mathbf{f}_2^k$, per window $\mathbf{S}^k$. Let $\mathbf{f}_i^k \in \mathbb{R}^{\tau \times d}$ for hidden dimension d of the convolutional kernels, such that:

$$\begin{aligned}
\mathbf{f}_0^k &= \mathbf{S}^k * \mathbf{C}_0^{1,1} + \mathbf{b}_{01} \\
\mathbf{f}_1^k &= (\mathbf{S}^k * \mathbf{C}_1^{1,1} + \mathbf{b}_{11}) * \mathbf{C}_2^{3,3} + \mathbf{b}_{23} \\
\mathbf{f}_2^k &= (\mathbf{S}^k * \mathbf{C}_2^{1,1} + \mathbf{b}_{21}) * \mathbf{C}_2^{5,5} + \mathbf{b}_{25}
\end{aligned} \tag{5.2}$$

where $*$ the convolutional operator, $\mathbf{C}_0^{1,1}$, $\mathbf{C}_1^{1,1}$, $\mathbf{C}_2^{1,1}$ convolutional kernels of size 1 and dilation rate 1, $\mathbf{C}_2^{3,3}$ a convolutional kernel of size 3 and dilation rate 3, $\mathbf{C}_2^{5,5}$ a convolutional kernel of size 5 and dilation rate 5, and $\mathbf{b}_{01}, \mathbf{b}_{11}, \mathbf{b}_{21}, \mathbf{b}_{23}, \mathbf{b}_{25}$ the corresponding bias terms.

The final representations per window k are obtained using a fully connected layer on the concatenated features $\mathbf{f}_0^k, \mathbf{f}_1^k, \mathbf{f}_2^k$, i. e., $\mathbf{z}^k = \mathrm{FC}(\mathbf{f}_0^k \| \mathbf{f}_1^k \| \mathbf{f}_2^k)$, such that $\mathbf{z}^k \in \mathbb{R}^{\tau \times d}$. In the next sections, we refer to each time step of the hidden representation of the feature extraction module in each window k as $\mathbf{z}_i^k, \forall i \in \{1, \dots \tau\}$.

5.2.3.2 *Graph Structure Learning*

The set $\mathcal{G} = \{\mathcal{G}^1, \mathcal{G}^2, \dots, \mathcal{G}^K\}$ describes the collection of graph structures that are parameterized for all individual sliding windows of length τ of the series, where K defines the total number of windows. The goal of the graph learning module is to learn each adjacency matrix $\mathbf{A}^k \in \{0,1\}^{\tau \times \tau}$ for a temporal window of observations $\mathbf{S}^k$. Following the works of [135, 227], we use the Gumbel softmax trick to sample a discrete adjacency matrix as described below.

Let $\mathbf{A}^k$ be a random variable of the matrix Bernoulli distribution parameterized by $\boldsymbol{\theta}^k \in [0,1]^{\tau \times \tau}$, so that $\mathbf{A}_{ij}^k \sim Ber(\theta_{ij}^k)$ is independent for pairs (i,j). By applying the Gumbel reparameterization trick [118] for enabling differentiability in sampling, we can obtain the following:

$$\begin{gathered} \mathbf{A}_{ij}^k = \sigma\Big(\big(\log(\theta_{ij}^k/(1-\theta_{ij}^k)) + (\mathbf{g}_{ij}^1 - \mathbf{g}_{ij}^2)\big)/s\Big), \\ \mathbf{g}_{ij}^1, \mathbf{g}_{ij}^2 \sim Gumbel(0,1) \forall i,j \end{gathered} \tag{5.3}$$

where $\mathbf{g}_{ij}^1, \mathbf{g}_{ij}^2$ vectors of i.i.d samples drawn from Gumbel distribution, σ the sigmoid activation and s a parameter that controls the smoothness of samples, so that the distribution converges to categorical values when $s \to 0$. The link predictor is applied on each pair of extracted features $(\mathbf{z}_i^k, \mathbf{z}_j^k)$ of window k and maps their similarity to a $\theta_{ij}^k \in [0,1]$ by applying fully connected layers and a sigmoid activation:

$$\theta_{ij}^k = \sigma\Big(\mathrm{FC}\big(\mathrm{FC}(\mathbf{z}_i^k \| \mathbf{z}_j^k)\big)\Big) \tag{5.4}$$

In order to obtain directed and forward (i. e., no look-back in previous time steps in the history) graph structures $\mathcal{G}$ we only learn the upper triangular part of the adjacency matrices.

5.2.3.3 *Graph Neural Network for Forecasting*

Once the collection $\mathcal{G}$ of learnable graph structures per sliding window k are sampled, standard GNN architectures can be applied for capturing the node-to-node relations, i. e., the temporal graph dynamics. GraphSAGE [102] was chosen as the basic building GNN block of the node embedding learning architecture. GraphSAGE can effectively generalize across different graphs with the same attributes, which is fitting for this task. GraphSAGE is an inductive framework that exploits node feature information and generates node embeddings (i. e., $\mathbf{h}_v$ for node v) via a learnable function, by sampling and aggregating features from a node's local neighborhood (i. e., $\mathcal{N}(v)$).

Let $(\mathcal{V}^k, \mathcal{E}^k)$ correspond to the set of nodes and edges of the learnable graph structure for each $\mathcal{G}^k$. The node embedding update process for each $p \in \{1, \ldots, P\}$ aggregation steps, employs the mean-based aggregator, namely convolutional, by calculating the element-wise mean of the vectors in $\{\mathbf{h}_v^{(p-1)}, \forall v \in \mathcal{N}(v)\}$, such that:

$$\mathbf{h}_v^{(p)} \leftarrow \sigma\big(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{(p-1)}\} \cup \{\mathbf{h}_u^{(p-1)} \forall u \in \mathcal{N}(v)\})\big) \tag{5.5}$$

where $\mathbf{W}$ trainable weights. The final normalized (i. e., $\tilde{\mathbf{h}}_u^{(p)}$) representation of the last node (i. e., time step) in each forward and directed graph denoted as $\mathbf{z}_{v_T} = \tilde{\mathbf{h}}_{v_T}^{(P)}$ is passed to the output module. The output module consists of two fully connected layers which reduce the vector into the final output dimension, so as to correspond to the forecasts $\hat{\mathbf{X}}_{:,t}, \hat{\mathbf{X}}_{:,t+1}, \ldots, \hat{\mathbf{X}}_{:,t+h-1}$. Figure 5.3 demonstrates the several components of the proposed TimeGNN architecture, including feature extraction, graph learning, GNN and output modules for forecasting.

5.2.3.4 *Training and Inference*

To train the parameters of Equation 5.1 for the time series point forecasting task, we use the mean absolute loss. Let $\hat{\mathbf{X}}^{(i)}, i \in \{1, ..., K\}$ denote the predicted vector values for K samples then the MAE loss is defined as:

$$\mathcal{L} = \frac{1}{K} \sum_{i=1}^{K} \|\mathbf{X}^{(i)} - \hat{\mathbf{X}}^{(i)}\|$$

The optimized weights for the feature extraction, graph structure learning and GNN and output modules are selected based on the minimum loss during training, which is evaluated as described in the experimental setup (Section 5.2.4.3).

5.2.4 *Experimental Evaluation*

We next describe the experimental setup, including the datasets and baselines we use for comparisons. We also demonstrate and analyze the results obtained by employing the proposed TimeGNN architecture and the baseline models.

5.2.4.1 *Datasets*

This work was evaluated on the following multivariate time series datasets:

EXCHANGE-RATE: This dataset consists of the daily exchange rates of 8 countries from 1990 to 2016, following the preprocessing of [147].

WEATHER: This data collection contains hourly observations of 12 climatological features over a period of four years [1], preprocessed as in [307].

1 https://www.ncei.noaa.gov/data/local-climatological-data/

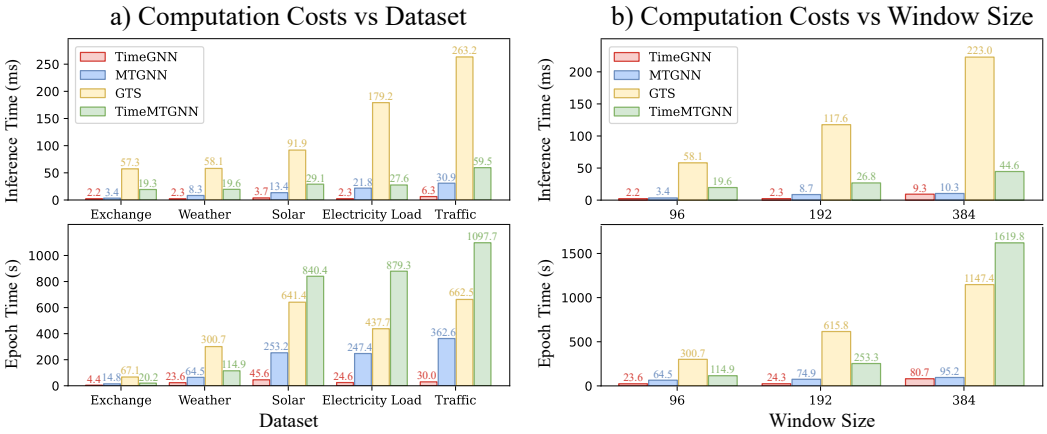


Figure 5.4: Computation costs of TimeGNN, TimeMTGNN and baseline models. a) the inference and epoch training time per epoch between datasets. b) the inference and epoch times with varying window sizes on the weather dataset

ELECTRICITY-LOAD: This dataset is based on the UCI Electricity Consuming Load dataset [2] that records the electricity consumption of 370 Portuguese clients from 2011 to 2014. As in [307], the recordings are binned into hourly intervals over the period of 2012 to 2014 and clients reduced to 321 due to missing information.

SOLAR-ENERGY: The dataset contains the solar power production records in 2006, sampled every 10 minutes from 137 PV plants in Alabama State [3].

TRAFFIC: This is a collection of 48 months, between 2015 and 2016, of hourly data from the California Department of Transportation [4]. The data describes the road occupancy rates (between 0 and 1) measured by different sensors on San Francisco Bay area freeways.

5.2.4.2 *Baselines*

We consider five baseline models for comparison with our TimeGNN proposed architecture.We chose two graph-based methods, MTGNN [272] and GTS [227], and three non graph-based methods, LSTNet [147], LSTM [110], and TCN [9]. Also, we evaluate the performance of TimeMTGNN, a variant of MTGNN that includes our proposed graph learning module. LSTM and TCN follow the size of the hidden dimension and number of layers of TimeGNN. Those were fixed to three layers with hidden dimensions of 32, 64 for the Exchange-Rate and Weather datasets and 128 for Electricity, Solar-Energy and Traffic. In the case of MTGNN, GTS, and LSTNet, parameters were kept as close as possible to the ones mentioned in their experimental setups.

2 https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014
3 http://www.nrel.gov/grid/solar-power-data.html
4 http://pems.dot.ca.gov

5.2.4.3 *Experimental Setup*

Each model is trained for two runs for 50 epochs and the average mean squared error (MSE) and mean absolute error (MAE) score on the test set are recorded. The model chosen for evaluation is the one that performs the best on the validation set during training. The same dataloader is used for all models where the train, validation, and test splits are 0.7, 0.1, and 0.2 respectively. The data is split first and each split is scaled using the standard scalar. The dataloader uses windows of length 96 and batch size 16. The forecasting horizons tested are 1, 3, 6, and 9 time steps into the future, where the exact value of the time step is dependent on the dataset (e. g., 3 time steps would correspond to 3 hours into the future for the weather dataset and 3 days into the future for the Exchange dataset). In this study, we use single-step forecasting for ease of comparison with other baseline methods. For training, we use the Adam optimizer with a learning rate of 0.001. Experiments for the Weather and Exchange datasets were conducted on an NVIDIA T4 and Electricity-Load, Solar, and Traffic on an NVIDIA A40 on the Alvis cluster at the National Academic Infrastructure for Supercomputing in Sweden (NAISS).

5.2.4.4 *Results*

SCALABILITY: We compare the inference and training times of the graph-based models TimeGNN, MTGNN, GTS in Figure 5.4. These figures also include recordings from the ablation study of the TimeMTGNN variant, which is described in the relevant paragraph below. Figure 5.4(a) shows the computational costs on each dataset. Among the baseline models, GTS is the most costly in both inference and training time due to the use of the entire training dataset for graph construction. In contrast, MTGNN learns static node features and is subsequently more efficient.

In inference time, as the number of variables increases there is a noticeable increase in inference time for MTGNN and GTS since their graph sizes also increase. TimeGNN's graph does not increase in size with the number of variables and consequently, the inference time scales well across datasets. The training epoch times follow the observations in inference time for the baseline models. GTS remains the most costly followed by MTGNN and finally TimeGNN.

Since the size of the graphs used by TimeGNN is based on window size, the cost of increasing the window size on the weather dataset is shown in Figure 5.4(b). As the window size increases, so does the cost of inference and training for all models. As the GTS graph learning module does not interact with the window size, the dramatic increase in cost can primarily be attributed to its encoder-decoder forecasting module. Similarly, MTGNN's graph learning module does not rely on the window size. MTGNN's inference times do not increase as dramatically as GTS's, showing the robustness of its forecasting modules. As the window size increases, TimeGNN's inference and training cost growth is slower than the other methods and remains the fastest of the GNN methods examined. The time-based graph learning module does not become overly cumbersome as window sizes increase.

Table 5.1: Forecasting performance for Exchange-Rate, Weather and Electricity-Load multivariate datasets and baselines for different horizons h - best in bold, second best underlined.

Exchange-Rate								
	Metric	**LSTM**	**TCN**	**LSTN**	**GTS**	**MTGNN**	**TimeGNN**	**TimeMTGNN**
h=1	mse	0.328 ± 0.007	0.094 ± 0.118	**0.004 ± 0.000**	0.005 ± 0.001	0.006 ± 0.002	0.129 ± 0.012	**0.004 ± 0.001**
	mae	0.475 ± 0.033	0.191 ± 0.163	**0.033 ± 0.000**	0.041 ± 0.004	0.048 ± 0.011	0.294 ± 0.029	0.034 ± 0.005
h=3	mse	0.611 ± 0.001	0.063 ± 0.035	0.013 ± 0.003	0.009 ± 0.000	0.012 ± 0.000	0.368 ± 0.059	**0.008 ± 0.001**
	mae	0.631 ± 0.031	0.190 ± 0.041	0.078 ± 0.012	0.063 ± 0.000	0.078 ± 0.000	0.501 ± 0.045	**0.061 ± 0.003**
h=6	mse	0.877 ± 0.105	0.189 ± 0.221	0.033 ± 0.005	**0.014 ± 0.001**	0.024 ± 0.001	0.354 ± 0.031	0.019 ± 0.004
	mae	0.775 ± 0.032	0.290 ± 0.214	0.139 ± 0.008	**0.081 ± 0.005**	0.111 ± 0.000	0.453 ± 0.052	0.099 ± 0.016
h=9	mse	0.823 ± 0.118	0.123 ± 0.030	0.030 ± 0.006	**0.020 ± 0.001**	0.035 ± 0.003	0.453 ± 0.149	0.034 ± 0.002
	mae	0.743 ± 0.080	0.277 ± 0.037	0.124 ± 0.011	**0.096 ± 0.001**	0.140 ± 0.008	0.543 ± 0.084	0.139 ± 0.010
Weather								
	Metric	**LSTM**	**TCN**	**LSTN**	**GTS**	**MTGNN**	**TimeGNN**	**TimeMTGNN**
h=1	mse	**0.162 ± 0.001**	0.176 ± 0.006	0.193 ± 0.001	0.209 ± 0.003	0.232 ± 0.008	0.178 ± 0.001	0.182 ± 0.003
	mae	0.202 ± 0.003	0.220 ± 0.011	0.236 ± 0.002	0.213 ± 0.004	0.230 ± 0.002	**0.185 ± 0.000**	0.186 ± 0.000
h=3	mse	**0.221 ± 0.000**	0.232 ± 0.003	0.233 ± 0.001	0.320 ± 0.005	0.263 ± 0.003	0.234 ± 0.001	0.234 ± 0.002
	mae	0.265 ± 0.000	0.275 ± 0.000	0.285 ± 0.000	0.320 ± 0.001	0.273 ± 0.000	**0.249 ± 0.001**	0.251 ± 0.001
h=6	mse	0.268 ± 0.004	0.274 ± 0.002	**0.266 ± 0.001**	0.374 ± 0.003	0.301 ± 0.003	0.287 ± 0.002	0.282 ± 0.007
	mae	0.320 ± 0.004	0.323 ± 0.001	0.321 ± 0.000	0.388 ± 0.002	0.311 ± 0.002	**0.297 ± 0.001**	0.300 ± 0.003
h=9	mse	0.292 ± 0.007	0.307 ± 0.009	**0.288 ± 0.000**	0.399 ± 0.002	0.329 ± 0.001	0.316 ± 0.001	0.311 ± 0.002
	mae	0.342 ± 0.003	0.350 ± 0.005	0.345 ± 0.003	0.420 ± 0.004	0.339 ± 0.004	**0.331 ± 0.001**	**0.331 ± 0.001**
Electricity-Load								
	Metric	**LSTM**	**TCN**	**LSTN**	**GTS**	**MTGNN**	**TimeGNN**	**TimeMTGNN**
h=1	mse	0.226 ± 0.002	0.267 ± 0.001	0.064 ± 0.001	0.135 ± 0.002	**0.046 ± 0.000**	0.211 ± 0.003	0.047 ± 0.000
	mae	0.323 ± 0.000	0.375 ± 0.002	0.167 ± 0.001	0.246 ± 0.001	**0.131 ± 0.000**	0.309 ± 0.001	0.135 ± 0.000
h=3	mse	0.255 ± 0.001	0.329 ± 0.015	**0.065 ± 0.001**	0.303 ± 0.019	0.079 ± 0.001	0.179 ± 0.003	0.077 ± 0.000
	mae	0.339 ± 0.000	0.406 ± 0.013	**0.163 ± 0.002**	0.388 ± 0.019	0.171 ± 0.000	0.320 ± 0.002	0.173 ± 0.000
h=6	mse	0.253 ± 0.005	0.331 ± 0.010	0.125 ± 0.006	0.334 ± 0.000	**0.097 ± 0.000**	0.246 ± 0.004	0.104 ± 0.015
	mae	0.340 ± 0.006	0.408 ± 0.009	0.238 ± 0.005	0.413 ± 0.000	**0.189 ± 0.001**	0.332 ± 0.004	0.200 ± 0.016
h=9	mse	0.271 ± 0.009	0.349 ± 0.022	0.144 ± 0.013	0.289 ± 0.021	0.108 ± 0.002	0.258 ± 0.010	**0.104 ± 0.001**
	mae	0.351 ± 0.003	0.410 ± 0.019	0.251 ± 0.013	0.368 ± 0.020	0.198 ± 0.002	0.344 ± 0.007	**0.196 ± 0.001**

FORECASTING QUALITY: Table 5.1 and Table 5.2 summarize the forecasting performance of the baseline models and TimeGNN for different horizons $h \in \{1, 3, 6, 9\}$. Table 5.1 provides the result for the first three datasets, i. e., Exchange-Rate, Weather, Electricity-Load and Table 5.2 for the rest datasets, i. e., Solar-Energy, Traffic.

In general, GTS has the best forecasting performance on the smaller Exchange-Rate dataset. The use of the training data during graph construction may give GTS an advantage over the other methods on this dataset. TimeGNN however shows signs of overfitting during training and is unable to match the other two GNNs. On the Weather dataset, the purely recurrent methods perform the best in MSE score across all horizons. TimeGNN is competitive with

Table 5.2: Forecasting performance for Solar-Energy and Traffic multivariate datasets and baselines for different horizons h - best in bold, second best underlined.

Solar-Energy								
	Metric	**LSTM**	**TCN**	**LSTN**	**GTS**	**MTGNN**	**TimeGNN**	**TimeMTGNN**
h=1	mse	0.019 ± 0.000	0.012 ± 0.000	0.007 ± 0.000	0.012 ± 0.001	**0.006 ± 0.000**	0.022 ± 0.000	**0.006 ± 0.000**
	mae	0.064 ± 0.000	0.055 ± 0.001	0.035 ± 0.000	0.046 ± 0.003	**0.026 ± 0.000**	0.059 ± 0.000	**0.026 ± 0.000**
h=3	mse	0.031 ± 0.000	0.030 ± 0.001	0.026 ± 0.000	0.044 ± 0.001	**0.022 ± 0.002**	0.030 ± 0.000	**0.022 ± 0.000**
	mae	0.086 ± 0.002	0.087 ± 0.004	0.080 ± 0.000	0.098 ± 0.003	**0.058 ± 0.002**	0.071 ± 0.000	**0.058 ± 0.000**
h=6	mse	0.046 ± 0.001	0.050 ± 0.000	0.049 ± 0.004	0.103 ± 0.001	**0.042 ± 0.000**	0.044 ± 0.000	0.043 ± 0.002
	mae	0.108 ± 0.005	0.121 ± 0.005	0.125 ± 0.013	0.163 ± 0.001	**0.086 ± 0.001**	0.090 ± 0.000	0.088 ± 0.004
h=9	mse	0.067 ± 0.003	0.073 ± 0.001	0.068 ± 0.000	0.167 ± 0.003	**0.055 ± 0.001**	0.060 ± 0.002	0.060 ± 0.000
	mae	0.138 ± 0.009	0.150 ± 0.005	0.154 ± 0.004	0.218 ± 0.006	**0.101 ± 0.001**	0.109 ± 0.001	0.110 ± 0.000
Traffic								
	Metric	**LSTM**	**TCN**	**LSTN**	**GTS**	**MTGNN**	**TimeGNN**	**TimeMTGNN**
h=1	mse	0.558 ± 0.007	0.594 ± 0.091	0.246 ± 0.002	0.520 ± 0.010	**0.233 ± 0.003**	0.567 ± 0.002	0.293 ± 0.026
	mae	0.296 ± 0.005	0.352 ± 0.025	0.203 ± 0.002	0.319 ± 0.013	**0.157 ± 0.002**	0.281 ± 0.000	0.162 ± 0.001
h=3	mse	0.595 ± 0.014	0.615 ± 0.002	0.447 ± 0.010	0.970 ± 0.027	**0.438 ± 0.001**	0.622 ± 0.006	0.465 ± 0.012
	mae	0.318 ± 0.007	0.363 ± 0.003	0.286 ± 0.009	0.456 ± 0.010	**0.205 ± 0.000**	0.306 ± 0.002	0.218 ± 0.007
h=6	mse	0.603 ± 0.001	0.680 ± 0.021	0.465 ± 0.005	0.938 ± 0.048	**0.450 ± 0.009**	0.623 ± 0.004	0.495 ± 0.012
	mae	0.321 ± 0.003	0.403 ± 0.013	0.288 ± 0.002	0.461 ± 0.023	**0.213 ± 0.003**	0.311 ± 0.007	0.239 ± 0.001
h=9	mse	0.614 ± 0.011	0.655 ± 0.017	**0.467 ± 0.010**	0.909 ± 0.024	0.471 ± 0.000	0.622 ± 0.002	0.494 ± 0.000
	mae	0.329 ± 0.010	0.382 ± 0.014	0.290 ± 0.006	0.453 ± 0.016	**0.220 ± 0.002**	0.313 ± 0.002	0.236 ± 0.005

the recurrent methods on these metrics and surpasses the recurrent models on MAE. This suggests TimeGNN is producing more significant outlier predictions than the recurrent methods and TimeGNN is the best performing GNN method.

On the larger Electricity-Load, Solar-Energy, and Traffic datasets, in general, MTGNN is the top performer with LSTNet close behind. However, for larger horizons, TimeGNN performs better than GTS and competitively with LSTNet and the other recurrent models.

This shows that time-domain graphs can successfully capture long-term dependencies within a dataset although TimeGNN struggles more with short-term predictions. This could also be attributed to the simplicity of TimeGNN's forecasting module compared to the other graph-based approaches.

ABLATION STUDY. To empirically examine the effects of the forecasting module and whether the proposed graph construction module in TimeGNN is capable of learning meaningful graphs, we conducted an ablation study where we replaced MTGNN's graph construction module with our own, so-called TimeMTGNN baseline. The rest modules and the hyperparameters in TimeMTGNN are kept as similar as possible to MTGNN. TimeMTGNN shows comparable forecasting performance to MTGNN on the larger Electricity-Load, Solar-Energy, and Traffic datasets and higher performance on the smaller Exchange-Rate and Weather

datasets. The TimeGNN graph construction module is capable of learning meaningful graph representations that do not impede and in some cases improve forecasting quality. As seen in Figure 5.4, the computational performance of TimeMTGNN suffers in comparison to MTGNN. A major contributing factor is the number of graphs produced. MTGNN learns a single graph for a dataset while TimeGNN produces one graph per window, accordingly, the number of GNN operations is greatly increased. However, the focus of this experiment was to confirm that the proposed temporal graph-learning module preserves or improves accuracy over static ones rather than to optimize efficiency.

5.2.5 *Conclusion*

We have presented a novel method of representing and dynamically generating graphs from raw time series. While conventional methods construct graphs based on the variables, we instead construct graphs such that each time step is a node. We use this method in TimeGNN, a model consisting of a graph construction module and a simple GNN-based forecasting module, and examine its performance against state-of-the-art neural networks, including some that perform jointly graph learning and forecasting. While TimeGNN's relative performance differs between datasets, this representation is clearly able to capture and learn the underlying properties of time series. Additionally, it is far faster and more scalable than existing graph methods as both the number of variables and the window size increase. However, there are several avenues for further improvement. The forecasting module, purposefully kept simple in order to examine the effects of the learnable temporal representations, could be extended to a more complex architecture. The graph learning module could also be modified to include edge weights learned on separately extracted features.

6 APPLICATIONS ON DYNAMICAL SYSTEMS

6.1 MODELING SPATIO-TEMPORAL DATA

Spatio-temporal data refers to data that have both spatial and temporal components. It involves observations that are recorded over both space (location) and time. Spatio-temporal data can represent various phenomena, such as weather patterns [201], population movements [1], traffic flow [32, 33], or any other process that varies over both geographical space and time. Examples of spatio-temporal data include satellite imagery capturing changes in land cover over time, GPS tracking data of moving objects, or climate data collected at different geographical locations over different time intervals [40, 58, 223]. A detailed overview of the most dominant deep learning approaches for modeling spatio-temporal data is provided in Section 5.1.4.

The main difference between spatio-temporal data and dynamical systems lies in their representation and nature. Spatio-temporal data is the observed data that capture variations over both space and time, while dynamical systems are mathematical models that describe the temporal evolution of a system based on certain rules or equations. In some cases, spatio-temporal data can be used to infer and validate dynamical systems, and dynamical models can help predict the behavior of spatio-temporal phenomena. However, they are two distinct concepts used to approach different aspects of understanding and analyzing phenomena that involve both spatial and temporal variations.

Performing tasks involving complex dynamical systems is significant across different scientific and engineering fields [242]. Prominent examples arise naturally in the fields of fluid dynamics [5], epidemiology [108], neuroscience [117], economics [62] and cosmology [251]. Dynamical systems are systems whose states evolve with time over a state space and whose behaviour is described by a set of predefined rules. Dynamical systems are rigorously represented and analyzed through the formulation of differential equations, serving as mathematical expressions that encapsulate the relationships between variables and their derivatives, thereby characterizing the evolution of the system over time or in spatial domains.

Following next, we present some precise mathematical descriptions and examples of dynamical systems in Section 6.1.1, along with an overview of current deep learning techniques used to address underlying challenges in Section 6.1.2, as outlined in relevant surveys [105, 255].

6.1.1 *Preliminaries: Dynamical Systems*

If the dynamical system evolves not only in time but also in space, it can be represented using partial differential equations (PDEs). In this case, the state variables depend on both

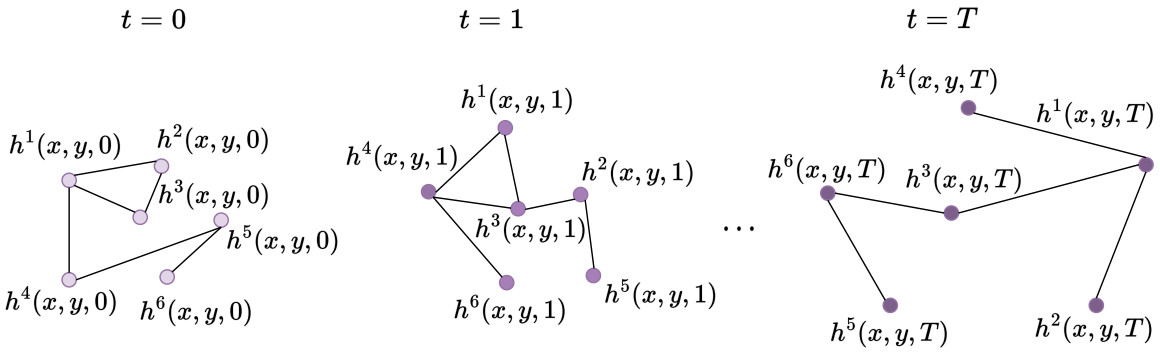


Figure 6.1: Example of the dynamical system of Equation 6.1 evolving in x, y and t coordinates.

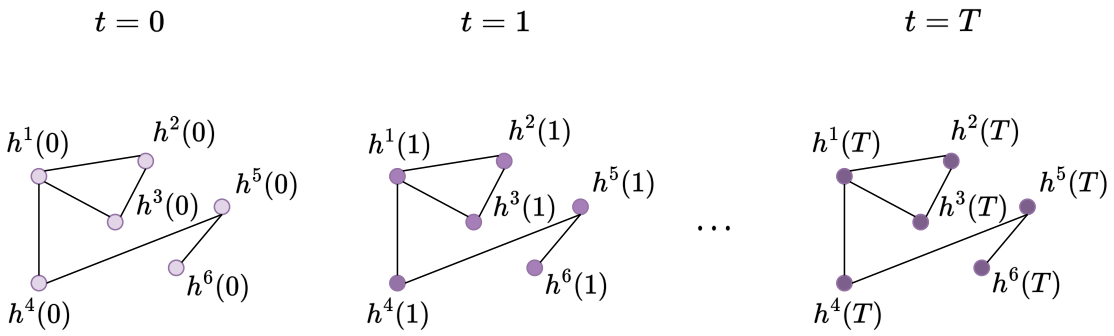


Figure 6.2: Example of the dynamical system of Equation 6.2 evolving in t coordinate.

time and spatial coordinates. We next provide the mathematical notation for a dynamical system described as a general system of PDEs [85].

Let $\mathbf{h} = (h^1, h^2, \ldots, h^d)$, $\mathbf{h} : \Omega \mapsto \mathbb{R}^d$ be functions in an open subset Ω of $\mathbb{R}^n$ and an n-dimensional state vector $\mathbf{x} \in \mathbb{R}^n$. The evolution of the system over time and space is described by the following k-th order system of partial differential equations for a fixed integer $k \geq 1$. Given $\mathcal{F} : \mathbb{R}^{mn^k} \times \mathbb{R}^{mn^{k-1}} \times \ldots \times \mathbb{R}^{mn} \times \mathbb{R}^m \times \Omega \mapsto \mathbb{R}^d$, this system is defined as follows:

$$\mathcal{F}(\mathcal{D}^k \mathbf{h}(\mathbf{x}), \mathcal{D}^{k-1} \mathbf{h}(\mathbf{x}), \ldots, \mathcal{D}\mathbf{h}(\mathbf{x}), \mathbf{h}(\mathbf{x}), \mathbf{x}) = 0 \tag{6.1}$$

where $\mathcal{D}^k, \mathcal{D}^{k-1}, \ldots, \mathcal{D}$ are differential operators of order up to k and $\mathcal{F}$ are the corresponding operators that model the evolution of the state vector. Solving these partial differential equations provides a spatio-temporal trajectory that describes the evolution of the system's state variables over time and space, by finding the functions $\mathcal{F}$ that satisfy Equation 6.1.

We then define a simpler form of a dynamical system that evolves only in time (i. e., for $k = 1$). Let $\mathbf{h} = (h^1, h^2, \ldots, h^d)$, $\mathbf{h} \in \mathbb{R}^d$ be the governing equations of the dynamical system,

and let t represent time. The evolution of the system over time is described by the following set of ordinary differential equations (ODEs):

$$\frac{d\mathbf{h}}{dt} = f(\mathbf{h}, t) \tag{6.2}$$

In this notation, each $\frac{dh^i}{dt}$ represents the time rate of change of each h^i for $i \in \{1, \ldots, d\}$ with respect to time t, and f are the corresponding functions that model the evolution of the state variable. This system of equations represents the dynamics of the system, where the state equations change over time based on the functions f that govern their behavior. Solving these differential equations provides a trajectory that describes the evolution of the system over time.

We provide in Figure 6.1 and Figure 6.2 two examples of dynamical systems for Equation 6.1 and Equation 6.2, evolving in both space and time or only in time, respectively. Let $h^i(t)$ describe the evolution of the state vector for each function $i \in \{1, \ldots, 6\}$ and for each time step $t \in \{0, \ldots, T\}$. We also consider a topology for the two dynamical system examples, where vertices represent the governing functions and edges represent the correlations or forces among those functions between different points of the considered space.

6.1.2 *Deep Learning Methods for Dynamical Systems*

NUMERICAL METHODS FOR APPROXIMATING SOLUTIONS. Numerically simulating dynamical systems governed by nonlinear differential equations poses a considerable challenge due to the inherent difficulty in mathematically tackling these equations [113, 165], resulting in prolonged computation times. Alternatively, the development of models with reduced complexity, guided by specific assumptions, provides a feasible approach [29]. However, in complex real-world scenarios where comprehensive knowledge of the dynamics is lacking, the task of deriving a simplified yet accurate model is challenging, potentially leading to equations that inadequately represent the system's evolution [255].

DEEP LEARNING APPROACHES AND UNDERLYING CHALLENGES. Deep learning techniques offer a promising avenue for comprehending and modeling complex spatio-temporal dynamics [63, 137, 158, 217], without the need for employing extensive traditional numerical methods. Despite their speed advantages, the data-driven nature of DL methods brings limitations, especially when data availability is restricted, where most cutting-edge machine learning methods lack robustness and convergence guarantees [254, 255]. The intrinsic nonlinear and chaotic dynamics of real-world systems introduce significant challenges, often leading to forecasts that contradict the underlying dynamics, unless explicit constraints are provided.

Recently, the integration of conventional physics-guided methods with deep learning models has garnered attention for modeling complex spatio-temporal data and dynamical systems [22, 128, 144, 266]. The so-called physics-guided deep learning methods remain fast and show improved generalization performance compared to data-driven approaches. By embedding

knowledge of system dynamics in the model architecture, faster convergence is enabled, since the parameter space is significantly constrained based on the underlying physical principles.

APPROACHES FOR INCREASED EFFICACY AND ROBUSTNESS. Deep neural networks are utilized to approximate solutions for problems associated with partial differential equations in dynamic systems, adhering to the framework of the physics-informed neural networks (PINNs) [27, 56, 203, 204]. By incorporating physics-based constraints into their loss function, PINNs guide the network to learn solutions aligning with the governing equations of the system. However, the performance of PINNs is dependent on the input properties, thus noisy or limited observations may impact their generalizability, especially when applied to data differing significantly from the training set [39, 277].

As an alternative to the above loss functions, physics-guided neural network architectures can be designed, imposing harder constraints for increased generalizability, while being highly demanding in terms of design. Neural ODE models [34], that leverage ordinary differential equations (ODEs) to model dynamic systems, constitute a prominent example of such methods and exhibit computational efficiency. Few recently proposed deep learning models, incorporating differential equations, have significantly outperformed purely data-driven approaches [168, 254]. Finally, several interesting research works extend the application of deep learning architectures to residual estimation and approximation of complex components of physics-based models [12, 247].

Following the recent advances in incorporating prior knowledge of the underlying dynamics in DL architectures for simulating spatio-temporal dynamical systems, in the next section (Section 6.2) we focus on the particular case of modeling spreading processes on networks. More specifically, we study epidemic spreading on networks of contacts under the SIR mathematical priors. Our proposed neural network architecture and the experimental design we follow can be adapted to different spreading phenomena on networks, thus constituting a notable application of physics-informed DL approaches for temporal dynamical systems.

6.2 NEURAL ORDINARY DIFFERENTIAL EQUATIONS FOR MODELING EPIDEMIC SPREADING

Mathematical models of infectious diseases have long been used for studying the mechanisms by which diseases spread, for predicting the spread of epidemics, and also for controlling their outbreaks. These models are based on some assumptions and different assumptions give rise to different models. Models on social networks of individuals which capture contact patterns are usually more realistic and can more accurately model contagion dynamics. Unfortunately, computing the output of realistic models is often hard. Thus, modeling the evolution of contagion dynamics over large complex networks constitutes a challenging task. In this study, we present a computational approach to model the contagion dynamics underlying infectious diseases. Specifically, we focus on the susceptible-infectious-recovered (SIR) epidemic model on networks [140]. Given that this model can be expressed by an intractable system of

ordinary differential equations, we devise a simpler system that approximates the output of the model. Then, we capitalize on recent advances in neural ordinary differential equations and propose a neural architecture that can effectively predict the course of an epidemic on the network. We apply the proposed architecture on several network datasets and compare it against state-of-the-art methods under different experimental settings. Our results indicate that the proposed method improves predictions in various spreading scenarios, paving the way for the extensive application of interpretable neural networks in the field of epidemic spreading. At the same time, the proposed model is highly efficient even when trained on very large networks where traditional algorithms become significantly slower.

6.2.1 *Introduction*

Spreading phenomena over complex networks are ubiquitous ranging from infectious diseases [51] and gossip in physical networks [43] to misinformation [23] and marketing campaigns [76] on social media. Modeling such spreading phenomena over complex networks has been lying at the core of various applications over the years. Indeed, such models would allow governments and policymakers to predict and control the spread of epidemics (e. g., COVID-19) on networks of contacts [169, 191], while they would also allow social media platforms to predict and prevent the spread of rumors and misinformation [245, 301]. Different mathematical models have been developed over the years. For instance, in epidemiology, compartmental models such as susceptible-infectious-recovered (SIR) and susceptible-infectious-susceptible (SIS), are often applied to the mathematical modeling of infectious diseases.

The outcome of a spreading process over a network is generally quantified as a node's probability of infection, for simple models like the Independent Cascade, or a quantity in compartmental models such as SIR. Several methods have been invented to derive a fast and reliable prediction of a spreading process over a given network. One can solve the system of differential equations that describes the epidemic model using computational methods [24]. Alternatively, the real state of the system can be approximated by simulating the spreading process multiple times in a Monte-Carlo fashion [131, 194]. The first option is fast enough but suffers from low accuracy, while the second is more accurate but too inefficient, as it requires typically several thousands or millions of simulations for an accurate approximation. A more balanced approach, dynamic message passing, approximates the solution using dynamic equations between nodes [127].

Recently, there has been an increasing interest in applying machine learning and artificial intelligence approaches to combinatorial optimization problems on networks [60, 125]. This approach usually involves training predictive models on instances of those problems. Once these models are trained, they can then be used for making predictions, but also for gaining insights into complex phenomena. These approaches usually rely on graph neural networks [271], a family of deep learning models that has attracted a lot of attention recently and which is particularly suited to problems that involve some kind of network structure. These models have been applied with great success to different problems such as predicting the quantum mechanical properties of molecules [94] and traffic prediction [70]. Thus, graph

neural networks could offer great potential to build effective data-driven dynamical models on networks. However, graph neural networks, on their own, might fail to fully capture the underlying dynamics of complex processes. For instance, in the case of mathematical models of infectious diseases such as the well-known SIR model, it might be challenging for the model to learn to predict the state of a node in a given time step. Fortunately, several of those compartmental models can be described mathematically by a set of differential equations. One can capitalize on such kind of information and incorporate structure into the learning process. This approach has already been applied to some problems (e. g., in physics) and the results indicate that it makes it easier for the model to encode the underlying dynamics [126].

In this work, we propose a novel deep neural network architecture for modeling and predicting spreading processes. We focus on the well-established susceptible-infectious-recovered (SIR) epidemiological model on arbitrary networks. In each time step, the network can be in one of 3^n states, where n is the number of nodes of the network. The dynamics of the SIR model is described by a Markov chain on a state space of dimension 3^n, while the time dependence of the probabilities of the states is governed by a system of 3^n linear ordinary differential equations. The exponential size of the system makes the analysis hard and thus, previous studies have resorted to large-scale simulations. However, for large networks, it is computationally very challenging to simulate the network SIR model, and hence, for such kind of networks, little is known about the long- but also short-term evolution of the model. Instead, in this study, we capitalize on recent advancements in the field of neural ordinary differential equations [34] and we propose a new architecture, so-called Graph Neural ODE Network (GN-ODE), to predict the probability that a node is in each one of the three states in a given time step. More specifically, we study each node individually and we employ a simpler system of differential equations which consists of $3n$ equations instead of 3^n. Not surprisingly, by decreasing the complexity, we obtain an approximation of the exact solution. This simpler system of differential equations is integrated into a neural network model which is responsible for fine-tuning the approximate system, thus leading to more accurate predictions. The output of the neural network is computed using a black-box differential equation solver with constant memory cost. It turns out that the proposed architecture employs a message passing mechanism similar to those of graph neural networks (GNNs) [94], that forms the temporal discretized approximation equations of the ODE solver, aiming to enhance their representational power in predicting epidemics spreading. To evaluate the proposed architecture, we conduct experiments on several networks of different sizes, including out of distribution testing of their generalization ability. We further investigate whether the proposed model can generalize to unseen networks by training on small networks and then evaluating the predictive performance on larger networks. Our results indicate that the proposed neural differential equation architecture outperforms vanilla GNNs in forecasting complex contagion phenomena, and thus can replace time-consuming simulations in several scenarios.

6.2.2 *Neural ODEs for Modeling Epidemic Spreading*

6.2.2.1 *Background*

PROBLEM DEFINITION. Epidemics are usually studied via compartmental models, where individuals can be in different states, such as susceptible, infected, or recovered. Contact networks have been considered in modeling epidemics, as a realistic simulation of the contact process in a social context. A contact network is composed of nodes representing individuals and links representing the contact between any pair of individuals. Following previous studies, we use an individual-based SIR approach to model the spread of epidemics in networks [282].

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a graph (a.k.a., network) where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. We will denote by n the number of vertices and by m the number of edges. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ of a network $\mathcal{G}$ is a matrix that encodes edge information in the network. The element of the i-th row and j-th column is equal to the weight of the edge between vertices $v_i, v_j \in \mathcal{V}$ if such an edge exists, and 0 otherwise. There are three states each node can belong to: (1) susceptible S; (2) infected I; or (3) recovered R.

The transmission of the disease is probabilistic. Thus, each edge $(v_i, v_j) \in \mathcal{E}$ is associated with a probability that v_i transmits the disease to v_j in case v_i becomes infected while v_j is susceptible. Also, let $\beta_{ij} \in [0, 1]$ be the infection rate of edge $(i \to j)$ and $\gamma_i \in [0, 1]$ be the recovery rate of node i. In this work, we assume uniform infection and recovery rates, i. e., $\beta_{ij} = \beta$ for all pairs of nodes (v_i, v_j) connected by an edge and $\gamma_i = \gamma$ for every node v_i of the network.

In the considered model, disease spread takes place at discrete time steps $t = 1, 2, \ldots, T$. For a given network $\mathcal{G}$ and some time step t, three different probabilities are associated with each node representing the probability that the node belongs to each one of the above three states. These probabilities are stored in vectors $\mathbf{s}^{(t)}, \mathbf{i}^{(t)}, \mathbf{r}^{(t)} \in \mathbb{R}^n$ for $t \in \{1, \ldots, T\}$. Given the structure of the network and some initial conditions, exactly computing those probabilities is intractable. Indeed, it has been shown that finding the probability of infection of an SIR model on a network is an NP-hard problem and that this problem is related to long-standing problems in the field of computer networks [228].

NEURAL ODES. These are deep neural network models which generalize standard layer to layer propagation to continuous depth models [34]. More specifically, the continuous dynamics of hidden units are parametrized using an ODE specified by a neural network:

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta)$$

where $t \in \{0, \ldots, T\}$ and $\boldsymbol{h}(t) \in \mathbb{R}^d$. Starting from the initial layer $\boldsymbol{h}(0)$, the output layer $\boldsymbol{h}(T)$ is the solution to this ODE initial value problem at some time T. This value can be obtained by a black-box differential equation solver. Euler's method is the simplest method for solving ODEs, among others (e. g., Runge-Kutta). For example, using Euler's/1st-order Runge-Kutta method the solution can be approximated by:

$$\boldsymbol{h}(t + s) = \boldsymbol{h}(t) + s\, f(\boldsymbol{h}(t), t, \theta)$$

where s is the step size. To compute gradients with respect to all inputs of any ODE solver [36] introduce a method that scalably backpropagates through the operations of the solver. This allows training with constant memory cost as a function of depth.

6.2.2.2 *The Proposed GN-ODE Model*

As already discussed, we use an individual-based SIR approach to model the spread of epidemics in networks. Nodes represent individuals, while edges represent the contact between pairs of individuals. Unfortunately, the exact computation of the epidemic spread in a network under this model is not feasible in practice. Therefore, approximate computation schemes have been proposed, and some of them are described by a system of ordinary differential equations (ODEs) [282]. We employ the following system of ODEs:

$$\begin{aligned} \frac{d\mathbf{S}}{dt} &= -\beta(\mathbf{A}\,\mathbf{I}_h) \odot \mathbf{S}_h \\ \frac{d\mathbf{I}}{dt} &= \beta(\mathbf{A}\,\mathbf{I}_h) \odot \mathbf{S}_h - \gamma \mathbf{I}_h \\ \frac{d\mathbf{R}}{dt} &= \gamma \mathbf{I}_h \end{aligned} \tag{6.3}$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the adjacency matrix of the network, $\mathbf{S}, \mathbf{I}, \mathbf{R} \in \mathbb{R}^n$ are vectors that represent the three different states of the SIR model for all the nodes of the network, $\mathbf{S}_h, \mathbf{I}_h, \mathbf{R}_h \in \mathbb{R}^n$ the hidden representations of the three states and $\odot$ denotes the elementwise product. We also denote as $\beta \in [0, 1]$ the infection rate of edges and as $\gamma \in [0, 1]$ the recovery rate of nodes. By solving the above ODEs (with some initial conditions), we can approximate the spread of the epidemic in the network.

Unfortunately, the above system of ODEs might fail to capture the complex dynamics of the epidemic, thus offering solutions that are not very accurate. Thus, to overcome these limitations, we capitalize on recent advancements in neural ODEs. Specifically, we parameterize the dynamics of the individual-based SIR model using a neural network. We compute a vector for each node v_i of the network, but we still require the ODEs of Equation 6.3 to hold (this time $\mathbf{S}_h, \mathbf{I}_h, \mathbf{R}_h \in \mathbb{R}^{n \times d}$ are matrices where nodes' representations are stored in their rows). The output of the network is computed using a black-box differential equation solver.

We next give more details about the proposed model. Let $\mathbf{s}^{(0)}, \mathbf{i}^{(0)}, \mathbf{r}^{(0)} \in \mathbb{R}^n$ denote the initial conditions of the SIR instance. Hence, $\mathbf{s}^{(0)}, \mathbf{i}^{(0)}, \mathbf{r}^{(0)}$ are binary vectors and a value equal to 1 in the i-th component of those vectors denotes that node v_i is in the corresponding state of SIR. Note that $\mathbf{s}^{(0)} + \mathbf{i}^{(0)} + \mathbf{r}^{(0)} = \mathbf{1}$ where $\mathbf{1}$ is the n-dimensional vector of ones. Therefore, each node initially belongs to exactly one of the three states of SIR. Those representations of the nodes are passed on to a fully-connected layer followed by the ReLU activation function,

and are thus transformed into vectors of dimension d (i. e., 0 and 1 integers are mapped to d-dimensional vectors), as follows:

$$\begin{aligned}\mathbf{S}^{(0)} &= \mathrm{ReLU}\big(\mathbf{s}^{(0)}\mathbf{W}_0 + \mathbf{b}_0\big)\\ \mathbf{I}^{(0)} &= \mathrm{ReLU}\big(\mathbf{i}^{(0)}\mathbf{W}_0 + \mathbf{b}_0\big)\\ \mathbf{R}^{(0)} &= \mathrm{ReLU}\big(\mathbf{r}^{(0)}\mathbf{W}_0 + \mathbf{b}_0\big)\end{aligned}$$

where $\mathbf{W}_0 \in \mathbb{R}^{1\times d}$ the weight matrix and $\mathbf{b}_0 \in \mathbb{R}^d$ the bias term. Thus, three vectors are associated with each node and each vector corresponds to one of the three states of the SIR model. These vectors correspond to the rows of three matrices $\mathbf{S}^{(0)}, \mathbf{I}^{(0)}, \mathbf{R}^{(0)} \in \mathbb{R}^{n\times d}$.

Then, these representations are fed to an ODE solver. The solver iteratively updates the representations of the nodes stored in matrices $\mathbf{S}^{(t)}, \mathbf{I}^{(t)}, \mathbf{R}^{(t)} \in \mathbb{R}^{n\times d}$ for $t \in \{1, \ldots, T\}$. In each iteration of the solver, first the representations of the previous iteration are further transformed using a fully-connected layer followed by the sigmoid activation function $\sigma(\cdot)$. Formally, the following updates take place:

$$\begin{aligned}\mathbf{S}_h^{(t)} &= \sigma\big(\mathbf{S}^{(t)}\mathbf{W}_h + \mathbf{b}_h\big)\\ \mathbf{I}_h^{(t)} &= \sigma\big(\mathbf{I}^{(t)}\mathbf{W}_h + \mathbf{b}_h\big)\\ \mathbf{R}_h^{(t)} &= \sigma\big(\mathbf{R}^{(t)}\mathbf{W}_h + \mathbf{b}_h\big)\end{aligned}$$

where $\mathbf{W}_h \in \mathbb{R}^{d\times d}$ the weight matrix and $\mathbf{b}_h \in \mathbb{R}^d$ the bias term. Then, the representations are re-updated based on the system of ODEs in Equation 6.3. We need to mention that training the model requires performing backpropagation through the ODE solver. Even though differentiating through the operations of the forward pass is straightforward, it incurs a high memory cost and it also introduces numerical error. Following recent advancements in the field of implicit differentiation [34], we treat the ODE solver as a black box, and compute gradients using the adjoint sensitivity method [198] which solves a second, augmented ODE backwards in time. This approach is computationally attractive since it scales linearly with problem size and has low memory requirements, while it also explicitly controls numerical error.

Once the solver has finalized its computations, the representations that correspond to the problem's discrete time steps are fed into a multi-layer perceptron (consisting of two fully connected layers) which for each node and time step outputs a 3-dimensional vector. The components of this vector correspond to the three states. Finally, the softmax function is applied to all those 3-dimensional vectors, and the emerging values can be thought of as the probabilities that a specific node belongs to state S, I or R in a given time step. These probabilities are then compared to the ground-truth probabilities that emerge from the simulations to compute the error. A high-level overview of the proposed model is given in Figure 6.3.

It is interesting to note that the update scheme of the ODE solver is related to a family of graph neural networks known as message passing neural networks [94]. These models employ a message passing procedure where they update the representation of each node by

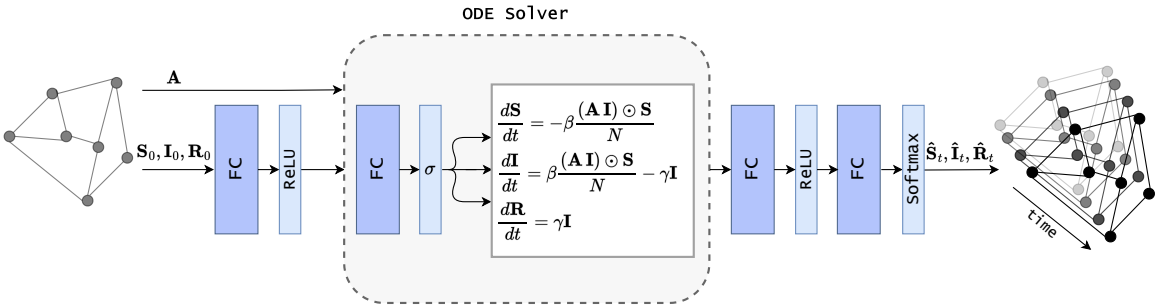


Figure 6.3: Overview of the proposed GN-ODE architecture.

aggregating information from its neighborhood. The matrix multiplication $\mathbf{A}\,\mathbf{I}$ performed by the solver to update the states of the nodes can be seen as a form of message passing. Indeed, for each node, the output of this operation produces a vector that aggregates the representations of state I of its neighbors. Then, the emerging representations are multiplied in an element-wise manner with $\mathbf{S}$. Therefore, it is evident that message passing models naturally emerge in different applications, and this perhaps justifies why these models have demonstrated great success in several problems.

6.2.3 *Experimental Evaluation*

In this section, we evaluate the proposed GN-ODE model on several real-world datasets. We first present the employed datasets, the baselines and other experimental details, and then, we present and discuss the experimental results.

6.2.3.1 *Experimental Setup*

DATASETS. We perform our experiments on real-world networks that represent social networks and are derived from online social networking and communication platforms (all datasets are publicly available). Specifically, we experiment with the following network datasets: (1) *karate* that contains social ties among the members of a University karate club; (2) *dolphins* representing a social network of bottlenose dolphins; (3) *fb-food* and (4) *fb-social* which represent the food page network of Facebook and private messages sent on a Facebook-like platform at UC-Irvine, respectively; (5) *openflights* that contains ties between two non-US-based airports and is downloaded from `Openflights.org`; (6) *Wiki-Vote*, a network created by all the voting data between administrators of Wikipedia; (7) *Enron*, an e-mail communication network; and (8) *Epinions*, an online social network created from the product review website `Epinions.com`. More details about the datasets are given in Table 6.1. The datasets are publicly available and can be derived from the following sources: *Wiki-Vote*, *Enron*, *Epinions* are available in `https://snap.stanford.edu/data/` and the rest five datasets in `https://networkrepository.com/` [212].

Table 6.1: Statistics of the 8 datasets that were employed in this study. All networks are undirected and are reduced to their largest connected component.

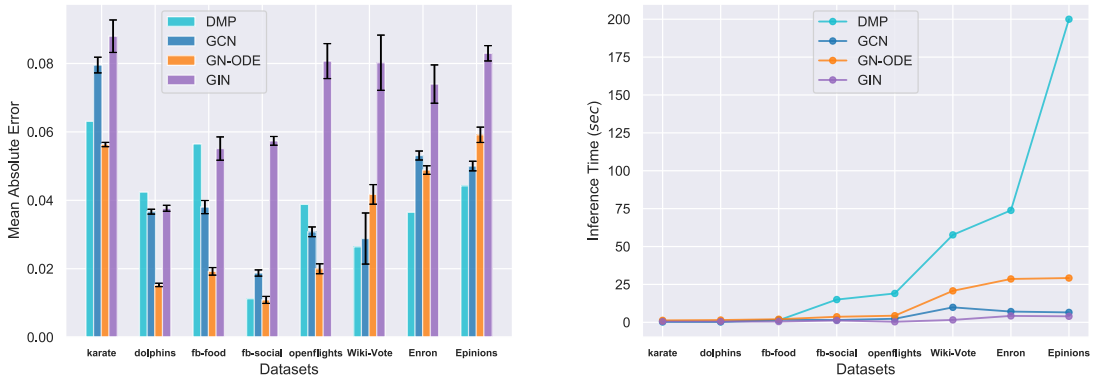
Dataset	karate	dolphins	fb-food	fb-social	openflights	Wiki-Vote	Enron	Epinions
#nodes	34	62	620	1,893	2,905	7,066	33,696	75,877
#edges	78	159	2,102	13,835	15,645	100,736	180,811	405,739
Transitivity	0.256	0.309	0.223	0.057	0.255	0.125	0.085	0.066
Density	0.1390	0.0841	0.0110	0.0077	0.0037	0.0040	0.0003	0.0001
Max. degree	17	12	134	255	242	1065	1383	3044

BASELINE MODELS. In all experiments, we compare the proposed model against three baseline methods, namely Dynamic Message Passing (DMP) [127, 167], Graph Convolution Network (GCN) [134] and Graph Isomorphism Network (GIN) [275]. DMP is an algorithm for inferring the marginal probabilities of stochastic spreading processes on networks. Under the individual-based SIR process, DMP is exact on trees and asymptotically exact on locally tree-like networks, while its complexity is linear in the number of edges and spreading time steps. Note that DMP is not a machine learning approach, but a purely combinatorial method. GCN and GIN are two well-established graph neural networks that have been recently applied with great success to different problems. Both models belong to the family of message passing neural networks. These architectures recursively update the representation of the nodes of a graph by aggregating information from the nodes' neighborhoods.

HYPERPARAMETERS. In order to select the combination of hyperparameters that leads to the best performance for each deep neural network architecture (GCN, GIN and GN-ODE), we performed grid search on a set of parameters and selected those that achieved the lowest error in the validation set. We chose learning rate from $\{0.0001, 0.001, 0.01\}$, batch size from $\{2, 4, 8, 16, 32, 64, 128\}$ and hidden dimension size for the trainable layers from $\{16, 32, 64, 128, 256, 512\}$. For larger datasets such as Wiki-Vote, Enron, Epinions we only tested the combinations of batch size and hidden dimension size that could fit into the memory of a single GPU (NVidia Quadro RTX 6000). We used the mean absolute error as our loss function and trained each architecture for 500 epochs. To make predictions, we used the model that achieved the lowest loss in the validation set. For the ODE solver in the case of GN-ODE, we used Euler's method with a step size equal to 0.5. The ground-truth values $\mathbf{s}^{(t)}, \mathbf{i}^{(t)}, \mathbf{r}^{(t)}$ were extracted after performing 10^4 simulations for 20 time-steps.

EVALUATION METRIC. We measure the mean absolute error (mae) across all nodes of all test instances, states and time steps. More specifically, the error is computed as follows:

$$mae = \frac{1}{3NnT} \sum_{i=1}^{N} \sum_{j=1}^{n} \sum_{t=1}^{T} \sum_{s \in \{S,I,R\}} |y_{i,j,t,s} - \hat{y}_{i,j,t,s}|$$



(a) (b)

Figure 6.4: (a) Mean absolute error (lower is better) achieved by the different approaches on the test set of datasets consisting of instances of a single network structure. The values of β and γ for the different network instances are sampled randomly. (b) Comparison of the inference time (in sec) of the different approaches on the test set of the considered datasets.

where N denotes the number of test samples, n the number of nodes of the graph (a.k.a., network), and T the number of time steps (i. e., 20 in our setting). Furthermore, $y_{i,j,t,s}$ denotes the probability that node j of the i-th test sample is in state s in time step t, and $\hat{y}_{i,j,t,s}$ the corresponding predicted probability.

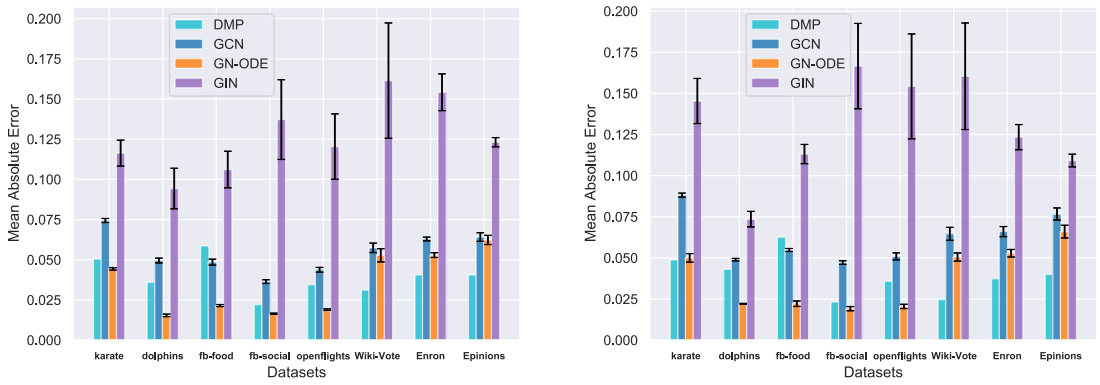
6.2.3.2 *Results*

We next present the experimental settings and the performance of the different models in different scenarios.

6.2.3.3 *Spreading Prediction on a Single Network*

WITHIN DISTRIBUTION PERFORMANCE. In the experimental results that follow, we investigate whether the different approaches can accurately estimate the spreading results of the individual-based SIR model. In these experiments, all approaches are trained and evaluated on instances of a single network.

To evaluate the performance of the different approaches, for each dataset, we created 200 samples by applying different instances of the SIR epidemic model to each network dataset. For each instance, we choose the values of hyperparameters β and γ of SIR randomly from $[0.1, 0.5]$ with uniform probability. This range for the hyperparameters is chosen so as to form a realistic model and to evaluate how useful each method could be in a real-world scenario [136]. We also choose two nodes randomly with uniform probability and set them to be in the infected state I, while the rest of the nodes are set to be in the susceptible state S. To estimate the marginal probabilities, we perform $10,000$ simulations, each running for 20



(a) Test Loss for β values

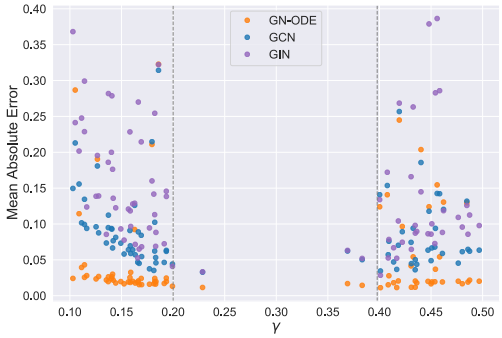
(b) Test Loss for γ values

Figure 6.5: Mean absolute error (lower is better) achieved by the different approaches on the test set of datasets consisting of instances of a single network structure. Most test instances have emerged from values of diffusion parameters β and γ different from those of training instances. Figures (a) and (b) illustrate the performance of the different approaches for out of distribution values of β and γ, respectively.

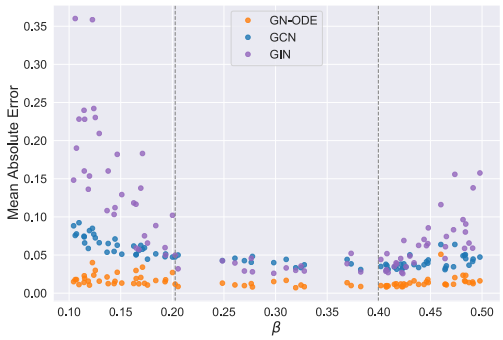
spreading time steps. The 200 samples were split into training, validation, and test sets with a $60:20:20$ split ratio, respectively.

Figure 6.4a illustrates the performance of the different methods. Note that each experiment is repeated 5 times, and Figure 6.4a illustrates the average mean absolute error along with the corresponding standard deviation. We observe that on most datasets, the proposed model outperforms the baselines. More specifically, GN-ODE is the best-performing approach on 5 out of the 8 benchmark datasets. On some datasets, the proposed model outperforms the baselines with wide margins. For instance, on the dolphins and fb-food datasets, it offers absolute improvements of 58.37% and 49.41% in mae, respectively, compared to the best competitor, respectively. Furthermore, on several datasets the proposed GN-ODE model achieves very low values of error, i. e., less than 0.02 which demonstrates that it can provide accurate predictions. With regards to the baselines, DMP and GCN perform comparably well in most cases, while GIN is the worst-performing method. This is an interesting result since GIN is known to be more powerful than GCN in terms of distinguishing non-isomorphic graphs [275]. However, it turns out that in this task, we are more interested in estimating distributions of the states of the neighbors of a node than the exact structure of the neighborhood.

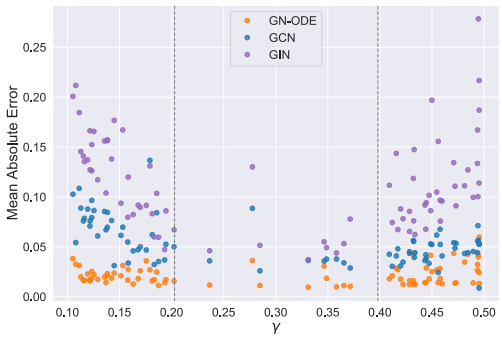
For the same set of experiments, we also demonstrate in Figure 6.4b the inference time of each model on the test set of the considered datasets. We can clearly observe that the inference time increases along with the size of the input networks. More specifically, the employed models show equivalent computational costs for relatively small networks such as the karate, dolphins and fb-food, where the proposed GN-ODE is slightly slower (a few seconds) compared to the fast, in those cases, DMP. However, DMP becomes dramatically



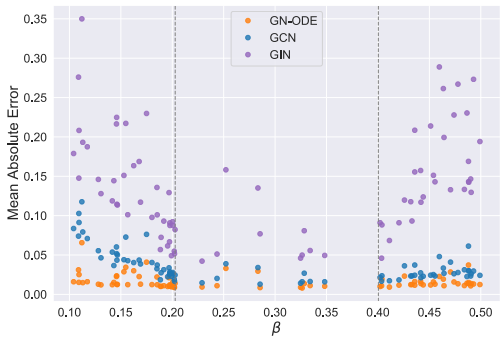
(a) (karate) Out of distribution γ values



(b) (dolphins) Out of distribution β values



(c) (fb-food) Out of distribution γ values



(d) (fb-social) Out of distribution β values

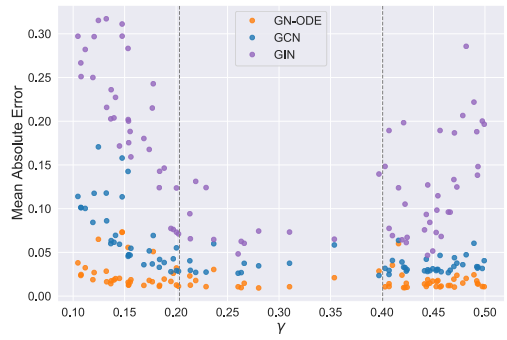
Figure 6.6: Mean absolute error (lower is better) achieved by the different approaches on each test sample (i. e., network) of a given dataset. Results provided for the following four datasets: karate, dolphins, fb-food, fb-social. Each figure is associated with one dataset and one parameter (β or γ). The out of distribution generalization performance of the different methods is evaluated. Test samples that appear in between the two dotted vertical lines correspond to test instances where values of β and γ were sampled from the same distribution as that of training instances. The rest of the samples correspond to instances where values of β and γ were sampled from different distributions than those of training instances.

slower on larger datasets, such as Epinions, where it suffers by an approximately ten times greater inference cost compared to the proposed model. This behaviour demonstrates the necessity of the development of accurate models that remain scalable on large datasets and can be employed as alternatives to algorithms such as DMP. We also observe that the vanilla GNNs (GCN and GIN) remain quite fast even for larger networks. The inference time of the proposed GN-ODE model becomes relatively worse than that of the GNN variants, especially on the three larger networks, which can be attributed to the intermediate step used for the computations of the ODE solver, as explained in Section 6.2.2.

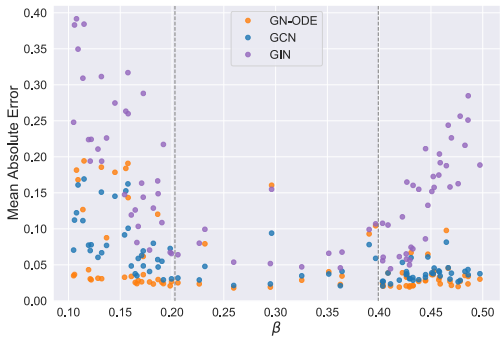
We also provide a visualization of the evolution of the diffusion process on the karate dataset (i. e., probabilities of infection for all the nodes of the network) in the Appendix.

OUT OF DISTRIBUTION GENERALIZATION. Neural network models might fail to generalize to unseen data. Thus, we also perform some experiments where we study whether the different methods can accurately predict the spreading process over instances of the network that are different from the ones the methods were trained on. To achieve this, we add to the test set of a dataset, instances that emerged from values of β and γ that fall outside of the range of values used to train the model. In order to create the dataset, the different values of β and γ (from the 200 instances described above) were divided into 5 bins. Then, 80 instances sampled from bins $2, 3$ and 4 constitute the training set. The validation and the test set both consist of some instances from bins $2, 3$ and 4 and some instances from bins 1 and 5. Overall, the validation set contains 40 samples, while the test set contains 80 samples. Note that the training set contains instances sampled exclusively from bins $2, 3$ and 4, while the test set mostly consists of samples from bins 1 and 5. Therefore, test instances can be considered as sampled from a different distribution compared to those of the training set.

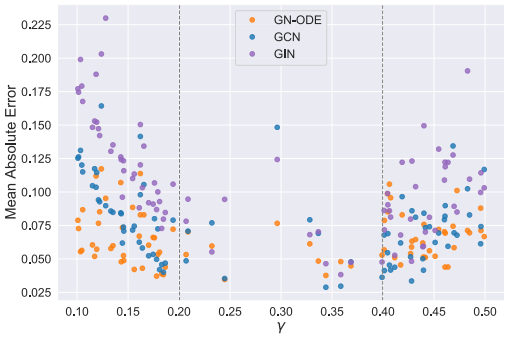
Figure 6.5 illustrates the performance of the different approaches on the eight datasets. We again report the mean absolute errors across all nodes of all test instances, states and time steps. The mean absolute errors are averaged over 5 runs. We observe that for out of distribution values of both β and γ, the GN-ODE model outperforms both GCN and GIN on all datasets. We can also see that the performance of the proposed architecture degrades on the largest networks (i. e., Wiki-Vote, Enron and Epinions) where DMP is the best-performing approach. GIN yields the worst results and achieves much higher values of mae than the rest of the methods. This might be due to the neighborhood aggregation method that this neural network model utilizes (i. e., sum function). Figure 6.6 and Figure 6.7 illustrate the error of the considered approaches for the different instances of each dataset (for clarity, we provide for each dataset a single plot illustrating the generalization performance with respect either to β or γ). The vertical lines distinguish bins 1 and 5 (those from which test samples emerged) from bins $2, 3$ and 4 (those from which training instances were sampled). The results indicate that the proposed GN-ODE model is relatively robust. In most cases, its generalization performance is similar to its within distribution performance, i. e., the obtained error for samples from bins $2, 3$ and 4 is similar to the error for samples from bins 1 and 5. On the other hand, the two baseline architectures achieve lower levels of performance



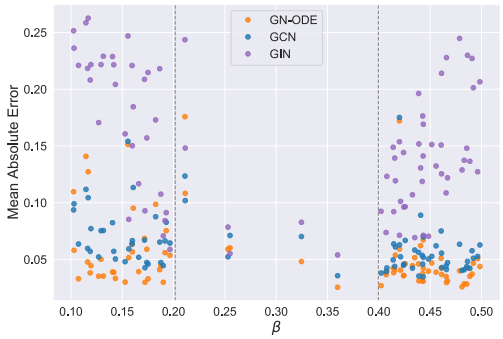
(a) (openflights) Out of distribution γ values



(b) (Wiki-vote) Out of distribution β values



(c) (Epinions) Out of distribution γ values



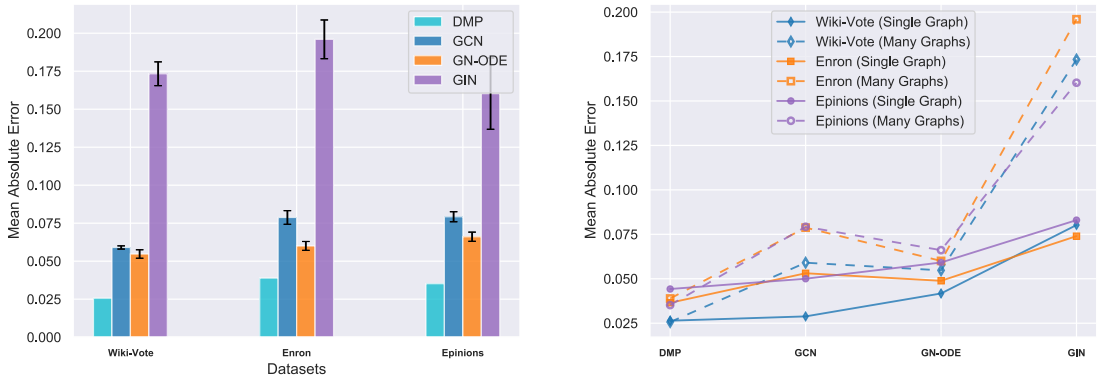
(d) (Enron) Out of distribution β values

Figure 6.7: Same as Figure 6.6. Results provided for the rest of the datasets: openflights, Wiki-vote, Enron, Epinions.

on instances where values of β or γ are different from those the models were trained on. It is interesting to note that GIN yields much higher errors for the out of distribution samples, thus the results suggest that this neural network model might not be useful in real-world scenarios. With regards to the proposed model, as already mentioned, it achieves very good levels of generalization performance on the karate, dolphins, fb-food, fb-social, and openflights datasets, while a decrease in performance occurs on the largest datasets, namely Wiki-Vote, Epinions, and Enron. Still, GN-ODE consistently outperforms the two baseline neural architectures, while the obtained errors are not prohibitive.

6.2.3.4 *Spreading Prediction on Multiple Networks*

We are also interested in investigating whether a model that is trained on one or more networks can generalize to networks different from the ones it is trained on. Thus, we designed a series of experiments where the model was trained on a subset of the datasets shown in Table 6.1, and evaluated on some dataset that was not contained in that subset.



(a) (b)

Figure 6.8: (a) Mean absolute error (lower is better) on the test set achieved by the different approaches when trained on instances of small networks and evaluated on instances of a large unseen network (as mentioned in the x-axis). (b) Comparison of the mean absolute error achieved by the different approaches when trained on instances of small networks and evaluated on instances of larger unseen networks (Many Graphs) vs. when both trained and evaluated on instances of the larger networks (Single Graph).

These experiments are of very high significance since for the model to be useful in real-world problems, it is necessary that it can generalize to unseen networks. That would suggest that a model trained on some networks could be then applied to any network.

More specifically, we investigated whether models trained on karate, dolphins, fb-food fb-social, and openflights networks can accurately predict the spreading process over Wiki-Vote, Enron, and Epinions. In the case of Wiki-Vote, training was performed on instances of the dolphins, fb-food, fb-social, and openflights networks. We used 45 instances of each of those networks, i. e., 180 training samples in total. The validation and test sets contain 60 instances of the Wiki-Vote network each. In the case of the Enron and Epinions networks, training was performed on instances of the dolphins, fb-food, fb-social, openflights, and Wiki-Vote networks. Specifically, 36 instances of each of those networks were generated giving rise to 180 training samples in total. The validation and test sets both contain 60 instances of the considered network (i. e., Enron and Epinions). With regards to the rest of the hyperparameters, for each instance, β and γ of SIR were randomly chosen from $[0.1, 0.5]$ with uniform probability. Furthermore, for each instance, two nodes were randomly chosen with uniform probability and were set to be in the infected state I, while the rest of the nodes were set to be in the susceptible state S. To estimate the marginal probabilities, we perform $10,000$ simulations, each running for 20 spreading time steps.

Figure 6.8a illustrates the performance of the different methods on the three datasets. Once again, we report the mean absolute error achieved by each method, where we compare the predicted probabilities that nodes belong to the different states against those that emerged from the simulations. Each experiment is repeated 5 times, and besides the average mean

absolute error, also standard deviations are provided. We observe that in all three experiments, the proposed GN-ODE model outperforms the baselines, i. e., the GCN and GIN models. Thus, the results suggest that the proposed model can generalize better to unseen networks than the baseline models. The GIN model fails to accurately predict the probabilities that nodes belong to the different states of SIR, thus achieving very high values of mean absolute error. On the other hand, the proposed model and GCN make more accurate predictions, and seem to be more robust since they can generalize to unseen networks.

We also investigate how the performance of the models on those three datasets compares to their performance when they are trained directly on them (i. e., results of Figure 6.4a and Figure 6.8a combined). Figure 6.8b shows the results. It is clear that all models achieve better performance when instances of the network in the test set also occur in the training set. However, for the proposed model GN-ODE, we can see that the difference in performance is very small, while for the baseline models it is much higher. In the case of GIN, there is a dramatic increase in mean absolute error when the test set contains unseen networks. In the case of GCN, the increase is not that large, but still significantly greater than that of the proposed model. Overall, the results indicate that the GN-ODE model is very robust and can achieve good levels of performance even on unseen data. DMP on the other hand, can be directly applied to the test set of the unseen networks, achieving comparably better performance than the rest methods, as shown in Figure 6.8a, with the disadvantage however of being significantly slower during inference and impractical to scale to large networks (as shown in Figure 6.4b).

6.2.4 *Conclusion*

The analysis and modeling of spreading processes have been a key issue in different fields, including physics, biology and computer science, among others. For instance, predicting the course of an epidemic is of paramount importance for governments and policymakers. Indeed, such predictions provide valuable information for adapting policies and protocols such that the spread of the disease is controlled. Mathematical models have traditionally been used to describe the underlying dynamics of different spreading phenomena. However, to capture the exact dynamics of most spreading processes, we need more realistic models which incorporate more parameters and are thus very complex. For example, most social and biological contagion processes require incorporating each individuals' contact patterns in the mathematical model. Unfortunately, most real-world systems exhibit very complex connectivity patterns, thus leading to models that are hard to solve. It turns out that even the well-established SIR model on general networks is computationally intractable. Therefore, there is a need for approximation techniques which can efficiently predict the model's output.

In the past years, machine learning has emerged as a promising tool for studying physical systems and has shown great potential in providing approximate solutions to complex problems. Even though machine learning approaches can learn useful patterns directly from empirical data, recently there is a trend towards embedding the knowledge of any physical laws that govern a given dataset in the learning process. In our setting, these physical laws

are described in the form of a system of ODEs. More specifically, to enhance the effectiveness of neural network models specifically for the SIR problem on networks, we incorporate knowledge on the evolution functions of the S,I,R using an approximate system of ODEs. Combining ODE solvers with neural networks has recently become an area of increasing interest for the research community. In this study, we are the first to apply task-specific neural ODEs for the SIR model, intending to advance the learning capabilities of a standard neural network model.

The obtained empirical results on a single network structure indicate that the proposed architecture outcompetes the baselines on almost all datasets and parameter settings. We need to stress that for a model to be useful, it is necessary to achieve high levels of performance even on instances of the problem that are different from the ones the model was trained on. This is, for instance, the case for most real-world datasets where training data might be available only for specific values of parameters (i. e., β and γ) and it is thus crucial for the model to learn to generalize to previously unseen parameter values. Therefore, a great deal of emphasis was placed on testing the generalization power of the models on parameters' combinations that are not seen during training. This is because neural networks are often prone to overfitting which results in a dramatic decrease in their performance when the parameter distribution of the test set is different from that of the training set. The obtained results demonstrate that the proposed model achieves better performance than GCN and GIN for all (out of distribution) combinations of β and γ, while it is competitive with DMP in the case of larger networks (Wiki-Vote, Enron, Epinions), where DMP achieves state-of-the-art performance at the cost of being significantly slower in terms of inference time. Besides the generalization performance with respect to the values of β and γ, many applications require a model to be able to generalize to unseen networks. For instance, one might employ a model trained on small networks (for which ground-truth labels might be available) to make predictions for larger networks. In this setting, algorithms like Dynamic Message Passing, which are directly calculated on the test data (in our case the final large network) cannot be implemented in a way to achieve faster inference. Thus, neural network architectures that can generalize well to unseen networks can be very useful for several real-world applications. Our empirical results demonstrate that graph neural networks and especially the proposed GN-ODE model are quite successful in this task, and can thus serve as a promising approach for the modeling of spreading processes on complex networks.

Overall, even though there exist mathematical models tailored to the specificities of complex spreading phenomena, these models are usually analytically intractable. This is more evident in the case of modern large-scale applications where large networks are involved and simulation methods are inapplicable. In such settings, there is a need for approximate techniques which can accurately predict the output of the mathematical models. In this work, we have introduced and evaluated a neural network model that is robust and scalable to large networks. The model employs prior knowledge in the form of a system of ODEs in order to increase the correctness of the function approximation. Hence, the model can make more accurate predictions and generalize well even with a small number of training examples. We believe that the proposed model can serve as a useful addition to the list of traditional

approximation approaches and motivate the further development of deep learning methods for capturing the dynamics of physical systems.

7 DISCUSSION

To conclude this thesis, we summarize our key findings and contributions. We also provide further ideas for extensions and future research directions for robust DL modeling of time series data.

THE TRAP OF LOSS FUNCTIONS AND EVALUATION METRICS. Regression problems treated with deep neural networks are traditionally optimized under measures that capture the deviation of predicted values from target values, e. g., MSE. Time series forecasting falls in this category of problems, in the sense that observations $x_t \in \mathbb{R}$ and the goal is to predict the future values $x_{t+1:t+h}$ for a specific horizon h. However, time series data often face distortions due to flawed measuring devices as well as cyclical and seasonal patterns of non-fixed periodicities, or lack of correlated variables and external knowledge that affect their evolution. In this sense, optimizing under L_p norms, while a straightforward solution, tends to overfit the data in the presence of noisy observations and abrupt changes. For instance, MSE penalizes stronger predicted values that deviate significantly from the expected ones. At the same time, the difference between the value of an observation and the value of the previous observation might be small, since observations close in time are often close in value. This makes MSE prone to learning to replicate past values for predictions [142]. The need for measures alternatives to L_p norms has recently led to the development of new differentiable loss functions, including differentiable variants of DTW [57, 178]. DTW-based losses capture meaningful statistical properties of time series such as the shape but ignore the temporal localization of changes. To tackle this, DILATE loss [149], combines soft-DTW variant with penalized TDI. However, such measures require extra preprocessing steps to become differentiable, while the time complexity of DTW variants becomes critical for large subsequences,i. e., long-horizons in multi-step forecasting. At the same time, TDI is calculated based on the DTW path, thus even DILATE loss remains sensitive to noise and scale of the observations.

In the proposed MSE with a regularization term, as presented in Section 3.2, we aim to tackle poor optimization when training with the standard MSE loss. Our proposed loss is a straightforward and mathematically solid extension of MSE, that remains scalable for large forecasting horizons. It also improves metrics among several real-world datasets including noisy observations and fewer informative correlated variables. The evaluation metrics we propose, e. g., accuracy of changes, highlight the need to capture shape rather than the point-wise deviations solely. In future work, we also aim to identify and formally define the exact properties (e. g., signal-to-noise ratio (SNR)) under which forecasting models are sensitive to the introduced phenomenon of 'mimicking'. At the same time, it would be interesting to highlight similar phenomenons when optimizing forecasting models with probabilistic

loss functions, following the work of [150] where the idea of DILATE loss is extended to probabilistic forecasting.

Probabilistic forecasting, which is based on probabilistic loss functions, has shown potential in datasets of several correlated variables that describe similar time series, e. g., the Electricity dataset that contains the electricity consumption of hundreds of clients (employed in experiments of Section 3.2 and Section 5.2). However, probabilistic forecasting models can face challenges when dealing with data of complex dependencies and variables that follow significantly different patterns. At the same time, selecting the appropriate probability distribution to represent the uncertainty in forecasts can be challenging. Often, there is no one-size-fits-all distribution, and the choice can impact the forecast quality.

To summarize, the field of time series forecasting lacks robust losses and evaluation metrics that apply to different large and real-world time series datasets. While most studies adopt MSE for training and evaluation purposes or task-specific metrics (e. g., profit for stock prediction), following some unified training and evaluation principles for forecasting is crucial for improved generalization to different datasets and interpretation of the quality of the forecasts.

CONTINUOUS MODELS: TRANSFORMING DISCRETE APPROACHES. Sequential models like RNNs and CNNs have gained prominence in deep learning for time series data due to their remarkable success in handling sequential data, e. g., text. However, the continuous nature of time series data necessitates models that can capture temporal dependencies and trends effectively, while modeling of discrete sequences, such as text data is more concerned with understanding the context relationships between discrete elements (e. g., words or characters). The discrepancy between the continuous nature of time series data and discrete architectures becomes more challenging in terms of irregular sampling. In situations characterized by fixed temporal intervals between observations, the conventional notational framework of standard sequential modules (e. g., RNNs, CNNs) remains applicable. However, this framework becomes inadequate when confronted with variable time intervals between temporal observations. Continuous-time models like Neural ODEs [34, 132, 214] offer an intuitively appealing approach by implicitly accounting for time. Nevertheless, their adoption can be challenging due to their relatively high computational demands and sensitivity to noisy data, limiting their scalability to large, real-world datasets. Meanwhile, state-of-the-art methods for irregular sampling [31, 234] rely on RNNs, which face optimization challenges when dealing with large time series inputs. This constraint could, for instance, restrict their utility to tasks that prioritize short-term dependencies over long-range forecasting.

Our objective is to address the research gap in irregularly sampled time series modeling, primarily dominated by RNN-based approaches, by extending conventional CNNs. In doing so, we leverage the parallelization and fast training advantages inherent in CNNs, as presented in Section 4.2. Notably, our proposed model, TPCNN, incorporates time functions that facilitate the interpretation of series evolution, while enhancing its effectiveness. However, it has demonstrated sensitivity to overfitting, which we attribute to the initialization of the kernel relying on learnable time functions. Despite our comprehensive examination of

parameter initialization methods for the convolutional kernel (explained in the Appendix A), there remains room for improvement. To address this challenge, one could intend to explore variational learning for the autoencoder component in the context of imputation. Additionally, considering the substantial class imbalance in the datasets under study, training stability could be enhanced by optimizing parameters jointly for the imputation and classification tasks. Initial experiments with the variational autoencoder revealed that the model tends to underfit during training. However, it appears that the sensitivity of kernel initialization methods holds the key to enhancing this aspect.

Exploring the application of continuous-time models, that incorporate time information implicitly and explicitly, to regularly sampled data can also be a worthwhile research direction [129]. It offers the potential to leverage the benefits of continuous-time modeling, such as capturing fine-grained dynamics, particularly when dealing with data of complex temporal dependencies and patterns. Additionally, there is a growing interest in exploring alternative methods to the dominant Neural ODEs model, that retain the advantages of continuous-time modeling but are less computationally expensive.

QUANTIZING CONTINUITY: DISCRETE MODELS FOR CONTINUOUS DATA. In contrast to the concept discussed in the preceding section, an intriguing avenue of investigation involves the development of algorithms for discretizing time series data. This discretization process can be founded upon inter-variable or inter-temporal dependencies within the data, with the primary objective of extracting latent embeddings that can furnish a more abstract input representation for deep neural networks. Such an abstraction facilitates enhanced modeling, especially in scenarios characterized by noise, strong correlations among variables, and long-term temporal dependencies. A direct and effective approach to creating a discrete structure capable of capturing dependencies from time series involves the construction of an underlying graph. Subsequently, neural networks like GNNs can be employed to process and generate informative embeddings derived from these correlations, which can, in turn, be harnessed for various subsequent tasks.

Nonetheless, there are several critical decisions to consider when applying the aforementioned approach, including the method used for constructing the graph, the specific GNN modules employed, and the choice of architecture for the subsequent tasks. Existing approaches for the challenging task of time series forecasting [135, 227, 272], capture parameterized inter-variable correlation graphs in a differentiable way and optimize them jointly with the forecasting module. In our novel approach, TimeGNN presented in Section 5.2, we introduce nodes that represent individual time steps within subsequences of the multivariate series, rather than nodes corresponding to the correlated variables. This deliberate choice explicitly encourages the formation of dynamic temporal graph representations, aligning more naturally with the sequential and chain-like nature of the series data [41, 42]. In contrast to graph-based baseline methods, our approach relies solely on a straightforward spatial network to extract information from the dynamically generated inter-temporal graphs, followed by a feedforward network for forecasting from the produced embeddings. Our method performs favorably compared to standard sequential approaches and, in certain cases, demonstrates

performance that is closely competitive with state-of-the-art graph-based methods. At the same time, it is significantly faster which is crucial for large time series datasets. Enhancing the graph-based feature extraction by incorporating memory mechanisms between the underlying dynamic graphs and extending to a more intricate forecasting architecture, while keeping the framework computationally efficient, holds promise for future improvements on TimeGNN.

In summary, the utilization of graph-structured learning in the context of time series, to extract informative embeddings through GNNs, represents a nuanced yet highly intricate approach to discretely model continuous data. Expanding upon this concept to address more complex scenarios like irregularly sampled or noisy data, and subsequently evaluating these methods for tasks like data generation, remains a relatively underexplored area of research with great potential.

DECIPHERING THE SIGNIFICANCE OF PRIOR KNOWLEDGE. Understanding the role of prior knowledge in time series modeling is of paramount importance in various domains, ranging from economics to natural sciences. One compelling example of this significance lies in the domain of physics-based knowledge for dynamical systems. In this context, the incorporation of prior knowledge about the underlying dynamics of a system can substantially enhance the predictive performance of time series models. Using known physical laws to constrain the modeling process provides valuable constraints and insights that help improve model accuracy, especially in cases where data may be limited or noisy.

Following this idea in Section 6.2, we study a particular case of a dynamical system and more specifically epidemic spreading on networks of contacts under the SIR compartmental model. A data-driven approach for modeling epidemic spreading, exemplified by the analysis of COVID-19 data, has become increasingly prevalent [191]. However, relying solely on historical data without incorporating any prior knowledge or constraints related to the spreading process can pose challenges. In such cases, the robustness of the applied method may be compromised, potentially limiting its ability to generalize effectively to unseen data, especially when trained on relatively small datasets. Therefore, we introduce GN-ODE, a continuous-time model for epidemic spreading on networks, based on message-passing Neural ODEs for the SIR model. In our research, we introduce innovative experimental configurations to evaluate the performance of GN-ODE. These setups involve out-of-distribution evaluations, where we vary infection hyperparameter rates in the test set to simulate different epidemic settings. Additionally, we assess GN-ODE's effectiveness by testing it on networks that were not part of its training data. These experiments provide valuable insights into the model's robustness and its ability to generalize to diverse epidemic scenarios and network structures. Our experimental results strongly affirm the substantial impact of incorporating prior knowledge about the dynamical system that underlies time series data.

To conclude, dynamical systems, as a subset of time series data, require a thoughtful approach that incorporates the underlying governing laws to derive meaningful predictions. Moreover, many time series datasets, typically constructed from historical observations, may adhere to underlying principles that are not fully captured by the temporal data alone. For

instance, stock prices are influenced by various factors including stock market trends, news events, and more. While deep learning, with its flexibility and data-driven approach, is often employed as a versatile tool for modeling time series data, it can fall short when dealing with datasets lacking prior or external knowledge about the underlying dynamics, resulting in poor generalization performance. Identifying whether benchmark datasets necessitate additional information or prior knowledge for specific applications represents a critical area for experimental exploration. Such an approach holds the potential to enable researchers and practitioners to perform thorough comparisons and evaluations of various deep learning models using standardized benchmark datasets across diverse applications.

UNLOCKING THE FUTURE: DEEP LEARNING FOR TIME SERIES AHEAD. Except for the aforementioned future ideas for increasing time series models' robustness, recent DL modeling techniques motivate several research directions for effectively capturing meaningful representations. Following the recent advances in text pre-trained embeddings and large language models (LLMs) [21, 202], the large amount of publicly available time series datasets describing different temporal phenomena currently motivates the development of large pre-trained models for their subsequent applications. Several research works have focused on extracting embeddings for time series using autoencoders. The concept of pre-training has been introduced in time series forecasting for weight initialization of the forecasting architecture by employing an LSTM Autoencoder (AE) [215]. In time series generation, architectures based on Generative Adversarial Networks (GANs) [183, 280], probabilistic variants [71, 119] and masked autoencoders [293] have been proposed. However, contrary to Natural Language Processing (NLP) and text modeling, generative time series architectures are evaluated on the performance of the generated data on downstream tasks rather than the quality of the learnable latent embeddings of the autoencoders. Additionally, transformer-based architectures (i. e., using attention modules) that are the most dominant architectures in NLP, have not been adequately explored for time series pre-training and generation.

As already mentioned, time series statistical properties and their acquisition process pose significant challenges to the robustness of deep neural network architectures when applied to raw data. Since time series are affected by distortions (as presented in Section 2.3.1), investigating various types of distortions and their effects on different facets of representational learning is crucial. Such distortions, except for being explicitly modeled by neural networks based on their design or the employed loss function, can be incorporated as perturbations in the input to enhance pre-training in an autoencoder fashion, similar to text pre-training denoising autoencoder techniques [200] (e. g., token masking, sentence permutation). Recently, there has been a notable expansion of diffusion models [240] into various applications related to time series data, encompassing tasks such as time series forecasting [157, 207] and imputation [2, 166]. Diffusion models are a class of probabilistic generative models that estimate data distributions through a process of iteratively transforming and diffusing data points to generate new samples from the underlying distribution. Since diffusion models incorporate stochasticity and noise in their generative process, they are suitable for capturing and simulating the continuous, dynamic patterns and irregularities often found in various time

series applications. However, while their application on time series data for extracting robust representations remains limited and computationally expensive, there exists significant room for future contributions (e. g., application to time series generation, pre-trained embeddings).

Inspired by the ideas of invariant and equivariant transformations for images [50], sets [291] and some recent works in time series [53], another research direction for time series embeddings could focus on learning invariant or equivariant representations for a specific set of input perturbations for pre-training. Such networks can directly learn to map distortions to the input to the same or similar representations, based on the type of distortion, during training. A possible research direction is towards the development of denoising autoencoders [250] that handle time series perturbations explicitly via equivariant neural networks, which constitutes a novel work in the field of time series. In a relevant research direction, several studies have focused on contrastive learning for time series. Contrastive learning is a self-supervised deep learning technique [187] with no explicit labels, that aims to learn useful representations by contrasting similar and dissimilar pairs of data, thereby enabling the model to capture meaningful patterns and features within the data. Contrastive pre-training has been applied among time-based and frequency-based representations evaluated for time series classification [297] and among trend and seasonal representations evaluated for forecasting [267]. A more unified representational approach [290], performs contrastive learning in a hierarchical way over augmented context views (i. e., by timestamp masking and random cropping of the input series) and is evaluated on time series classification, forecasting and anomaly detection. However, relevant works in contrastive learning for time series follow different augmentation and consistency strategies to define positive and negative pairs and their evaluation is limited to a few specific datasets and downstream tasks. Challenging the capabilities of pre-training models on larger and more complex datasets (e. g., including irregular sampling, noise and long-term dependencies) and defining versatile evaluation metrics or tasks for the extracted representation is still an open field for experimentation and research.

The use of large pre-trained embeddings and universal representation methods has the potential to revolutionize the field of time series by enabling more accurate and efficient modeling of complex patterns and unlocking new insights from the data, that were previously unattainable. Moreover, pre-trained embeddings can provide a solid foundation for transfer learning and domain adaptation, enabling these models to be applied to new, previously unseen domains and accelerate the development of novel time series applications.

CONCLUSION. In conclusion, this thesis has systematically examined prominent challenges in robust deep learning modeling of time series data. By addressing a wide range of distortions and complexities inherent in time series data, our contributions have introduced novel techniques and architectures that enhance the generalization and employment of deep neural networks across diverse time series tasks. The potential for future research in this field is significant, as the demand for robust models capable of handling noisy, irregular, and high-dimensional time series data continues to grow. Further exploration could focus on advancing the interpretability of deep learning models, integrating domain-specific knowledge,

and harnessing cutting-edge generative models. Moreover, optimizing and scaling these models to accommodate increasingly large datasets is essential for real-world applications. In summary, the future of robust deep learning for time series holds great promise, with ongoing developments poised to contribute substantially to the progress of scientific and engineering fields, as well as industrial applications involving real-world time series data.

Part III

APPENDIX

A APPENDIX

A.1 NEURAL ODES FOR EPIDEMIC SPREADING - ADDITIONAL RESULTS & FORMULATIONS

A.1.1 *Individual-based SIR Model*

We next give more details about the considered individual-based SIR model. We consider a system Γ. In our setting, Γ corresponds to a complex epidemiological system, i. e., the SIR epidemiological model. There are 3^n different states in total (where n denotes the number of nodes of the network) and each state is denoted by Γ^α where $\alpha \in \{1, 2, \ldots, 3^n\}$. Then, the probability $P(\Gamma = \alpha)$ that the system is in state Γ^α is given by the master equations:

$$\frac{dP(\Gamma = \Gamma^\alpha)}{dt} = \sum_{\beta=1}^{3^n} \Big[R^{\beta\alpha}\, P(\Gamma = \Gamma^\beta) - R^{\alpha\beta}\, P(\Gamma = \Gamma^\alpha) \Big]$$

where $R^{\beta\alpha}$ denotes the transition rate from state Γ^β to state Γ^α. By solving these equations, we can obtain the complete evolution of the probabilities of the states of the stochastic system Γ. However, solving these equations is only feasible for very small networks.

We suppose that within the system Γ, there exist well-defined smaller systems (i. e., subsystems). Such a set of subsystems is formed by the individuals themselves. Let $P(X_i = S)$ denote the probability that node v_i is susceptible, $P(X_j = I \cap X_i = S)$ denotes the probability that node v_j is infectious and node v_i is susceptible. Probabilities and joint probabilities for the other states are defined in a similar manner. Then, the following system is an exact description of node probability dynamics for an SIR model on a network:

$$\begin{aligned}
\frac{dP(X_i = S)}{dt} &= -\beta \sum_{j=1}^{n} \mathbf{A}_{ij} P(X_j = I \cap X_i = S) \\
\frac{dP(X_i = I)}{dt} &= \beta \sum_{j=1}^{n} \mathbf{A}_{ij} P(X_j = I \cap X_i = S) - \gamma P(X_i = I) \\
\frac{dP(X_i = R)}{dt} &= \gamma P(X_i = I)
\end{aligned}$$

The above system is indeed exact, i. e., it gives the exact evolution of the probabilities of being susceptible, infectious or recovered during an epidemic. Unfortunately, it is not closed and

thus, has no solution. We can obtain a closed system if we assume statistical independence in the states of individuals:

$$\frac{dP(X_i = S)}{dt} = -\beta \sum_{j=1}^{n} \mathbf{A}_{ij} P(X_j = I)\, P(X_i = S)$$
$$\frac{dP(X_i = I)}{dt} = \beta \sum_{j=1}^{n} \mathbf{A}_{ij} P(X_j = I)\, P(X_i = S) - \gamma P(X_i = I)$$
$$\frac{dP(X_i = R)}{dt} = \gamma P(X_i = I)$$

The above approximate set of equations (from which our model is inspired) focuses on an individual level and can be employed to evaluate the evolution of complex epidemics on networks of individuals. The accuracy of the above system depends on how much the independence assumption used to derive it holds in practice. Previous studies have shown that the above system is less accurate than more complex models (e. g., pair-based models) [229]. Roughly speaking, the proposed approach uses a neural network architecture to refine the output of the above system.

A.1.2 *Comparison against System of ODEs*

The system of ODEs of Equation 6.3, which motivated the proposed GN-ODE model, can also be used to predict the spread of epidemics of networks as a function of time. More specifically, by solving the system for some initial values $\mathbf{s}^{(0)}, \mathbf{i}^{(0)}, \mathbf{r}^{(0)} \in \mathbb{R}^n$, we can obtain for each time step t a set of vectors $\hat{\mathbf{S}}^{(t)}, \hat{\mathbf{I}}^{(t)}, \hat{\mathbf{R}}^{(t)} \in \mathbb{R}^n$ that describe the nodes' states. Note that no trainable parameters are involved in this system. We compare in Table A.1 the proposed GN-ODE model against the solution of the system of Equation 6.3. To solve the system, we utilized the ODE-RK method. ODE-RK follows the implementation of the SciPy package[1] and solves the fixed system of ODEs with a Runge-Kutta solver of order 5(4) [79]. The results reported in Table A.1 highlight the poor performance of the approximate system when the representations that emerge at the different iterations of the solver are not refined by a neural network model. We can observe that for all datasets, the fixed system fails to capture the dynamics of the SIR process since it performs significantly worse compared to GN-ODE, and the rest of the methods of Figure 6.4.

A.1.3 *Visualization of the Spreading Process*

We provide a visualization of the evolution of the diffusion process on the karate dataset in Figure A.1. The results correspond to the experimental setting of Section 6.2.3.3 and within distribution hyperparameter selection. More specifically, we illustrate the probabilities of infection (i. e., probability of a node being in state I) for all the nodes of the network,

1 https://docs.scipy.org/doc/scipy/reference/integrate.html

Table A.1: Mean absolute error achieved by GN-ODE and simple fixed ODE system with Runge-Kutta solver, ODE-RK, on the test set of datasets consisting of instances of a single network structure. The values of β and γ for the different network instances are sampled randomly.

Dataset	**MODELS**	
	ODE-RK	**GN-ODE**
karate	0.09608	0.05631 ± 0.00062
dolphins	0.10653	0.01527 ± 0.00049
fb-food	0.19109	0.01924 ± 0.00111
fb-social	0.11061	0.01089 ± 0.00102
openflights	0.16087	0.02000 ± 0.00145
Wiki-Vote	0.12287	0.04173 ± 0.00287
Enron	0.16572	0.04885 ± 0.00125
Epinions	0.15917	0.05915 ± 0.00224

starting from a fixed initial set of infected nodes and fixed values of the transmission β and recovery γ rates, for several subsequent time steps ($t = 4$, $t = 8$ and $t = 12$). We compare the predictions obtained by applying the proposed GN-ODE model against the ground truth probabilities extracted via Monte-Carlo simulations on the test set. The color bars on the right demonstrate the ranges of the probability of infection per time step, with dark red and blue indicating the highest and lowest probabilities respectively. Not surprisingly, based on the low score in Figure 6.4a, it is clearly observed that the proposed GN-ODE architecture gives highly accurate predictions in comparison to the probabilities that emerge from the Monte-Carlo simulations. Due to the small size of the considered network, we notice that within a few time steps, many nodes become infected. In contrast, others obtain low probabilities of infection, probably by transitioning to the recovered set.

A.1.4 *Out of Distribution Generalization - Complementary Figures*

We provide complementary results in Figure A.2 for the out of distribution performance of the different methods for each dataset and different values of parameters β, γ. These results complement those of Figure 6.6 and Figure 6.7. The results show the error of the considered approaches for different instances of each dataset, including samples outside of the ranges of the parameters β or γ in the training set. Following the observations made for Figure 6.6 and Figure 6.7, in the plots for each dataset and the remaining out of distribution parameters β or γ in Figure A.2, the proposed GN-ODE method seems to have comparatively more robust performance when generalizing to unseen data compared to the other GNN models.

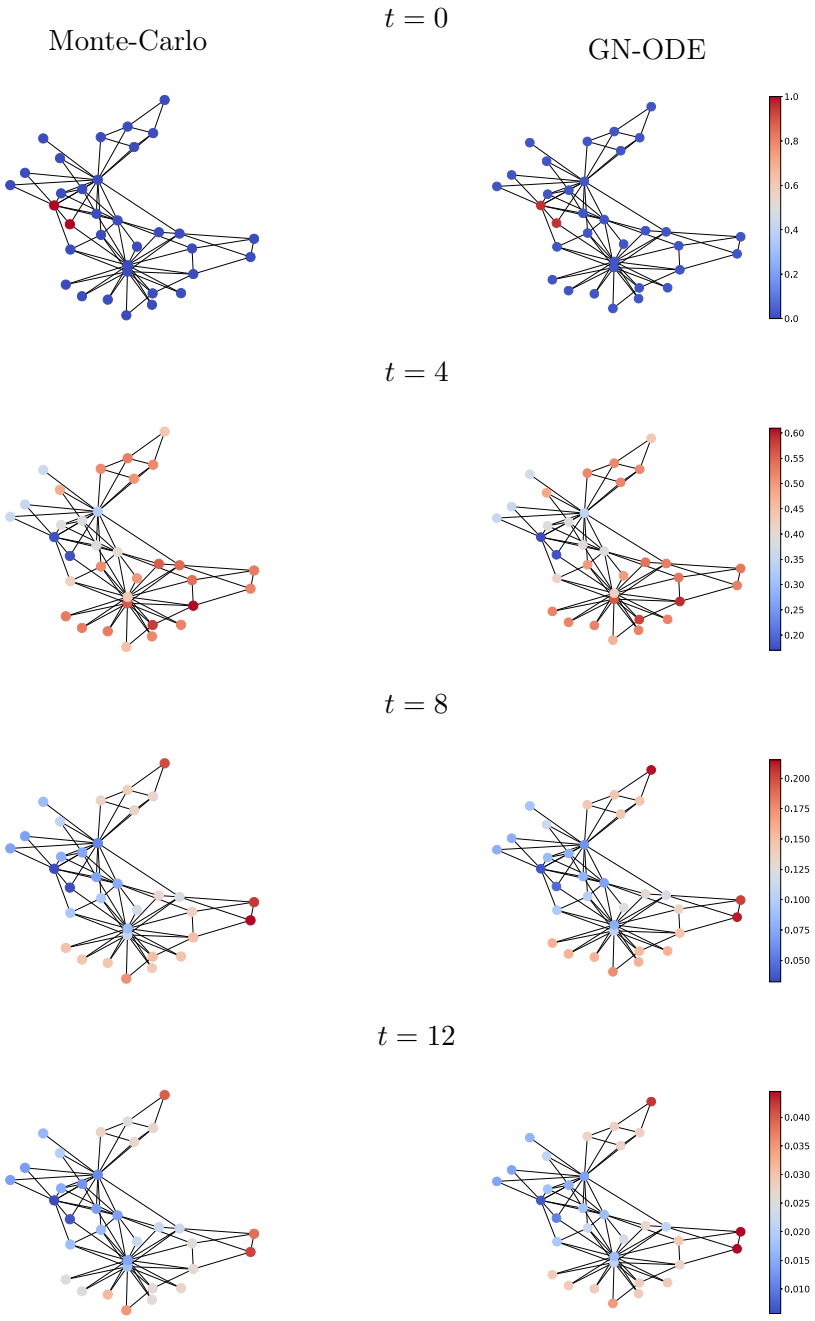
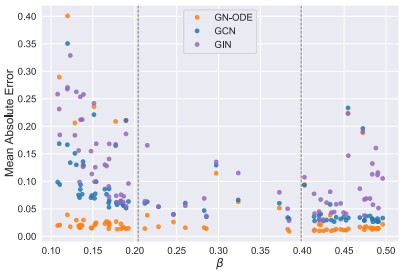
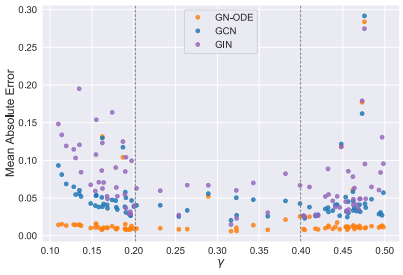


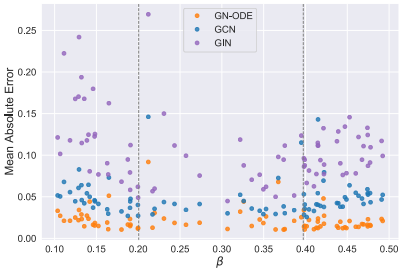
Figure A.1: Visualization of the evolution of infection over time on the karate dataset. Given the initially infected nodes (with red at $t = 0$), we compare the predictions (probability that a node is in state I) of the proposed GN-ODE model (right) against the ground truth probabilities obtained through Monte-Carlo simulations (left).



(a) (karate) Out of distribution β values

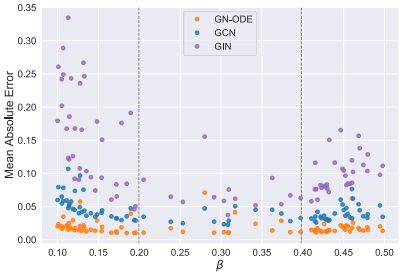


(b) (dolphins) Out of distribution γ values

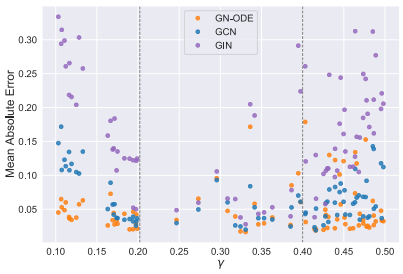


(c) (fb-food) Out of distribution β values

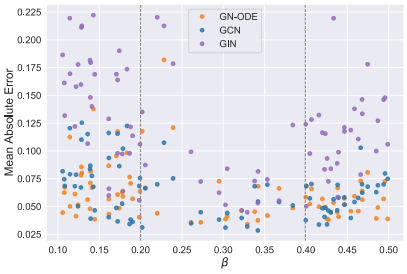
(d) (fb-social) Out of distribution γ values



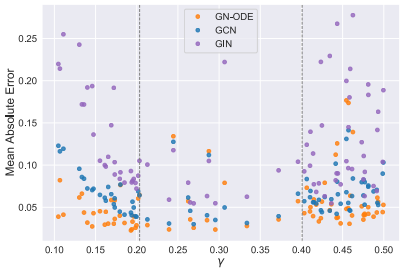
(e) (openflights) Out of distribution β values



(f) (Wiki-vote) Out of distribution γ values



(g) (Epinions) Out of distribution β values



(h) (Enron) Out of distribution γ values

Figure A.2: Mean absolute error (lower is better) achieved by the different approaches on each test sample (i. e., network) of a given dataset. Each figure is associated with one dataset and one parameter (β or γ). The out of distribution generalization performance of the different methods is evaluated. Complementary figures for each dataset and the out of distribution parameter (β or γ) are shown in Figure 6.6 and Figure 6.7.

A.2 TIME-PARAMETERIZED CNNS - ADDITIONAL RESULTS & SPECIFICATIONS

A.2.1 *Experimental Details*

HYPERPARAMETER TUNING. To identify the optimal set of hyperparameters for each deep neural network architecture, we conducted a comprehensive grid search across specified parameter ranges. Our selection process defined the combinations that yielded the minimum error on the validation dataset. The hyperparameters subjected to grid search included the learning rate, drawn from the set $\{0.0001, 0.001, 0.01\}$, the batch size, taken from $\{16, 32, 64, 128, 256, 512\}$, and the hidden dimension size for the trainable layers, chosen from $\{16, 32, 64, 128, 256\}$. From the set of the time functions defined for the parameterization of the proposed TPC layer, we tested all different combinations for employing up to 2, 4 and 8 functions simultaneously as well as each single time function alone. For the kernel sizes of the proposed TPC layer, we tested the following set of lengths $\{3, 6, 9, 15, 25\}$, whereas for the stacked vanilla convolutions we selected sizes from $\{2, 4, 8\}$. The deep learning framework utilized for the development of the code and the experimental evaluation is the PyTorch library [195]. Experiments were conducted on a single GPU, specifically an NVidia Quadro RTX 6000. Each architecture underwent training for up to 300 epochs, while early stopping was enabled in the case that the validation loss was not improved for 20 epochs. To generate predictions, we employed the model that exhibited the lowest loss on the validation dataset.

The final set of hyperparameters for interpolation on PhysioNet includes a batch size equal to 32, hidden size for the TPC layer equal to 128, a kernel size equal to 25 parameterized by the $\exp(\cdot)$ time function, while the model was trained for 300 epochs with learning rate 0.001. For the classification task on PhysioNet, the obtained results correspond to the optimal parameters of batch size 64, hidden size 128 for each time function, kernel size 3 for the TPC layer, and the $\{\sin(\cdot), \cos(\cdot), \text{lin}(\cdot), \exp(\cdot)\}$ time functions for its parameterization. We employ two stacked vanilla convolutions after the TPC layer of the same hidden dimensionality and kernel size 2, while we use 3 max pooling layers after each convolution (with kernel 8 and stride 4 for the first, kernel 2 and stride 2 for the last two). The hyperparameters are similar for classification on MIMIC-III but with batch size equal to 512, hidden size equal to 256 and the $\exp(\cdot)$ time function. Models for both datasets are trained for up to 200 epochs with early stopping and a learning rate equal to 0.0001. A final fully-connected layer maps the representation to a hidden dimension of 64 and the final classifier is a multi-layer perceptron with a hidden dimension of 300. Similarly for the classification of the Human Activity dataset, we chose a batch size of 256 and hidden size 64 for each time function in the set of $\{\sin(\cdot), \cos(\cdot), \text{lin}(\cdot), \exp(\cdot)\}$. For this dataset, we use three average pooling layers, one after each convolution, with kernel 4 and stride 4 for the first, kernel 2 and stride 2 for the last two. The rest hyperparameters are the same as for the other two datasets. For both classification and interpolation, each convolutional or fully-connected layer is followed by a Leaky ReLU non-linear activation function, except for the final fully-connected layer. In the case of classification, the last penultimate fully-connected layer is followed by a ReLU activation function.

WEIGHT INITIALIZATION FOR THE TPC LAYER. An important aspect of the fast convergence and smooth training of the proposed architecture is the weight initialization methods applied to the weights of the TPC layer. The trainable weights θ_1 and θ_3 for each input dimension m and output dimension p are initialized using the kaiming uniform distribution [107] with a negative slope equal to 10 and the Leaky ReLU non-linearity. The bias terms θ_2, θ_3 are initialized with uniform distribution in the range $(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}})$.

A.2.2 *Experiments on synthetic data.*

Following the line of work of [234], we reproduce their synthetic sinusoidal dataset that consists of 1000 samples, each describing a time series of 100 time points where $t \in [0, 1]$. Given 10 reference points, an RBF kernel with bandwidth 100 is used to obtain local interpolations at the 100 time steps. For each sample, 20 time points are randomly selected so as to represent an irregularly spaced series. A split of 80% and 20% extracts the respective train and test sets. We employ the encoder-decoder interpolation framework of Figure 4.1 (Right). Contrary to the interpolation setting for PhysioNet, we give as input the 20 irregular time steps, without the missing points, and reconstruct each observation based on the rest using TPCNN with the functions $h_2(x) = \sin(x)$ (blue points) and $h_5(x) = \exp(x)$ (green points). We visualize the obtained reconstructions for 3 samples of the test set in Figure A.3. Each plot consists of the true values (ground truth) for a test sample, while the dark markers represent the 20 observed input data points (observed data), the blue markers and the green markers the 20 predicted values (reconstruction) using $\sin(\cdot)$ and $\exp(\cdot)$ functions respectively. By employing the function $h_2(x) = \sin(x)$, we are able to achieve a lower MSE loss compared to the ones achieved with the rest of the time functions defined in Section 4.2.3.2. We should mention here that in case domain knowledge is available, it can be incorporated into the proposed TPCNN method via the employed time function, which is likely to lead to performance improvements.

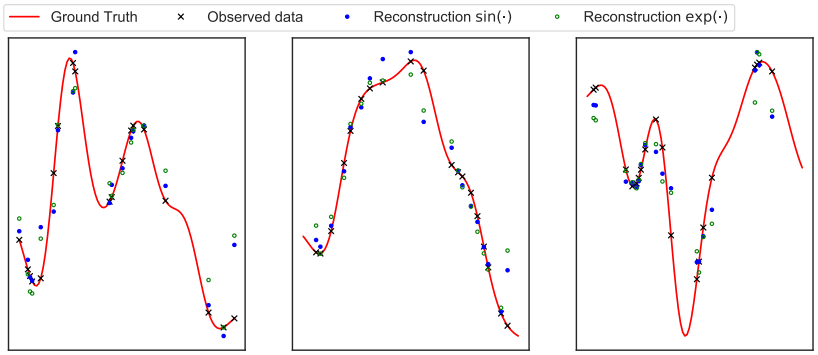


Figure A.3: Reconstruction results using the proposed TPCNN model on the synthetic dataset. Three different samples of the test set are visualized.

BIBLIOGRAPHY

[1] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. "Social lstm: Human trajectory prediction in crowded spaces." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 961–971.

[2] Juan Miguel Lopez Alcaraz and Nils Strodthoff. "Diffusion-based time series imputation and forecasting with structured state space models." *arXiv preprint arXiv:2208.09399* (2022).

[3] Pierre-Olivier Amblard and Olivier JJ Michel. "On directed information theory and Granger causality graphs." *Journal of computational neuroscience* 30.1 (2011), pp. 7–16.

[4] Samaneh Aminikhanghahi and Diane J Cook. "A survey of methods for time series change point detection." *Knowledge and Information Systems* 51.2 (2017), pp. 339–367.

[5] John David Anderson, Gérard Degrez, Erik Dick, and Roger Grundmann. *Computational fluid dynamics: an introduction*. Springer Science & Business Media, 2013.

[6] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. "A closer look at memorization in deep networks." In: *International conference on machine learning*. PMLR. 2017, pp. 233–242.

[7] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances." *Data mining and knowledge discovery* 31 (2017), pp. 606–660.

[8] 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), pp. 17804–17815.

[9] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling." *arXiv:1803.01271* (2018).

[10] Kasun Bandara, Peibei Shi, Christoph Bergmeir, Hansika Hewamalage, Quoc Tran, and Brian Seaman. "Sales demand forecast in e-commerce using a long short-term memory neural network methodology." In: *Neural Information Processing: 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12–15, 2019, Proceedings, Part III 26*. Springer. 2019, pp. 462–474.

[11] Grigory A Bautin, Valery A Kalyagin, Alexander P Koldanov, Petr A Koldanov, and Panos M Pardalos. "Simple measure of similarity for the market graph construction." *Computational Management Science* 10 (2013), pp. 105–124.

[12] Filipe De Avila Belbute-Peres, Thomas Economon, and Zico Kolter. "Combining differentiable PDE solvers and graph neural networks for fluid flow prediction." In: *international conference on machine learning*. PMLR. 2020, pp. 2402–2411.

[13] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Yuyang Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, et al. "Deep learning for time series forecasting: Tutorial and literature survey." *ACM Computing Surveys* 55.6 (2022), pp. 1–36.

[14] Rianne van den Berg, Thomas N Kipf, and Max Welling. "Graph convolutional matrix completion." *arXiv preprint arXiv:1706.02263* (2017).

[15] 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 (2020), pp. 2995–3007.

[16] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A Lozano. "A review on outlier/anomaly detection in time series data." *ACM Computing Surveys (CSUR)* 54.3 (2021), pp. 1–33.

[17] Mathieu Blondel, Arthur Mensch, and Jean-Philippe Vert. "Differentiable Divergences Between Time Series." In: *AISTATS'21*. 2021, pp. 3853–3861.

[18] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[19] E Oran Brigham. *The fast Fourier transform and its applications*. Prentice-Hall, Inc., 1988.

[20] Peter J Brockwell and Richard A Davis. *Introduction to time series and forecasting*. Springer, 2002.

[21] 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 (2020), pp. 1877–1901.

[22] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. "Machine learning for fluid mechanics." *Annual review of fluid mechanics* 52 (2020), pp. 477–508.

[23] Ceren Budak, Divyakant Agrawal, and Amr El Abbadi. "Limiting the spread of misinformation in social networks." In: *Proceedings of the 20th international conference on World wide web*. 2011, pp. 665–674.

[24] John Charles Butcher. "A history of Runge-Kutta methods." *Applied Numerical Mathematics* 20.3 (1996), pp. 247–260.

[25] Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Congrui Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, et al. "Spectral temporal graph neural network for multivariate time-series forecasting." *Advances in neural information processing systems* 33 (2020), pp. 17766–17778.

[26] 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).

[27] Giuseppe Carleo and Matthias Troyer. "Solving the quantum many-body problem with artificial neural networks." *Science* 355.6325 (2017), pp. 602–606.

[28] Di Chai, Leye Wang, and Qiang Yang. "Bike flow prediction with multi-graph convolutional networks." In: *Proceedings of the 26th ACM SIGSPATIAL international conference on advances in geographic information systems*. 2018, pp. 397–400.

[29] Bruno Chaouat. "The state of the art of hybrid RANS/LES modeling for the simulation of turbulent flows." *Flow, turbulence and combustion* 99 (2017), pp. 279–327.

[30] Sucheta Chauhan and Lovekesh Vig. "Anomaly detection in ECG time signals via deep long short-term memory networks." In: *2015 IEEE international conference on data science and advanced analytics (DSAA)*. IEEE. 2015, pp. 1–7.

[31] 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 (2018), pp. 1–12.

[32] Meng Chen, Xiaohui Yu, and Yang Liu. "PCNN: Deep convolutional networks for short-term traffic congestion prediction." *IEEE Transactions on Intelligent Transportation Systems* 19.11 (2018), pp. 3550–3559.

[33] Quanjun Chen, Xuan Song, Harutoshi Yamada, and Ryosuke Shibasaki. "Learning deep representation from big and heterogeneous data for traffic accident inference." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[34] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. "Neural ordinary differential equations." In: *Advances on Neural Information Processing Systems*. 2018, pp. 6572–6583.

[35] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[36] Wei Chen, Chi Wang, and Yajun Wang. "Scalable influence maximization for prevalent viral marketing in large-scale social networks." In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2010, pp. 1029–1038.

[37] Weiqi Chen, Ling Chen, Yu Xie, Wei Cao, Yusong Gao, and Xiaojie Feng. "Multi-range attentive bicomponent graph convolutional network for traffic forecasting." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 04. 2020, pp. 3529–3536.

[38] Yitian Chen, Yanfei Kang, Yixiong Chen, and Zizhuo Wang. "Probabilistic forecasting with temporal convolutional neural network." *Neurocomputing* 399 (2020), pp. 491–501.

[39] Zhao Chen, Yang Liu, and Hao Sun. "Physics-informed learning of governing equations from scarce data." *Nature communications* 12.1 (2021), p. 6136.

[40] Weiyu Cheng, Yanyan Shen, Yanmin Zhu, and Linpeng Huang. "A neural attention model for urban air quality inference: Learning the weights of monitoring stations." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[41] Ziqiang Cheng, Yang Yang, Shuo Jiang, Wenjie Hu, Zhangchi Ying, Ziwei Chai, and Chunping Wang. "Time2Graph+: Bridging Time Series and Graph Representation Learning via Multiple Attentions." *IEEE Transactions on Knowledge and Data Engineering* (2021).

[42] Ziqiang Cheng, Yang Yang, Wei Wang, Wenjie Hu, Yueting Zhuang, and Guojie Song. "Time2graph: Revisiting time series modeling with dynamic shapelets." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 2020, pp. 3617–3624.

[43] Flavio Chierichetti, Silvio Lattanzi, and Alessandro Panconesi. "Rumor spreading in social networks." *Theoretical Computer Science* 412.24 (2011), pp. 2602–2610.

[44] Vinay Kumar Reddy Chimmula and Lei Zhang. "Time series forecasting of COVID-19 transmission in Canada using LSTM networks." *Chaos, Solitons & Fractals* 135 (2020), p. 109864.

[45] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation." In: *EMNLP'14*. 2014, pp. 1724–1734.

[46] 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).

[47] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. "A recurrent latent variable model for sequential data." *Advances in neural information processing systems* 28 (2015).

[48] Andrea Cini, Ivan Marisca, and Cesare Alippi. "Filling the g_ap_s: Multivariate time series imputation by graph neural networks." *arXiv preprint arXiv:2108.00298* (2021).

[49] Robert B Cleveland, William S Cleveland, Jean E McRae, and Irma Terpenning. "STL: A seasonal-trend decomposition." *J. Off. Stat* 6.1 (1990), pp. 3–73.

[50] Taco Cohen and Max Welling. "Group equivariant convolutional networks." In: *International conference on machine learning*. PMLR. 2016, pp. 2990–2999.

[51] Vittoria Colizza, Alain Barrat, Marc Barthelemy, Alain-Jacques Valleron, and Alessandro Vespignani. "Modeling the worldwide spread of pandemic influenza: baseline case and containment interventions." *PLoS medicine* 4.1 (2007), e13.

[52] Fabienne Comte and Eric Renault. "Long memory continuous time models." *Journal of Econometrics* 73.1 (1996), pp. 101–149.

[53] Romain Cosentino and Behnaam Aazhang. "Learnable group transform for time-series." In: *International conference on machine learning*. PMLR. 2020, pp. 2164–2173.

[54] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. "A survey on network embedding." *IEEE transactions on knowledge and data engineering* 31.5 (2018), pp. 833–852.

[55] Zhicheng Cui, Wenlin Chen, and Yixin Chen. "Multi-scale convolutional neural networks for time series classification." *arXiv preprint arXiv:1603.06995* (2016).

[56] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. "Scientific machine learning through physics–informed neural networks: Where we are and what's next." *Journal of Scientific Computing* 92.3 (2022), p. 88.

[57] Marco Cuturi and Mathieu Blondel. "Soft-DTW: a Differentiable Loss Function for Time-Series." In: *ICML'17*. 2017, pp. 894–903.

[58] Sina Dabiri and Kevin Heaslip. "Inferring transportation modes from GPS trajectories using a convolutional neural network." *Transportation research part C: emerging technologies* 86 (2018), pp. 360–371.

[59] Estela Bee Dagum and Silvia Bianconcini. *Seasonal adjustment methods and real time trend-cycle estimation*. Springer, 2016.

[60] Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. "Learning combinatorial optimization algorithms over graphs." In: *Advances on Neural Information Processing Systems*. 2017, pp. 6351–6361.

[61] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. "Language modeling with gated convolutional networks." In: *International conference on machine learning*. PMLR. 2017, pp. 933–941.

[62] Richard H Day et al. "Complex economic dynamics-vol. 1: An introduction to dynamical systems and market mechanisms." *MIT Press Books* 1 (1994).

[63] Emmanuel De Bézenac, Arthur Pajot, and Patrick Gallinari. "Deep learning for physical processes: Incorporating prior scientific knowledge." *Journal of Statistical Mechanics: Theory and Experiment* 2019.12 (2019), p. 124009.

[64] Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. "GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series." *Advances in neural information processing systems* 32 (2019).

[65] Jan G De Gooijer and Rob J Hyndman. "25 years of time series forecasting." *International Journal of Forecasting* 22.3 (2006), pp. 443–473.

[66] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering." *Advances in neural information processing systems* 29 (2016).

[67] Angus Dempster, François Petitjean, and Geoffrey I Webb. "ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels." *Data Mining and Knowledge Discovery* 34.5 (2020), pp. 1454–1495.

[68] Ailin Deng and Bryan Hooi. "Graph neural network-based anomaly detection in multivariate time series." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 5. 2021, pp. 4027–4035.

[69] Houtao Deng, George Runger, Eugene Tuv, and Martyanov Vladimir. "A time series forest for classification and feature extraction." *Information Sciences* 239 (2013), pp. 142–153.

[70] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. "Eta prediction with graph neural networks in google maps." In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 3767–3776.

[71] Abhyuday Desai, Cynthia Freeman, Zuhui Wang, and Ian Beaver. "Timevae: A variational auto-encoder for multivariate time series generation." *arXiv preprint arXiv:2111.08095* (2021).

[72] Rahul Dey and Fathi M Salem. "Gate-variants of gated recurrent unit (GRU) neural networks." In: *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE. 2017, pp. 1597–1600.

[73] Zulong Diao, Xin Wang, Dafang Zhang, Yingru Liu, Kun Xie, and Shaoyao He. "Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 890–897.

[74] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. "Querying and mining of time series data: experimental comparison of representations and distance measures." *Proceedings of the VLDB Endowment* 1.2 (2008), pp. 1542–1552.

[75] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. "Deep learning for event-driven stock prediction." In: *Twenty-fourth international joint conference on artificial intelligence*. 2015.

[76] Pedro Domingos and Matt Richardson. "Mining the network value of customers." In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 2001, pp. 57–66.

[77] Wei Dong, Charikar Moses, and Kai Li. "Efficient k-nearest neighbor graph construction for generic similarity measures." In: *Proceedings of the 20th international conference on World wide web*. 2011, pp. 577–586.

[78] Reik V Donner, Yong Zou, Jonathan F Donges, Norbert Marwan, and Jürgen Kurths. "Recurrence networks—a novel paradigm for nonlinear time series analysis." *New Journal of Physics* 12.3 (2010), p. 033025.

[79] John R Dormand and Peter J Prince. "A family of embedded Runge-Kutta formulae." *Journal of computational and applied mathematics* 6.1 (1980), pp. 19–26.

[80] Ziheng Duan, Haoyan Xu, Yueyang Wang, Yida Huang, Anni Ren, Zhongbin Xu, Yizhou Sun, and Wei Wang. "Multivariate time-series classification with hierarchical variational graph pooling." *Neural Networks* 154 (2022), pp. 481–490.

[81] James Durbin and Siem Jan Koopman. *Time series analysis by state space methods*. Vol. 38. OUP Oxford, 2012.

[82] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee Keong Kwoh, Xiaoli Li, and Cuntai Guan. "Time-series representation learning via temporal and contextual contrasting." *arXiv preprint arXiv:2106.14112* (2021).

[83] Philippe Esling and Carlos Agon. "Time-series data mining." *ACM Computing Surveys (CSUR)* 45.1 (2012), pp. 1–34.

[84] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. "Real-valued (medical) time series generation with recurrent conditional gans." *arXiv preprint arXiv:1706.02633* (2017).

[85] Lawrence C Evans. *Partial differential equations*. Vol. 19. American Mathematical Society, 2022.

[86] Chenguang Fang and Chen Wang. "Time series data imputation: A survey on deep learning approaches." *arXiv preprint arXiv:2011.11347* (2020).

[87] 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*. 2021, pp. 364–373.

[88] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. "Protein interface prediction using graph convolutional networks." *Advances in neural information processing systems* 30 (2017).

[89] Joseph Futoma, Sanjay Hariharan, and Katherine Heller. "Learning to detect sepsis with a multitask Gaussian process RNN classifier." In: *International conference on machine learning*. PMLR. 2017, pp. 1174–1182.

[90] Everette S Gardner Jr. "Exponential smoothing: The state of the art." *Journal of forecasting* 4.1 (1985), pp. 1–28.

[91] Johannes Gasteiger, Janek Groß, and Stephan Günnemann. "Directional message passing for molecular graphs." *arXiv preprint arXiv:2003.03123* (2020).

[92] 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*. PMLR. 2019, pp. 1901–1910.

[93] Xu Geng, Yaguang Li, Leye Wang, Lingyu Zhang, Qiang Yang, Jieping Ye, and Yan Liu. "Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 3656–3663.

[94] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. "Neural message passing for quantum chemistry." In: *Proceedings of the 34th International Conference on Machine Learning*. 2017, pp. 1263–1272.

[95] Vladimir Gligorijević, P Douglas Renfrew, Tomasz Kosciolek, Julia Koehler Leman, Daniel Berenberg, Tommi Vatanen, Chris Chandler, Bryn C Taylor, Ian M Fisk, Hera Vlamakis, et al. "Structure-based protein function prediction using graph convolutional networks." *Nature communications* 12.1 (2021), p. 3168.

[96] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. "Recent advances in convolutional neural networks." *Pattern recognition* 77 (2018), pp. 354–377.

[97] Vincent Le Guen and Nicolas Thome. "Shape and Time Distortion Loss for Training Deep Time Series Forecasting Models." In: *NeurIPS'19*. 2019, pp. 4189–4201.

[98] Vincent Le Guen and Nicolas Thome. "Probabilistic Time Series Forecasting with Shape and Temporal Diversity." In: *NeurIPS'20*. 2020, pp. 4427–4440.

[99] Kan Guo, Yongli Hu, Yanfeng Sun, Sean Qian, Junbin Gao, and Baocai Yin. "Hierarchical graph convolution network for traffic forecasting." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 1. 2021, pp. 151–159.

[100] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 922–929.

[101] Mohammad M Hamed, Hashem R Al-Masaeid, and Zahi M Bani Said. "Short-term prediction of traffic volume in urban arterials." *Journal of Transportation Engineering* 121.3 (1995).

[102] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." *Advances in neural information processing systems* 30 (2017).

[103] William L Hamilton, Rex Ying, and Jure Leskovec. "Representation learning on graphs: Methods and applications." *arXiv preprint arXiv:1709.05584* (2017).

[104] Jindong Han, Hao Liu, Hengshu Zhu, Hui Xiong, and Dejing Dou. "Joint air quality and weather prediction based on multi-adversarial spatiotemporal networks." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 5. 2021, pp. 4081–4089.

[105] Zhongkai Hao, Songming Liu, Yichi Zhang, Chengyang Ying, Yao Feng, Hang Su, and Jun Zhu. "Physics-informed machine learning: A survey on problems, methods and applications." *arXiv preprint arXiv:2211.08064* (2022).

[106] Frank E Harrell et al. *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis*. Vol. 608. Springer, 2001.

[107] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[108] Herbert W Hethcote. "The mathematics of infectious diseases." *SIAM review* 42.4 (2000), pp. 599–653.

[109] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. "Classification of time series by shapelet transformation." *Data mining and knowledge discovery* 28 (2014), pp. 851–881.

[110] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997), pp. 1735–1780.

[111] Charles C Holt. "Forecasting seasonals and trends by exponentially weighted moving averages." *International Journal of Forecasting* 20.1 (2004), pp. 5–10.

[112] Max Horn, Michael Moor, Christian Bock, Bastian Rieck, and Karsten Borgwardt. "Set functions for time series." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 4353–4363.

[113] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

[114] Rob J Hyndman and Anne B Koehler. "Another look at measures of forecast accuracy." *International journal of forecasting* 22.4 (2006), pp. 679–688.

[115] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. "Deep learning for time series classification: a review." *Data mining and knowledge discovery* 33.4 (2019), pp. 917–963.

[116] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F Schmidt, Jonathan Weber, Geoffrey I Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. "Inceptiontime: Finding alexnet for time series classification." *Data Mining and Knowledge Discovery* 34.6 (2020), pp. 1936–1962.

[117] Eugene M Izhikevich. *Dynamical systems in neuroscience*. MIT press, 2007.

[118] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical reparameterization with gumbel-softmax." *arXiv preprint arXiv:1611.01144* (2016).

[119] Daniel Jarrett, Ioana Bica, and Mihaela van der Schaar. "Time-series generation by contrastive imitation." *Advances in Neural Information Processing Systems* 34 (2021), pp. 28968–28982.

[120] 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* (2023).

[121] Guangyin Jin, Min Wang, Jinlei Zhang, Hengyu Sha, and Jincai Huang. "STGNN-TTE: Travel time estimation via spatial–temporal graph neural network." *Future Generation Computer Systems* 126 (2022), pp. 70–81.

[122] 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* (2023).

[123] 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).

[124] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. "MIMIC-III, a freely accessible critical care database." *Scientific data* 3.1 (2016), pp. 1–9.

[125] Nikolaos Karalias and Andreas Loukas. "Erdős goes neural: an unsupervised learning framework for combinatorial optimization on graphs." In: *Advances on Neural Information Processing Systems*. 2020, pp. 6659–6672.

[126] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. "Physics-informed machine learning." *Nature Reviews Physics* 3.6 (2021), pp. 422–440.

[127] Brian Karrer and Mark EJ Newman. "Message passing approach for general epidemic models." *Physical Review E* 82.1 (2010), p. 016101.

[128] Karthik Kashinath, M Mustafa, Adrian Albert, JL Wu, C Jiang, Soheil Esmaeilzadeh, Kamyar Azizzadenesheli, R Wang, A Chattopadhyay, A Singh, et al. "Physics-informed machine learning: case studies for weather and climate modelling." *Philosophical Transactions of the Royal Society A* 379.2194 (2021), p. 20200093.

[129] 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).

[130] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. "Molecular graph convolutions: moving beyond fingerprints." *Journal of computer-aided molecular design* 30 (2016), pp. 595–608.

[131] David Kempe, Jon Kleinberg, and Éva Tardos. "Maximizing the spread of influence through a social network." In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2003, pp. 137–146.

[132] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. "Neural controlled differential equations for irregular time series." *Advances in Neural Information Processing Systems* 33 (2020), pp. 6696–6707.

[133] Sung-Suk Kim. "Time-delay recurrent neural network for temporal correlations and prediction." *Neurocomputing* 20.1-3 (1998), pp. 253–263.

[134] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks." In: *In 5th International Conference on Learning Representations*. 2017.

[135] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. "Neural relational inference for interacting systems." In: *International conference on machine learning*. PMLR. 2018, pp. 2688–2697.

[136] Maksim Kitsak, Lazaros K Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H Eugene Stanley, and Hernán A Makse. "Identification of influential spreaders in complex networks." *Nature physics* 6.11 (2010), pp. 888–893.

[137] Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. "Machine learning–accelerated computational fluid dynamics." *Proceedings of the National Academy of Sciences* 118.21 (2021), e2101784118.

[138] Stephan Kolassa. "Why the "best" point forecast depends on the error or accuracy measure." *International Journal of Forecasting* 36.1 (2020), pp. 208–211.

[139] Xiangjie Kong, Wenfeng Zhou, Guojiang Shen, Wenyi Zhang, Nali Liu, and Yao Yang. "Dynamic graph convolutional recurrent imputation network for spatiotemporal traffic missing data." *Knowledge-Based Systems* 261 (2023), p. 110188.

[140] Chrysoula Kosma, Giannis Nikolentzos, George Panagopoulos, Jean-Marc Steyaert, and Michalis Vazirgiannis. "Neural Ordinary Differential Equations for Modeling Epidemic Spreading." *Transactions on Machine Learning Research* (2023).

[141] Chrysoula Kosma, Giannis Nikolentzos, and Michalis Vazirgiannis. "Time-Parameterized Convolutional Neural Networks for Irregularly Sampled Time Series." *arXiv preprint arXiv:2308.03210* (2023).

[142] Chrysoula Kosma, Giannis Nikolentzos, Nancy Xu, and Michalis Vazirgiannis. "Time Series Forecasting Models Copy the Past: How to Mitigate." In: *Artificial Neural Networks and Machine Learning–ICANN 2022: 31st International Conference on Artificial Neural Networks, Bristol, UK, September 6–9, 2022, Proceedings, Part I*. Springer. 2022, pp. 366–378.

[143] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012).

[144] J Nathan Kutz. "Deep learning in fluid dynamics." *Journal of Fluid Mechanics* 814 (2017), pp. 1–4.

[145] Denis Kwiatkowski, Peter CB Phillips, Peter Schmidt, and Yongcheol Shin. "Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?" *Journal of econometrics* 54.1-3 (1992), pp. 159–178.

[146] Lucas Lacasa, Bartolo Luque, Fernando Ballesteros, Jordi Luque, and Juan Carlos Nuno. "From time series to complex networks: The visibility graph." *Proceedings of the National Academy of Sciences* 105.13 (2008), pp. 4972–4975.

[147] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. "Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks." In: *SIGIR'18*. 2018, pp. 95–104.

[148] Thomas A Lasko. "Efficient inference of Gaussian-process-modulated renewal processes with application to medical event data." In: *Uncertainty in artificial intelligence: proceedings of the... conference. Conference on Uncertainty in Artificial Intelligence*. Vol. 2014. NIH Public Access. 2014, p. 469.

[149] Vincent Le Guen and Nicolas Thome. "Shape and time distortion loss for training deep time series forecasting models." *Advances in neural information processing systems* 32 (2019).

[150] Vincent Le Guen and Nicolas Thome. "Deep time series forecasting with shape and temporal criteria." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.1 (2022), pp. 342–355.

[151] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. "Data augmentation for time series classification using convolutional neural networks." In: *ECML/PKDD workshop on advanced analytics and learning on temporal data*. 2016.

[152] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[153] Mengzhang Li and Zhanxing Zhu. "Spatial-temporal fusion graph neural networks for traffic flow forecasting." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 5. 2021, pp. 4189–4196.

[154] 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." In: *NeurIPS'19*. 2019, pp. 5243–5253.

[155] Steven Cheng-Xian Li and Benjamin M Marlin. "A scalable end-to-end gaussian process adapter for irregularly sampled time series classification." *Advances in neural information processing systems* 29 (2016).

[156] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting." *arXiv preprint arXiv:1707.01926* (2017).

[157] Yan Li, Xinjiang Lu, Yaqing Wang, and Dejing Dou. "Generative time series forecasting with diffusion, denoise, and disentanglement." *Advances in Neural Information Processing Systems* 35 (2022), pp. 23009–23022.

[158] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. "Fourier neural operator for parametric partial differential equations." *arXiv preprint arXiv:2010.08895* (2020).

[159] Bryan Lim and Stefan Zohren. "Time-series forecasting with deep learning: a survey." *Philosophical Transactions of the Royal Society A* 379.2194 (2021), p. 20200209.

[160] Bryan Lim, Stefan Zohren, and Stephen Roberts. "Recurrent neural filters: Learning independent bayesian filtering steps for time series prediction." In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–8.

[161] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. "Experiencing SAX: a novel symbolic representation of time series." *Data Mining and knowledge discovery* 15 (2007), pp. 107–144.

[162] Min Lin, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).

[163] Jason Lines, Sarah Taylor, and Anthony Bagnall. "Time series classification with HIVE-COTE: The hierarchical vote collective of transformation-based ensembles." *ACM Transactions on Knowledge Discovery from Data (TKDD)* 12.5 (2018), pp. 1–35.

[164] Zachary C Lipton, David Kale, and Randall Wetzel. "Directly modeling missing data in sequences with rnns: Improved classification of clinical time series." In: *Machine learning for healthcare conference*. PMLR. 2016, pp. 253–270.

[165] Vadim Lisitsa, Galina Reshetova, and Vladimir Tcheverda. "Finite-difference algorithm with local time-space grid refinement for simulation of waves." *Computational geosciences* 16 (2012), pp. 39–54.

[166] Mingzhe Liu, Han Huang, Hao Feng, Leilei Sun, Bowen Du, and Yanjie Fu. "PriSTI: A Conditional Diffusion Framework for Spatiotemporal Imputation." *arXiv preprint arXiv:2302.09746* (2023).

[167] Andrey Y Lokhov, Marc Mézard, and Lenka Zdeborová. "Dynamic message-passing equations for models with unidirectional dynamics." *Physical Review E* 91.1 (2015), p. 012811.

[168] Yun Long, Xueyuan She, and Saibal Mukhopadhyay. "Hybridnet: integrating model-based and data-driven learning to predict evolution of dynamical systems." In: *Conference on Robot Learning*. PMLR. 2018, pp. 551–560.

[169] Lars Lorch, Heiner Kremer, William Trouleau, Stratis Tsirtsis, Aron Szanto, Bernhard Schölkopf, and Manuel Gomez-Rodriguez. "Quantifying the effects of contact tracing, testing, and containment measures in the presence of infection hotspots." *ACM Transactions on Spatial Algorithms and Systems* 8.4 (2022), pp. 1–28.

[170] Mantas Lukoševičius and Herbert Jaeger. "Reservoir computing approaches to recurrent neural network training." *Computer science review* 3.3 (2009), pp. 127–149.

[171] Yonghong Luo, Xiangrui Cai, Ying Zhang, Jun Xu, et al. "Multivariate time series imputation with generative adversarial networks." *Advances in neural information processing systems* 31 (2018).

[172] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. "The concrete distribution: A continuous relaxation of discrete random variables." *arXiv preprint arXiv:1611.00712* (2016).

[173] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. "M5 accuracy competition: Results, findings, and conclusions." *International Journal of Forecasting* 38.4 (2022), pp. 1346–1364.

[174] Srikanth Malla, Chiho Choi, and Behzad Dariush. "Social-STAGE: Spatio-temporal multi-modal future trajectory forecast." In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 13938–13944.

[175] Ivan Marisca, Andrea Cini, and Cesare Alippi. "Learning to reconstruct missing data from spatiotemporal graphs with sparse observations." *Advances in Neural Information Processing Systems* 35 (2022), pp. 32069–32082.

[176] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[177] Hongyuan Mei and Jason M Eisner. "The neural hawkes process: A neurally self-modulating multivariate point process." *Advances in neural information processing systems* 30 (2017).

[178] Arthur Mensch and Mathieu Blondel. "Differentiable dynamic programming for structured prediction and attention." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3462–3471.

[179] Abduallah Mohamed, Kun Qian, Mohamed Elhoseiny, and Christian Claudel. "Social-stgcnn: A social spatio-temporal graph convolutional neural network for human trajectory prediction." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 14424–14432.

[180] Douglas C Montgomery, Cheryl L Jennings, and Murat Kulahci. *Introduction to time series analysis and forecasting*. John Wiley & Sons, 2015.

[181] Michael C Mozer, Denis Kazakov, and Robert V Lindsey. "Discrete event, continuous time rnns." *arXiv preprint arXiv:1710.04110* (2017).

[182] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. "Phased lstm: Accelerating recurrent network training for long or event-based sequences." *Advances in neural information processing systems* 29 (2016).

[183] Hao Ni, Lukasz Szpruch, Marc Sabate-Vidales, Baoren Xiao, Magnus Wiese, and Shujian Liao. "Sig-Wasserstein GANs for time series generation." In: *Proceedings of the Second ACM International Conference on AI in Finance*. 2021, pp. 1–8.

[184] Nima Noorshams, Saurabh Verma, and Aude Hofleitner. "Ties: temporal interaction embeddings for enhancing social media integrity at facebook." In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, pp. 3128–3135.

[185] Elzbieta Olejarczyk and Wojciech Jernajczyk. "Graph-based analysis of brain connectivity in schizophrenia." *PloS one* 12.11 (2017), e0188629.

[186] 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).

[187] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation learning with contrastive predictive coding." *arXiv preprint arXiv:1807.03748* (2018).

[188] Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting." *arXiv preprint arXiv:1905.10437* (2019).

[189] Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting." In: *ICLR'20*. 2020.

[190] Zheyi Pan, Yuxuan Liang, Weifeng Wang, Yong Yu, Yu Zheng, and Junbo Zhang. "Urban traffic prediction from spatio-temporal data using deep meta learning." In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 1720–1730.

[191] George Panagopoulos, Giannis Nikolentzos, and Michalis Vazirgiannis. "Transfer graph neural networks for pandemic forecasting." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 6. 2021, pp. 4838–4845.

[192] Cheonbok Park, Chunggi Lee, Hyojin Bahng, Yunwon Tae, Seungmin Jin, Kihwan Kim, Sungahn Ko, and Jaegul Choo. "ST-GRAT: A novel spatio-temporal graph attention networks for accurately forecasting dynamically changing road speed." In: *Proceedings of the 29th ACM international conference on information & knowledge management*. 2020, pp. 1215–1224.

[193] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. "Deepsdf: Learning continuous signed distance functions for shape representation." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 165–174.

[194] Romualdo Pastor-Satorras and Alessandro Vespignani. "Epidemic dynamics and endemic states in complex networks." *Physical Review E* 63.6 (2001), p. 066117.

[195] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. "Pytorch: An imperative style, high-performance deep learning library." *Advances in neural information processing systems* 32 (2019).

[196] Jose Manuel Pavía-Miralles et al. "A survey of methods to interpolate, distribute and extra-polate time series." *Journal of Service Science and Management* 3.04 (2010), p. 449.

[197] Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. "Predicting healthcare trajectories from medical records: A deep learning approach." *Journal of biomedical informatics* 69 (2017), pp. 218–229.

[198] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. CRC press, 1987.

[199] Yao Qin, Dongjin Song, Haifeng Cheng, Wei Cheng, Guofei Jiang, and Garrison W Cottrell. "A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction." In: *IJCAI'17*. 2017, pp. 2627–2633.

[200] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. "Pre-trained models for natural language processing: A survey." *Science China Technological Sciences* 63.10 (2020), pp. 1872–1897.

[201] Evan Racah, Christopher Beckham, Tegan Maharaj, Samira Ebrahimi Kahou, Mr Prabhat, and Chris Pal. "Extremeweather: A large-scale climate dataset for semi-supervised detection, localization, and understanding of extreme weather events." *Advances in neural information processing systems* 30 (2017).

[202] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. "Language models are unsupervised multitask learners." *OpenAI blog* 1.8 (2019), p. 9.

[203] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations." *Journal of Computational physics* 378 (2019), pp. 686–707.

[204] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations." *arXiv preprint arXiv:1711.10561* (2017).

[205] Alvin Rajkomar, Eyal Oren, Kai Chen, Andrew M Dai, Nissan Hajaj, Michaela Hardt, Peter J Liu, Xiaobing Liu, Jake Marcus, Mimi Sun, et al. "Scalable and accurate deep learning with electronic health records." *NPJ digital medicine* 1.1 (2018), pp. 1–10.

[206] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. "Deep State Space Models for Time Series Forecasting." In: *NeurIPS'18*. 2018, pp. 7785–7794.

[207] 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*. PMLR. 2021, pp. 8857–8868.

[208] Kashif Rasul, Abdul-Saboor Sheikh, Ingmar Schuster, Urs Bergmann, and Roland Vollgraf. "Multivariate probabilistic time series forecasting via conditioned normalizing flows." *arXiv preprint arXiv:2002.06103* (2020).

[209] François Rivest and Richard Kohar. "A New Timing Error Cost Function for Binary Time Series Prediction." *IEEE TNNLS* 31.1 (2019), pp. 174–185.

[210] Juan José Rodriguez, Ludmila I Kuncheva, and Carlos J Alonso. "Rotation forest: A new classifier ensemble method." *IEEE transactions on pattern analysis and machine intelligence* 28.10 (2006), pp. 1619–1630.

[211] David W Romero, Anna Kuzina, Erik J Bekkers, Jakub M Tomczak, and Mark Hoogendoorn. "Ckconv: Continuous kernel convolution for sequential data." *arXiv preprint arXiv:2102.02611* (2021).

[212] Ryan Rossi and Nesreen Ahmed. "The network data repository with interactive graph analytics and visualization." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 29. 1. 2015.

[213] François Rousseau and Michalis Vazirgiannis. "Graph-of-word and TW-IDF: new approach to ad hoc IR." In: *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 2013, pp. 59–68.

[214] Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. "Latent ordinary differential equations for irregularly-sampled time series." *Advances in neural information processing systems* 32 (2019).

[215] Alaa Sagheer and Mostafa Kotb. "Unsupervised pre-training of a deep LSTM-based stacked autoencoder for multivariate time series forecasting problems." *Scientific reports* 9.1 (2019), pp. 1–16.

[216] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. "DeepAR: Probabilistic forecasting with autoregressive recurrent networks." *International Journal of Forecasting* 36.3 (2020), pp. 1181–1191.

[217] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. "Learning to simulate complex physics with graph networks." In: *International conference on machine learning*. PMLR. 2020, pp. 8459–8468.

[218] Leonidas Sandoval Jr. "Structure of a global network of financial companies based on transfer entropy." *Entropy* 16.8 (2014), pp. 4443–4482.

[219] Aliaksei Sandryhaila and José MF Moura. "Discrete signal processing on graphs." *IEEE transactions on signal processing* 61.7 (2013), pp. 1644–1656.

[220] Victor Garcia Satorras, Syama Sundar Rangapuram, and Tim Januschowski. "Multivariate time series forecasting with latent graph inference." *arXiv preprint arXiv:2203.03423* (2022).

[221] Patrick Schäfer. "The BOSS is concerned with time series classification in the presence of noise." *Data Mining and Knowledge Discovery* 29 (2015), pp. 1505–1530.

[222] Patrick Schäfer and Mikael Högqvist. "SFA: a symbolic fourier approximation and index for similarity search in high dimensional datasets." In: *Proceedings of the 15th international conference on extending database technology*. 2012, pp. 516–527.

[223] Sebastian Scher. "Toward data-driven weather and climate forecasting: Approximating a simple general circulation model with deep learning." *Geophysical Research Letters* 45.22 (2018), pp. 12–616.

[224] Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. "Schnet: A continuous-filter convolutional neural network for modeling quantum interactions." *Advances in neural information processing systems* 30 (2017).

[225] Rajat Sen, Hsiang-Fu Yu, and Inderjit S Dhillon. "Think Globally, Act Locally: A Deep Neural Network Approach to High-Dimensional Time Series Forecasting." In: *NeurIPS'19*. 2019, pp. 4837–4846.

[226] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. "Structured sequence modeling with graph convolutional recurrent networks." In: *Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part I 25*. Springer. 2018, pp. 362–373.

[227] Chao Shang, Jie Chen, and Jinbo Bi. "Discrete graph structure learning for forecasting multiple time series." *arXiv preprint arXiv:2101.06861* (2021).

[228] Michael Shapiro and Edgar Delgado-Eckert. "Finding the probability of infection in an SIR network is NP-Hard." *Mathematical biosciences* 240.2 (2012), pp. 77–84.

[229] Kieran J Sharkey. "Deterministic epidemiological models at the individual level." *Journal of Mathematical Biology* 57 (2008), pp. 311–331.

[230] 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).

[231] Shun-Yao Shih, Fan-Keng Sun, and Hung-yi Lee. "Temporal pattern attention for multivariate time series forecasting." *Machine Learning* 108 (2019), pp. 1421–1441.

[232] Satya Narayan Shukla and Benjamin M Marlin. "Interpolation-prediction networks for irregularly sampled time series." *arXiv preprint arXiv:1909.07782* (2019).

[233] 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).

[234] Satya Narayan Shukla and Benjamin M Marlin. "Multi-time attention networks for irregularly sampled time series." *arXiv preprint arXiv:2101.10318* (2021).

[235] 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 (2013), pp. 83–98.

[236] Ikaro Silva, George Moody, Daniel J Scott, Leo A Celi, and Roger G Mark. "Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012." In: *2012 Computing in Cardiology*. IEEE. 2012, pp. 245–248.

[237] Jelena Simeunović, Baptiste Schubnel, Pierre-Jean Alet, and Rafael E Carrillo. "Spatio-temporal graph neural networks for multi-site PV power forecasting." *IEEE Transactions on Sustainable Energy* 13.2 (2021), pp. 1210–1220.

[238] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. "Implicit neural representations with periodic activation functions." *Advances in Neural Information Processing Systems* 33 (2020), pp. 7462–7473.

[239] Slawek Smyl and N Grace Hua. "Machine learning methods for GEFCom2017 probabilistic load forecasting." *International Journal of Forecasting* 35.4 (2019), pp. 1424–1431.

[240] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. "Deep unsupervised learning using nonequilibrium thermodynamics." In: *International conference on machine learning*. PMLR. 2015, pp. 2256–2265.

[241] Huan Song, Deepta Rajan, Jayaraman Thiagarajan, and Andreas Spanias. "Attend and diagnose: Clinical time series analysis using attention models." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 2018.

[242] Steven H Strogatz. *Nonlinear dynamics and chaos with student solutions manual: With applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[243] Souhaib Ben Taieb and Amir F Atiya. "A bias and variance analysis for multistep-ahead time series forecasting." *IEEE transactions on neural networks and learning systems* 27.1 (2015), pp. 62–76.

[244] Qingxiong Tan, Mang Ye, Baoyao Yang, Siqi Liu, Andy Jinhua Ma, Terry Cheuk-Fung Yip, Grace Lai-Hung Wong, and PongChi Yuen. "Data-gru: Dual-attention time-aware gated recurrent unit for irregular multivariate time series." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 2020, pp. 930–937.

[245] Samia Tasnim, Md Mahbub Hossain, and Hoimonty Mazumder. "Impact of rumors and misinformation on COVID-19 in social media." *Journal of Preventive Medicine and Public Health* 53.3 (2020), pp. 171–174.

[246] Sean J Taylor and Benjamin Letham. "Forecasting at scale." *The American Statistician* 72.1 (2018), pp. 37–45.

[247] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. "Accelerating eulerian fluid simulation with convolutional networks." In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3424–3433.

[248] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

[249] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. "Graph attention networks." *arXiv preprint arXiv:1710.10903* (2017).

[250] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. "Extracting and composing robust features with denoising autoencoders." In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1096–1103.

[251] John Wainwright and George Francis Rayner Ellis. *Dynamical systems in cosmology*. 1997.

[252] Chunnan Wang, Kaixin Zhang, Hongzhi Wang, and Bozhou Chen. "Auto-STGCN: Autonomous Spatial-Temporal Graph Convolutional Network Search." *ACM Transactions on Knowledge Discovery from Data* 17.5 (2023), pp. 1–21.

[253] Chunyang Wang, Yanmin Zhu, Tianzi Zang, Haobing Liu, and Jiadi Yu. "Modeling inter-station relationships with attentive temporal graph convolutional network for air quality prediction." In: *Proceedings of the 14th ACM international conference on web search and data mining*. 2021, pp. 616–634.

[254] Rui Wang, Danielle Maddix, Christos Faloutsos, Yuyang Wang, and Rose Yu. "Bridging physics-based and data-driven modeling for learning dynamical systems." In: *Learning for Dynamics and Control*. PMLR. 2021, pp. 385–398.

[255] Rui Wang and Rose Yu. "Physics-guided deep learning for dynamical systems: A survey." *arXiv preprint arXiv:2107.01272* (2021).

[256] Senzhang Wang, Jiannong Cao, and S Yu Philip. "Deep learning for spatio-temporal data mining: A survey." *IEEE transactions on knowledge and data engineering* 34.8 (2020), pp. 3681–3700.

[257] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. "Deep parametric continuous convolutional neural networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2589–2597.

[258] Shuo Wang, Yanran Li, Jiang Zhang, Qingye Meng, Lingwei Meng, and Fei Gao. "PM2. 5-GNN: A domain knowledge enhanced graph neural network for PM2. 5 forecasting." In: *Proceedings of the 28th international conference on advances in geographic information systems*. 2020, pp. 163–166.

[259] Xiaoyang Wang, Yao Ma, Yiqi Wang, Wei Jin, Xin Wang, Jiliang Tang, Caiyan Jia, and Jian Yu. "Traffic flow prediction via spatial temporal graph neural network." In: *Proceedings of the web conference 2020*. 2020, pp. 1082–1092.

[260] Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. "Experimental comparison of representation methods and distance measures for time series data." *Data Mining and Knowledge Discovery* 26 (2013), pp. 275–309.

[261] Yuyang Wang, Alex Smola, Danielle Maddix, Jan Gasthaus, Dean Foster, and Tim Januschowski. "Deep Factors for Forecasting." In: *ICML'19*. 2019, pp. 6607–6617.

[262] Zhiguang Wang, Weizhong Yan, and Tim Oates. "Time series classification from scratch with deep neural networks: A strong baseline." In: *2017 International joint conference on neural networks (IJCNN)*. IEEE. 2017, pp. 1578–1585.

[263] Qingsong Wen, Linxiao Yang, Tian Zhou, and Liang Sun. "Robust time series analysis and applications: An industrial perspective." In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2022, pp. 4836–4837.

[264] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. "A multi-horizon quantile recurrent forecaster." *arXiv preprint arXiv:1711.11053* (2017).

[265] Paul J Werbos. "Backpropagation through time: what it does and how to do it." *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[266] Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. "Integrating scientific knowledge with machine learning for engineering and environmental systems." *ACM Computing Surveys* 55.4 (2022), pp. 1–37.

[267] Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven Hoi. "CoST: Contrastive learning of disentangled seasonal-trend representations for time series forecasting." *arXiv preprint arXiv:2202.01575* (2022).

[268] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting." *Advances in Neural Information Processing Systems* 34 (2021), pp. 22419–22430.

[269] 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*. Vol. 35. 5. 2021, pp. 4478–4485.

[270] Yuankai Wu, Dingyi Zhuang, Mengying Lei, Aurelie Labbe, and Lijun Sun. "Spatial aggregation and temporal convolution networks for real-time kriging." *arXiv preprint arXiv:2109.12144* (2021).

[271] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. "A comprehensive survey on graph neural networks." *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2020), pp. 4–24.

[272] 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*. 2020, pp. 753–763.

[273] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. "Graph wavenet for deep spatial-temporal graph modeling." *arXiv preprint arXiv:1906.00121* (2019).

[274] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. "Self-attention with functional time representation learning." *Advances in neural information processing systems* 32 (2019).

[275] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. "How Powerful are Graph Neural Networks?" In: *7th International Conference on Learning Representations*. 2019.

[276] Nancy Xu, Chrysoula Kosma, and Michalis Vazirgiannis. "TimeGNN: Temporal Dynamic Graph Learning for Time Series Forecasting." *arXiv preprint arXiv:2307.14680* (2023).

[277] Liu Yang, Xuhui Meng, and George Em Karniadakis. "B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data." *Journal of Computational Physics* 425 (2021), p. 109913.

[278] Lexiang Ye and Eamonn Keogh. "Time series shapelets: a novel technique that allows accurate, interpretable and fast classification." *Data mining and knowledge discovery* 22 (2011), pp. 149–182.

[279] 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*. Springer. 2021, pp. 241–252.

[280] Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. "Time-series generative adversarial networks." *Advances in neural information processing systems* 32 (2019).

[281] Jinsung Yoon, William R Zame, and Mihaela van der Schaar. "Estimating missing data in temporal data streams using multi-directional recurrent neural networks." *IEEE Transactions on Biomedical Engineering* 66.5 (2018), pp. 1477–1490.

[282] Mina Youssef and Caterina Scoglio. "An individual-based approach to SIR epidemics in contact networks." *Journal of theoretical biology* 283.1 (2011), pp. 136–144.

[283] Bing Yu, Haoteng Yin, and Zhanxing Zhu. "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting." *arXiv preprint arXiv:1709.04875* (2017).

[284] 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).

[285] Cunjun Yu, Xiao Ma, Jiawei Ren, Haiyu Zhao, and Shuai Yi. "Spatio-temporal graph transformer networks for pedestrian trajectory prediction." In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XII 16*. Springer. 2020, pp. 507–523.

[286] Fisher Yu and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions." *arXiv preprint arXiv:1511.07122* (2015).

[287] Hongyuan Yu, Ting Li, Weichen Yu, Jianguo Li, Yan Huang, Liang Wang, and Alex Liu. "Regularized graph structure learning with semantic knowledge for multi-variates time-series forecasting." *arXiv preprint arXiv:2210.06126* (2022).

[288] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S Dhillon. "Temporal Regularized Matrix Factorization for High-dimensional Time Series Prediction." In: *NeurIPS'16*. 2016, pp. 847–855.

[289] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. "A review of recurrent neural networks: LSTM cells and network architectures." *Neural computation* 31.7 (2019), pp. 1235–1270.

[290] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. "Ts2vec: Towards universal representation of time series." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 8. 2022, pp. 8980–8987.

[291] 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).

[292] Daochen Zha, Kwei-Herng Lai, Kaixiong Zhou, and Xia Hu. "Towards similarity-aware time-series classification." In: *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*. SIAM. 2022, pp. 199–207.

[293] Mengyue Zha, SiuTim Wong, Mengqi Liu, Tong Zhang, and Kani Chen. "Time series generation with masked autoencoder." *arXiv preprint arXiv:2201.07006* (2022).

[294] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. "An end-to-end deep learning architecture for graph classification." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[295] Qi Zhang, Jianlong Chang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. "Spatio-temporal graph structure learning for traffic forecasting." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 01. 2020, pp. 1177–1185.

[296] Xiang Zhang, Marko Zeman, Theodoros Tsiligkaridis, and Marinka Zitnik. "Graph-guided network for irregularly sampled multivariate time series." *arXiv preprint arXiv:2110.05357* (2021).

[297] Xiang Zhang, Ziyuan Zhao, Theodoros Tsiligkaridis, and Marinka Zitnik. "Self-supervised contrastive pre-training for time series via time-frequency consistency." *Advances in Neural Information Processing Systems* 35 (2022), pp. 3988–4003.

[298] Yuan Zhang. "ATTAIN: Attention-based Time-Aware LSTM Networks for Disease Progression Modeling." In: *In Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI-2019), pp. 4369-4375, Macao, China.* 2019.

[299] Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. "Convolutional neural networks for time series classification." *Journal of Systems Engineering and Electronics* 28.1 (2017), pp. 162–169.

[300] 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 (2019), pp. 3848–3858.

[301] Zhe Zhao, Paul Resnick, and Qiaozhu Mei. "Enquiring minds: Early detection of rumors in social media from enquiry posts." In: *Proceedings of the 24th International Conference on World Wide Web*. 2015, pp. 1395–1405.

[302] 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*. Vol. 34. 01. 2020, pp. 1234–1241.

[303] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. "Improving the robustness of deep neural networks via stability training." In: *Proceedings of the ieee conference on computer vision and pattern recognition*. 2016, pp. 4480–4488.

[304] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. "Time series classification using multi-channels deep convolutional neural networks." In: *International conference on web-age information management*. Springer. 2014, pp. 298–310.

[305] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. "Exploiting multi-channels deep convolutional neural networks for multivariate time series classification." *Frontiers of Computer Science* 10 (2016), pp. 96–112.

[306] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. "Learning deep features for discriminative localization." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2921–2929.

[307] 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*. Vol. 35. 12. 2021, pp. 11106–11115.

[308] Yunyue Zhu and Dennis Shasha. "Statstream: Statistical monitoring of thousands of data streams in real time." In: *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier. 2002, pp. 358–369.

[309] Eric Zivot and Jiahui Wang. "Vector autoregressive models for multivariate time series." *Modeling financial time series with S-PLUS®* (2006), pp. 385–429.

[310] 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).

COLOPHON

This document was typeset with the typographical style provided by `classicthesis`, a design created by André Miede and Ivo Pletikosić. The aesthetic inspiration for this style draws from Robert Bringhurst's influential work on typography, "*The Elements of Typographic Style*". The `classicthesis` package is compatible with both LaTeX and LyX, and you can find it at:

https://bitbucket.org/amiede/classicthesis/

Final Version as of January 19, 2024 (classicthesis v4.6).

ECOLE DOCTORALE

Titre : Vers des Méthodes Robustes d'Apprentissage Profond pour les Données de Séries Temporelles et leurs Applications

Mots clés : Apprentissage automatique, Apprentissage profond, Séries temporelles, Réseaux de neurones

Résumé : Les données de séries temporelles sont abondantes dans divers domaines, notamment la finance, l'énergie et les réseaux sociaux. L'extraction de connaissances à partir des séries temporelles, pour effectuer des tâches ultérieures telles que la prévision, l'interpolation et la classification, est donc devenue essentielle dans plusieurs domaines scientifiques et industriels. Les modèles d'apprentissage profond (DL), se sont révélés être des outils puissants dans diverses applications de données, car ils fournissent des méthodes flexibles pour l'extraction de connaissances et peuvent facilement apprendre à partir de sources de données multiples. Cependant, les modèles d'apprentissage profond nécessitent généralement de grands ensembles de données pour une meilleure généralisation et manquent souvent d'interprétabilité, de sorte que leur application industrielle reste limitée.

En ce qui concerne plus particulièrement la modélisation des séries temporelles, l'adaptabilité des réseaux neuronaux à la dynamique à temps continu de ces données reste une question ouverte. Pour combler cette lacune, il est essentiel d'évaluer systématiquement l'efficacité des architectures de DL en fonction de diverses caractéristiques et distorsions des séries temporelles, telles que le bruit, l'échantillonnage irrégulier, les fortes corrélations inter-variables et inter-temporelles et les connaissances préalables.

Nos contributions comprennent le développement de fonctions de perte robustes, l'adaptation d'architectures neuronales, par ex. CNNs, pour un échantillonnage irrégulier, et la création d'algorithmes capables de capturer des dépendances temporelles et inter-variables complexes. L'application s'étend aux données spatio-temporelles et aux systèmes dynamiques, en intégrant des modèles basés sur la physique.

En conclusion, la thèse fournit un examen complet des approches d'apprentissage profond de pointe pour la modélisation des séries temporelles, révélant les limites, proposant de nouvelles contributions, et esquissant des orientations futures, y compris l'exploration de la modélisation générative, pour améliorer de manière significative la robustesse de la modélisation neuronale et automatique pour les données de séries temporelles.

Title : Towards Robust Deep Learning Methods for Time Series Data and their Applications

Keywords : Machine Learning, Deep learning, Time Series, Neural Networks

Abstract : Time series data is abundant in various fields including finance, energy, and social networks. Extracting knowledge from time series, to perform subsequent tasks such as forecasting, interpolation, and classification, has therefore become essential in several scientific and industrial domains. Deep Learning (DL) models, have emerged as powerful tools in various data applications since they provide flexible methods for knowledge extraction and can easily learn from multiple sources of data. However, DL models typically require large datasets for improved generalization and often lack interpretability, thus their industrial application remains limited.

Specifically for time series modeling, the adaptability of neural networks to the continuous-time dynamics of such data remains an open question. To bridge this gap, systematically assessing the effectiveness of DL architectures under diverse time series characteristics and distortions, such as noise, irregular sampling, strong inter-variable and inter-temporal correlations and prior knowledge, is crucial. The current study critically evaluates the performance of existing DL models across these challenges and proposes new modeling approaches.

Our contributions include the development of robust loss functions, the adaptation of neural architectures, e.g., CNNs, for irregular sampling, and the creation of algorithms capable of capturing intricate temporal and inter-variable dependencies. The application extends to spatio-temporal data and dynamic systems, integrating physics-informed models.

In conclusion, the thesis provides a comprehensive examination of cutting-edge deep learning approaches for time series modeling, revealing limitations, proposing novel contributions, and outlining future directions, including the exploration of generative modeling, to significantly enhance the robustness of neural and automatic modeling for time series data.

Institut Polytechnique de Paris
91120 Palaiseau, France