Learning neural representation for time series

Étienne Le Naour

Sorbonne Université

Doctoral School **École Doctorale Informatique, Télécommunications et Electronique (ED130)**
Joint Research Unit *Institut des Systèmes Intelligents et de Robotique*

Learning Neural Representations for Time Series

By **Etienne Le Naour**

Academic Field **Computer Science**

Thesis supervised by	Vincent Guigue	Supervisor
	Nicolas Baskiotis	Co-Monitor

Committee members.

Referees	Germain Forestier	Professor at Université de Haute-Alsace
	Romain Tavenard	Professor at Université de Rennes 2
Examiners	Isabelle Bloch	Professor at Sorbonne Université *Committee President*
	Marc Sebban	Professor at Université de Saint-Étienne
Supervisors	Vincent Guigue	Professor at AgroParisTech
	Nicolas Baskiotis	Associate Professor at Sorbonne Université
Invited guests	Ghislain Agoua	Research Scientist at EDF
	Patrick Gallinari	Professor at Sorbonne Université

Sorbonne Université

École Doctorale Informatique, Télécommunications et Electronique (ED130)

Laboratoire *Institut des Systèmes Intelligents et de Robotique*

Apprentissage de représentations neuronales pour les séries temporelles

---

Par **Etienne Le Naour**

Thèse de doctorat d'**Informatique**

Thèse dirigée par	Vincent Guigue	Directeur de thèse
	Nicolas Baskiotis	Co-encadrant

Membres du jury.

Rapporteurs	Germain Forestier	Professeur à l'Université de Haute-Alsace
	Romain Tavenard	Professeur à l'Université de Rennes 2
Examinateurs	Isabelle Bloch	Professeure à Sorbonne Université *Présidente du jury*
	Marc Sebban	Professeur à l'Université de Saint-Étienne
Superviseurs	Vincent Guigue	Professeur à AgroParisTech
	Nicolas Baskiotis	Maître de conférence à Sorbonne Université
Invités	Ghislain Agoua	Ingénieur Chercheur à EDF
	Patrick Gallinari	Professeur à Sorbonne Université

À Jade et à mes parents

Learning neural representations for time series

Abstract

Time series analysis has become increasingly important in various fields, including industry, finance, and climate science. The proliferation of sensors and the data heterogeneity necessitate effective time series modeling techniques. While complex supervised machine learning models have been developed for specific tasks, representation learning offers a different approach by learning data representations in a new space without explicitly focusing on solving a supervised task. The learned representation is then reused to improve the performance of supervised tasks applied on top of it. Recently, deep learning has transformed time series modeling, with advanced models like convolutional and attention-based neural networks achieving state-of-the-art performance in classification, imputation, or forecasting. The fusion of representation learning and deep learning has given rise to the field of neural representation learning. Neural representations have a greater ability to extract intricate features and patterns compared to non-neural representations, making them more powerful and effective in handling complex time series data. Recent advances in the field have significantly improved the quality of time series representations, enhancing their usefulness for various downstream tasks.

This thesis focuses on advancing the field of neural representation learning for time series, targeting both industrial and academic needs. This research addresses open problems in the domain, such as creating interpretable neural representations, developing continuous time series representations that handle irregular and unaligned time series, and creating adaptable models for distribution shifts. By tackling these challenges, the thesis aims to bridge the gap between cutting-edge academic research and practical industrial applications.

This manuscript offers multiple contributions to tackle the previously mentioned challenges in neural representation learning for time series. (i) Firstly, we propose an interpretable discrete neural representation model for time series based on a vector-quantization encoder-decoder architecture, which facilitates interpretable classification. (ii) Secondly, we design a continuous implicit neural representation model, called TimeFlow, for time series imputation and forecasting that can handle unaligned and irregular samples. This model leverages per-context modulations, enabling it to adapt to new samples and unseen contexts by adjusting the representations. (iii) Lastly, we demonstrate that TimeFlow learns relevant features, making the representation space effective for downstream tasks such as data generation.

Apprentissage de représentations neuronales pour les séries temporelles

Résumé

L'analyse des séries temporelles est cruciale dans divers domaines tels que l'industrie, la finance et la science du climat. La prolifération des capteurs et l'hétérogénéité croissante des données nécessitent des techniques efficaces de modélisation des séries temporelles. Alors que des modèles complexes d'apprentissage automatique supervisé ont été développés pour des tâches spécifiques, l'apprentissage de représentation offre une approche différente en apprenant des représentations des données dans un nouvel espace sans se concentrer explicitement sur une tâche supervisée. Par la suite, les représentations extraites sont réutilisées pour améliorer les performances des tâches supervisées en aval. Récemment, l'apprentissage profond a révolutionné la modélisation des séries temporelles, avec des modèles tels que les réseaux de neurones convolutifs et les réseaux basés sur les mécanismes d'attention. Ces modèles atteignent des performances à l'état de l'art pour les tâches de classification, d'imputation ou encore de prévision. La fusion de l'apprentissage de représentation et de l'apprentissage profond a donné naissance au domaine de l'apprentissage de représentation neuronale. Les représentations neuronales pour les séries temporelles, comparées aux représentations non neuronales, possèdent une meilleure capacité à extraire des caractéristiques complexes au sein d'un nouvel espace structuré. Les progrès récents dans ce domaine ont considérablement amélioré la qualité des représentations des séries temporelles, améliorant ainsi leurs utilités pour les tâches en aval.

Cette thèse vise à contribuer au domaine de l'apprentissage des représentations neuronales pour les séries temporelles, en ciblant à la fois des besoins industriels et académiques. Ce manuscrit aborde des problèmes ouverts dans le domaine, tels que la construction de représentations neuronales interprétables, le développement de modèles de représentations continues capables d'apprendre à partir de séries temporelles irrégulières et non alignées, ainsi que la création de modèles adaptés pour les changements de distribution.

Ce manuscrit propose plusieurs contributions pour relever les défis mentionnés ci-dessus. (i) Premièrement, nous proposons un modèle de représentation neuronale discrète et interprétable pour les séries temporelles, basé sur une architecture encoder-decoder avec un mécanisme de discrétisation. (ii) Deuxièmement, nous concevons un modèle continu en temps de représentation neuronale implicite pour l'imputation et la prévision des séries temporelles qui peut traiter des échantillons non alignés et irréguliers. Ce modèle se base sur des représentations modulables, ce qui lui permet de s'adapter à de nouveaux échantillons et à des contextes inédits en ajustant les représentations. (iii) Enfin, nous démontrons que le modèle proposé ci-dessus apprend des caractéristiques pertinentes, créant un espace de représentation structuré et efficace pour des tâches en aval telle que la génération de données.

Remerciements

Je tiens à remercier chaleureusement toutes les personnes qui m'ont accompagné de près ou de loin lors de cette aventure qu'est le doctorat.

Tout d'abord, je souhaite remercier mes encadrants académiques : Vincent, pour nos échanges scientifiques et son aide précieuse pour sortir des impasses, ainsi que Nicolas, pour ses conseils et propositions tout au long de la thèse.

Ensuite, je tiens à exprimer ma gratitude envers mon encadrant industriel, Ghislain, pour m'avoir proposé ce sujet de thèse passionnant. Merci de m'avoir fait confiance, de m'avoir accordé une grande liberté, et d'avoir toujours été présent.

Je remercie aussi l'ensemble des membres du jury. Un grand merci à Romain Tavenard et Germain Forestier d'avoir accepté de rapporter cette thèse. Vos travaux académiques ont grandement contribué à ma compréhension du machine learning pour les séries temporelles. Je tiens également à remercier Isabelle Bloch et Marc Sebban d'avoir accepté d'examiner ce travail de recherche.

Je souhaite ensuite remercier l'équipe MLIA pour son environnement propice au travail. J'ai une pensée particulière pour Yuan, Léon, et Louis pour nos collaborations scientifiques et la qualité de nos échanges. Travailler avec vous a été un vrai plaisir. Une mention spéciale pour Louis, nos discussions permanentes et nos travaux m'ont énormément fait progresser ; merci à toi. Je voudrais également remercier chaleureusement Patrick pour son aide et les nombreux enseignements tirés de notre collaboration. Je remercie aussi les autres membres de l'équipe pour les moments conviviaux partagés, notamment Paul, Lise, Armand, Tanguy, Thomas, Tristan et tous les autres.

Mes remerciements vont également à toutes les personnes que j'ai pu côtoyer à EDF, en particulier les membres de l'équipe SOAD. J'ai eu beaucoup de plaisir à travailler avec vous pendant trois ans et je suis heureux de continuer l'aventure avec vous. Un merci spécial à Tahar pour ses conseils avisés, nos échanges scientifiques et sa grande disponibilité. Je remercie aussi les autres doctorants rencontrés à EDF,

notamment Julie avec qui j'ai eu une très belle collaboration sur la prévision de production éolienne, et Adrien avec qui j'ai beaucoup échangé et appris.

Une pensée particulière pour mes amis qui m'ont accompagné tout au long de cette aventure : Camille, Julien, Salim, Fabien, Jules, Louis, Xavier et tant d'autres. Je pense également à tous les membres de mon équipe de futsal, avec qui j'ai passé de beaux moments me permettant de penser à autre chose qu'au travail.

Enfin, je tiens à remercier ma famille : Emmanuelle, Elsa, Jean, Grégory ainsi que les petits neveux et nièce arrivés en cours de route. Les moments partagés avec vous sont un vrai bonheur. Un merci du fond du cœur à mes parents pour leur soutien inconditionnel, l'énergie déployée pour le bien-être de leurs enfants, et les belles valeurs transmises. Je remercie aussi Matthieu, Nathalie et Alban pour votre accueil et tous les moments partagés depuis de nombreuses années, vous êtes comme une deuxième famille pour moi. Enfin, je remercie Jade de partager ma vie depuis toutes ces belles années. Il est difficile de trouver les mots pour décrire tout ce que tu m'apportes, mais je suis convaincu que tu es en grande partie responsable du fait que cette thèse ait été agréable à mener. Tu es et seras toujours une source de motivation et de réconfort.

Pensée pour Nalou qui m'accompagne à l'heure où j'écris ces lignes.

Symbols

The manuscript uses the following symbols throughout, with additional notations occasionally defined within specific chapters or sections.

Domain	Symbol	Description
Time series	$\mathbf{x}$	Time series
	$\mathbf{z}$	Time series representation (sometimes referred to as code)
	t	Timestamp index
	T	Number of timestamps
	L	Look-back window length (forecasting)
	H	Horizon window length (forecasting)
	$\mathcal{T}$	Temporal support
	$\mathbb{A}$	Symbolic values support
	c	Number of input channels
	y	Time series label
General	i, j	Samples index
	n	Number of samples
	$\mathcal{L}$	Loss
	$\mathbf{W}$, $\mathbf{b}$	Weights and bias of a linear layer
	$p(\mathbf{x})$	Probabilistic distribution
	$\hat{p}_\theta(\mathbf{x})$	Probabilistic distribution approximation
	$\mathcal{N}(.;.)$	Gaussian distribution
	∇	Gradient
	τ	Sampling rate
Neural network	θ, θ'	Weights of a neural network
	ϕ	Encoder
	ψ	Decoder
	$h_{\mathbf{w}}$	Hypernetwork
	α	Learning rate
	$\mathcal{B}$	Batch

Contents

List of Figures

List of Tables

Part I

Research Context

Chapter 1

Introduction

1.1 Context and challenges

This work falls within the research field of neural representation learning for time series while also addressing industrial needs. In this section, we outline the context that motivated this thesis.

1.1.1 Motivating neural representation for time series data

Time series analysis and modeling are crucial in diverse fields such as industry, finance, and climate science. These data consist of sequences of measurements taken at successive points in time and can reflect phenomena caused by human activities, physical processes, or a combination of both. For example, in the electricity industry, time series can represent the wind power generation of a wind farm over time.

The increasing variety, heterogeneity, and number of deployed sensors present new challenges in dealing with real-world problems. As the volume of time series data grows, there is a need for effective modeling of these data. Complex machine learning models have been developed to address this need, excelling at tasks such as forecasting future values, classifying sequences, and imputing missing values. These models are typically task-specific and not easily reusable for other tasks or settings; we refer to them as supervised models.

In contrast, a new paradigm has emerged with representation learning models (also known as unsupervised or self-supervised models). Time series representation models aim to learn data representations in a new space without explicitly solving a particular supervised task. These representations capture the essential features and patterns in the data, projecting the time series into a new space called the representation space. In a second stage, the learned representations can be reused to solve specific downstream tasks, see Figure 1.1 for a visualization of the representation learning process for time series.

Operating in a two-stage manner, first learning the representation and then solving the task, offers various benefits. For instance, the representation can be reused for several downstream tasks, avoiding the need to retrain a new model each time and thus saving computational costs. This approach can lead to better task performance as the critical features are already identified during the representation learning stage. It is particularly useful in few-shot learning scenarios. In addition, for tasks such as anomaly detection and generation, the representation space may be more meaningful than the original time series space.

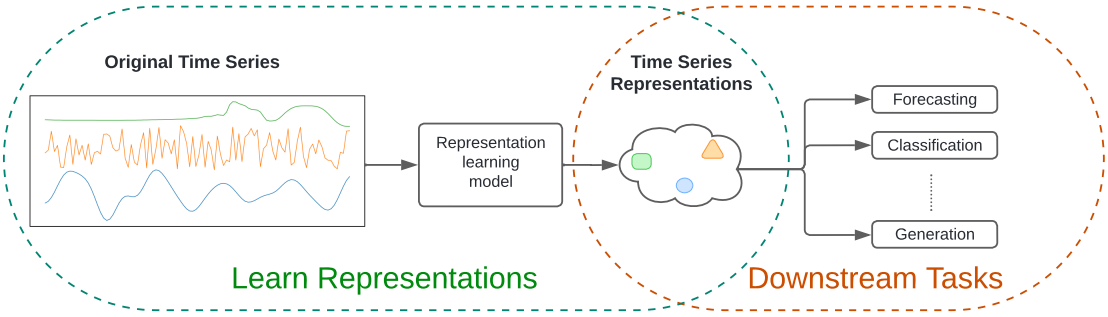


Figure 1.1: Visualization of the time series representation learning pipeline. Stage 1: Learn the time series representation through unsupervised learning. Stage 2: Train a model using this representation to address the downstream task. This graphic is inspired by Trirat et al. (2024).

Recently, the advent of deep learning has revolutionized time series modeling. Powerful deep learning models, including Multi-Layers (MLPs) (Haykin, 1994), Convolutional Neural Networks (CNNs) (LeCun et al., 1995), Recurrent Neural Networks (RNNs) (Schuster and Paliwal, 1997), attention-based neural networks (transformers) (Vaswani et al., 2017), have achieved state-of-the-art performance in classification (Ismail Fawaz et al., 2020), imputation (Cao et al., 2018), and forecasting (Nie et al., 2022). These models capture complex temporal patterns and dependencies, surpassing traditional machine learning approaches. Consequently, neural representation learning for time series has emerged, leading to substantial improvements in the quality of time series representations. These models enhance downstream task performance (Yue et al., 2022), reduce the need for labeled data

during training (Franceschi et al., 2019), and facilitate the transfer of learned representations across different datasets (Nie et al., 2022). Today, neural representation learning is an active field of research. The taxonomy of its various branches is illustrated in Figure 1.2.

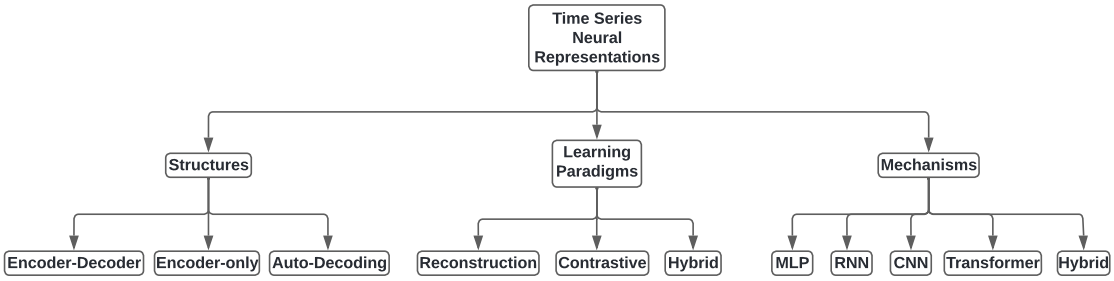


Figure 1.2: Taxonomy of research branches in neural representation learning for time series.

The primary objective of this thesis is to develop new representation learning models for time series that meet industrial needs. This research is conducted in collaboration with Electricité de France (EDF), a company that collects millions of time series annually and has significant requirements for advanced neural representation modeling. By addressing EDF's specific challenges, this work aims to bridge the gap between cutting-edge academic research and practical industrial applications.

1.1.2 Industrial context and objectives

EDF is a French public energy company that operates as both a producer and supplier of electricity. EDF collects and analyzes large volumes of time series data from various sources, including: • Electricity load curves from households and businesses. • Sensors on production sites such as wind farms, solar farms, nuclear and hydraulic power plants.

EDF has developed strong expertise in time series modeling for tasks such as forecasting, classification, and anomaly detection. Existing neural representation learning models for time series can address some of EDF's current challenges, such as creating efficient classification algorithms with limited labeled data. However, there are several unresolved issues that this research aims to tackle.

1. **Interpretable neural representations.** While deep learning methods are highly effective, decision-makers need to understand the model's outputs. Therefore, it is crucial to develop interpretable neural representations of time series, achieving high performance and providing interpretable results.

2. **Adaptable models for new time series and distribution shifts.** EDF frequently encounters new time series samples, such as data from new clients

or new sensors. Retraining a model for each new sample is impractical. Therefore, developing models that can quickly adapt to new samples or contexts is essential. Leveraging neural representation learning is a promising avenue to address this challenge effectively.

3. **Handling irregular and unaligned time series.** EDF collects numerous unaligned and irregular time series. For instance, household load curves will soon be collected at 15-minute intervals instead of 30-minute intervals, and sensors at power production sites may experience irregular sampling. The challenge is to develop a model that can effectively capture neural representations of all this data, regardless of these irregularities. Consequently, the irregularities would not affect downstream tasks using these representations.

1.2 Contributions

This thesis aims to address the above challenges by designing new neural representation models for time series.

(i) First, we propose a discrete interpretable neural representation architecture for time series, which is used subsequently for interpretable classification. (ii) Then, we design a continuous neural model for time series imputation and forecasting that can handle unaligned and irregular samples. This model, called TimeFlow, leverages the representation of time series data, allowing it to adapt to new samples and unseen contexts by adjusting these representations. (iii) Finally, we show that the representation learned by TimeFlow captures relevant features, enabling the representation space to be effectively utilized for downstream tasks such as data generation.

Our contributions, which will be discussed in detail in Part II, are summarized below.

Interpretable time series neural representation for classification purposes. Deep learning has advanced time series representation by identifying complex patterns but lacks interpretability. While some traditional representation learning methods offer interpretability, they fail to capture complex patterns. This contribution introduces a set of requirements for creating interpretable neural representations of univariate time series. We propose a novel unsupervised neural architecture that produces consistent, discrete, interpretable, and visualizable representations. The model is task-agnostic, ensuring robustness. We demonstrate its effectiveness through classification experiments on the University of California Riverside archive datasets, comparing it to other interpretable and neural representation models. Our model quantitatively outperforms other interpretable approaches, and additional qualitative evaluations confirm its interpretability.

Le Naour, E., Agoua, G., Baskiotis, N., and Guigue, V. **Interpretable time series neural representation for classification purposes.** *IEEE 10th International Conference on Data Science and Advanced Analytics (IEEE DSAA)* 2023. Best research paper award.

Time series continuous modeling for imputation and forecasting with implicit neural representations. In this contribution, we present a novel modeling approach called "TimeFlow" for time series imputation and forecasting that addresses challenges in real-world data, such as irregular samples, missing data, and unaligned measurements from multiple sensors. Our method utilizes a continuous-time model of the series' evolution dynamics and incorporates conditional, implicit neural representations for sequential data. A modulation mechanism, driven by meta-learning, allows adaptation to new samples and long-term extrapolation beyond observed time windows. This model offers a flexible and unified framework for both imputation and forecasting tasks in diverse scenarios. It demonstrates state-of-the-art performance on classic benchmarks and surpasses other time-continuous models.

Le Naour, E., Serrano, L., Migus, L., Yin, Y., Agoua, G., Baskiotis, N., Gallinari, P., and Guigue, V. **Time Series Continuous Modeling for Imputation and Forecasting with Implicit Neural Representations.** *Transactions on Machine Learning Research (TMLR)* 2024.

Exploring the representation learning capabilities of TimeFlow. We then explore TimeFlow's representation learning capabilities, demonstrating how it effectively handles irregular data and extracts semantically rich representations. Through post-hoc experiments in latent space, we demonstrate TimeFlow's robustness and ability to capture the underlying structure of time series. We demonstrate the practical utility of the learned representations in downstream tasks such as time series generation through a structured two-step approach. Experiments validate the utility of performing generation in latent space rather than directly in time series space.

In an internal collaboration at EDF, we explored the application of implicit neural representation learning models, combined with a regressor, to predict national wind production from wind speed maps. Our approach was compared against purely supervised deep learning methods and automated deep learning methods. As this work is not based on representation learning for time series, we present the results in Appendix A.

Keisler, J., Le Naour, E. **WindDragon: Enhancing Wind Power Forecasting with Automated Deep Learning.** *International Conference on Learning Representations (ICLR), Tackling Climate Change with Machine Learning Workshop*, 2024.

1.3 Structure of the thesis

This thesis is organized as follows.

- Chapter 2 introduces essential concepts for the manuscript, starting with the definition and historical treatment of time series and reviews the evolution of models due to massive data collection. Then, the chapter examines time series representations, their benefits, and limitations, and explores how neural representations address these issues. It concludes by identifying open problems in time series neural representation.

- Chapter 3 presents a novel unsupervised neural architecture that creates consistent, discrete, interpretable, and visualizable representations. The model's robustness and effectiveness are demonstrated through classification experiments.

- Chapter 4 introduces TimeFlow, a novel model for time series imputation and forecasting. TimeFlow addresses real-world challenges like irregular samples, missing data, and unaligned sensor measurements using a continuous-time model through implicit neural representations.

- Chapter 5 explores TimeFlow's ability to extract semantically rich representations from potential irregular time series. We show the practical utility of these representations in downstream tasks, such as unconditional time series generation using a two-step approach.

- Chapter 6 presents a global summary of the thesis and identifies some perspectives of future works based on the observed limitations.

Chapter 2

Background and Related Work

This chapter introduces basic concepts that are essential for understanding the rest of the manuscript. Section 2.1 begins by defining time series, providing practical examples of their applications, and discussing their historical treatment. We then explore how the massive collection of data has driven the evolution of models to adapt to time series datasets, leading to the emergence of new tasks. Next, in Section 2.2, we discuss the motivations of time series representations compared to traditional supervised machine learning models. We review several time series representation methods, discuss their advantages, and highlight existing limitations. In the final Section 2.3, we introduce deep learning mechanisms for time series and discuss their effectiveness. We then define neural representations, discuss their advantages over non-neural representations, and present popular methods. Finally, we conclude this section by identifying open problems in learning neural representations for time series.

2.1 Machine learning for time series: an introduction

This section introduces time series data, offers a historical overview of their analysis, and outlines the tasks they are used for.

2.1.1 Single time series analysis

A time series is a sequential collection of observation points of a measured underlying phenomenon recorded at successive timestamps. These observations can typically be obtained at regular intervals, such as hourly, daily, monthly, or annually, depending on the nature of the phenomenon being studied. Each data point in a time series is associated with a specific timestamp or time index that reflects the temporal order of the observations. We define a time series as below.

Definition 2.1.1 (Time Series). *Consider a temporal phenomenon represented by a continuous function of time* $\mathbf{x}$*:* $t \in \mathcal{T} \rightarrow \mathbf{x_t} \in \mathbb{R}^c$ *where* c *denotes the number of input channels. Given a sequence of observed timestamps* $(t_1, t_2, \ldots, t_T)$*. The time series* $\boldsymbol{x}$*, can be defined as* $\boldsymbol{x} = (\mathbf{x_{t_1}}, \mathbf{x_{t_2}}, \ldots, \mathbf{x_{t_T}}) \in \mathbb{R}^{c \times T}$*.*

Time series data can represent various phenomena, including economic indicators, weather records, stock prices, physiological signals, and more. The underlying measured phenomena can be caused by human activities (such as national nuclear power generation, Figure 2.1), physical phenomena (such as the measured temperature at a location), or a combination of both (such as national hydraulic power generation, Figure 2.2).

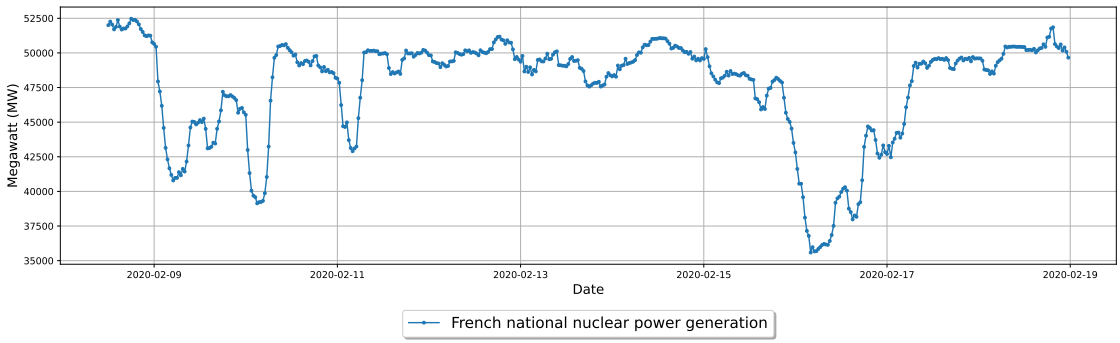


Figure 2.1: Visualization of French national nuclear power generation time series from February 9th, to February 19th, 2020. Data source: eCO2mix.

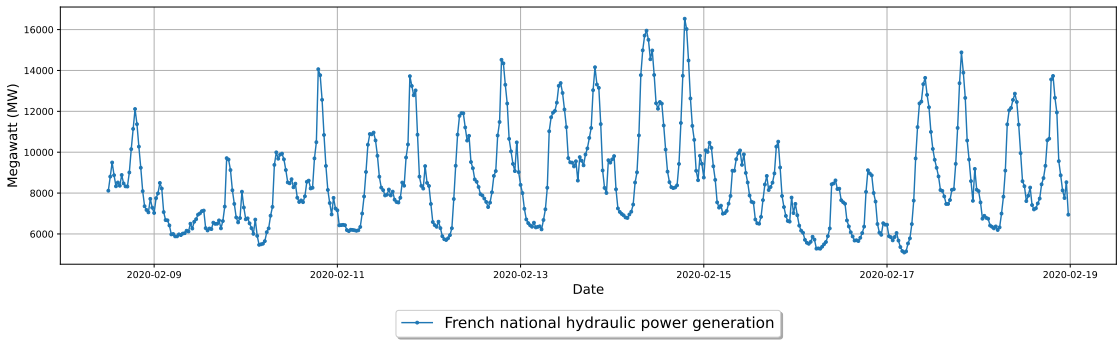


Figure 2.2: Visualization of french national hydraulic power generation time series from February 9th, to February 19th, 2020. Data source: eCO2mix.

Historically, time series analysis has predominantly been associated with forecasting tasks (see Definition 2.1.2). Time series data are characterized by their sequential nature and dependence on past observations. Hence, forecasting future values based on historical patterns is a natural task. Forecasting is paramount in fields as diverse as economics, epidemiology, meteorology, and engineering, where accurate predictions of future outcomes are essential for decision making and planning.

Definition 2.1.2 (Time series forecasting)**.** *Time series forecasting consists in predicting future values of a temporal phenomenon for a horizon window of length* H $(\mathbf{x_{t+1}}, \mathbf{x_{t+2}}, \ldots, \mathbf{x_{t+H}})$ *based on a sequence of past observations, the look-back window, of length* L $(\mathbf{x_{t-(L-1)}}, \mathbf{x_{t-(L-2)}}, \ldots, \mathbf{x_t})$.

As a consequence, early research and development efforts in time series analysis focused on building models specifically tailored for forecasting, relying primarily on autoregressive mechanisms. Among the early approaches, traditional statistical models such as Exponential Smoothing (Brown, 1959; Holt, 2004) and Autoregressive Integrated Moving Average (ARIMA) (Box et al., 2015) stood out. Exponential Smoothing, introduced in the late 1950s, provides a simple and intuitive approach by using weighted averaging of past observations to predict the future. Conversely, ARIMA models, developed in the 1970s, incorporate differencing, autoregressive, and moving average components to capture the temporal dependencies inherent in the data. These methods established the foundation for time series forecasting and are still widely used today. They are valued for their simplicity, interpretability, and effectiveness in capturing various patterns within time series data (see Figure 2.3).

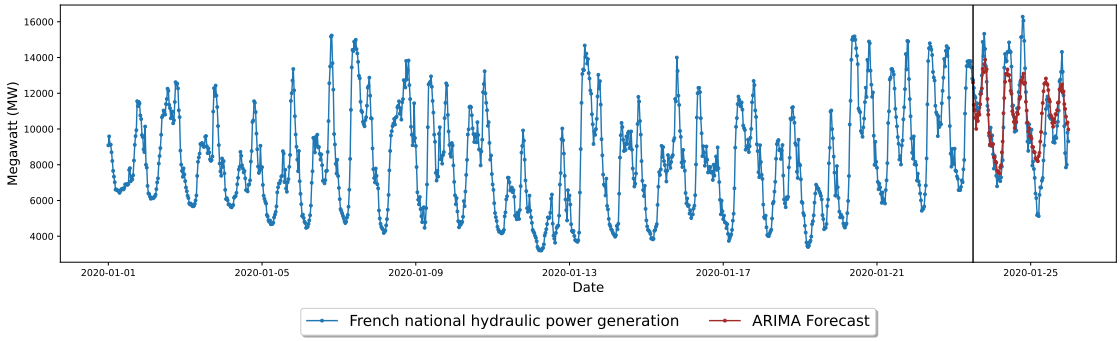


Figure 2.3: Visualization of an ARIMA($p = 24, d = 1, q = 1$) forecast on the french national hydraulic power generation time series.

Over time, with the proliferation of data collection on an unprecedented scale, the focus of time series analysis has shifted. While traditional statistical approaches focused primarily on individual time series, contemporary analysis now encompasses a broader perspective. With access to vast amounts of data, attention has increasingly turned to the analysis of jointly observed time series. Rather than isolating a single time series for analysis, researchers and practitioners now seek to understand the collective behavior and patterns exhibited by groups of related time series. This encourages the exploration of time series methods capable of capturing the diversity of entire datasets. This development has paved the way for the emergence of machine learning applied to time series datasets.

2.1.2 Time series datasets

A time series dataset can be defined as a collection of time series that share a similar structure or measure similar phenomena. We define it below.

Definition 2.1.3 (Time series dataset)**.** *A time series dataset can be defined as a collection of n time series $\{\boldsymbol{x}^{(j)}\}_{j=1}^{n}$ where each time series $\boldsymbol{x}^{(j)}$ can be written as $\boldsymbol{x}^{(j)} = (\boldsymbol{x}^{(j)}_{t_1^{(j)}}, \boldsymbol{x}^{(j)}_{t_2^{(j)}}, \ldots, \boldsymbol{x}^{(j)}_{t_T^{(j)}}) \in \mathbb{R}^{c \times T^{(j)}}$.*
We will refer to $\{\boldsymbol{x}^{(j)}\}_{j=1}^{n}$ as an unsupervised dataset in contrast to a supervised dataset $\{\boldsymbol{x}^{(j)}, y^{(j)}\}_{j=1}^{n}$ where $y^{(j)}$ can denote the label in the classification task or the target variable in the regression task associated with the time series $\boldsymbol{x}^{(j)}$.

Time series datasets are very common in many real-world applications. They can range from electricity load consumption at different locations (*Electricity* dataset[1], see Figure 2.4) to traffic road measurement at different roads (*Traffic* dataset[2]) to ECG signals at various periods (*ECG5000* dataset[3]).

[1]https://archive.ics.uci.edu/dataset/321/electricityloaddiagrams20112014
[2]https://pems.dot.ca.gov/
[3]https://www.timeseriesclassification.com/description.php?Dataset=ECG5000

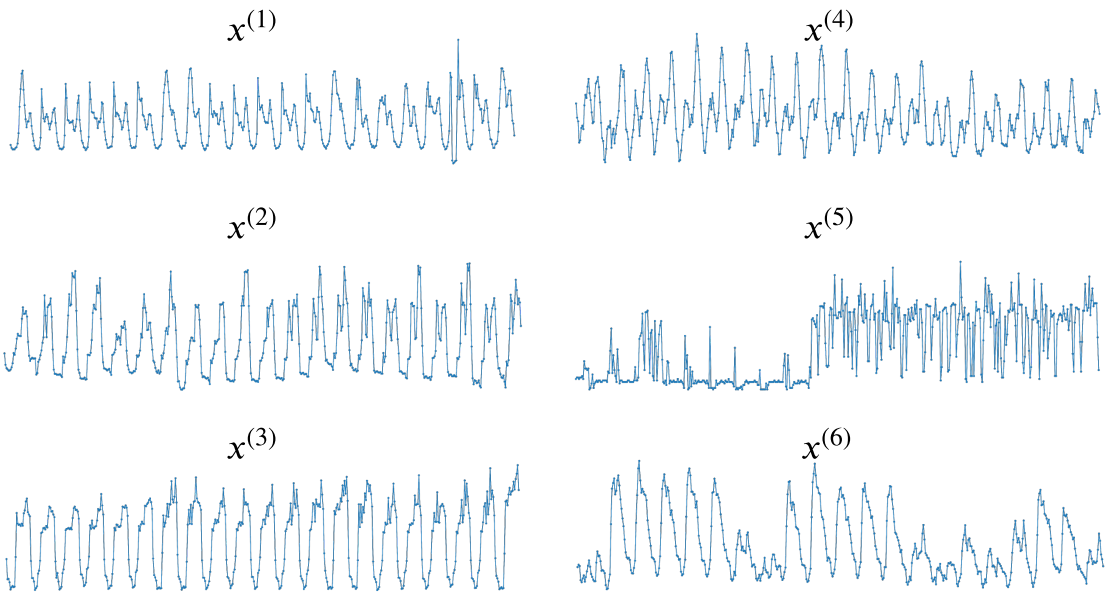


Figure 2.4: Visualization of six samples covering three-week intervals from the *Electricity* dataset.

The growing availability of time series datasets has not only improved the handling of existing tasks, such as time series imputation (see Definition 2.1.4), but has also led to the emergence of new tasks, such as classification (see Definition 2.1.5) and, more recently, generation (see Definition 2.1.6).

Definition 2.1.4 (Time series imputation)**.** *For a time series $\boldsymbol{x}^{(j)}$, we denote the set of observed timestamps as $\mathcal{T}_{in}^{(j)}$ and the dense support of timestamps as $\mathcal{T}^{(j)}$ ($\mathcal{T}_{in}^{(j)} \subset \mathcal{T}^{(j)}$). The observed time grids may be irregularly spaced and can vary across different time series ($\mathcal{T}_{in}^{(j_1)} \neq \mathcal{T}_{in}^{(j_2)}$, for $j_1 \neq j_2$). The objective of time series imputation is to predict the missing value $\boldsymbol{x}_t^{(j)}$ for any unobserved $t \in \mathcal{T}^{(j)}$ based on the observed values.*

Definition 2.1.5 (Time series classification)**.** *Let $y^{(j)} \in \mathcal{C}$ ($\mathcal{C} \subset \mathbb{N}$) denote the label associated with the time series $\boldsymbol{x}^{(j)}$. The objective of the classification task is to learn a classifier $f(.)$ that establishes a mapping between the time series and its label. Formally, we seek to find f such that $f(\boldsymbol{x}^{(j)}) = y^{(j)}$, $\forall j \in \{1, ..., n\}$.*

Definition 2.1.6 (Time series generation)**.** *We consider a time series dataset* $\{\boldsymbol{x}^{(j)}\}_{j=1}^{n}$ *which are sampled from an unknown probability distribution* $p(\mathbf{x})$*. At training we aim to approximate the true distribution* $p(\mathbf{x})$ *with an estimated distribution* $\hat{p}_\theta(\mathbf{x})$*. At generation we aim to draw new time series from the estimated distribution* $\boldsymbol{x}^{(\boldsymbol{gen})} \sim \hat{p}_\theta(\mathbf{x})$*.*

The proliferation of time series datasets has driven the development of innovative machine learning techniques. Machine learning models such as decision trees and support vector machines (SVM) have been applied to capture the intricate temporal patterns and dependencies within time series datasets (Geurts, 2001; Kampouraki et al., 2008). By harnessing the power of machine learning approaches, researchers have gained valuable insights from time series datasets, paving the way for innovative applications in various domains like automatic classification of heart diseases based on ECG signal (Owis et al., 2002).

2.1.3 Machine learning for time series datasets

Machine Learning approaches aim to leverage the information between time series to enhance the resolution of various machine learning tasks. Rather than relying solely on autoregressive information, which is based on the observed values of each individual series, joint modelisation seeks to harness the relationships between the series. This section provides some intuition as to why time series datasets are valuable for various machine learning tasks. We start with a practical example focusing on the imputation task and then elaborate on the classification task and the concept of distance in time series, which are important in this manuscript.

2.1.3.1 Motivation of joint modelisation: time series imputation

While time series imputation methods can be purely autoregressive, the most effective approaches learn dependencies between time series samples. Leveraging relationships between time series can intuitively help fill in missing values more accurately than relying solely on autoregressive information (see Figure 2.5).

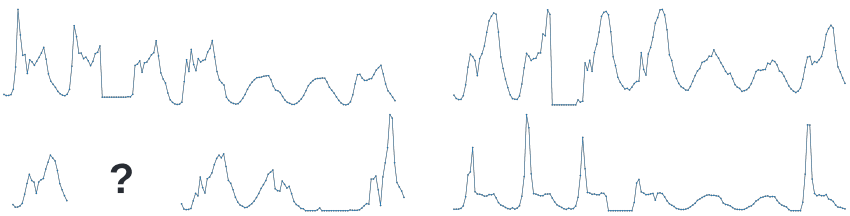


Figure 2.5: Time series missing values. The complete time series shows a repeating pattern, providing insight into imputing the incomplete time series.

Thus, numerous imputation methods for time series have been developed that model the entire time series as a closed system, such as matrix factorization methods (Grabocka et al., 2012; Mei et al., 2017). Explicitly modeling the relationships between time series in a matrix allows for an effective extraction of dependencies.

In the next section, we delve into the time series classification task, introducing concepts that will serve as recurring examples throughout the manuscript.

2.1.3.2 Introducing time series classification and distance metrics

Classification is a prevalent task in time series analysis. It finds application across diverse fields, from ECG-based identification of myocarditis to device classification via electrical signals. This task aims to discriminate time series based on their labels. However, developing universal methods for time series classification is challenging as discriminative factors can be local, global, or a mixture of both.

k-NN approaches for time series classification. A simple and widely used approach to classification problems is the k-Nearest Neighbors (k-NN) method (Fix, 1985). This method does not require any specific parameters to be learned. The unlabeled time series is assigned the average label of its k nearest neighbors.

- In the 1-NN setup, the unlabeled time series inherits the label of its nearest neighbor.
- For $k > 1$, the method involves assigning the most frequent class among the k-nearest neighbors. There is the possibility to give the more distant neighbors less weight than the closest neighbors in the majority voting procedure.

This classification method raises an intriguing question: *how do we determine the distance between two time series*? A straightforward answer is to compute the **Euclidean distance** pointwise. We formally define the Euclidian distance between two time series in Definition 2.1.7.

Definition 2.1.7 (Euclidian Distance (ED), univariate case)**.** *We consider two time series $\boldsymbol{x}^{(j)}$ and $\boldsymbol{x}^{(i)} \in \mathbb{R}^{1\times T}$. The ED between the two time series can be defined as:*

$$d_{eucl}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \left(\sum_{t=1}^{T}(x_t^{(i)} - x_t^{(j)})^2\right)^{1/2}. \tag{2.1}$$

We note that calculating the ED between two series is suitable when dealing with time series that share the same number of well-aligned timestamps. However, the ED, being the sum of pointwise distances, overlooks shape similarities between two curves. Consequently, two series exhibiting identical shapes but with a time

shift may appear significantly distant regarding the ED. This limitation can be a concern in many applications. For instance, consider a scenario where we aim to determine household ownership of an electric car based on the aggregated electricity consumption curve. In such cases, our primary interest lies in discerning the shape of the curve rather than the timing of charging activities.

The Dynamic Time Warping (DTW) distance (Vintsyuk, 1968; Sakoe and Chiba, 1978) addresses this problem of shape similarity. This distance is defined in such a way that two time series with the same shapes but with a temporal misalignment should have a distance of zero. We formally present the DTW distance in Definition 2.1.8 following the notations defined in Tavenard (2021).

Definition 2.1.8 (The DTW distance, univariate case)**.** *The DTW distance between two time series* $\boldsymbol{x}^{(j)}$ *and* $\boldsymbol{x}^{(i)}$ *aims to identify an optimal alignment path, denoted as* π^* *(illustrated in Figure 2.6). This path minimizes the cumulative Euclidean distance between corresponding points of the time series along the alignment. The formal definition of DTW is provided below:*

$$DTW(\boldsymbol{x}^{(j)}, \boldsymbol{x}^{(i)}) = \min_{\pi \in \mathcal{A}(x^{(j)}, x^{(i)})} \left(\sum_{(t,\tau)\in\pi} (x_t^{(j)} - x_\tau^{(i)})^2 \right)^{\frac{1}{2}}. \tag{2.2}$$

(i) π *represents an admissible alignment path, which is a sequence of index pairs matching between* $\boldsymbol{x}^{(j)}$ *and* $\boldsymbol{x}^{(i)}$*.*

(ii) $\mathcal{A}(\boldsymbol{x}^{(j)}, \boldsymbol{x}^{(i)})$ *denotes the set of all admissible paths.*

(iii) An admissible path π *must satisfy three conditions:*

- *The first and last pairs of* π *must match the first and last time indexes of the two time series, respectively.*
- *All time indexes of both time series must appear in the path* π*.*
- *The sequence of index pairs must be monotonically increasing, meaning that for any given element* (t_k, τ_k) *in the sequence, the possible next elements are* (t_{k+1}, τ_k)*,* (t_k, τ_{k+1})*, or* (t_{k+1}, τ_{k+1})*.*

The computation of the DTW typically involves dynamic programming, where the cumulative distance is recursively computed based on the distances of neighboring points, and the path with the minimum cumulative distance is chosen. Most of the time, the optimal path is searched within a warping window of size w (wich represent the maximum distance from the anti-diagonal path).

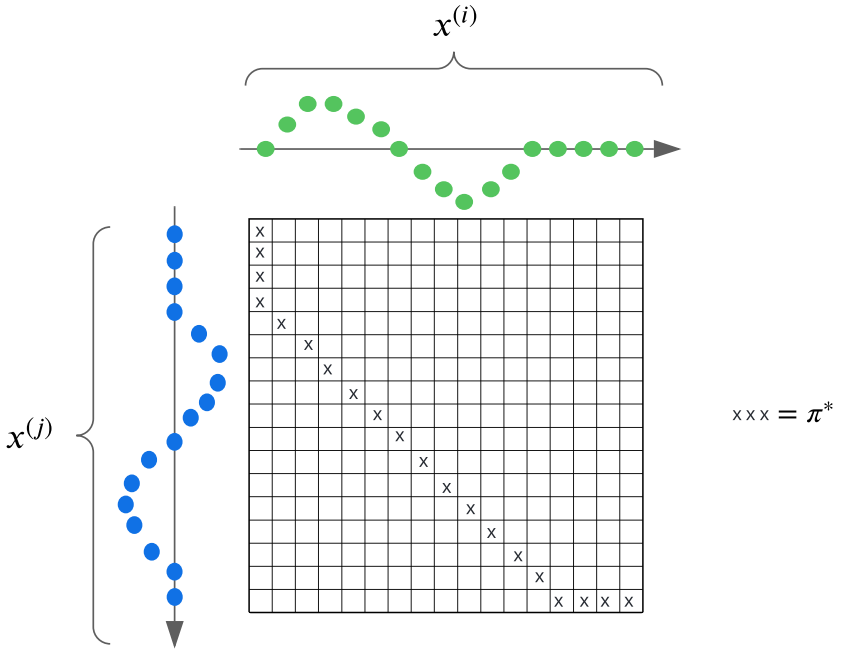


Figure 2.6: DTW explanation: illustration of the optimal warping path π^*.

Now that we have introduced the k-NN classification algorithm along with the ED and DTW distances, another question arises: *which distance measure is most appropriate for classifying time series using a* 1*-NN approach?*

Setting. We present in Figure 2.7 the classification results for the 1-NN ED, 1-NN DTW (w=100) and 1-NN DTW (learned w) on the well know University California Riverside (UCR) archive (Dau et al., 2019). We also compare to a naive baseline which always predict the majority class.

Note 2.1.1 : The UCR archive

The UCR Time Series Classification Archive is a repository of 128 time series datasets curated for the purpose of benchmarking and evaluating univariate time series classification algorithms [a]. It was created to provide a standardized platform for researchers to compare the performance of their classification methods on various time series datasets (sensor, simulated, image, motion, and so on). The time series datasets vary significantly in their length, number of samples, and classes.

[a] https://www.cs.ucr.edu/%7Eeamonn/time_series_data_2018/

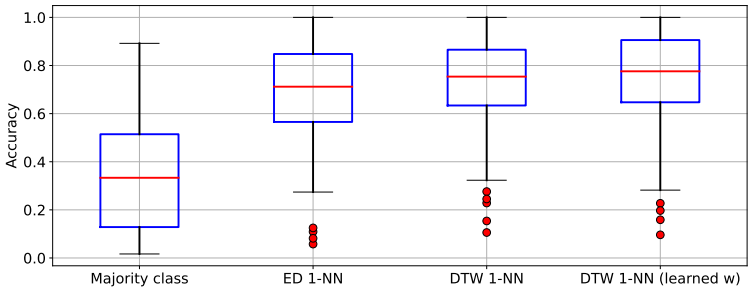


Figure 2.7: Accuracy results of the 1-NN classification methods on the 128 datasets of the UCR time series classification archive.The closer to one, the better.

Results. We note two phenomena in Figure 2.7:

(i) Applying a simple algorithm like 1-NN is considerably more efficient than consistently predicting the majority class.

(ii) Opting for DTW distance over simple ED generally yields better results on average, although this is not universally true and varies depending on the application.

Many purely supervised classification methods have been proposed to successfully improve the results on the UCR archive. Based on recent comparisons (Middlehurst et al., 2024), state-of-the-art methods include ensemble methods that combine multiple classifiers (Middlehurst et al., 2021) and convolution-based methods (Ismail Fawaz et al., 2020; Dempster et al., 2023).

Another promising approach to time series classification is representation learning, which extracts meaningful features from the series in an unsupervised manner and then transforms them into a representation designed to capture relevant patterns effectively (Lin et al., 2003; Franceschi et al., 2019; Yue et al., 2022). In a second step, these representations are leveraged by downstream classification models, potentially improving interpretability, performance, and efficiency.

Representation learning models are widely used for downstream classification tasks, but they can also be applied to a wide range of tasks and settings in time series modeling. In the next section, we propose to define and explore representation learning for time series.

2.2 Learning time series representations

In statistics and machine learning, a representation provides an alternative view of the original data. In the time series context, it involves projecting the original time series into a latent space that differs from its original temporal domain. These representations can vary widely, from simple descriptive statistics to the output of complex functions. It is important to note the ambiguity surrounding time series representation. (i) In the "pure" form, these representations are entirely unsupervised or self-supervised, allowing them to be reused for various downstream tasks. (ii) In the supervised cases, a model trained for a specific task (e.g., forecasting) may be partially reused, with minor adaptations, for the same task but with new samples or contexts. (iii) Additionally, there is a third scenario where the hyperparameters of the representation are already optimized according to the downstream task. In this case, the representation is already influenced by the task.

All these scenarios can be considered forms of time series representation. Most of this manuscript and our contributions focus on the unsupervised approach, though we occasionally address the other scenarios.

2.2.1 Introduction and intuition

The primary purpose of learning time series representations is to extract insightful features that can be reused in subsequent machine learning tasks, known as downstream tasks. This approach offers several advantages over building supervised machine learning models from scratch. These benefits include enhanced performance, reduced computation time, good performance with few examples (few-shot learning), and improved interpretability (Bengio et al., 2013). We define time series representation as follows.

Definition 2.2.1 (Time series representation)**.** *Given a time series* $\boldsymbol{x}^{(j)} \in \mathbb{R}^{c \times T}$*, we define its representation as* $\boldsymbol{z}^{(j)} \in \mathbb{R}^{d \times T'}$*, where* $\boldsymbol{z}^{(j)} = \phi(\boldsymbol{x}^{(j)})$ *and* ϕ *is an encoding function.*

In the context of time series representation, the dimensions hold specific significance:

- *The* d *dimension typically refers to the latent space dimensionality, such as the number of channels in a convolution output.*
- *The dimension* T' *denotes the temporal aspect of the representation. In general,* $T' \leq T$*.*

If $T' = 1$ (as in the example below), it implies that temporal information has

been lost, resulting in a representation shaped as a d-dimensional vector.

A primary example. A time series representation could be a simple descriptive statistics vector. Consider a univariate time series dataset $\{\boldsymbol{x}^{(j)}\}_{j=1}^{n}$. For each sample $\boldsymbol{x}^{(j)}$, we can construct a representation vector $\boldsymbol{z}^{(j)}$ that captures descriptive statistics such as mean, variance, series minimum, series maximum, etc. (see Figure 2.8). In a second step, these vectors can then be reused, for example, to solve a classification task. Let us consider a dataset of time series with associated labels $\{\boldsymbol{x}^{(j)}, y^{(j)}\}_{j=1}^{n}$. We can fit a classifier (e.g., an XGboost classifier) between the constructed $\boldsymbol{z}^{(j)}$ and the associated $y^{(j)}$ labels. If the extracted statistics are discriminative between the classes, then the intermediate representation will have been helpful, as the valuable information for classification will have been extracted before.

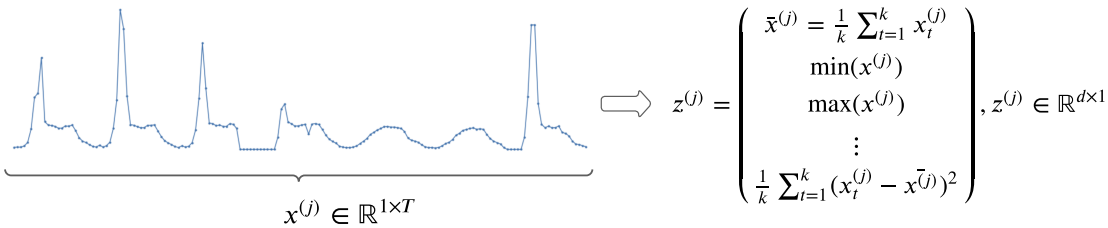


Figure 2.8: Basic time series representation vector of statistic features.

The type of representation presented above has certain advantages and may be sufficient for specific simple tasks. However, it is important to note that such representations often involve a loss of information. For example, the temporal structure is lost, and potential frequency information may be compromised. In the next subsection, Section 2.2.2, we will delve into examples of more advanced representations.

2.2.2 Exploring popular time series representation methods

This section explores two widely used time series representations in machine learning. First, in Section 2.2.2.1, we describe the decomposition of structured time series into trend, seasonality, and residual components, a particularly efficient technique for downstream forecasting tasks. Second, in Section 2.2.2.2, we present the symbolic aggregate approximation representation widely used for downstream classification tasks.

2.2.2.1 The seasonal / trend / residual decomposition

The Seasonal / Trend / Residual (STR) decomposition (Bandara et al., 2021; Taylor and Letham, 2018) is a technique used in time series analysis to decompose a structured time series into its constituent components, namely:

- **Seasonal component.** This component captures the repeating patterns in the data. These patterns occur at fixed intervals over time, such as daily, weekly, monthly, or yearly cycles.
- **Trend component.** This component represents the long-term trend of the data. It captures the overall increasing or decreasing pattern in the data over time.
- **Residual component.** Also call the "noise", it represents the remaining variation in the data after removing the seasonal and trend components.

STR decomposition is useful for understanding the global underlying structure of time series data and can help in trend analysis, forecasting, and anomaly detection. Various methods, such as additive or multiplicative decomposition, and other approaches like empirical mode decomposition (Rilling et al., 2003), can be used for STR decomposition depending on the characteristics of the data and the specific objectives of the analysis. This subsection discusses the additive decomposition, expressed mathematically in Equation (2.3).

$$x(t_i) = g(t_i) + s_1(t_i) + s_2(t_i) + \cdots + s_m(t_i) + r(t_i). \tag{2.3}$$

where: (i) $t_i \in t_1, \ldots, t_T$ represents the timestamps. (ii) $g(t_i)$ represents the trend component. (iii) $s_k(t_i)$ represents the k-th seasonal component. (iv) $r(t_i)$ represents the residual component.

Mathematical expression of the components. The different components can take various forms; below are some examples.

- **The trend component** $g(.)$ can be polynomial:

$$g(t_i) = \sum_{p=0}^{P} w_p t_i^p.$$

  The w_p are learnable weights and P stands for the polynomial degree.

  For the trend component, we note that there are many other possible forms.

- **The seasonal components** $s(.)$ can be expressed using Fourier series:

$$s(t_i) = \sum_{n=1}^{N} (a_n \cos(\frac{2\pi n t_i}{P}) + b_n \sin(\frac{2\pi n t_i}{P})).$$

  where P represents the seasonality of interest (e.g., 24 for hourly timesteps and daily seasonality).

- **The Residual Component** represents the difference between the estimated trend and seasonalities and the observed series.

Time series representation form in the STR additive decomposition case. The representation $\boldsymbol{z}$ can be expressed as:

$$\boldsymbol{z}_{\boldsymbol{t_1:t_T}} = \begin{pmatrix} g(t_1) & \dots & g(t_T) \\ s_1(t_1) & \dots & s_1(t_T) \\ \vdots & \ddots & \vdots \\ s_m(t_1) & \dots & s_m(t_T) \\ r(t_1) & \dots & r(t_T) \end{pmatrix} \in \mathbb{R}^{(m+2)\times T}.$$

In this representation, we note that the temporal dimensions do not contract between the original time series and the representation.

Example on the French national electricity consumption curve. We assume that the French national electricity consumption is composed of daily and weekly seasonalities. We re-used this prior information to extract an additive STR decomposition of this time series. For this purpose, we use the *statsmodels* package[4]. We present the decomposition results in Figure 2.9 for approximately three weeks in 2020.

Results. We observe in Figure 2.9 that the additive combination of the two chosen seasonalities and the trend characterize accurately the underlined time series. Indeed, the residual component magnitude is quite small compare to the original time series.

Potential downstream task. The extracted representation can be re-used for different downstream tasks. If we consider the forecasting task with a look-back window of length L and an horizon window of length H we can use $\boldsymbol{z}_{\boldsymbol{t_{T-L-1}:t_T}}$ to predict $\boldsymbol{z}_{\boldsymbol{t_{T+1}:t_{T+H}}}$. For example a simple forecast method would be to only forecast the trend component, keep the seasonalities the same and set the residual term to zero. In a second step, we compute the additive re-composition of $\boldsymbol{z}_{\boldsymbol{t_{T+1}:t_{T+H}}}$ to retrieve the forecast in the original time series domain.

In the next section, Section 2.2.2.2, we explore the symbolic aggregate approximation time series representation method, which is mainly used for downstream classification tasks.

[4]https://www.statsmodels.org/dev/generated/statsmodels.tsa.seasonal.MSTL.html

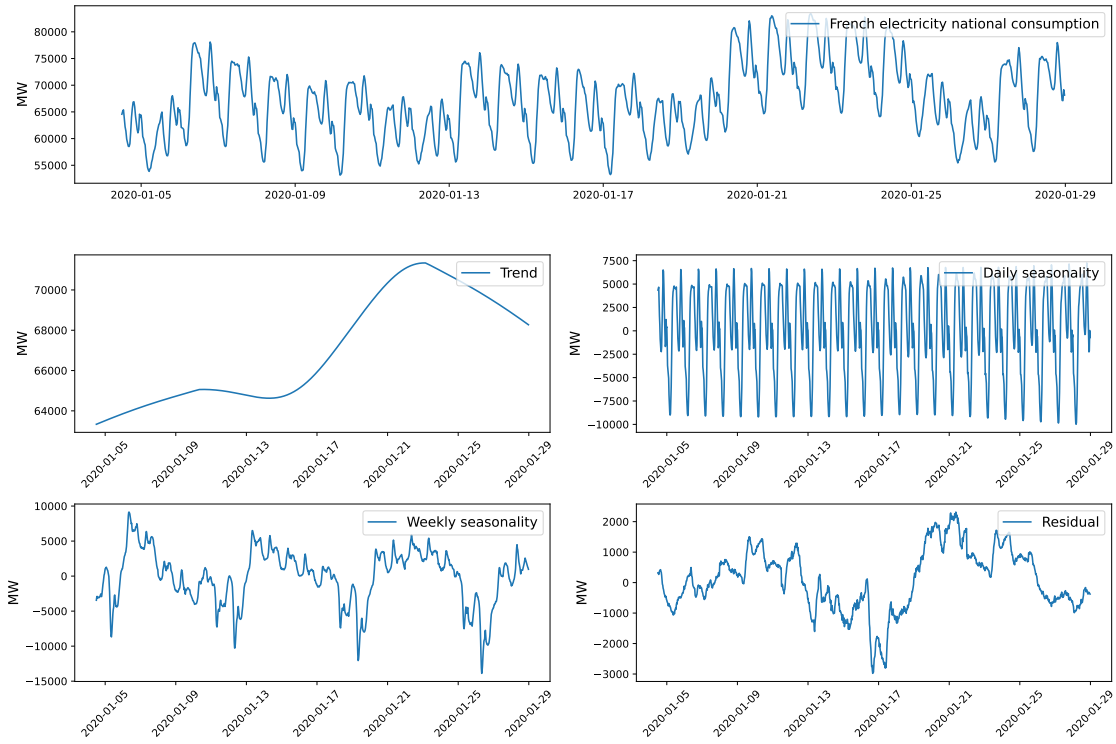


Figure 2.9: Additive STR decomposition on the french national electricity consumption over three weeks in January 2020.

2.2.2.2 The symbolic aggregate approximation representation

The Symbolic Aggregate Approximation (SAX) method (Lin et al., 2003, 2007) is a symbolic approach that discretizes a univariate time series of length T into a sequence of symbols of length T'. The SAX method relies on a two-stage approach:

(i) Dimensionality reduction via piecewise aggregate approximation.

(ii) Turn the piecewise aggregate approximation representation into a symbolic representation.

Piecewise Aggregate Approximation (PAA). PAA involves computing local means of disjoint, side-by-side windows with a length of $\frac{T}{T'}$. For a regularly sampled time series $x_1, ..., x_T$, we define the PAA as $\bar{z}_1, ..., \bar{z}_{T'}$, where:

$$\bar{z}_k = \frac{T'}{T} \sum_{i=\frac{T}{T'}(k-1)+1}^{\frac{T}{T'}k} x_i. \tag{2.4}$$

It is important to note that PAA is performed on **z-normalized** time series.

Note 2.2.1 : z-normalization

z-normalization is a per-sample j normalization method defined as:

$$\boldsymbol{x}_{\text{norm}}^{(j)} = \frac{\boldsymbol{x}^{(j)} - \text{mean}(\boldsymbol{x}^{(j)})}{\text{std}(\boldsymbol{x}^{(j)})}. \tag{2.5}$$

It's worth noting that applying z-normalization to an entire dataset removes the level information between the time series while retaining the variation and pattern information.

Example of PAA on the french national hydraulic power generation time series. In Figure 2.10 we show the PAA applied to a French national hydraulic power generation time series using the *Tslearn* package (Tavenard et al., 2020). The length of the initial time series is $T = 100$ and we set the length of the PAA to $T' = 10$. While the approximation works well when the series has minimal variations, it becomes noticeably imprecise for segments characterized by significant internal variations.

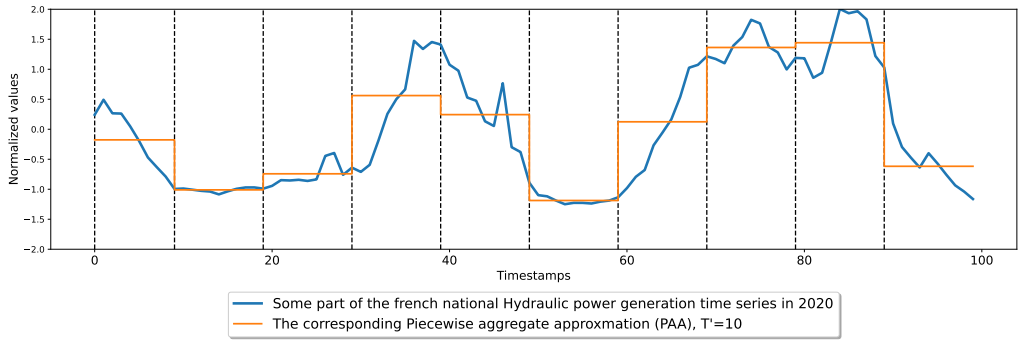


Figure 2.10: PAA of some part of the french national hydraulic power generation time series in 2020.

Symbolic Aggregate Approximation (SAX) representation. After computing the PAA, the next step is to apply a discretization technique that produces symbols with equi-probability.

(i) First, we need to select the cardinality of the support for the SAX representation, i.e., the number of possible different symbols. Let us denote the support of possible symbols as $\mathbb{A}$ (the vocabulary). We denote the number of symbols as p ($Card(\mathbb{A}) = p$).

(ii) Second, since PAA operates on z-normalized time series, the density of the time series tends to be similar to an isotropic Gaussian distribution (Larsen and Marx, 2005). Then, choosing a vocabulary size of p necessitates defining

a sorted list of breakpoints, $\beta = (\beta_1, \ldots, \beta_{p-1})$, such that the area under the standard normal distribution $\mathcal{N}(0,1)$ equals $\frac{1}{p}$ for each segment. Additionally, we set $\beta_0 = -\infty$ and $\beta_p = \infty$.

(iii) Then, each segment $[\beta_i, \beta_{i+1}]$ is assigned to a specific symbol from the vocabulary $\mathbb{A}$. This mapping is denoted by the function $q(\cdot)$ (for quantization). Thus, each transformed value $\bar{z}_k$ is represented by a symbol $z_k \in \mathbb{A}$, depending on its corresponding interval.

SAX Representation Dimension and Mechanism. The SAX representation vector $\boldsymbol{z}$ can be expressed as

$$\boldsymbol{z_{t_1:t_{T'}}} = \begin{pmatrix} q(\bar{z}_{t_1}) & \ldots & q(\bar{z}_{t_{T'}}) \end{pmatrix} \in \mathbb{A}^{1\times T'}.$$

Two important phenomena are observed:

- Because of the PAA, there is a reduction in the time dimension.
- The elements of the representation belong to a discrete, relatively small support of size $p = Card(\mathbb{A})$.

On the one hand, the compression of the temporal dimension and the quantization mechanism lead to the loss of some information within the time series. On the other hand, the representation is "simplified" for downstream tasks, significantly reducing the series' complexity. Later, we will explore how this reduction in complexity allows some form of interpretability of classification tasks based on the SAX representation.

Example of SAX representation on the French National Hydraulic Power Generation Time Series using *Tslearn*. We revisit the previous example where we computed the PAA for $T' = 10$ (see Figure 2.10). For the SAX representation, we opt to set the vocabulary size to 4 ($p = 4$). The breakpoints are defined as $(\beta_0 = -\infty, \beta_1 = -0.67, \beta_2 = 0, \beta_3 = 0.67, \beta_4 = \infty)$, corresponding to the vocabulary $\mathbb{A} = \{a, b, c, d\}$. The entire process is illustrated in Figure 2.11. As a result, we obtain the following symbolic representation: $\boldsymbol{z} = (c\ d\ d\ b\ b\ d\ b\ a\ a\ c)$.

SAX representations with time series datasets. Above, we presented the SAX representation for a single time series. However, the real power of SAX comes when applied to time series datasets. Given a time series dataset $\{\boldsymbol{x^{(j)}}\}_{j=1}^n$, we can use the SAX method to build a corresponding set of representations, denoted as $\{\boldsymbol{z^{(j)}}\}_{j=1}^n$. Subsequently, the SAX representations can be reused to solve classification tasks efficiently, as presented in the following paragraph.

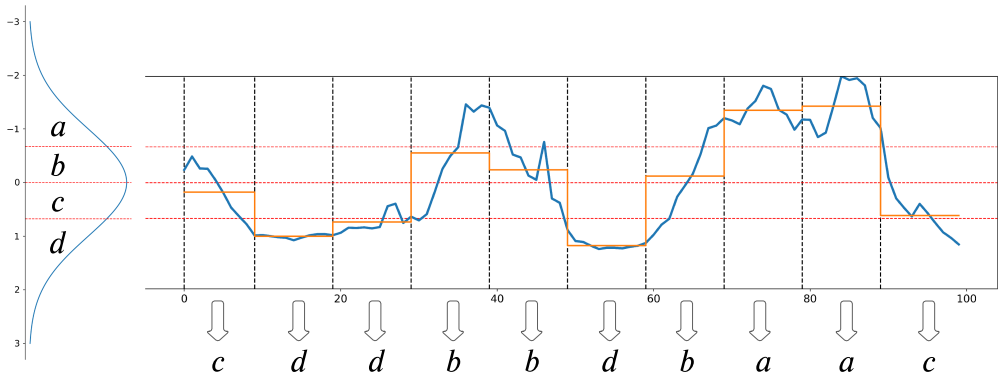


Figure 2.11: SAX representation of some part of the french national Hydraulic power generation time series in 2020.

Exploring the SAX-Vector Space Model (SAX-VSM) for Classification Tasks (Senin and Malinchik, 2013). Let us consider a dataset containing time series paired with associated classes $\{\boldsymbol{x}^{(j)}, y^{(j)}\}_{j=1}^{n}$, where labels $y^{(j)} \in \{0, 1\}$. In the SAX-VSM approach, authors propose constructing a subsequence space vector for each class, where each element of the vector represents the frequency of occurrences of SAX subsequences based on a Term Frequency $*$ Inverse Document Frequency (TF$*$IDF) scheme (Salton and McGill, 1984). These subsequences are obtained by sliding a window over the original SAX representation. Consequently, discriminative subsequences exhibit high scores for one class and near-null scores for the other. During inference, we compute the scores vector based on symbolic subsequences for a new SAX representation without associated labels and assign a class based on similarity to class 0 and class 1. This procedure is briefly depicted in Figure 2.12.

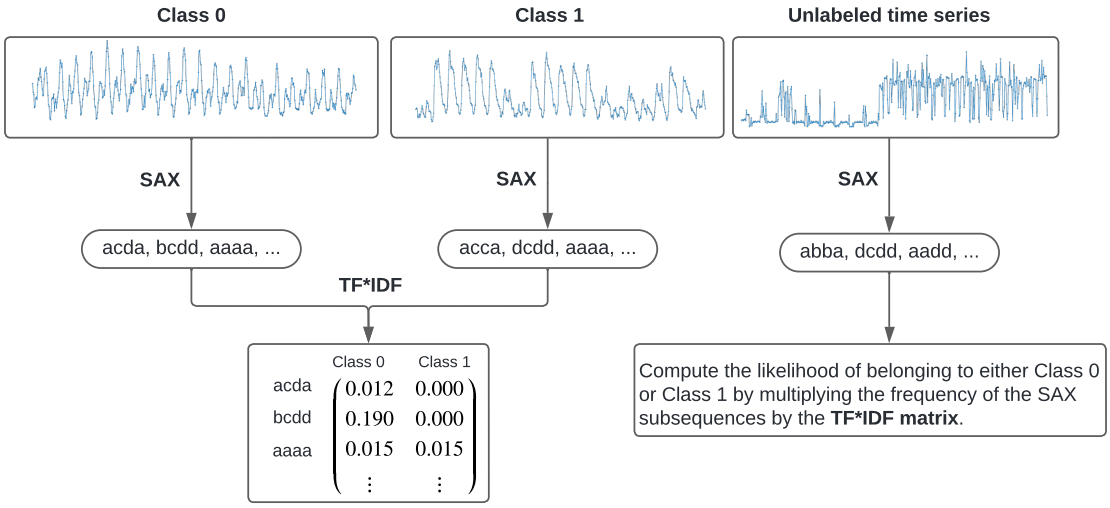


Figure 2.12: Visualization of the SAX-VSM classification decision. This graphic is inspired by Senin and Malinchik (2013).

Interpretability. The SAX-VSM method provides a form of local interpretability for the classification decision. For example, if the SAX subsequence (b d c a) discriminates between the two classes in the TF*IDF matrix, we can find and highlight the time series subsequences that produce this symbolic subsequence. The advantage of using this representation is that it reduces the complexity of the data, making it easier to interpret the classification algorithms applied to the SAX representation.

SAX-based approaches in literature and limitations. In Lin et al. (2003), SAX is presented as a versatile unsupervised method applicable to various downstream tasks. In practice, however, SAX representations are often paired with classifiers that are applied on top of the representation. Several methodologies exist for classification that utilize SAX representations. We have previously discussed SAX-VSM method (Senin and Malinchik, 2013), which relies on subsequence frequencies within each class. Another approach, SAX-SEQL (Nguyen et al., 2017), explores the entire subsequence space to identify the most discriminatory subsequences using logistic regression based on coordinate descent. SAX techniques have effectively facilitated interpretable classification decisions through the symbolic representation they construct. Nevertheless, this interpretability is limited in scope. While identifying symbolic subsequences allows to highlight corresponding segments of the time series, we cannot fully reconstruct what the model has learned, as multiple subseries can yield the same symbolic subsequence. Moreover, as highlighted in Lipton (2018), excessive use of representations for classification, as seen in multi-SAX-SEQL (Nguyen et al., 2019), can hinder the interpretation of classification results.

2.2.3 From time series representations to time series neural representations

The two examples above are just a few of the many ways to learn time series representations. We describe below a non-exhaustive list of other time series representation method in traditional machine learning (non deep learning):

- **The Principal Component Analysis (PCA)** which is a particular case of singular value decomposition, is widely used in time series modelisation. It can be used to compute distances between multivariate time series (Yang and Shahabi, 2004), anomaly detection (Hyndman et al., 2015) and so on.
- **The Symbolic Fourier Approximation (SFA)** method (Schäfer and Högqvist, 2012) is quite related to the SAX method but operates in the frequency domain. First the time series is approximated with a discrete Fourier transform, then the Fourier coefficients are mapped to a symbol. The final representation is a sequence of symbols such as SAX. The SFA representations are mainly

used for classification and proved to be very efficient with ensemble classifiers (Schäfer, 2015).

- **Wavelet decompositions** are popular in time series (Percival and Walden, 2000) and offer an alternative to Fourier decomposition. Wavelets offer several advantages over Fourier series in time series analysis. They provide localized frequency information, making them more suitable for analyzing signals with non-stationary or transient characteristics. Additionally, wavelets offer better time-frequency resolution, allowing for precise localization of features in both time and frequency domains. This enables wavelet analysis to capture fine details and abrupt changes in signals. These representations are useful for time series forecasting (Joo and Kim, 2015) or clustering (Zhang et al., 2006).

There are many other possible time series representation such as singular value decomposition (when it not the PCA case) (Cadzow et al., 1983; Weng and Shen, 2008), random mapping such as random convolution kernels (Dempster et al., 2020) and so on. Conversely with the rise of deep learning (LeCun et al., 2015), new possibilities in time series representation emerge.

Traditional machine learning relies on pre-designed models and often requires manual feature engineering, where features are selected and transformed before being fed into the model. Deep learning, however, utilizes artificial neural networks, which are highly flexible and can learn directly from raw data without extensive feature engineering. These neural networks consist of multiple layers of neurons, allowing them to capture intricate patterns and relationships within the data. This flexibility has been mathematically studied in the seminal work on the universal approximation theorem (Cybenko, 1989).

While traditional machine learning relies on handcrafted features and pre-defined models, deep learning learns complex patterns directly from the data itself. In the context of neural representation for time series, leveraging the expressive capabilities of neural networks offers several advantages over traditional machine learning representations:

- Improve the handling of downstream tasks by capturing the underlying structure of time series data more accurately.
- Build representations that can be re-used for several downstream task.
- Train a representation extractor (encoder neural network) that can be reused across multiple datasets without retraining.
- Allow the development of tasks which where under-explored such as time series generation.
- Build meaningful latent spaces.

2.3 Time series neural representations

As traditional time series representation, time series neural representations provide an alternative view of the original temporal data. The only difference is that the projection from the original time series space to the representation space (also called latent space) is performed by a Neural Network (NN).

Definition 2.3.1 (A simple definition of a neural network.)**.** *A neural network, denoted as $f_{\boldsymbol{\theta}}$, can be conceptualized as a series of interconnected layers $L(.)$, each adjustable through learnable parameters $\boldsymbol{\theta}$, and intersected by activation functions $\sigma(.)$.*

$$f_{\boldsymbol{\theta}}(\boldsymbol{x^{(j)}}) = \sigma^{(K)} \circ L^{(K)}_{\boldsymbol{\theta^{(K)}}} \circ \ldots \circ \sigma^{(2)} \circ L^{(2)}_{\boldsymbol{\theta^{(2)}}} \circ \sigma^{(1)} \circ L^{(1)}_{\boldsymbol{\theta^{(1)}}} \circ \boldsymbol{x^{(j)}}. \tag{2.6}$$

Here, K stands for the number of layers. The parameters to be learned, represented as $\boldsymbol{\theta} = \{\boldsymbol{\theta^{(1)}}, \ldots, \boldsymbol{\theta^{(K)}}\}$, are optimized by minimizing a loss function $\mathcal{L}$. For instance, in binary classification tasks, this involves minimizing $\mathcal{L}(f_{\boldsymbol{\theta}}(\boldsymbol{x^{(j)}}), y^{(j)})$, where $\mathcal{L}$ is a binary cross-entropy loss which computes the discrepancy between predictions and ground truth labels.

The learnable layers $L^{(k)}_{\boldsymbol{\theta^{(k)}}}$ can take many forms, in the following section we will describe popular deep learning mechanisms.

2.3.1 Deep Learning mechanisms

Note 2.3.1 : Warning about input/output spaces illustrations

In the following deep learning mechanism descriptions, we represent the time series space as the input layer and the latent space as the output layer. It is important to note that this is purely for illustrative purposes. In reality, the input/output spaces may vary depending on the architecture.

Dense layers. Dense Layers (Haykin, 1994), also known as Multi-Layer Perceptron (MLP), illustrated in Figure 2.13, is a sequence of K linear layers interspersed with non-linear activations. We define it below.

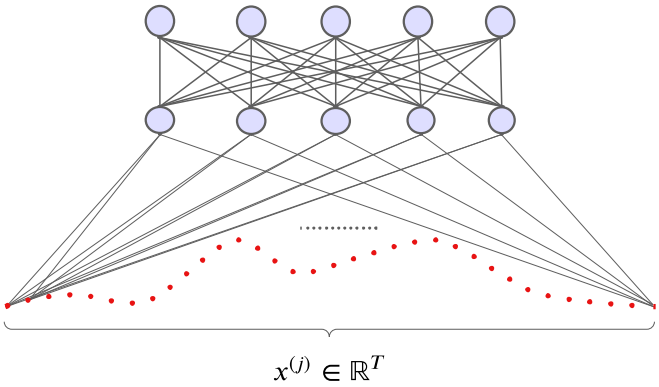


Figure 2.13: Simplified illustration of dense layers.

Definition 2.3.2 (Dense layers.)**.** *Dense layers can be broken down into three types:*

- *The input layer which projects the input from the input domain into an hidden space of dimension h. For instance, if we consider the input space to be the series space, this layer can be written as:*

$$\boldsymbol{A^{(1)}} = \sigma^{(1)}(\boldsymbol{x\theta^{(1)}} + \boldsymbol{b^{(1)}}). \tag{2.7}$$

  Where $\boldsymbol{\theta^{(1)}} \in \mathbb{R}^{T\times h}$, $\boldsymbol{b^{(1)}} \in \mathbb{R}^h$. σ denotes a non-linear pointwise activation, e.g. $\sigma(u) = max(0, u)$.

- *The hidden layers.*

$$\boldsymbol{A^{(p)}} = \sigma^{(p)}(\boldsymbol{A^{(p-1)}\theta^{(p)}} + \boldsymbol{b^{(p)}}). \tag{2.8}$$

  Where $\boldsymbol{\theta^{(p)}} \in \mathbb{R}^{h\times h}$, $\boldsymbol{b^{(p)}} \in \mathbb{R}^h$ and $1 < p < K$.

- *The output layer projects the last hidden state to an output space. For example, if we consider the output space to be the latent space of dimension d, it can be written:*

$$\boldsymbol{z} = \boldsymbol{A^{(K)}} = \sigma^{(K)}(\boldsymbol{A^{(K-1)}\theta^{(K)}} + \boldsymbol{b^{(K)}}). \tag{2.9}$$

  Where $\boldsymbol{\theta^{(K)}} \in \mathbb{R}^{h\times d}$, $\boldsymbol{b^{(K)}} \in \mathbb{R}^d$ and $\boldsymbol{z} \in \mathbb{R}^d$ is the latent representation.

MLP networks are not widely used in the time series literature. However, a few attempts have been made to couple MLP networks with the DTW distance to exploit the structure of time series (Iwana et al., 2016, 2020). The limited use of

MLP networks is due to their structure. While these models can learn complex nonlinear relationships, they do not consider the specific structure of time series, such as the proximity between neighbors or the presence of frequencies.

Recurrent layers. Recurrent layers (Schuster and Paliwal, 1997) are specially designed to handle sequential data. The idea is to represent the time series by a sequence of hidden states that incorporate both past and present information. A simplified schematic of deep recurrent models is presented in Figure 2.14.

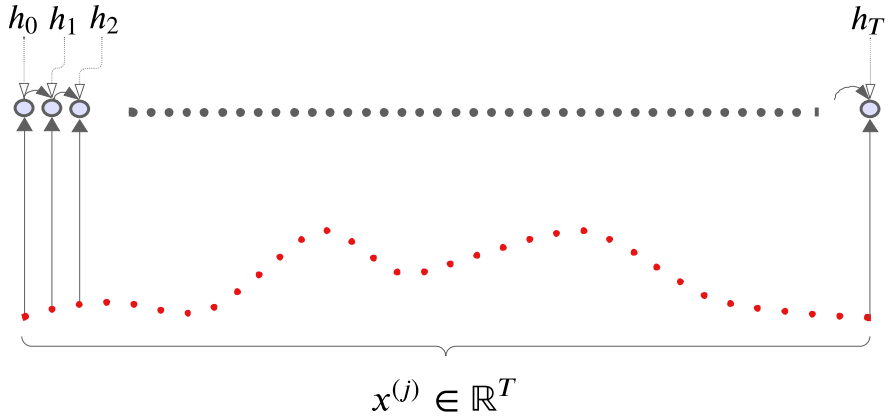


Figure 2.14: Simplified illustration of recurrent layers.

Definition 2.3.3 (Vanilla recurrent layers.)**.** *Hidden recurrent layers can take many forms. In the case of a vanilla RNN, the hidden state* $\mathbf{h_t}$ *at time* t *can be written as a combination of the previous hidden state* $t-1$ *and the time series input at current time* t*:*

$$\mathbf{h_t} = tanh(\boldsymbol{\theta_x} x_t^{(j)} + \boldsymbol{\theta_h} \mathbf{h_{t-1}}). \tag{2.10}$$

Where the weight matrices $\boldsymbol{\theta_x}$ *and* $\boldsymbol{\theta_h}$ *are learnable and not time dependent.*

Although seemingly suitable for time series, the Equation (2.10) of the hidden state structure in practice leads to problems during optimization. During backpropagation, gradients propagate poorly through the sequence: this is known as the gradient vanishing problem. This phenomenon is more pronounced for long sequences and prevents the model from learning correlations between distant timestamps. To overcome this limitation, more complex recurrent mechanisms have been proposed in which the hidden states allow both short-term and long-term information to pass through the network. Such mechanisms include the Gated Recurrent Unit (GRU) (Chung et al., 2014) and the Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) cells.

Due to their sequential nature, simple RNNs were quickly applied to time series (Hüsken and Stagge, 2003) before giving way to more complex RNNs for different task such as classification or imputation (Dennis et al., 2019; Cao et al., 2018).

Convolutional layers. Convolutional layers (LeCun et al., 1995) are very efficient for pattern extraction due to the weight sharing mechanism. In addition, they are relatively easier to train than RNNs and have fewer weights. The structure of 1-dimensional convolutional layers is illustrated in Figure 2.15 and defined below.

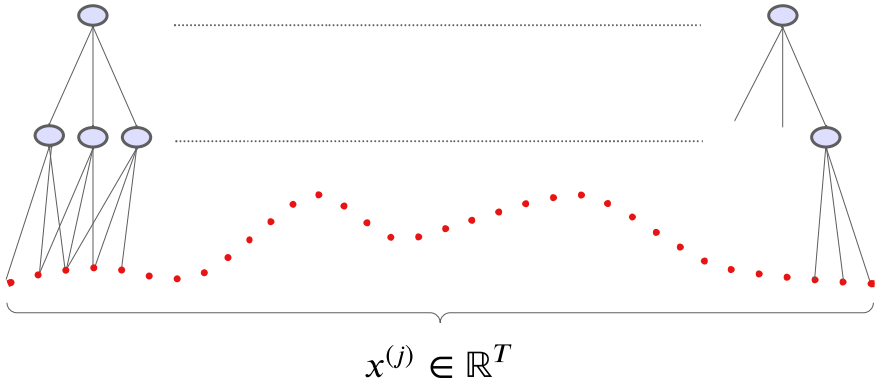


Figure 2.15: Simplified illustration of convolutions layers.

Definition 2.3.4 (1-dimensional convolution layers.)**.** *Let $\boldsymbol{x}$ be an entry vector (e.g. the time series) and $\boldsymbol{g}$ be a kernel, respectively of size T and m. Then the convolution operations at timestamp t can be written as:*

$$(x*g)(t) = \sum_{r=1}^{m} g(r)\cdot x(t-r+\lceil m/2 \rceil). \quad \textit{Where } \lceil . \rceil \textit{ stands for the ceiling function.} \tag{2.11}$$

- *The kernel $\boldsymbol{g}$ is a learnable vector $[w_1, w_2, ..., w_m]$, (e.g. $m = 3$).*
- *In practice, convolution layers use not just one, but several dozen kernels, e.g. $\{\boldsymbol{g^1}, ..., \boldsymbol{g^{64}}\}$, to extract different types of local relationships.*

Convolutions allow to extract similar patterns along the time series, thanks to the weight sharing of $\boldsymbol{g}$. This local mechanism is well suited for time series with local (neighborhood) information: for example, for a load consumption curve, we can assume that there is a correlation between the consumption at 9 a.m. and the consumption at 10 a.m..

Convolutional networks are widely used in modern time series classification research. Inspired by their success in computer vision, equivalent architectures have been adapted for time series. First, a variant of ResNet for time series (Ismail Fawaz et al., 2019) that incorporated residual connections paved the way. Later, an adaptation of AlexNet (Ismail Fawaz et al., 2020) was introduced to better capture signal frequencies by incorporating kernels of different sizes adapted to different convolution channels. CNNs are also used in forecasting. The temporal convolutional network (Wan et al., 2019), which uses causal convolution, is considered a strong baseline. Recently, however, attention mechanisms have gained popularity, overshadowing convolutions for time series forecasting.

Attention layer. Transformers (Vaswani et al., 2017) have recently gained popularity in time series after their success in natural language processing. They are capable of efficiently extracting distant correlations in sequences, thanks to the self-attention mechanism. Self-attention layers enable the network to efficiently select the relationships between data for a given task. We define the self-attention mechanism below and illustrate it in Figure 2.16.

Definition 2.3.5 (Self-attention layers.)**.** *Self-attention layers are used to calculate the importance of different relationships within a sequence. The self-attention mechanism is defined as follows:*

$$\textit{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathbf{A}\boldsymbol{V}, \quad \textit{where } \mathbf{A} := \textit{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\boldsymbol{T}}}{\sqrt{d_k}}\right) \in \mathbb{R}^{T\times T}. \quad (2.12)$$

Where for an input $\boldsymbol{x} \in \mathbb{R}^{d\times T}$

- $\boldsymbol{Q} = \boldsymbol{x}^{\boldsymbol{t}}\boldsymbol{W}^{\boldsymbol{q}}$ *and* $\boldsymbol{W}^{\boldsymbol{q}} \in \mathbb{R}^{d\times d_k}$.
- $\boldsymbol{K} = \boldsymbol{x}^{\boldsymbol{t}}\boldsymbol{W}^{\boldsymbol{k}}$ *and* $\boldsymbol{W}^{\boldsymbol{k}} \in \mathbb{R}^{d\times d_k}$.
- $\boldsymbol{V} = \boldsymbol{x}^{\boldsymbol{t}}\boldsymbol{W}^{\boldsymbol{v}}$ *and* $\boldsymbol{W}^{\boldsymbol{v}} \in \mathbb{R}^{d\times d_v}$.

The weight matrices $\boldsymbol{W}^{\boldsymbol{q}}$, $\boldsymbol{W}^{\boldsymbol{k}}$ *and* $\boldsymbol{W}^{\boldsymbol{v}}$ *are learnable. The coefficient* $\mathbf{A}_{ij}$ *of the attention matrix* $\mathbf{A}$ *can be interpreted as the attention between the* i^{th} *and* j^{th} *elements of the input sequence.*

In time series modeling, transformer models have recently been applied to tasks such as classification (Zerveas et al., 2021), forecasting (Zhou et al., 2021; Nie et al., 2022), and imputation (Du et al., 2023). The first generation of transformers for forecasting (Zhou et al., 2021; Wu et al., 2021) used pointwise attention, which proved inefficient as simple linear models outperformed them (Zeng et al., 2022).

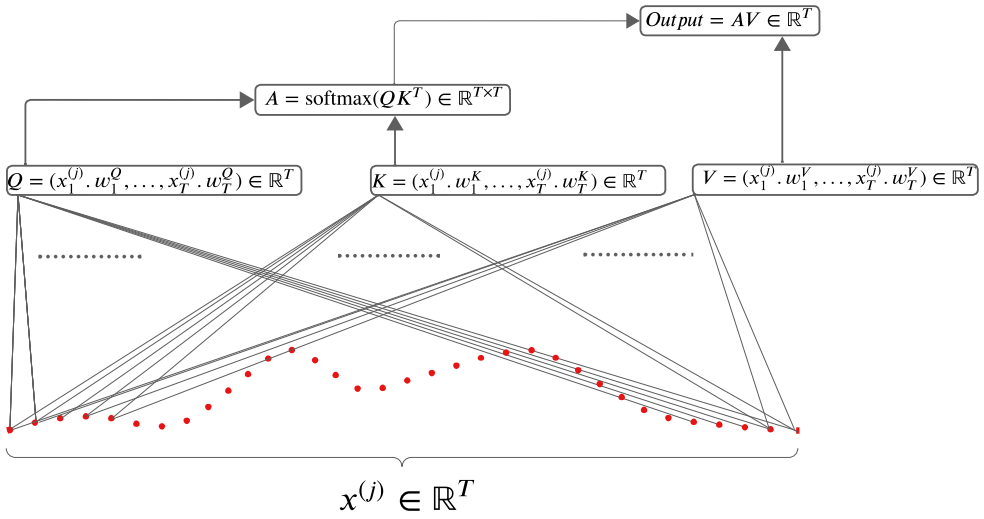


Figure 2.16: Simplified illustration of an attention layer.

However, the second generation (Nie et al., 2022) used patch-wise attention (similar to vision transformer (Dosovitskiy et al., 2020)), which is much more efficient.

Deep learning mechanisms are often mixed within the same architecture. In deep learning, various mechanisms are frequently combined within the same architecture to leverage their complementary strengths. For example, the well-known VGG model (Simonyan and Zisserman, 2014), an image classifier, incorporates both convolutional and dense layers. This combination enables the model to extract complex features from input data, leading to more robust and accurate results.

When it comes to learning neural representations, the importance goes beyond the choice of specific mechanisms; it encompasses the structure of the architecture that is responsible for acquiring these representations. In the following subsection, we explore three structural paradigms for learning time series neural representations.

2.3.2 Structures of time series neural representation architecture

The time series neural representation can be defined similarly to Definition 2.2.1 where $\phi_{\boldsymbol{\theta}}$ is a learnable neural network. Below are presented three structures of time series neural representation architecture.

Encoder-decoder structure. The encoder-decoder structure (also called auto-encoder) is the most popular architectures for learning representations (Kramer, 1991; Vincent et al., 2008). This architecture can be described as the composition

of an encoder and a decoder which output an approximation of the original signal i.e. $\psi_{\theta'}(\phi_\theta(\boldsymbol{x}^{(j)})) = \hat{\boldsymbol{x}}^{(j)}$. It can be decomposed in a two steps process:

(i) Learn the latent representation through encoding: $\boldsymbol{z}^{(j)} = \phi_\theta(\boldsymbol{x}^{(j)})$.

(ii) Learn to approximate the signal trough decoding: $\hat{\boldsymbol{x}}^{(j)} = \psi_{\theta'}(\boldsymbol{z}^{(j)})$.

In the simple case, the purpose of this architecture is to learn to reconstruct the original signal $\boldsymbol{x}^{(j)}$ in the presence of a bottleneck. The presence of a bottleneck is essential to ensure that the latent representation $\boldsymbol{z}^{(j)}$ captures useful information. Without a bottleneck, the input signal could be perfectly reconstructed by setting both the encoder $\phi_{\boldsymbol{\theta}}(.)$ and the decoder $\psi_{\boldsymbol{\theta'}}(.)$ to the identity function Id(.). However, in this case, the representation would not learn anything meaningful. In general, for vanilla time series encoder-decoder we have $T^{'} < T$ which mean that the encoder contracts the temporal dimension. Then, $\boldsymbol{\theta}, \boldsymbol{\theta'}$ are learned by minimizing iteratively (through gradient-descent) a pointwise reconstruction loss (e.g. $||\hat{\boldsymbol{x}}^{(j)} - \boldsymbol{x}^{(j)}||_2^2$). This structure is widely used in time series neural representation (Malhotra et al., 2017; Nie et al., 2022). We will elaborate on this kind of architecture in Section 2.3.3.2. We provide an overview of this structure in Figure 2.17.

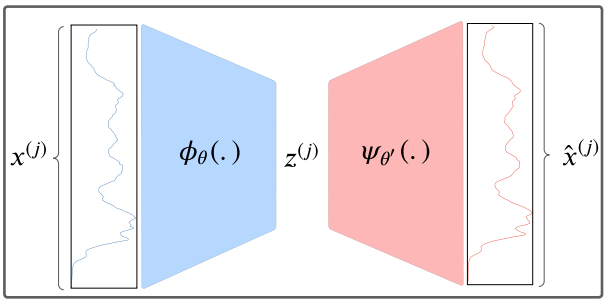
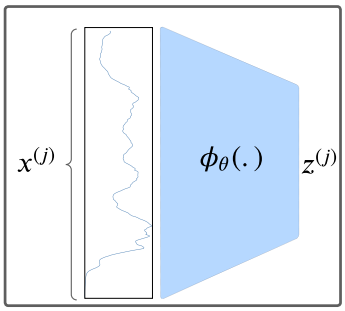


Figure 2.17: Time series neural representation: encoder-decoder structure.

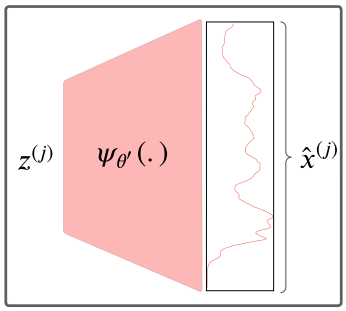
Encoder-only structure. Encoder-only structures rely on a single projection of the time series into the latent space (i.e. $\boldsymbol{z}^{(j)} = \phi_{\boldsymbol{\theta}}(\boldsymbol{x}^{(j)})$ only). There is no reconstruction loss, most of the time $\boldsymbol{\theta}$ is optimized through a contrastive loss (Chopra et al., 2005) which gathers similar time series and spreads apart, dissimilar time series. We will elaborate on this kind of architecture in Section 2.3.3.1. These architectures are widely used in time series neural representation (Franceschi et al., 2019; Yue et al., 2022). Most of the time, the obtained representations are used for classification, anomaly detection and clustering downstream tasks. We provide an overview of this structure in Figure 2.18a.

Auto-decoding structure. Auto-decoder structures (Park et al., 2019) are not widely explored in time series neural representation. Similar to encoder-decoder structures, these architectures aim to reconstruct the signal. The $\boldsymbol{z}^{(j)}$ representation is a vector (or matrix) of optimizable weights which are randomly initialized. The decoder takes as input this representation and outputs an approximation of the

ground truth signal $\boldsymbol{x}^{(j)}$. Then, the $\boldsymbol{z}^{(j)}$ are updated thanks to the back-propagation (e.g. according to the reconstruction loss $||\hat{\boldsymbol{x}}^{(j)} - \boldsymbol{x}^{(j)}||_2^2$). With this structure, the back-propagation can be see as encoding. We will used the auto-decoding structure later in the manuscript. We provide an overview of this structure in Figure 2.18b.



(a) Time series neural representation: encoder-only architecture.



(b) Time series neural representation: auto-decoding architecture.

Figure 2.18: Time series neural representation illustrations.

2.3.3 Advanced neural representation methods

This section explores two models from the time series neural representation learning literature. These models are built upon the concepts previously introduced.

2.3.3.1 T-Loss: An encoder-only convolutional neural network based on contrastive learning

The Triplet Loss (T-loss) model of Franceschi et al. (2019) is inspired by the Word2Vec model (Mikolov et al., 2013) proposed in Natural Language Processing (NLP) for vector representation of words. T-Loss contains three main features:

(i) An encoder-only architecture (Figure 2.20) based on dilated causal convolutions, a design popularized by WaveNet (Van Den Oord et al., 2016).

(ii) The architecture is trained with a contrastive loss to produce representations with meaningful geometric properties.

(iii) The contrastive loss employed is a triplet loss (Figure 2.19), which uses a sampling scheme that generates anchor, positive and negative samples.

Intuition. The intuition behind contrastive loss in NLP is that the context representation of a word should be close to the words that make it up but also distant from random words elsewhere. Thus, the loss to be optimized is chosen so that one pair (context, word in the context) and another pair (context, random word) are separable in latent space.

Triplet loss. The objective of Franceschi et al. (2019) is to apply the same principle to time series. We consider a subsegment $\boldsymbol{x^{ref}}$ that is randomly (uniformly) drawn from a time series $\boldsymbol{x^{(j)}}$. On the one hand, we want the representation of $\boldsymbol{x^{ref}}$ to be close to the representations of the $\boldsymbol{x^{pos}}$ ($\boldsymbol{x^{pos}} \subset \boldsymbol{x^{ref}}$, see Figure 2.19). On the other hand, we want any other subseries $\boldsymbol{x^{neg}}$ randomly chosen from another series $\boldsymbol{x^{(l)}}$ or from $\boldsymbol{x^{(j)}}$ but not include in $\boldsymbol{x^{ref}}$ to have a representation far away from $\boldsymbol{x^{ref}}$. In practice, to facilitate convergence, it is preferable to repeat the sampling procedure and draw K negative subsequences $\{\boldsymbol{x_k^{neg}}\}_{k=1}^{K}$.

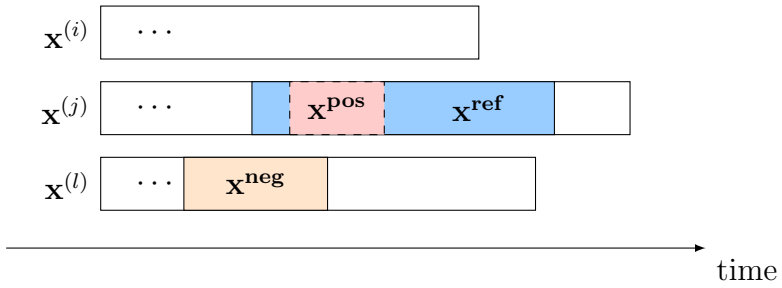


Figure 2.19: Generation of the triplet $(\boldsymbol{x}^{\text{ref}}, \boldsymbol{x}^{\text{pos}}, \boldsymbol{x}^{\text{neg}})$ in the univariate case, for the T-loss (Franceschi et al., 2019) model.

Let $\phi_{\boldsymbol{\theta}}(.)$ be a given encoder that outputs a latent vector and σ the sigmoid function. The loss function to be minimized is the following loss:

$$\mathcal{L}_\theta = -\log\left(\sigma\left(\phi_{\boldsymbol{\theta}}(\boldsymbol{x^{ref}})^\intercal \phi_{\boldsymbol{\theta}}(\boldsymbol{x^{pos}})\right)\right) - \sum_{k=1}^{K} \log\left(\sigma\left(-\phi_{\boldsymbol{\theta}}(\boldsymbol{x^{ref}})^\intercal \phi_{\boldsymbol{\theta}}(\boldsymbol{x_k^{neg}})\right)\right). \quad (2.13)$$

Encoder architecture. The $\phi_{\boldsymbol{\theta}}$ encoder consists of a sequence of causally dilated convolutions (see Section 2.3.1 for convolution definition). A max-pooling mechanism is then applied to the last convolution output, flattening the temporal dimension to one. This practical and efficient architecture allows arbitrary sequence sizes to be inputs using the convolution layers and projected onto a fixed-size vector using the max-pooling mechanism. The encoder architecture is shown in Figure 2.20.

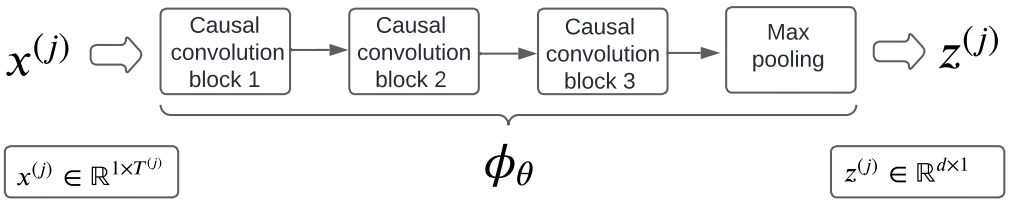


Figure 2.20: Simplified illustration of the causal CNN encoder with 3 causal convolution blocks applied on a univariate time series.

The question at hand is whether the latent space in which the $\{\boldsymbol{z}^{(j)}\}_{j=1}^{n}$ reside has a more organized structure compared to the original time series space in which the $\{\boldsymbol{x}^{(j)}\}_{j=1}^{n}$ reside. When we refer to a better structured space, we imply that similar series are close in the latent space, while dissimilar ones are distant. Consequently, tasks that rely on similarity, such as classification, clustering, or anomaly detection, become more manageable in this latent space than in the original time series space. We will discuss this aspect further below.

Experimental Setup. To estimate whether time series are more effectively separable in latent space than in their original domain, we perform classification experiments using the *FordA* dataset from the UCR archive (Dau et al., 2019). This dataset comprises 3601 training instances and 1320 test instances, with each time series consisting of 500 measurements of engine noise recorded by an engine sensor. The task entails automatically detecting the presence of a specific engine problem, thus constituting a balanced binary classification task.

We employ the model from Franceschi et al. (2019) to generate representations $\{\boldsymbol{z}^{(j)}\}_{j=1}^{n}$ and subsequently employ a 1-NN classifier with the Euclidean distance (ED) metric in this latent space to label the test series based on their proximity to the training series. In our evaluation, we compare this approach with the following methods:

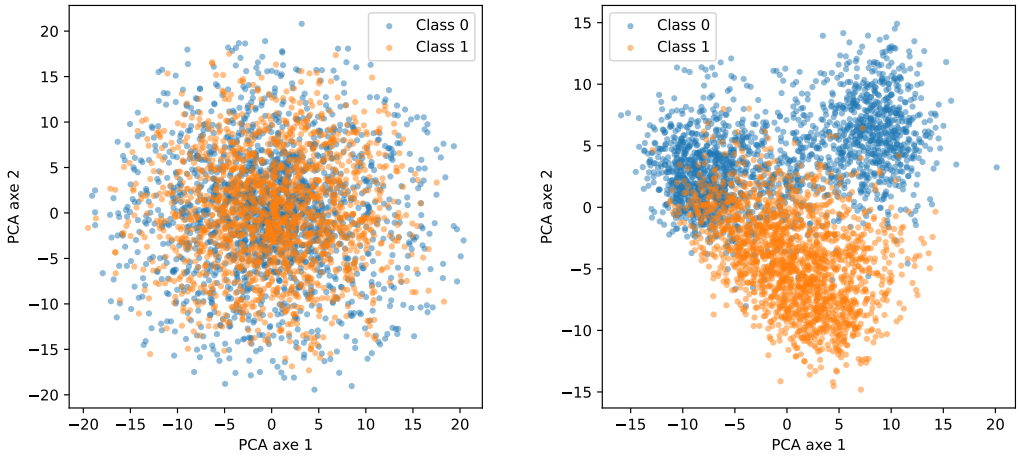
- Application of the 1-NN classifier in the original series space using the Euclidean distance metric.
- Application of the 1-NN classifier in the original series space using the dynamic time warping distance metric.
- The naive predictor that predicts the majority class in the training dataset for all test instances.

The results are presented in Table 2.1.

Table 2.1: Comparison of the 1-NN classifier in time series space and latent space for the *FordA* dataset.

	1-NN ED in latent space	**1-NN ED in time series space**	**1-NN DTW (learned w) in time series space**	**Majority class**
Accuracy	**88.48** %	66.52 %	69.09 %	51.59 %

Results. We observe a notable improvement in performance with the 1-NN ED method applied in latent space compared to the two 1-NN methods operating in the original time series space. This observation highlights the ability of the encoder to more effectively discriminate dissimilar series in the *FordA* dataset compared to the original space. To provide a visual insight into this phenomenon, we present in Figure 2.21 scatterplots where we have applied Principal Component Analysis (PCA) to both the time series and their corresponding representations. The visualizations clearly show that the distributions of the different classes are better separated in the latent space.



(a) PCA applied on the initial time series. (b) PCA applied on the representations.

Figure 2.21: Visualization of the PCA applied to the initial time series and their representations learned with T-Loss on the FordA test dataset. (UCR archive).

Additional noteworthy results. Below are some advantages of the representations constructed using this model, as evidenced in the original paper:

- The representations effectively separate dissimilar series, making classifiers based on space separators, such as Support Vector Machines (SVMs) (Cortes and Vapnik, 1995), highly efficient.
- The geometric properties of the constructed space enable classifiers (such as k-NN or SVMs) to perform well even with very few labels available for training.
- Once trained on a dataset, the $\phi_{\boldsymbol{\theta}}$ encoder can be reused without retraining to build representations for other datasets.

A method at the forefront of a new field of time series research. This method, which learns neural representations for time series based on contrastive loss, has initiated a rapidly growing field of research. New models have sought to:

- Improve the positive/negative sample criteria in contrastive loss (Tonekaboni et al., 2021).
- Change the encoder architecture (Yue et al., 2022)
- Consider time domain and frequency domains (Zhang et al., 2022b).
- Apply other downstream tasks on top of the representations, such as forecasting (Zheng et al., 2024).

These are just a few examples. Numerous new unsupervised neural methods based on contrastive learning are introduced each year. For a comprehensive review of contrastive methods, please refer to the extensive literature review by Zhang et al. (2024).

2.3.3.2 Unsupervised PatchTST: An encoder-decoder representation learning method based on attention mechanism

Intuition. The PatchTST model (Nie et al., 2022), inspired by the Vision Transformer (ViT) (Dosovitskiy et al., 2020), stands out as a highly effective time series modeling model. This model decomposes time series into a sequence of patches and considers these patches as tokens. Then, several attention blocks are applied on top of this sequence of patches. The author demonstrates that the patch attention is more effective for time series than the pointwise attention. In the original paper, the authors introduced both supervised and unsupervised variants. This section focuses on the unsupervised version, whose training is based on patch masking (He et al., 2022).

Model description. The forward process of the unsupervised PatchTST can be described as follows:

(i) Segmentation of the time series into contiguous, non-overlapping patches.

(ii) Random masking of 40% of these patches.

(iii) Linear projection of patches + additive position encoding.

(iv) Application of several self-attention blocks.

(v) The output of these attention blocks is the latent representation $\mathbf{z}$.

(vi) A linear layer is then applied on top of the latent representation $\mathbf{z}$ to reconstruct the masked patches.

The parameters of this architecture are optimized according to the reconstruction loss, which is exclusively computed over the masked patches. The architecture is illustrated in Figure 2.22.

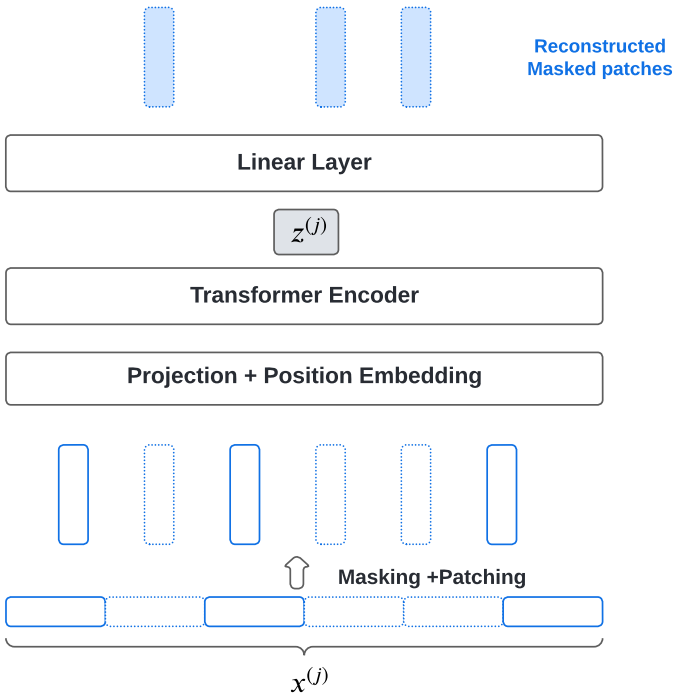


Figure 2.22: PatchTST unsupervised architecture.

This masked auto-encoder transformer is a powerful feature extractor, and the representation learned just before linear projection contains valuable information for downstream tasks. The authors also show that the unsupervised PatchTST can serve as a reusable representation extractor. Once trained, it can be used to extract high-quality representations for datasets not seen during training. This result is described in the next section.

Generalization experiments from the paper. The authors evaluated the generalizability of the representation learned by unsupervised PatchTST for the downstream forecasting task. The experimental setup is as follows:

(i) First, unsupervised PatchTST is pre-trained on the *Electricity* dataset.

(ii) The pre-trained architecture is then applied to other datasets to extract a latent representation.

(iii) On top of the extracted representation, a simple linear layer is trained to forecast the horizon for the considered datasets.

In their experiments, the authors compare the effectiveness of the PatchTST representation coupled with linear layer against several supervised forecasting baselines: Dlinear (Zeng et al., 2022), FEDformer (Zhou et al., 2022), Autoformer (Wu et al., 2021), and Informer (Zhou et al., 2021). This comparison is made over the *Traffic* and *Weather* datasets. Forecasting results are presented in Table 2.2.

Table 2.2: Forecasting experiments comparison from the original paper. Trained linear layer on top of the PatchTST representation trained on the *Electricity* dataset compare with supervised forecast baselines. H stands for the horizon. Bolds stands for the best result and underline for the second best result.

		PatchTST unsupervised		DLinear		FEDformer		Autoformer		Informer	
	H	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Weather	96	**0.163**	**0.216**	0.176	0.237	0.238	0.314	0.249	0.329	0.354	0.405
	192	**0.205**	**0.252**	0.220	0.282	0.275	0.329	0.325	0.370	0.419	0.434
	336	**0.253**	**0.289**	0.265	0.319	0.339	0.377	0.351	0.391	0.583	0.543
	720	**0.320**	**0.336**	0.323	0.362	0.389	0.409	0.415	0.426	0.916	0.705
Traffic	96	**0.400**	0.288	0.410	**0.282**	0.576	0.359	0.597	0.371	0.733	0.410
	192	**0.412**	0.293	0.423	**0.287**	0.610	0.380	0.607	0.382	0.777	0.435
	336	**0.425**	0.307	0.436	**0.296**	0.608	0.375	0.623	0.387	0.776	0.434
	720	**0.457**	0.317	0.466	**0.315**	0.621	0.375	0.639	0.395	0.827	0.466

Results. Table 2.2 clearly shows that the representation extracted by PatchTST, when combined with the linear layer, outperforms or is on par with other methods on all horizons for both datasets examined. This suggests that unsupervised PatchTST, pre-trained on one dataset, can extract meaningful representations for other datasets.

Encoder-decoder based on a reconstruction loss used for time series representation learning. Many models have utilized encoder-decoder architectures based on time series reconstruction to learn representations. Malhotra et al. (2017) were pioneers in this approach, employing an LSTM-based architecture. Later, masking strategies combined with transformers were introduced to enhance the effectiveness of learning time series features (Zerveas et al., 2021; Li et al., 2023). Typically, the representations derived from these models are used for downstream forecasting tasks, but they can also be applied to classification, anomaly detection, and other tasks.

2.3.4 Discussion and open problems

The two methods presented above are just a small sample of all the neural representation learning methods for time series that have proliferated in recent years. For example, we have presented contrastive objective and reconstruction objective separately, but recent papers combine the two to learn neural representations for time series (Chowdhury et al., 2023; Dong et al., 2024). Additionally, as discussed in the introduction of Section 2.2, literature on pre-trained supervised neural networks for a specific task that reuse the trained architecture for another task can be considered as neural representation learning (Ma et al., 2023). This includes all the recent literature on foundational models for time series forecasting, more precisely referred to as "flexible zero-shot forecasters" (Das et al., 2023; Liang et al., 2024). The concept of these models is to jointly train a deep learning forecasting model on multiple datasets and then be able to make forecasts without training on a new dataset (never seen during training). For an extensive review of neural representation learning for time series, please refer to Trirat et al. (2024).

However, despite the rise of neural representation methods for time series and the various improvements compared to traditional representation learning, there still remain critical open issues:

- **Intepretability of the neural representation.** Unsupervised representation learning approaches aim to build representation for time series by capturing their underlying distribution without expert knowledge or human supervision. They have demonstrated good performances for clustering, classification, missing values imputation or forecasting. Despite these good performances for downstream tasks, the neural representations models in the literature lack interpretability. In Bengio et al. (2013), a review of representation learning, the authors emphasize that good representations should have the ability to extract *Explanatory Factors* and should guarantee *Temporal Consistency*. Current neural approaches do not meet these criteria. Indeed, for most existing approaches, the representation results from mapping signals to a latent vector with no temporal consistency and in which values have no meaning. These representations fail to provide interpretability when used for downstream tasks, which is problematic for critical decision-making.

- **Adaptable models for new time series and temporal distribution shifts.** One of the major open problems in learning time series representations is the ability to create adaptive models that can effectively handle distribution shifts. These challenges are exacerbated when new data samples or different temporal contexts are introduced, requiring models to adapt to shifts in data distribution without extensive retraining. Ensuring that models can dynamically adapt to new and evolving data while maintaining high performance

under varying conditions is essential. The need for robust and flexible models that respond to temporal changes remains a critical area of research in time series analysis.

- **Capture neural representation from unaligned and irregular time series.** In time series analysis, the variety, heterogeneity, and increasing number of deployed sensors present new challenges that current methods often fail to address. Time series data frequently originate from diverse sources, exhibiting irregular sampling, containing missing values, and lacking alignment, particularly when collected from distributed sensors. While time-dependent continuous models can handle such data, they struggle to learn effective representations from them. Furthermore, their performance has significantly lagged behind models designed for regular discrete grids. This highlights the need for advanced approaches that can effectively capture neural representations from unaligned and irregular time series, ensuring robust performance across various real-world scenarios.

In the contribution part of the manuscript, we propose to tackle these open issues and design time series neural representation learning models that address these challenges.

Part II

Contributions

Chapter 3

Interpretable Time Series Neural Representation for Classification Purposes

In this chapter, we address the problem of interpretability of neural representations. Recently, deep learning has made significant progress in creating efficient representations of time series data by automatically identifying complex patterns. However, these approaches lack interpretability because the time series is transformed into a latent vector or matrix that is not easily interpretable. On the other hand, Symbolic Aggregate Approximation (SAX) methods allow the creation of symbolic representations that can be interpreted, but do not effectively capture complex patterns. In this work, we propose a set of requirements for a neural representation of univariate time series to be interpretable. We propose a new unsupervised neural architecture that satisfies these requirements. The proposed model produces consistent, discrete, interpretable, and visualizable representations. The model is learned in an unsupervised setting independent of any downstream tasks to ensure robustness. To demonstrate the effectiveness of the proposed model, we propose experiments on classification tasks using UCR archive datasets. The obtained results are extensively compared with other interpretable models and state-of-the-art neural representation learning models. The experiments show that the proposed model provides, on average, better results than other interpretable approaches on several datasets. We also present qualitative experiments to evaluate the interpretability of the approach.

Le Naour, E., Agoua, G., Baskiotis, N., and Guigue, V. **Interpretable time series neural representation for classification purposes.** *IEEE 10th International Conference on Data Science and Advanced Analytics (IEEE DSAA)* 2023. Best research paper award.

3.1 Introduction

Unsupervised representation learning approaches aim to build representation for time series by capturing their underlying distribution without expert knowledge or human supervision. They have demonstrated good performances for clustering (Ma et al., 2019), classification (Franceschi et al., 2019; Malhotra et al., 2017; Yue et al., 2022), missing values imputation or forecasting (Zerveas et al., 2021). Despite these good performances for downstream tasks, the neural representations models in the literature lack interpretability. In Bengio et al. (2013) a review of representation learning, the authors emphasize that good representations should have the ability to extract *Explanatory Factors* and should guarantee *Temporal Consistency*. Current approaches do not meet these criteria. Indeed, for most existing approaches (Franceschi et al., 2019; Yue et al., 2022; Zhang et al., 2022a), the representation results from mapping signals to a latent vector with no temporal consistency and in which weights have no meaning. These representations fail to provide interpretability when used for downstream tasks like classification, which is problematic for critical decision-making.

However, interpretability is a concept that is not universally agreed upon (Lipton, 2018), with confusion arising from the different meanings of interpretability and

explicability. For time series models, Wang (2021) offers a clear taxonomy of the interpretability shown in Figure 3.1. Post-hoc interpretability refers to methods that analyze the model after the training and are generally model-agnostic. It is often related to the eXplainable Artificial Intelligence (XAI) research field. However, post-hoc methods only explain the decision for a specific instance (or specific features), and additional methods are required to understand the overall model.

On the other hand, in-situ interpretable models are self interpretable. The interpretability arises directly from the model without any other process being applied after the training phase (Rudin, 2019; Wang, 2021). The level of this interpretability can be local or global. In Lipton (2018), global interpretability is defined as a model that is easy for a human to understand and requires low computational complexity. In contrast, local interpretability is a way to interpret a model's decision for a particular instance. Global interpretability often implies local interpretability, but the reverse is not true.

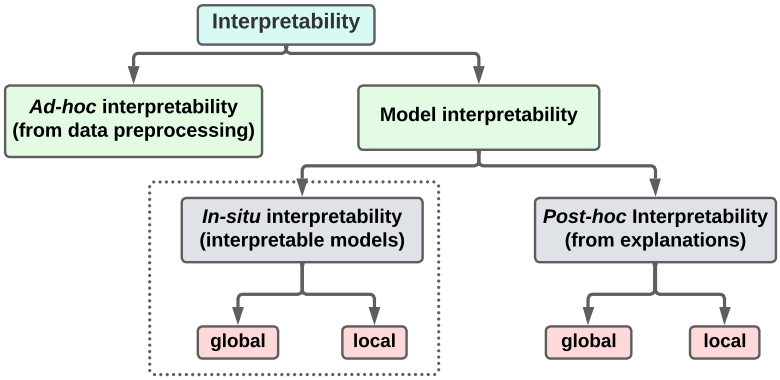


Figure 3.1: Interpretable time series model taxonomy introduced in Wang (2021).

In this work, we focus on global in-situ interpretability rather than post-hoc explainability, as global in-situ models are inherently interpretable and can be understood both for individual instances and the model as a whole. This paper aims to develop a global in-situ interpretable neural method for time series representation. The first contribution of this work is to define the requirement to bridge the gap between symbolic representation and neural representation to ensure a global interpretable neural symbolic representation for time series data. Indeed, most successful interpretable models come from symbolic machine learning, such as symbolic aggregate approximation, which creates interpretable symbolic representations of time series data. However, the information captured by these symbols is limited and does not provide global interpretability of the representation. On the other hand, neural representation learning methods achieve great performances but are definitively not in-situ interpretable. Section 3.3 analyses the criteria that must be respected to guarantee global in-situ interpretability.

We propose a novel unsupervised neural network that fills these requirements in Section 3.4. The neural network is based on an auto-encoder architecture and vector quantization mechanism (Gersho and Gray, 1991; van den Oord et al., 2017; Fortuin et al., 2019). The unsupervised setting is a crucial choice for the generalization of the learned representation. Moreover, it allows to re-use the extracted representation for several classification tasks.

To demonstrate the qualities of the proposed architecture, Section 3.5 presents an application of our learned symbolic neural representation to classification tasks. A simple linear classifier over the interpretable symbolic representations is used to solve classification tasks efficiently. The linearity of the classifier preserves interpretability in the representation, providing both global and local interpretability for understanding the decision made by the classifier.

Our main contributions can be summarized as follows:

- We define and formalize the fundamental requirements to construct interpretable symbolic neural representations for time series.
- We propose an unsupervised neural network architecture that satisfies the above requirements.
- We use these representations for downstream classification tasks (while preserving interpretability of the representation) and evaluate them through quantitative experiments on the UCR archive.
- We provide qualitative experiments to capture local and global interpretability.

3.2 Related content

Constructing time series representations is a fundamental challenge that can be performed unsupervised or based on a specific task. This related content presents both neural representation methods, which are not easy to interpret, and classical representation methods, which are interpretable. In most cases, classical representation methods are combined with a classification task.

Unsupervised neural representation learning for time series classification. In recent works, models of neural representations of time series have emerged. These models typically learn the representation in an unsupervised way and then use the learned representation for a specific task in a second step. In Malhotra et al. (2017), the authors use a deep unsupervised representation to solve a subsequent classification task for time series. They used an architecture composed of Recurrent Neural Networks to build a representation that would later be used for classification. In

Franceschi et al. (2019), authors build a time series representation using a convolutional encoder and a contrastive loss (Mikolov et al., 2013). The representation space brings together series (and subseries) that are similar. Afterward, a support vector machine (SVM) is applied on top of the representation to solve the classification problem. Then, several papers attempted to construct vector representations of time series using contrastive loss (Yue et al., 2022; Zhang et al., 2022a). Some recent works (Zerveas et al., 2021) have tried to build an unsupervised representation of time series using transformers mechanisms (Vaswani et al., 2017) inside an auto-encoder. These neural representations can capture a lot of information, and the downstream tasks learned from them are very efficient. However, the representations and thus the downstream tasks cannot be interpreted with these models.

Attempt to construct an interpretable neural representation for time series. Recently, progress has been made in making neural representations of time series interpretable. In Li et al. (2022), the authors have attempted to decompose time series into disentangled semantic factors (for both individual factors and group segment factors) using Variational Auto-Encoder (VAE), LSTM, and a disentanglement strategy. The limitation of disentangled representations is that it is difficult to assess the disentanglement of latent factors when the initial semantic properties of the time series are unknown. This unsupervised representation is interesting for generation, but difficult to adapt for classification. Other work, such as Luo et al. (2022), has attempted to obtain interpretable differential operators from multivariate time series. However, they are specific to forecasting.

Although neural methods for representing time series are relatively new, non neural interpretable methods for representing time series are widely studied.

The Symbolic Aggregate approximation (SAX). SAX methods (Lin et al., 2003, 2007) create a symbolic representation of a time series by combining local statistics, which are calculated by taking the average of different segments of the time series. Each average is assigned a symbol based on its value. A sequential symbolic representation of the original time series is formed by mapping these local averages to symbols. Multiple SAX representations can be obtained by taking more or less local averages. SAX symbolic elements operate in the time domain and are interpretable, unlike the Symbolic Fourier approximation (SFA) method (Schäfer and Högqvist, 2012), which operates in the frequency domain. In Lin et al. (2003), the authors introduce the SAX method as a versatile unsupervised approach for various downstream tasks, but in practice, SAX methods are typically used in combination with a classifier on top of the representation. There are several methods to classify on top of SAX representations. For example, the *SAX-VSM* method (Senin and Malinchik, 2013) is based on the frequency of subsequences within each class. The *SAX-SEQL* method (Nguyen et al., 2017) searches the entire space of sub-

sequences for the most discriminating subsequences using logistic regression based on coordinate descent. SAX methods have proven useful for making interpretable classification decisions from the constructed symbolic representation. However, this interpretability is only local. Indeed, when we obtain a discriminating symbolic subsequence, we can highlight the corresponding subpart of the time series. Nevertheless, we cannot reconstruct what the model has learned because several subseries can give the same symbolic subsequence. Additionally, as stated in Lipton (2018), when the number of representations used for classification is excessive, such as in the case of multi-SAX-SEQL (Nguyen et al., 2019), it becomes difficult to interpret the classification results.

Shapelet methods. These methods aim to find the subsequences of the time series that most discriminate between classes (Ye and Keogh, 2011). This representation method is supervised because it relies on a downstream task. In addition, the resulting representations are partial, since only the shapelets that discriminate between classes are extracted at the end of the process. Once the optimal shapelets are found, if the classifier applied on top of them is interpretable (linear regression or decision tree), the whole process is interpretable at both local and global levels. However, the original method is costly because it is necessary to search the whole space for possible subsequences. In *Fast Shapelets* (FS) (Keogh and Rakthanmanon, 2013), the authors proposed to speed up the discovery of discriminative subsequences by using a SAX representation to discover the subparts of the series where shapelets should be searched. Although the method speeds up the discovery of discriminative shapelets, it remains computationally expensive and the accuracy could be better. To avoid these problems, in Grabocka et al. (2014), authors propose the *Learning Shapelets* (LTS) method. The goal is to learn the discriminating shapelets rather than to search for them in the whole space of possibilities. This method has led to improvements in accuracy. However, it generates a large number of shapelets that are almost the same. This negatively affects interpretability. Afterward, methods have been proposed to learn a limited number of shapelets and thus reinforce the interpretability of the model. In Wang et al. (2020), authors propose learning a small number of discriminating shapelets by training a Generative Adversarial Network (GAN) and a classifier.

To the best of our knowledge, there is no unsupervised learning method capable of learning an interpretable neural representation for time series. In the next section, we set out the requirements for constructing interpretable symbolic neural representations. These criteria will link traditional methods of representing time series using symbols and unsupervised neural representation methods.

3.3 Requirements for an interpretable symbolic neural representation

In order to ensure that a neural representation of a time series is interpretable, we set out several requirements that we consider essential.

For this purpose, we introduce some notation for this section. We consider that we have a dataset of n samples. For an instance i ($i \in \{1, ..., n\}$), the univariate time series is denoted by the vector $\boldsymbol{x^{(i)}}$ of length T: $\boldsymbol{x^{(i)}} = \left(x_1^{(i)}, \ldots, x_T^{(i)}\right) \in \mathbb{R}^T$. Let $\boldsymbol{z^{(i)}}$ be the symbolic neural representation composed of $T^{'}$ elements for $\boldsymbol{x^{(i)}}$. We denote by $\mathbb{A}$ the support (alphabet) common to all these elements: $\boldsymbol{z^{(i)}} = (z_1^{(i)}, \ldots, z_{T^{'}}^{(i)}) \in \mathbb{A}^{T^{'}}$. Then ϕ_θ is the function that maps the time series into the representation, and $\psi_{\theta^{'}}$ is the function that goes from the representation to the reconstruction space of the time series. To simplify the reading, we omit the indices i for the vectors $\boldsymbol{z}$ and $\boldsymbol{x}$.

Requirement n°1 - discrete symbolic representation. The purpose of a symbolic neural representation method is to capture complex phenomena within the representation (as neural representations do) while being able to interpret and visualize the representation elements (as symbolic representations tend to do). Thus, the support $\mathbb{A}$ of each element must be discrete and limited (e.g. $Card(\mathbb{A}) = 32$). In addition, the support must be common to all elements of the symbolic representation. This limits the number of possible patterns. Once we obtain the symbolic representation $\boldsymbol{z}$, we can use the classifiers used in the dictionary methods (Schäfer, 2015; Nguyen et al., 2017, 2019) (see Section 3.5).

Requirement n°2 - temporal consistency. For a time series $\boldsymbol{x}$ of length T, learning a contracted representation $\boldsymbol{z}$ of length $T^{'}$ will mechanically lead to a contraction of the time dimension ($T^{'} < T$). We then define temporal consistency by two properties. First, each element of the representation is a function of a portion of the original time series. Thus, for each element of the representation, we must be able to compute the pre-image of the element. Second, the representation must preserve the original temporal order despite the contraction of the temporal dimension. To illustrate this property, consider the case where $z_{t_1^{'}}$ is an element of the representation and $z_{t_2^{'}}$ is another element of the representation that occurs after $z_{t_1^{'}}$. As shown in Figure 3.2, the temporal consistency of the representation ensures that the pre-image of $z_{t_1^{'}}$ must precede the pre-image of $z_{t_2^{'}}$ in time.

Requirement n°3 - a decodable representation. Being able to visualize the portion of the time series related to a specific element of the representation is important, but it is also important to be able to see what the model has learned overall.

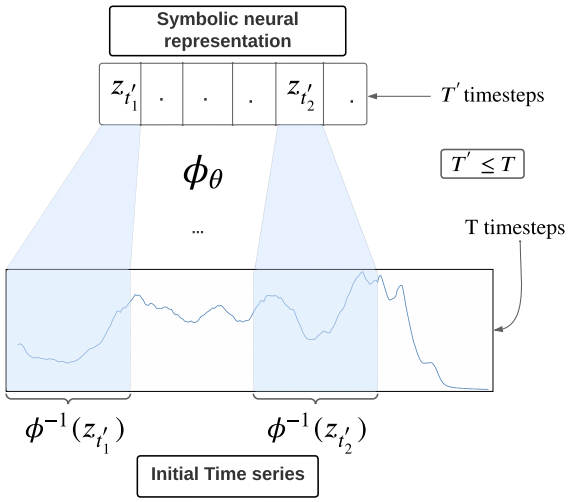


Figure 3.2: Temporal consistency visualization. *Coffee dataset.*

As shown in Figure 3.3, $\psi_{\theta'}$ should be able to reconstruct the entire time series, as well as specific parts of it, while maintaining temporal consistency.

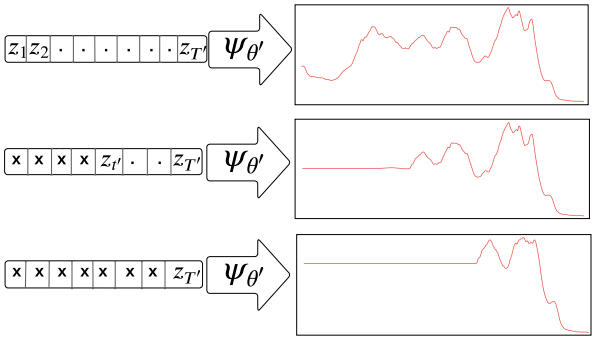


Figure 3.3: Visualization of how the representation is decoded for the complete representation, as well as for half of the representation and for only the last element. *Coffee dataset.*

Requirement n°4 - shift equivariance properties. We want the shift equivariance property for both ϕ_θ and $\psi_{\theta'}$. The ϕ_θ shift equivariance means that two patterns in the initial time series that are identical but do not occur at the same time should be encoded with the same value but not at the same place in the representation. The $\psi_{\theta'}$ shift equivariance means that two elements of the representation that have the same value but do not occur at the same time should represent the same pattern when decoded (with a time shift). This property is essential to interpret the representation elements and ensure that the same value in the representation, regardless of its position, represents the same pattern. We illustrated the shift equivariance

property for $\psi_{\theta'}$ in Figure 3.4.

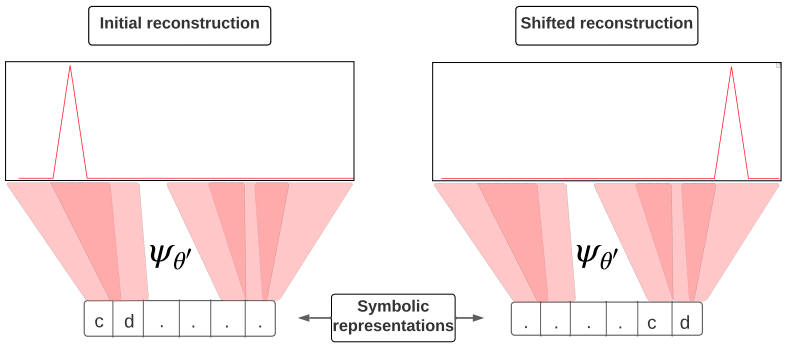


Figure 3.4: Shift equivariance of $\psi_{\theta'}$. The subsequence c d appear at two different locations in the representation: $\psi_{\theta'}$ should decode the same pattern with a shift in time.

Let's formally define the shift equivariance property. Let S be an element of a sequence (a single element or a subsequence), and G_T the group of discrete translations along the temporal axis. If we take τ to be any discrete translation in G_T and f to be a function equivariant by discrete translation for G_T, then there exists $\tau' \in G_{T'}$ such that: $f(\tau(S)) = \tau'(f(S))$.

Requirement n°5 - a representation adjustable to the frequency level. Like an image that cannot be interpreted at the pixel level, a time series is difficult to interpret at the point level. This requirement aims to control the amount of information captured when creating the representation. For the representation to be easily understood, it is crucial to capture the appropriate frequency levels that define the time series. As illustrated in Figure 3.5, the depth of the representation determines whether it focuses on lower or higher frequency features, which in turn affects the length of the representation.

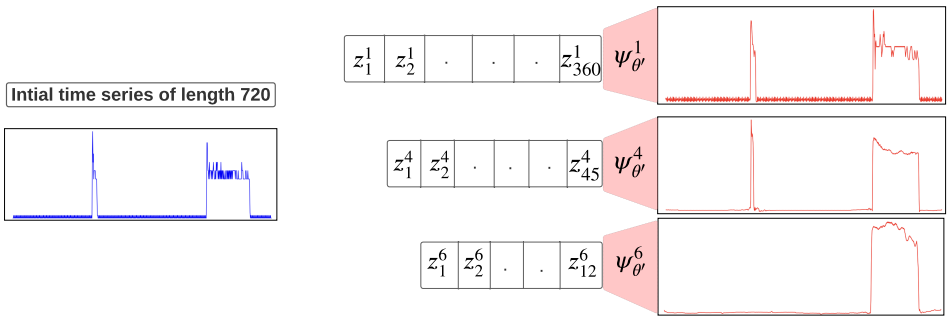


Figure 3.5: Visualization of different reconstructions for different level of representation. We can see that the deeper the architecture, the shorter the representation and the smoother the reconstruction. *Computers dataset.*

3.4 Model

This section proposes an unsupervised model that respects the different requirements for building an interpretable symbolic neural representation.

3.4.1 Proposed architecture

Architecture overview. The proposed unsupervised model architecture consists of an encoder-decoder structure with a discretization mechanism within the representation space. (i) First, the time series $\boldsymbol{x}$ is given as input to the encoder. (ii) Second, the output of the encoder is discretized using the vector quantization mechanism. This step allows us to obtain $\boldsymbol{z}$ after the learning process. (iii) Finally, the discretized elements are passed to the decoder, which returns a reconstruction of the time series $\hat{\boldsymbol{x}}$.

The architecture takes elements from the Vector Quantization Variationnal Auto-Encoder (VQ-VAE) (van den Oord et al., 2017), which was the first model to establish a latent space of discrete representation within an auto-encoder. However, we add constraints to the architecture and remove the variational part in order to meet the requirements defined in Section 3.3. The different parts of the architecture are described below and Figure 3.8 illustrates the global unsupervised architecture. The components of the architecture are described below.

Encoder. The encoder can be divided into a sequence of consecutive blocks with the same structure. As shown in Figure 3.6, a block consists in three operations: a 1D convolution layer, a downsampling operation, and a non-linear activation function.

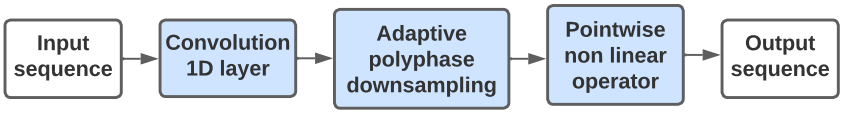


Figure 3.6: Inside an encoder block.

The convolution layer is non-strided and has a dilation factor of zero. The input sequence is padded with a zero on each side, and the kernel size is three. Therefore, the convolution layer's output sequence length remains unchanged. Subsequently, we perform a downsampling operation on the sequence. This operation aims to reduce the sequence length by two while keeping the shift equivariance property. For this purpose, we use the adaptive polyphase downsampling (APS-D) proposed in Chaman and Dokmanic (2021).

To explain the APS-D operation, let us introduce $s(t)$ the convolutions output sequence of length T_0. We can then define the two subsequences s_0 and s_1 such

that $s_0(t) = s(2t)$ and $s_1(t) = s(2t+1)$. The APS-D operation D_2^A consists in subsampling the subsequence s_l such that:

$$D_2^A(s) = s_l \quad \text{where} \quad l = \arg\max ||s_j||_1 \text{ (for } j \in \{0, 1\})$$

When the downsampling process is applied to the sequence, its length is reduced by half. Finally, a non-linear pointwise operation is applied. In practice, this operation is the LeakyReLU.

All blocks inside the encoder have the same structure. Consider an encoder with B blocks and convolutions with Z channels. Then the initial time series is projected into the representation space on a sequence of $T^{'}(T' \approx T/2^B)$ timestamps, where each point of the sequence is a vector of size Z.

Decoder. Now we can define the decoder blocks symmetrically to the encoder. The decoder is a function that projects a sequence of T' vectors of size Z into a reconstructed time series $\hat{\boldsymbol{x}}$ of size T. If the encoder has B blocks, then the decoder will have B blocks. However, the structure of a decoder block is slightly different. First, an upsampling operation (adaptive polyphase upsampling) increases the length of the sequence by two. In practice, the values of the input sequence are re-used and 0's are inserted between each value to double the size of the sequence. The 0's are inserted in an even or odd manner according to the subsequence extracted in the corresponding encoder block during the downsampling phase (if l was 0 or 1). Next, a 1D non-strided convolutional layer with a kernel size of 3 is applied. Finally, a nonlinear pointwise operation is applied (except in the last block).

Discretization mechanism. Now that the encoder and decoder are defined, we need to specify the discretization method used within the representation space. After passing the time series to the encoder and thus obtaining a sequence of vectors (sequence of embeddings: $(\boldsymbol{e_1}, \boldsymbol{e_2}, ..., \boldsymbol{e_{T'}})$), we use the vector quantization (VQ) mechanism (Gersho and Gray, 1991; van den Oord et al., 2017). Given a set of K centroids $\{\boldsymbol{c_j} \in \mathbb{R}^Z,\ j \in 1, ..., K\}$, the VQ mechanism consists of assigning an encoder output point $\boldsymbol{e_{t'}} \in \mathbb{R}^Z$ to the nearest centroid:

$$\boldsymbol{e}^{\boldsymbol{q}}_{\boldsymbol{t'}} \leftarrow \boldsymbol{c_k} \quad \text{where} \quad k = \arg\min_j ||\boldsymbol{e_{t'}} - \boldsymbol{c_j}||_2^2. \tag{3.1}$$

In this way, the sequence of embedding vectors is transformed into a sequence of quantized vectors (each quantized vector has only K possible values). This mechanism is illustrated in the Figure 3.7. Then, the sequence of quantized vectors ($\boldsymbol{e}^{\boldsymbol{q}}_{\boldsymbol{1}}$, $\boldsymbol{e}^{\boldsymbol{q}}_{\boldsymbol{2}}$, ..., $\boldsymbol{e}^{\boldsymbol{q}}_{\boldsymbol{T'}}$) is given as input to the decoder. We describe how centroids are moving through epochs in the training section.

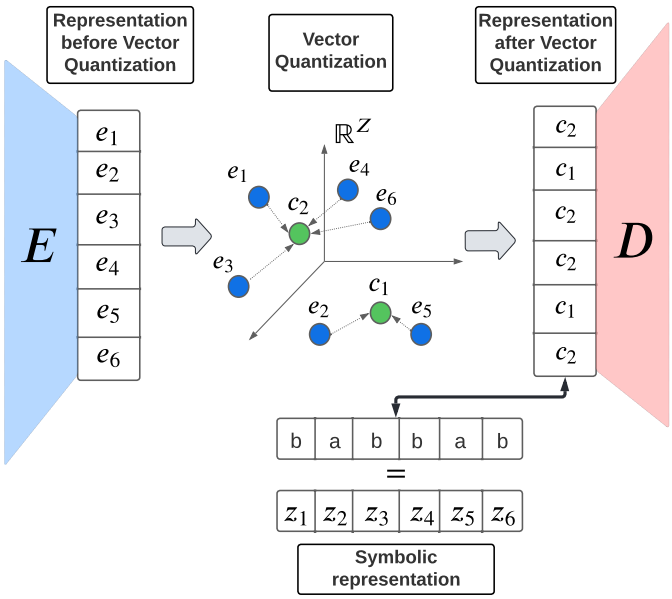


Figure 3.7: Visualization of the vector quantization mechanism in the context of an auto-encoder. E and D stand respectively for Encoder and Decoder.

3.4.2 Meeting the requirements

The proposed architecture aims to meet the requirements for an interpretable neural representation. Below, we discuss how each requirement is addressed in the architecture's design.

Discrete symbolic representation. The VQ mechanism, also known as a self-organizing map (Kohonen, 1990), allows the centroids to move as iterations progress. This mechanism assigns nearby vectors to the same centroid while maintaining temporal consistency. Thus, our architecture encourages similar patterns to be quantized to the same vector. The symbolic representation is obtained directly from the centroid indices used during vector quantization (see Figure 3.7). Since the index represents the vector it characterizes, the sequence of centroid indices obtained during the vector quantization stage is a faithful symbolic representation of the time series.

Temporal consistency. The proposed architecture uses convolution operations that guarantee the temporal consistency of the representation. We can easily compute the input that produced a given output from a convolution sequence by determining the receptive field (see Appendix B.2 for the details). In addition, convolutions are local operations that slide over the input, which allows them to preserve the temporal order of the input. Using a pointwise VQ mechanism does not affect the temporal consistency property of our architecture.

A decodable representation. The symmetric and unbiased encoder-decoder design guarantees that the representation is decodable. The reconstruction criterion ensures that the reconstruction converges towards the original time series. It enables the visualization of the learned representation while maintaining temporal consistency and enabling separate decoding of each representation component.

Shift equivariance properties. Due to the adaptive polyphase upsampling and downsampling (Chaman and Dokmanic, 2021), both encoder and decoder are shift equivariant. The shift equivariance properties would be lost with classical strided convolutions (Cohen and Welling, 2016). This property is crucial for understanding the meaning of the elements in the symbolic representation. It ensures that identical symbolic elements always represent the same pattern (when there are no edge effects, see Section 3.7.3), regardless of their position.

An adjustable representation. By changing the number of blocks B in the encoder (and by the symmetry in the decoder), we can adjust the information learned in the representation, enabling us to adapt the representation to different frequency levels. The higher the number of blocks, the shorter the representation sequence and the more global the information captured for an element of the representation. Thus, we define the number of blocks as a hyperparameter of the architecture.

3.4.3 Training

The unsupervised architecture (see Figure 3.8) is trained according to Equation (3.2). To simplify the notations, we present the loss for an instance $\boldsymbol{x}$. The notations remain the same as those introduced above, except for the following. The operator sg is the stop gradient operator whose derivative is zero during the backward computation time. The encoder is defined as ϕ_θ and the decoder as $\psi_{\theta'}$. In Section 3.3, ϕ_θ characterizes the function that maps the time series to the symbolic representation. Here ϕ_θ stands for the function that maps the time series to the embeddings (before the discretization phase). On the other hand, here, $\psi_{\theta'}$ characterizes the function that maps the quantized vector representation to the reconstruction. The temporal vector sequence after quantization is denoted $\boldsymbol{E^q} \in \mathbb{R}^{T' \times Z}$. The unsupervised architecture is then optimized by solving the following optimization problem.

$$\arg\min_{\theta,\theta',E} ||\boldsymbol{x} - \psi_{\theta'}(\boldsymbol{E^q})||_2^2 + ||sg[\phi_\theta(\boldsymbol{x})] - \boldsymbol{E^q}||_2^2 + \beta||\phi_\theta(\boldsymbol{x}) - sg[\boldsymbol{E^q}]||_2^2. \tag{3.2}$$

- The first term of the loss refers to the reconstruction ability of the model. In practice, we consider the mean square error pointwise between the ground truth time series and the decoder output. This part of the loss optimizes the

decoder and the encoder. It is important to note that the arg min operator in Equation (3.1) is not differentiable at the VQ level. In van den Oord et al. (2017), the authors suggest passing the gradients of the quantized vectors to their corresponding encoder vector outputs. This allows the reconstruction loss to optimize the encoder.

- The second term in the loss updates the used centroids $\boldsymbol{E^q}$ by moving them toward the embedding vectors $\phi_\theta(\boldsymbol{x})$ assigned to them. The centroids are initialized with a Gaussian distribution. This approach has proven to be stable in training and allows an independent choice of solver for the rest of the loss. In the model implementation, the Adam solver (Kingma and Ba, 2015) is used for the other terms of the loss.

- The last part of the loss is the commitment loss and ensures that the encoder outputs do not land too far from the used centroids. It guarantees better training stability.

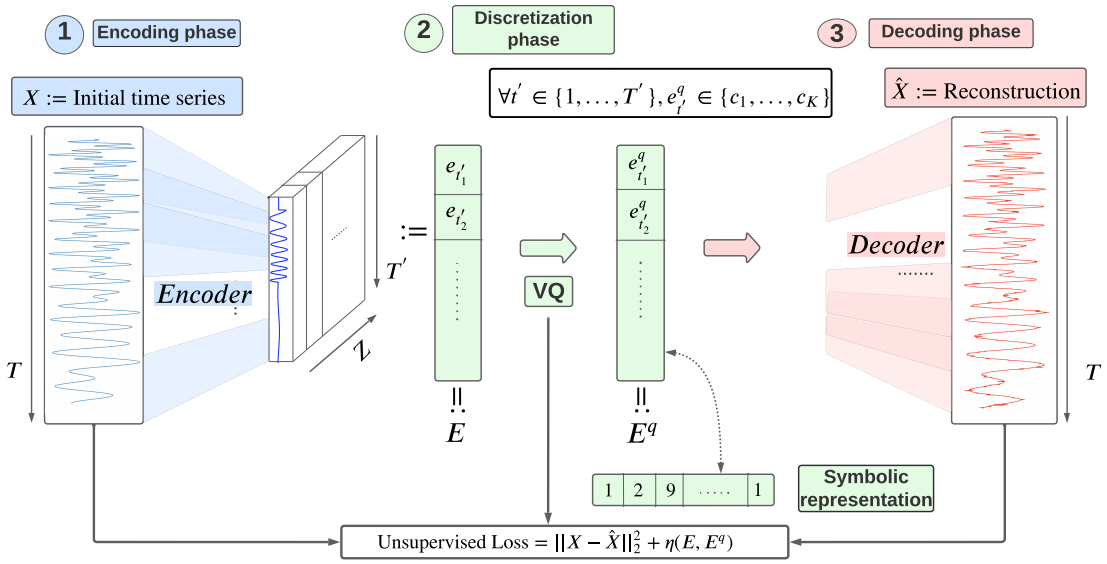


Figure 3.8: Global unsupervised architecture overview. η refers the last two terms in Equation (3.2).

3.5 Downstream task: classification over extracted representations

Similar to dictionary methods for time series classification (Lin et al., 2003; Schäfer and Högqvist, 2012), the proposed representation learning process leads to an interpretable sequential symbolic representation of the time series. The series is represented by the sequence of the indices of the quantized vectors (the symbolic representation). We propose to evaluate the obtained symbolic representation with downstream classification tasks. When classifying symbolic representations, a common practice is to extract histograms of symbolic subsequences and feed them into a k-NN, or linear classifier (Nguyen et al., 2017, 2019). For example, in Nguyen et al. (2017), authors propose an efficient algorithm for a logistic regression classifier based on coordinate gradient descent to find the most relevant subsequences. In the following, we propose a linear classifier based on the presence or absence of discriminative subsequences inspired by existing methods. This classification method preserves the interpretability provided by the representation and ensures an in-situ, global interpretable classification decision.

3.5.1 Classification using a unique symbolic representation

In our classification process, we focus only on using subsequences of lengths one and two to characterize the symbolic representation. This approach effectively captures local dynamics while maintaining a low-dimensional feature space. Let ν be the cardinal of the set of possible symbolic subsequences of length one and two, and $\boldsymbol{z^{(i)}}$ be a symbolic representation (extracted using our auto-encoder) for sample i. We can extract for the representation $\boldsymbol{z^{(i)}}$, the vector $\boldsymbol{h^{(i)}}$ which indicates if a subsequence (of length one or two) is present in $\boldsymbol{z_i}$. Thus, $\boldsymbol{h^{(i)}}$ is a vector of size ν composed of 0 and 1 elements ($\boldsymbol{h^{(i)}} \in \{0,1\}^{\nu}$). The feature vectors $\{\boldsymbol{h^{(i)}}, i \in \{1,...,n\}\}$ representing the training set are then used to solve the classification problem using logistic regression as follows.

$$\arg\min_{w,b} \frac{1-\rho}{2}\boldsymbol{w^T w} + \rho\|\boldsymbol{w}\|_1 + \lambda \sum_{i=1}^{n} \log\left(\exp\left(-y_i\left(\boldsymbol{h^{(i)^T} w} + b\right)\right) + 1\right). \quad (3.3)$$

with λ the regularization parameter, ρ the trade-off between the ℓ_1 penalty and the ℓ_2 penalty and b the bias. Sparsity in interpretable classification models is desirable to more easily understand the decision (Wu et al., 2018; Li et al., 2019). In the case of multi-class classification, the usual One vs All method (Rifkin and Klautau, 2004) is applied to provide better interpretability.

However, classification using a unique representation (e.g., a representation where

the encoder-decoder is composed of B blocks) does not allow the use of features of different frequencies. Therefore, it is desirable to train several representations corresponding to different temporal dimension reductions (deeper or shallower architectures) in order to classify using different frequency features.

3.5.2 Classification using multiple symbolic representations

According to Section 3.4.1 the depth of the architecture is directly related to the number of blocks B in the encoder (and by symmetry in the decoder). We train the proposed unsupervised model for D different depths ($B \in \{1, 2, ..., D\}$). Thus, we get D representations of different length, which capture different features ($\boldsymbol{h}^{(i)^1}$, $\boldsymbol{h}^{(i)^2}, ..., \boldsymbol{h}^{(i)^D}$). Then we apply the following steps:

1. A penalized logistic regression (Equation (3.3)) is performed separately on each extracted features vector $\boldsymbol{h}^{(i)}$.
2. For each $\boldsymbol{h}^{(i)}$, we recover the features whose logistic regression coefficients have non-zero values. Thus, we obtain an aggregated discriminative set of features for the D representations.
3. We perform a final penalized logistic regression on the obtained set.

It is important to highlight two observations. • Firstly, this classification method retains interpretability because each feature is interpretable by the construction of the representation, making it straightforward to interpret the coefficients of the logistic regression associated with a feature. • Secondly, the number of representations we use corresponds to the number of times the initial time series can be halved while being greater than a threshold (T' should be greater than this threshold). In practice, the number of representations we use for the UCR datasets rarely exceeds five. Keeping the number of representations small is essential for preserving interpretability. In particular, a large number of representations has a strong negative impact on interpretability (Lipton, 2018).

3.6 Experiments

In our experiments, the unsupervised architectures are trained using only the training set. The symbolic test representations are constructed by passing the test data through the trained architecture. We also set the number of available centroids to 32 and the dimension of the latent space is 64. For training the unsupervised models, we set β to 0.25 in Equation (3.2) as realized in van den Oord et al. (2017).

3.6.1 Quantitative experiments

The quantitative experiments are performed on 25 selected datasets from the UCR archive (Dau et al., 2019). These datasets meet specific criteria: variety of application types (sensor, motion, image, device), a minimum of 50 training and 50 test instances, and a maximum of seven classes. Table 3.1 presents the results of our model compared to the interpretable in-situ methods SAX SEQL, SAX VSM, FS and LTS presented in Section 4.2. We also compare our accuracy results to the well-known k-NN classifier coupled with Dynamical Time Warping (DTW) distance (Rakthanmanon et al., 2012). This classifier is sometimes presented as interpretable, but it does not allow the extraction of localizable discriminative features. We reuse the accuracy results from Nguyen et al. (2019) for comparison.

Table 3.1: Accuracy on 25 UCR datasets compare to in-situ interpretable methods. The best results are in bold and the second best results are underlined.

Datasets	Ours	SAX SEQL	SAX VSM	FS	LTS	DTW CV
Coffee	0.964	**1.000**	0.929	0.929	**1.000**	**1.000**
Computers	**0.728**	0.676	0.620	0.500	0.584	0.620
DistalPhalanxOAG	0.755	0.818	**0.842**	0.655	0.779	0.626
DistalPhalanxOC	0.732	0.718	0.728	**0.750**	0.719	0.725
DistalPhalanxTW	0.640	**0.748**	0.604	0.626	0.626	0.633
Earthquakes	0.734	**0.789**	0.748	0.705	0.741	0.727
ECG5000	**0.932**	0.924	0.910	0.923	**0.932**	0.925
FordA	0.883	0.851	0.827	0.787	**0.957**	0.691
GunPoint	0.940	0.987	0.987	0.947	**1.000**	0.913
Ham	0.705	0.705	**0.810**	0.648	0.667	0.600
Herring	**0.656**	0.578	0.625	0.531	0.625	0.531
ItalyPowerDemand	0.906	0.734	0.816	0.917	**0.970**	0.955
LargeKitchenApp	0.864	0.760	**0.877**	0.560	0.701	0.795
PhalangesOC	0.748	0.717	0.710	0.744	**0.765**	0.761
ProximalPhalanxOC	0.818	0.818	0.828	0.804	**0.834**	0.790
ProximalPhalanxOAG	0.839	0.844	0.824	0.780	**0.849**	0.785
ProximalPhalanxTW	0.771	**0.792**	0.610	0.702	0.776	0.756
RefrigerationDevices	0.533	0.541	**0.653**	0.333	0.515	0.440
ScreenType	0.499	0.461	**0.512**	0.413	0.429	0.411
ShapeletSim	0.994	0.994	0.717	**1.000**	0.950	0.700
SmallKitchenApp	**0.795**	0.776	0.579	0.333	0.664	0.672
Strawberry	**0.962**	0.954	0.957	0.903	0.911	0.946
Wafer	0.975	0.993	**0.999**	0.997	0.996	0.995
Wine	0.759	0.556	**0.963**	0.759	0.500	0.611
Worms	**0.714**	0.536	0.558	0.649	0.610	0.532
Mean	**0.793**	0.770	0.769	0.715	0.764	0.725

Referring to Table 3.1, our method gives better results on average than the other in-situ methods interpretable on these datasets. Our method results in a minimum 2.2 percentage points increase compared to other interpretable in-situ methods. Even if our method does not come first in all cases, the quantitative results are very promising for two main reasons. First, only a small subpart of all possible symbolic subsequences are used as our method aims to enforce sparsity in order to favour interpretability. Secondly, our representations are learned without any supervision and thus are not guided by the underlined task.

In addition, we evaluate the accuracy of logistic regression on our symbolic neural representation against SVM on top of neural representation. This classification framework is often used to evaluate the best unsupervised neural representation when the neural representation is a simple vector (Yue et al., 2022; Franceschi et al., 2019). Previous research has found that the best-unsupervised methods with this framework are TS2Vec (Yue et al., 2022) and T-Loss (Franceschi et al., 2019). We compare these two methods with our own on the 25 previous datasets and find that all three perform similarly in accuracy (0.789 ± 0.15 for T-Loss and 0.807 ± 0.15 for TS2Vec compared to 0.793 ± 0.13 for our method). These results suggest that the constraints we impose on our architecture to satisfy the interpretability requirement do not significantly deteriorate the expressiveness of the unsupervised representation.

3.6.2 Qualitative experiments

Among the 25 previous UCR datasets, not all are suitable for interpretability because a limited number of features cannot discriminate between the classes. In most papers dealing with in-situ interpretability for time series, the commonly used datasets are GunPoint, ShapeletSim, or Coffee datasets (Nguyen et al., 2017; Senin and Malinchik, 2013). For the qualitative analysis, we focus on the GunPoint and ShapeletSim datasets (Dau et al., 2019).

3.6.2.1 GunPoint dataset

The GunPoint dataset consists of two actors performing a movement with their right hand, with two classes: Gun-Draw (class 0) and Point (class 1).

Representations. The time series are z-normalized for each instance. We train five models with different architecture depths to capture different features in the time series. The architecture is only trained using the train time series; then, a simple forward pass is used to get the representation for each time series from the test dataset. After training, we obtain five representations of different lengths. As shows in Table 3.2, the representation generalizes easily to the test. The train and test reconstructions for this dataset are good. It may seem surprising that the

pointwise train MAE increases slightly with depth, but this is because the other terms in Equation (3.2) are higher for less deep representations.

Table 3.2: Information on the different representations learned for the *GunPoint* dataset.

Depth	Temporal downscaling	Symbolic representation length	Pointwise train MAE	Pointwise test MAE
$B = 1$	$2^1 = 2$	75	0.044	0.045
$B = 2$	$2^2 = 4$	38	0.038	0.045
$B = 3$	$2^3 = 8$	19	0.032	0.047
$B = 4$	$2^4 = 16$	10	0.027	0.085
$B = 5$	$2^5 = 32$	5	0.023	0.085

Classification. For each trained representation $\boldsymbol{z}^{\boldsymbol{B}}$ we construct the binary vector $\boldsymbol{h}^{\boldsymbol{B}}$ ($B \in \{1, 2, 3, 4, 5\}$). We then fit a logistic regression for each representation separately. For each logistic regression, ρ and λ are found by cross-validation (Pedregosa et al., 2011). We encourage a strong penalty ℓ_1 to set to zero the coefficients for non-discriminating features for each representation. Table 3.3 shows for each logistic regression the initial number of features and the number of features whose regression coefficient is different from zero.

Table 3.3: Information on logistic regression for each representation (step 1 in process 3.5.2).

Depth	$\boldsymbol{h}^{\boldsymbol{B}}$ size	Number of features actually used to classify
$B = 1$	115	23
$B = 2$	203	26
$B = 3$	293	24
$B = 4$	328	26
$B = 5$	187	11

Then, we fit the final logistic regression on each extracted feature (whose regression coefficients differ from 0). The final logistic regression performs on 110 features. After training, 89 coefficients are set to zero in this regression because of the penalty effect. Initially, the concatenation of feature vectors is a vector of size 1126, but only 21 features are ultimately used for the classification problem.

The test accuracy is 0.94, and the train accuracy is 1. Now that the final logistic regression is fitted, we can look at the coefficients of this regression. Table 3.4 shows the most critical features (whose relative importance is greater than 5 percent).

Table 3.4: Details of the most discriminative features in the final logistic regression (step 3 in process 3.5.2), intercept is 5.13.

Depth	Symbolic subsequence	Logistic regression coefficients	Relative importance
$B = 1$	bb	- 0.92	7.0 %
$B = 1$	fc	- 0.93	7.2 %
$B = 1$	hk	- 1.20	9.2 %
$B = 1$	kg	- 0.98	7.4 %
$B = 1$	kh	- 0.76	5.8 %
$B = 3$	Fx	1.23	9.4 %

Our representation is interpretable and allows us to decode symbolic subsequences, so we can use the extracted features and logistic regression coefficients to gain insight into the problem. We can interpret the classification decision at the global level as well as at the local level.

Global interpretability. It consists in visualizing which feature at the model level allows the classification of the time series correctly. With our architecture, decoding the discriminative symbolic subsequences suffices to understand what the unsupervised model learned, and visualize the reconstructed subseries. Let us consider the symbolic subsequence *'Fx'* (for depth $B = 3$ in Table 3.4). This subsequence is the most discriminative subsequence for class one. When we decode this subsequence in Figure 3.9, we obtain a subseries that characterizes the way the finger is raised. This subseries differs from the way the gun is raised.

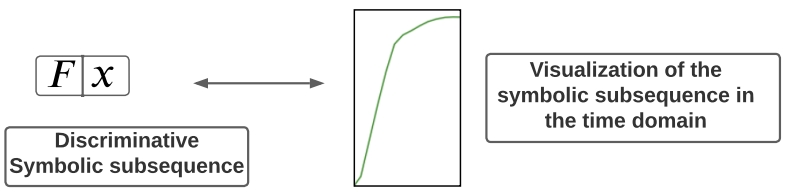


Figure 3.9: Global interpretability decision. Visualization of the most discriminative symbolic subsequence for class 1 (Point).

Local interpretability. It consists in visualizing for an instance the regions of the time series that make the decision. For example, take the symbolic subsequence

'hk' (for depth $B = 1$ in Table 3.4), which is the most discriminating subsequence for class 0. Figure 3.10 presents an instance whose representation ($B = 1$) contains the subsequence *'hk'*. Then, we highlight in red the pre-image of this subsequence using the receptive fields of the convolutions (see Appendix B.2 for the details).

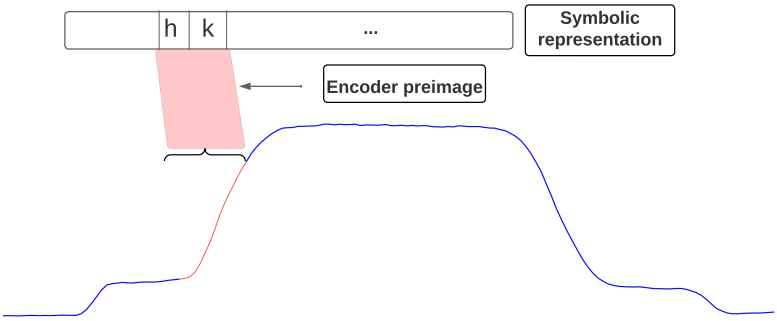


Figure 3.10: Local interpretability for a class 0 (Gun) instance.

3.6.2.2 ShapeletSim dataset

The *ShapeletSim* dataset comprises two classes, class 0 is purely white noise, and class 1 includes a triangle shape at a random location. The classification problem is easily interpretable because only the presence or absence of a triangle characterizes the difference between the two classes.

As in the previous use case, we construct the different symbolic representations in an unsupervised manner and then extract the discriminative features. The highest coefficient in the final logistic regression is associated with the symbolic subsequence *'wjdddjw'*. Using the decoder, we decoded the subsequence *'wjdddjw'* and examined the resulting decoded shape. Figure 3.11 presents visualization for global interpretable classification decision.

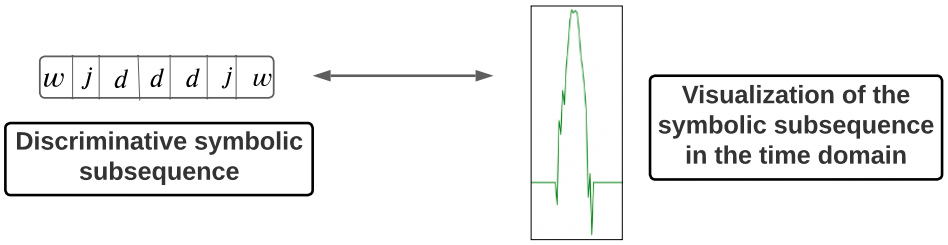


Figure 3.11: Global interpretability decision. Visualization of the most discriminative symbolic subsequence for class 1 (includes triangle).

We observed in Figure 3.11 that the decoded shape matches the vertex of the decoded triangle. It is worth noting that this demonstration of global interpretability does not require any specific instance for visualization.

On the other hand, Figure 3.12 shows a visualization of the local interpretability for the symbolic subsequence *'wjdddjw'* for a given sample. We retrieve the triangular shape by computing the pre-image of the discriminating subsequence for this given sample.

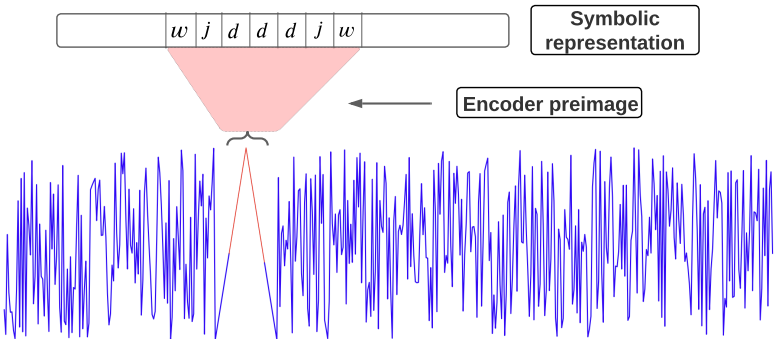


Figure 3.12: Local interpretability for a class 1 (includes triangle) instance.

3.7 Limitations

In this section, we identify and discuss the main limitations of the proposed method. We will cover the following points: (i) The impact of key hyperparameters on the method's effectiveness. (ii) A comparative analysis of the method's performance against supervised neural networks. (iii) A discussion of the edges effects of reconstructions on interpretability. (iv) The application of the method to multivariate time series.

3.7.1 Critical hyperparameters in the proposed method

The vector quantization encoder-decoder that extracts the discrete neural representation relies heavily on two hyperparameters that are difficult to tune.

- **The compression level (ratio of series length to latent representation length).** Depending on the use case and the time series, finding the optimal compression level is not straightforward. To tackle this problem, we have proposed using several levels of representations with different compression levels to capture information at various frequencies. However, considering too many levels of representation can hinder the interpretability of the classifier built on top of it (Lipton, 2018).

- **The symbolic vocabulary size.** The vocabulary size is an essential element: the larger the vocabulary, the better the reconstruction and the less expressive the representation (and the more computationally expensive the downstream

classification method). This hyperparameter is difficult to optimize by cross-validation, because we do not necessarily want the best reconstruction at all costs, but the most expressive representation with the best possible reconstruction. We propose an ablation study in Appendix B.3, which shows that the vocabulary size is a critical hyperparameter.

In addition, classification requires optimization of the following key hyperparameters.

- **The ℓ_1 and ℓ_2 regularization hyperparameters for logistic regression.** These hyperparameters can be determined through cross-validation, using tools like scikit-learn (Pedregosa et al., 2011). However, there is a trade-off to consider. We prefer a strong ℓ_1 regularization to yield only a few discriminative subsequences, even if it results in a slightly lower accuracy score.
- **The length of the symbolic subsequences used to identify discriminating patterns.** In the extensive classification experiment on 25 UCR datasets, we searched for discriminative symbolic subsequences of length 1 or 2 due to computational constraints. While exploring the entire subsequence space would yield more accurate and interpretable results, it is computationally prohibitive with traditional gradient descent optimization.

3.7.2 Comparison with supervised neural network classifiers

As shown in Section 3.6.1, our proposed method outperforms or matches existing interpretable classifiers and is on part with state-of-the-art unsupervised neural representation models paired with SVM classifiers.

However, an important question remains: *How does our method compare to the best supervised neural networks?* In Table 3.5, we present a comparison of our method with purely supervised neural networks on the 25 datasets investigated in Table 3.1. The baseline results are derived from the works of Fawaz et al. (2018) and Ismail Fawaz et al. (2020).

Table 3.5: Classification results on 25 UCR datasets. Comparison of our proposed model with supervised neural networks.

	Our proposed method	**ResNet**	**AlexNet**	**CNN**	**MLP**
Accuracy	79.3 %	**82.3** %	82.0 %	72.2 %	69.5 %

In Table 3.5, the accuracy scores of ResNet and AlexNet outperform our method by 3% and 2.7%, respectively. However, our method significantly outperforms the

CNN and MLP models. These results indicate a small gap between our proposed method and the state-of-the-art neural network classifiers. In Section 6.2, we suggest future directions to improve the accuracy of our method while preserving its interpretability.

3.7.3 Edge effects of reconstructions on interpretability

Due to the structure of the decoder, we note that for a given neural symbolic subsequence, the symbolic elements to the left and right of this subsequence can influence its reconstruction. This neighborhood effect implies that, in practice, the reconstruction of the same symbolic subsequence may differ at the edges between different symbolic neural representations. Such variations may affect the global interpretability of the visualization of discriminative subsequences. However, we found that these effects are primarily noticeable at the edges of the extracted patterns rather than within the patterns themselves. Future research could focus on mitigating these edge effects.

3.7.4 A method only tested on univariate time series

This work focuses on univariate time series, but the proposed method could be adapted to multivariate series using 1D convolutions with multiple input and output channels. Regarding the discrete representation, we hypothesize that a larger vocabulary size would be required to characterize multivariate time series accurately. While an increase in the number of possible symbols may reduce interpretability, we believe that future research should explore the viability of testing this approach with a small number of input channels, such as two, which could maintain strong performance and interpretability.

3.8 Conclusion

We first present essential requirements for building an interpretable neural representation for time series and then present an architecture that satisfies these requirements. The proposed unsupervised symbolic neural model fills a gap between symbolic and neural representations for time series. It has the advantage of allowing global interpretability of downstream classification tasks, while guaranteeing high expressiveness and good performance. The constructed representation has been evaluated both qualitatively and quantitatively on classification tasks. We show promising accuracy results compared to both in-situ interpretable and neural methods. Additionally, the proposed interpretability provides an understanding of the model's classification decisions at both global and local levels for a broad range of time series. We believe that the design of interpretable and explainable deep learning

models will become increasingly important with the new European AI regulations, i.e., the AI Act (Panigutti et al., 2023).

Much of our work has been devoted to building an unsupervised neural architecture that meets the interpretability requirements. For future work, the use of coordinate descent, as described in Ifrim and Wiuf (2011), can improve the accuracy of these models by identifying longer symbolic subsequences that help distinguish between classes. In addition, it seems interesting to explore the choice of reconstruction loss in the unsupervised architecture.

Chapter 4

Time Series Continuous Modeling for Imputation and Forecasting with Implicit Neural Representations

In this chapter, we present a novel modeling approach for time series imputation and forecasting using a continuous representation mechanism. The proposed model addresses the challenges often encountered in real-world data, such as irregular samples, missing data, or unaligned measurements from multiple sensors. Our method relies on a continuous-time-dependent model of the series' evolution dynamics. It leverages adaptations of conditional, implicit neural representations for sequential data. A modulation mechanism, driven by a meta-learning algorithm, allows adaptation to unseen samples and extrapolation beyond observed time-windows for long-term predictions. The model provides a highly flexible and unified framework for imputation and forecasting tasks across a wide range of challenging scenarios. It achieves state-of-the-art performance on classical benchmarks and outperforms alternative time-continuous models.

Le Naour, E., Serrano, L., Migus, L., Yin, Y., Agoua, G., Baskiotis, N., Gallinari, P., and Guigue, V. **Time Series Continuous Modeling for Imputation and Forecasting with Implicit Neural Representations.** *Transactions on Machine Learning Research (TMLR)* 2024.

4.1 Introduction

Time series analysis and modeling are ubiquitous in a wide range of fields, including industry, medicine, and climate science. The variety, heterogeneity and increasing number of deployed sensors, raise new challenges when dealing with real-world problems for which current methods often fail. For example, data are frequently irregularly sampled, contain missing values, or are unaligned when collected from distributed sensors (Schulz and Stattegger, 1997; Clark and Bjørnstad, 2004). Recent advancements in deep learning have significantly improved state-of-the-art performance in both data imputation (Cao et al., 2018; Du et al., 2023) and forecasting tasks (Zeng et al., 2022; Nie et al., 2022). Many state-of-the-art models, such as transformers, have been primarily designed for dense and regular grids (Wu et al., 2021; Nie et al., 2022; Du et al., 2023). They struggle to handle irregular data and often suffer from significant performance degradation (Chen et al., 2001; Kim et al., 2019).

Our objective is to explore alternatives to state-of-the-art (SOTA) transformers able to handle, in a unified framework, imputation and forecasting tasks for irregularly, arbitrarily sampled, and unaligned time series sources. Time-dependent continuous models (Rasmussen and Williams, 2006; Garnelo et al., 2018; Rubanova et al., 2019) offer such an alternative. However, until now, their performance has lagged significantly behind that of models designed for regular discrete grids. A few years ago, implicit neural representations (INRs) emerged as a powerful tool for representing images as continuous functions of spatial coordinates (Sitzmann et al., 2020; Tancik et al., 2020) with recent new applications such as image generation (Dupont et al., 2022) or even modeling dynamical systems (Yin et al., 2023).

In this work, we leverage the potential of conditional INR models within a meta-learning approach to introduce TimeFlow: a unified framework designed for modeling continuous time series and addressing imputation and forecasting tasks with irregular and unaligned observations. Our key contributions are:

- We propose a novel framework that excels in modeling time series as continuous functions of time, accepting arbitrary time step inputs, thus enabling the handling of irregular and unaligned time series for both imputation and forecasting tasks. This is one of the very first attempts to adapt INRs that enables efficient handling of both imputation and forecasting tasks within a unified framework. The methodology which leverages the synergy between the model components, evidenced in the context of this application, is a pioneering contribution to the field.

- We conducted an extensive comparison with state-of-the-art continuous and discrete models. It demonstrates that our approach outperforms continuous and discrete SOTA deep learning approaches for imputation. As for long-term forecasting, it outperforms existing continuous models both on regular and irregular samples. It is on par with SOTA discrete models on regularly sampled time series while allowing for a much greater flexibility for irregular samplings, allowing to cope with situations where discrete models fail. Furthermore, we prove that our method effortlessly handles previously unseen time series and new time windows, making it well-suited for real-world applications.

4.2 Related work

Discrete methods for time series imputation and forecasting. Recently, Deep Learning (DL) methods have been widely used for both time series imputation and forecasting. For imputation, BRITS (Cao et al., 2018) uses a bidirectional recurrent neural network (RNN). Alternative frameworks were later explored, e.g., GAN-based (Luo et al., 2018, 2019; Liu et al., 2019b), VAE-based (Fortuin et al., 2020), diffusion-based (Tashiro et al., 2021), matrix factorization-based (TIDER, Liu et al., 2023) and transformer-based (SAITS, Du et al., 2023) approaches. These methods cannot handle irregular time series. In situations involving multiple sensors, such as those placed at different locations, incorporating new sensors necessitates retraining the entire model, thereby limiting their usability. For forecasting, most recent DL SOTA models are based on transformers. Initial approaches apply plain transformers directly to the series, each token being a series element (Zhou et al., 2021; Liu et al., 2022; Wu et al., 2021; Zhou et al., 2022). These transformers may underperform linear models as shown in (Zeng et al., 2022). PatchTST (Nie et al., 2022) significantly improved transformers SOTA performance by considering sub-

series as tokens of the series. However, all these models cannot handle properly irregularly sampled look-back windows.

Continuous methods for time series. Gaussian Processes (Rasmussen and Williams, 2006) have been a popular family of methods for modeling time series as continuous functions. They require choosing an appropriate kernel (Corani et al., 2021) and may suffer limitations in large dimensions settings. Neural Processes (NPs) (Garnelo et al., 2018; Kim et al., 2019) parameterize Gaussian processes through an encoder-decoder architecture leading to more computationally efficient implementations. NPs have been used to model simple signals for imputation and forecasting tasks, but struggle with more complex signals. Bilos et al. (2023) parameterizes a Gaussian Process through a diffusion model, but the model has difficulty adapting to a large number of timestamps. Other approaches such as Brouwer et al. (2019) and Rubanova et al. (2019) model time series continuously with latent ordinary differential equations. mTAN (Shukla and Marlin, 2021), a transformer model, uses an attention mechanism to impute irregular time series. While these approaches have shown significant progress in continuous modeling for time series, we observed that their performances on imputation and forecasting tasks are inferior compared to the aforementioned discrete models (Table 4.2, Table 4.5).

Implicit neural representations. The recent development of implicit neural representations (INRs) has led to impressive results in computer vision (Sitzmann et al., 2020; Tancik et al., 2020; Fathony et al., 2021; Mildenhall et al., 2021). INRs can represent data as a continuous function, which can be queried at any coordinate. While they have been applied in other fields such as physics (Yin et al., 2023) and meteorology (Huang and Hoefler, 2023), there has been limited research on INRs for time series analysis. Prior works (Fons et al., 2022; Jeong and Shin, 2022) focused on time series generation for data augmentation and on time series encoding for reconstruction but are limited by their fixed grid input requirement. DeepTime (Woo et al., 2022) is the closest work to our contribution. DeepTime learns a set of basis INR functions from a training set of multiple time series and combines them using a Ridge regressor. This regressor allows it to adapt to new time series. It has been designed for forecasting only. The original version cannot handle imputation properly and was adapted to do so for our comparisons. In our experiments, we will demonstrate that TimeFlow significantly outperforms DeepTime in imputation and also in forecasting tasks when dealing with missing values in the look-back window. TimeFlow also shows a slight advantage over DeepTime in forecasting regularly sampled series.

4.3 The TimeFlow framework

4.3.1 Problem setting

We aim to develop a unified framework for time series imputation and forecasting that reduces dependency on a fixed sampling scheme for time series. We introduce the following notations for both tasks. During training, in the imputation setting, we have access to time series in an observed temporal grid denoted as $\mathcal{T}_{in}$, which is a subset of the dense temporal support $\mathcal{T}$. In the forecasting setting, we observe time series within a limited past time grid, referred to as the 'look-back window' and denoted as $\mathcal{T}_{in}$ (a subset of $\mathcal{T}$), as well as a future grid, the 'horizon', denoted as $\mathcal{T}_{out}$ (also a subset of $\mathcal{T}$). At test time, in both cases, and given observed values in a temporal grid $\mathcal{T}^*_{in}$ included in a possibly new temporal window $\mathcal{T}^*$, our objective is to infer the time series values within $\mathcal{T}^*$. $\mathcal{T}^* = \mathcal{T}$ if we infer values in the training temporal support (e.g. in the classical imputation scenario, see Section 4.4.1), $\mathcal{T}^* \neq \mathcal{T}$ if we infer for a new temporal support(e.g. in the forecasting setting, see Section 4.4.2).

4.3.2 Key components

Our framework is articulated around three key components:

(i) **INR-based time-continuous functions**: a discrete time series $\boldsymbol{x} = (\boldsymbol{x_{t_1}}, \boldsymbol{x_{t_2}}, \ldots, \boldsymbol{x_{t_k}})$ can be represented by an underlying time-continuous function $\mathbf{x}$: $t \in \mathbb{R}_+ \rightarrow \boldsymbol{x_t} \in \mathbb{R}^d$. We want to approximate the ground-truth $\mathbf{x}$ by employing implicit neural representations (INRs), which are neural networks capable of learning a parameterized continuous function f_θ from discrete data by minimizing the reconstruction loss between observed data and network's outputs.

(ii) **Conditional INRs with modulations**: An INR can represent only one function, whether it's an image or a time series. To effectively represent a collection of time series $(\boldsymbol{x^{(j)}})_j$ using INRs, we improve their encoding by incorporating per-sample modulations, which we denote as $\psi^{(j)}$. These modulations condition the parameters $\boldsymbol{\theta}$ of the INRs. We use the notation $f_{\boldsymbol{\theta},\psi^{(j)}}$ to refer to the conditioned INR with the modulations $\psi^{(j)}$.

(iii) **Optimization-based encoding**: the conditioning modulation parameters $\psi^{(j)}$ are calculated as a function of codes $\boldsymbol{z^{(j)}}$ that represent the individual sample series. We acquire these codes $\boldsymbol{z^{(j)}}$ through a meta-learning optimization process using an auto-decoding strategy. Notably, auto-decoding has been found to be more efficient for this purpose than set encoders (Kim et al., 2019).

In the following sections, we will elaborate on each component of our method. Given that the choices made for each component and the methodology developed to enhance their synergy are essential aspects, we provide a discussion of the various choices involved in Section 4.3.4.

INR-based time-continuous functions. We implement our INR with Fourier features and a feed-forward network (FFN) with ReLU activations, i.e. for a time coordinate $t \in \mathcal{T}$, the output of the INR $f_{\boldsymbol{\theta}}$ is given by $f_{\boldsymbol{\theta}}(t) = \text{FFN}(\boldsymbol{\gamma(t)})$. The Fourier Features $\gamma(\cdot)$ are a frequency embedding of the time coordinates used to capture high-frequencies (Tancik et al., 2020; Mildenhall et al., 2021). In our case, we chose $\boldsymbol{\gamma(t)} := (\sin(\pi t), \cos(\pi t), \cdots, \sin(2^{N-1}\pi t), \cos(2^{N-1}\pi t))$, with N the number of fixed frequencies. For an INR with L layers, the output is computed as follows: (i) we get the frequency embedding $\boldsymbol{\phi_0} = \boldsymbol{\gamma(t)}$, (ii) we update the hidden states according to $\boldsymbol{\phi_l} = \text{ReLU}(\boldsymbol{\theta_l \phi_{l-1}} + \boldsymbol{b_l})$ for $l = 1, \ldots, L$, (iii) we project onto the output space $f_{\boldsymbol{\theta(t)}} = \boldsymbol{\theta_{L+1} \phi_L} + \boldsymbol{b_{L+1}}$.

Conditional INRs with modulations. As indicated, sample conditioning of the INR is performed through modulations of its parameters. In order to adapt rapidly the model to new samples, the conditioning should rely only on a small number of the INR parameters. This is achieved by modifying only the biases of the INR through the introduction of an additional bias term $\boldsymbol{\psi}_l^{(j)}$ for each layer l, also known as *shift modulation*. To further limit the versatility of the conditioning, we generate the instance modulations $\boldsymbol{\psi}^{(j)}$ from compact codes $\boldsymbol{z}^{(j)}$ through a linear hypernetwork h with parameters $\boldsymbol{w}$, i.e., $\boldsymbol{\psi}^{(j)} = h_{\boldsymbol{w}}(\boldsymbol{z}^{(j)})$. Consequently, the approximation of a time series $\boldsymbol{x}^{(j)}$, denoted globally as $f_{\boldsymbol{\theta}, h_{\boldsymbol{w}}(\boldsymbol{z}^{(j)})}$, will depend on shared parameters θ and w that are common among all the INRs involved in modeling the series family and on the code $\boldsymbol{z}^{(j)}$ specific to series $\boldsymbol{x}^{(j)}$. The output of the l-th layer of the modulated INR is given by $\boldsymbol{\phi_l} = \text{ReLU}(\boldsymbol{\theta_l \phi_{l-1}} + \boldsymbol{b_l} + \boldsymbol{\psi}_l^{(j)})$, where $\boldsymbol{\psi}_l^{(j)} = \boldsymbol{W_l z}^{(j)}$, and $\boldsymbol{w} := (\boldsymbol{W_l})_{l=1}^{L}$ are the parameters of the hypernetwork $h_{\boldsymbol{w}}$. This design enables gathering information across samples into the common parameters of the INR and hypernetwork, while the codes contain only specific information about their respective time-series samples. The architecture is illustrated in Figure 4.1.

Optimization-based encoding. We condition the INR using the data from $\mathcal{T}_{in}$, and learn the shared INR and hypernetwork parameters $\boldsymbol{\theta}$ and $\boldsymbol{w}$ using $\mathcal{T}_{in}$ for both imputation and forecasting, and $\mathcal{T}_{out}$ for forecasting only. We achieve the conditioning on $\mathcal{T}_{in}$ by optimizing the codes $\boldsymbol{z}^{(j)}$ through gradient descent. The joint optimization of the codes and common parameters is challenging. In TimeFlow, it is achieved through a meta-learning approach, adapted from Dupont et al. (2022) and Zintgraf et al. (2019). The objective is to learn shared parameters so that the code $\boldsymbol{z}^{(j)}$ can be adapted in just a few gradient steps for a new series $\boldsymbol{x}^{(j)}$. For

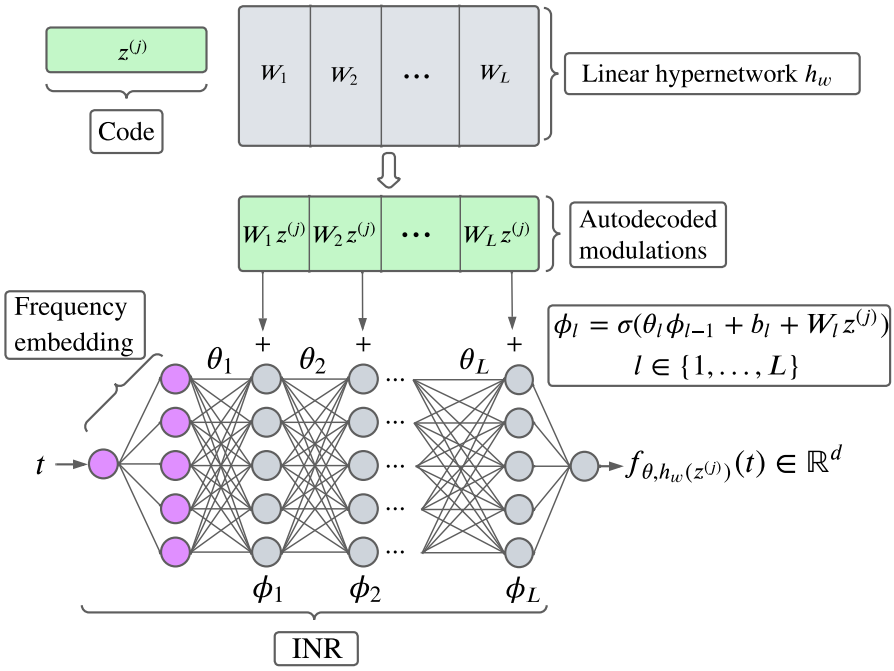


Figure 4.1: Overview of TimeFlow architecture. Forward pass to approximate the time series $x^{(j)}$. σ stands for the ReLU activation function.

training, we perform parameter optimization at two levels: the inner-loop and the outer-loop. The inner-loop adapts the code $\boldsymbol{z}^{(j)}$ to condition the network on the set $\mathcal{T}_{in}^{(j)}$, while the outer-loop updates the common parameters using $\mathcal{T}_{in}^{(j)}$ and also $\mathcal{T}_{out}^{(j)}$ for forecasting (see Appendix C.6 for more detailed intuition.). We present our training optimization in Algorithm 1. At each training epoch and for each batch of data $\mathcal{B}$ composed of time series $\boldsymbol{x}^{(j)}$ sampled from the training set, we first update individually the codes $\boldsymbol{z}^{(j)}$ in the inner loop, before updating the common parameters in the outer loop using a loss over the whole batch. We introduce a parameter λ to weight the importance of the loss over $\mathcal{T}_{out}$ w.r.t. the loss over $\mathcal{T}_{in}$ for the outer-loop. In practice, when $\mathcal{T}_{out}$ exists, i.e. for forecasting, we set $\lambda = 1$ and $\lambda = 0$ otherwise. We use an MSE loss over the observations grid $\mathcal{L}_{\mathcal{T}}(x_t, \tilde{x_t}) := \mathbb{E}_{t \sim \mathcal{T}}[(x_t - \tilde{x_t})^2]$. We denote α and η the learning rates of the inner-loop and outer-loop. Using $K = 3$ steps for training and testing is sufficient for our experiments thanks to the use of second-order meta-learning as explained in Section 4.3.4.

Algorithm 1: TimeFlow Training

while *no convergence* **do**
  Sample batch $\mathcal{B}$ of data $(x^{(j)})_{j\in\mathcal{B}}$;
  Set codes to zero $z^{(j)} \leftarrow 0, \forall j \in \mathcal{B}$;
  `// inner loop for encoding:`
  for $j \in \mathcal{B}$ *and step* $\in \{1, ..., K\}$ **do**
    $z^{(j)} \leftarrow z^{(j)} - \alpha \nabla_{z^{(j)}} \mathcal{L}_{\mathcal{T}_{in}^{(j)}}(f_{\theta,h_w(z^{(j)})}, x^{(j)})$;
  end
  `// outer loop step:`
  $[\theta, w] \leftarrow$
   $[\theta, w] - \eta \nabla_{[\theta,w]} \frac{1}{|\mathcal{B}|} \sum_{j\in\mathcal{B}} [\mathcal{L}_{\mathcal{T}_{in}^{(j)}}(f_{\theta,h_w(z^{(j)})}, x^{(j)}) + \lambda \mathcal{L}_{\mathcal{T}_{out}^{(j)}}(f_{\theta,h_w(z^{(j)})}, x^{(j)})]$;
end

4.3.3 TimeFlow inference

During the inference process, we aim to infer the time series value for each timestamp in the dense grid $\mathcal{T}^{*(j)}$ based on the partial observation grid $\mathcal{T}_{in}^{*(j)} \subset \mathcal{T}^{*(j)}$. We can encounter two scenarios: (i) One where we observe the same time window as during training ($\mathcal{T}^{*(j)} = \mathcal{T}^{(j)}$) as in the imputation setting in Section 4.4.1. (ii) One, where we are dealing with a newly observed time window ($\mathcal{T}^{*(j)} \neq \mathcal{T}^{(j)}$), as in the forecasting setting in Section 4.4.2. At inference, the parameters $\boldsymbol{\theta}$ and $\boldsymbol{w}$ are kept fixed to their final training values. We optimize the individual parameters $\boldsymbol{z^{*(j)}}$ based on the newly observed grid $\mathcal{T}_{in}^{*(j)}$ using the K inner-steps of the meta-learning algorithm as described in Algorithm 2. We are then in position to query $f_{\theta,h_{\boldsymbol{w}}(\boldsymbol{z^{*(j)}})}(t)$ for any given timestamp $t \in \mathcal{T}^{*(j)}$.

Algorithm 2: TimeFlow Inference with trained θ, w

For the j-th series $(x^{(j)})$, set code to zero $z^{*(j)} \leftarrow 0$;
for *step* $\in \{1, ..., K\}$ **do**
  $z^{*(j)} \leftarrow z^{*(j)} - \alpha \nabla_{z^{*(j)}} \mathcal{L}_{\mathcal{T}_{in}^{*(j)}}(f_{\theta,h_w(z^{*(j)})}, x_t)$
end
Query $f_{\theta,h_w(z^{*(j)})}(t)$ for any $t \in \mathcal{T}^{*(j)}$

4.3.4 Discussion on implementation choices

As indicated before, adapting the components and enhancing their synergy for the tasks of imputation and forecasting is not trivial and requires careful choices. We conducted several ablation studies to provide a comprehensive examination of key implementation choices of our framework.

Our findings can be summarized as follows.

- **Choice of INR**: An FFN with Fourier Features outperformed other popular INRs for the tasks considered in this study. Unlike SIREN (Sitzmann et al., 2020), which does not explicitly incorporate frequencies but uses sine activation functions, the Fourier features network can more effectively capture a wider range of frequencies, especially at low sampling rates. This is crucial for accurately capturing high frequencies in sparsely observed time series. Our experiments, detailed in Section C.2.2.1 and Table C.1, demonstrate this superiority across various datasets.

- **Choice of encoding / meta-learning**: TimeFlow with a set encoder for learning the compact conditioning codes $\boldsymbol{z}$ in place of the auto-decoding strategy used here, proved much less effective on complex datasets. This is further elaborated in Section C.2.2.4 and Table C.8. Additionally, replacing the 2nd-order meta-learning optimization for a 1st-order one, such as REPTILE (Nichol et al., 2018), led to unstable training, as shown in Table C.7.

- **Choice of modulations**: Complexifying the modulation by introducing scaling parameters in addition to shift parameters did not provide performance gains. Our experiments on the *Electricity* dataset, detailed in Section C.2.2.5 and Table C.9, indicate that shift-only modulation is more efficient.

For TimeFlow, across all experiments, we used a code dimension of 128, an FFN with a depth of 5 and a width of 256, and 64 Fourier features. We used 3 inner steps and a learning rate of 0.01 for the inner-loop, and a learning rate of 5×10^{-4} for the outer-loop. We performed a comprehensive analysis to understand notably the **influence of the z dimension**: a latent code dimension of 128 was suitable for our tasks; this is supported by results in Section C.2.2.2 and Table C.2 - and the **influence of the number of inner steps**: using 3 inner steps for training and inference struck a favorable balance between reconstruction capabilities and computational efficiency, as detailed in Section C.2.2.3.

4.4 Experiments

We conducted a comprehensive evaluation of our TimeFlow framework across three different tasks, comparing its performance to state-of-the-art continuous and discrete baseline methods. In Section 4.4.1, we assess TimeFlow's capabilities to impute sparsely observed time series under various sampling rates. Section 4.4.2 focuses on long-term forecasting, where we evaluate TimeFlow over standard long-term forecasting horizons. In Section 4.4.3, we tackle a challenging task forecasting with incomplete look-back windows, thus combining the challenges of imputation and forecasting. The code for the experiments is available at this link

Datasets. We tested our framework on three extensive datasets where a single phenomenon is measured at multiple locations over time, namely *Electricity*, *Traffic* and *Solar*. The *Electricity* dataset comprises hourly electricity load curves of 321 customers in Portugal, spanning the years 2012 to 2014. The *Traffic* dataset is composed of hourly road occupancy rates from 862 locations in San Francisco during 2015 and 2016. Lastly, the *Solar* dataset contains measurements of solar power production from 137 photovoltaic plants in Alabama, recorded at 10-minute intervals in 2006. Additionally, we have created an hourly version, *SolarH*, for the sake of consistency in the forecasting section. These datasets exhibit diversity in various characteristics: • They exhibit diverse temporal frequencies, including daily and weekly seasonality observed in the *Traffic* and *Electricity* datasets, while the *Solar* dataset possesses only daily frequency. • There is individual variability across data samples and more pronounced trends in the *Electricity* dataset compared to the *Traffic* and *Solar* datasets.

Datasets source. *Electricity* dataset is available here, *Traffic* dataset here and *Solar* dataset here. Table 4.1 provides a concise overview of the main information about the datasets used for forecasting and imputation tasks.

Table 4.1: Summary of datasets information.

Dataset name	Number of samples	Number of time steps	Sampling frequency	Location	Years
Electricity	321	26 304	hourly	Portugal	2012 – 2014
Traffic	862	17 544	hourly	San Francisco bay	2015 – 2016
Solar	137	52 560	10 minutes	Alabama	2006
SolarH	137	8 760	hourly	Alabama	2006

TimeFlow relative improvement score. In the following experiments, the relative improvement score of TimeFlow is provided. Its purpose is to quantify the average incremental benefit of TimeFlow over the method under consideration. It is computed as follows:

$$\text{TimeFlow improvement} = \frac{1}{L}\sum_{l=1}^{L}\frac{s_l(\text{baseline}) - s_l(\text{TimeFlow})}{s_l(\text{baseline})}$$

• s stands for the Mean Absolute Error score of the considered method against the ground truth at line l. • L stands for the number of line in the table.

4.4.1 Imputation

4.4.1.1 Imputation for known time series.

We consider the classical imputation setting where n time series are partially observed over a given time window. Using our approach, we can predict for each time series the value at any timestamp t in that time window based on partial observations.

Setting. For a time series $\boldsymbol{x^{(j)}}$, we denote the set of observed points as $\mathcal{T}_{in}^{(j)}$ and the ground truth set of points as $\mathcal{T}^{(j)}$. The observed time grids may be irregularly spaced and may differ across the different time series ($\mathcal{T}_{in}^{(j_1)} \neq \mathcal{T}_{in}^{(j_2)}, \forall j_1 \neq j_2$). The model is trained for each $x^{(j)}$ following Algorithm 1. Then, we aim to infer for any unobserved $t \in \mathcal{T}^{(j)}$ the missing value $x_t^{(j)}$ conditioned on $\mathcal{T}_{in}^{(j)}$ according to Algorithm 2.

For this imputation task, the TimeFlow training and inference procedures are detailed in Appendix A.2 and illustrated in Figure 4.2. For comparison with the SOTA imputation baselines, we assume that the ground truth time grid is the same for each sample. The subsampling rate τ is define as the rate of observed values.

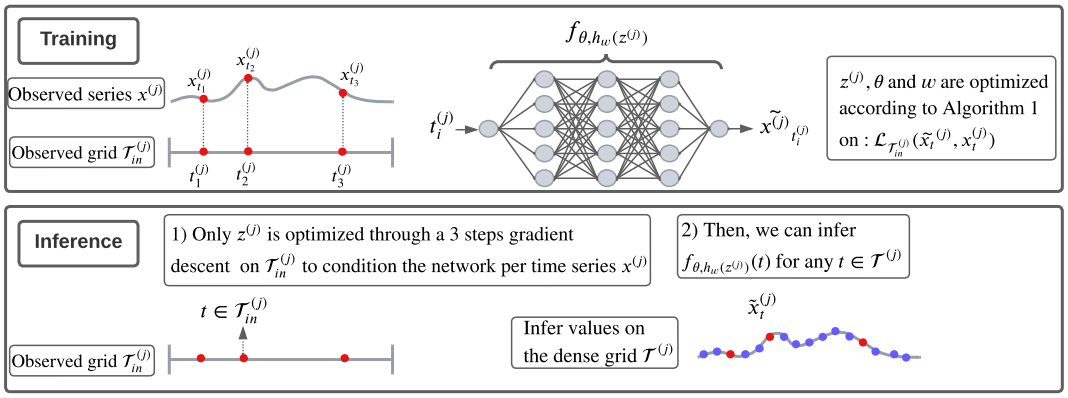


Figure 4.2: Training and inference procedures of TimeFlow for imputation. (i) During training, for each time series $\boldsymbol{x^{(j)}}$, our observations (red dots ●) are restricted to the sparsely sampled grid, denoted as $\mathcal{T}_{in}^{(j)}$. (ii) During inference, our objective is to infer the values over the dense grids $\mathcal{T}^{(j)}$, on the unobserved data points (such as the blue dots ● on the figure).

Baselines. We compare TimeFlow with various baselines, including discrete imputation methods, such as CSDI (Tashiro et al., 2021), SAITS (Du et al., 2023), BRITS (Cao et al., 2018), and TIDER (Liu et al., 2023), and continuous ones, such as Neural Process (NP, Garnelo et al., 2018), mTAN (Shukla and Marlin, 2021), and DeepTime with slight adjustments (Woo et al., 2022) (details cf. Appendix C.3.2).

Table 4.2: Mean MAE imputation results on the missing grid only. Each time series is divided into 5 time windows onto which imputation is performed, and the performances are averaged over the 5 windows. In the table, τ stands for the subsampling rate, i.e. the proportion of observed points considered for each time window. Bold results are best, underlined results are second best. TimeFlow improvement represents the overall percentage improvement achieved by TimeFlow compared to the specific method being considered.

		Continuous methods				Discrete methods			
	τ	TimeFlow	DeepTime	mTAN	Neural Process	CSDI	SAITS	BRITS	TIDER
Electricity	0.05	**0.324 ± 0.013**	0.379 ± 0.037	0.575 ± 0.039	0.357 ± 0.015	0.462 ± 0.021	0.384 ± 0.019	0.329 ± 0.015	0.427 ± 0.010
	0.10	**0.250 ± 0.010**	0.333 ± 0.034	0.412 ± 0.047	0.417 ± 0.057	0.398 ± 0.072	0.308 ± 0.011	0.287 ± 0.015	0.399 ± 0.009
	0.20	**0.225 ± 0.008**	0.244 ± 0.013	0.342 ± 0.014	0.320 ± 0.017	0.341 ± 0.068	0.261 ± 0.008	0.245 ± 0.011	0.391 ± 0.010
	0.30	**0.212 ± 0.007**	0.240 ± 0.014	0.335 ± 0.015	0.300 ± 0.022	0.277 ± 0.059	0.236 ± 0.008	0.221 ± 0.008	0.384 ± 0.009
	0.50	0.194 ± 0.007	0.227 ± 0.012	0.340 ± 0.022	0.297 ± 0.016	**0.168 ± 0.003**	0.209 ± 0.008	0.193 ± 0.008	0.386 ± 0.009
Solar	0.05	**0.095 ± 0.015**	0.190 ± 0.020	0.241 ± 0.102	0.115 ± 0.015	0.374 ± 0.033	0.142 ± 0.016	0.165 ± 0.014	0.291 ± 0.009
	0.10	**0.083 ± 0.015**	0.159 ± 0.013	0.251 ± 0.081	0.114 ± 0.014	0.375 ± 0.038	0.124 ± 0.018	0.132 ± 0.015	0.276 ± 0.010
	0.20	**0.072 ± 0.015**	0.149 ± 0.020	0.314 ± 0.035	0.109 ± 0.016	0.217 ± 0.023	0.108 ± 0.014	0.109 ± 0.012	0.270 ± 0.010
	0.30	**0.061 ± 0.012**	0.135 ± 0.014	0.338 ± 0.05	0.108 ± 0.016	0.156 ± 0.002	0.100 ± 0.015	0.098 ± 0.012	0.266 ± 0.010
	0.50	**0.054 ± 0.013**	0.098 ± 0.013	0.315 ± 0.080	0.107 ± 0.015	0.079 ± 0.011	0.094 ± 0.013	0.088 ± 0.013	0.262 ± 0.009
Traffic	0.05	0.283 ± 0.016	**0.246 ± 0.010**	0.406 ± 0.074	0.318 ± 0.014	0.337 ± 0.045	0.293 ± 0.007	0.261 ± 0.010	0.363 ± 0.007
	0.10	**0.211 ± 0.012**	0.214 ± 0.007	0.319 ± 0.025	0.288 ± 0.018	0.288 ± 0.017	0.237 ± 0.006	0.245 ± 0.009	0.362 ± 0.006
	0.20	**0.168 ± 0.006**	0.216 ± 0.006	0.270 ± 0.012	0.271 ± 0.011	0.269 ± 0.017	0.197 ± 0.005	0.224 ± 0.008	0.361 ± 0.006
	0.30	**0.151 ± 0.007**	0.172 ± 0.008	0.251 ± 0.006	0.259 ± 0.012	0.240 ± 0.037	0.180 ± 0.006	0.197 ± 0.007	0.355 ± 0.006
	0.50	**0.139 ± 0.007**	0.171 ± 0.005	0.278 ± 0.040	0.240 ± 0.021	0.144 ± 0.022	0.160 ± 0.008	0.161 ± 0.060	0.354 ± 0.007
TimeFlow improvement		/	24.14 %	50.53 %	31.61 %	36.12 %	20.33 %	18.90 %	53.40 %

See Appendix C.3.1.1 for the baseline training procedure and hyperparameter selection. For each dataset, we divide the series into five independent time windows (consisting of 2000 timestamps for *Electricity* and *Traffic*, and 10,000 timestamps for *Solar*), perform imputation on each time window and average the performance to obtain robust results. We evaluate the quality of the models for different subsampling rates, from the easiest $\tau = 0.5$ to the most difficult $\tau = 0.05$. All the scores presented in the experiments are reported as Mean Absolute Error (MAE).

Results. We show in Table 4.2 that TimeFlow outperforms both discrete and continuous models across almost all τ's for the given datasets. The relative improvements of TimeFlow over the baselines are significant, ranging from 15% to 50%. Especially for the lowest sampling rate $\tau = 0.05$, TimeFlow outperforms all discrete baselines, demonstrating the advantages of continuous modeling. Additionally, it achieves lower imputation errors compared to continuous models in all but one cases. Qualitatively, we see on example series in Figure 4.3 that our model shows significant imputation capabilities, with a subsampling rate at $\tau = 0.1$ on the *Electricity* dataset. It captures well different frequencies and amplitudes in a challenging case (sample 35), although it underestimates the amplitude of some peaks. In a more challenging scenario (sample 25), where the series exhibit additional trend changes and frequency variations within the data, TimeFlow correctly imputes most timestamps, outperforming BRITS, which is the best-performing method for the

Electricity dataset.

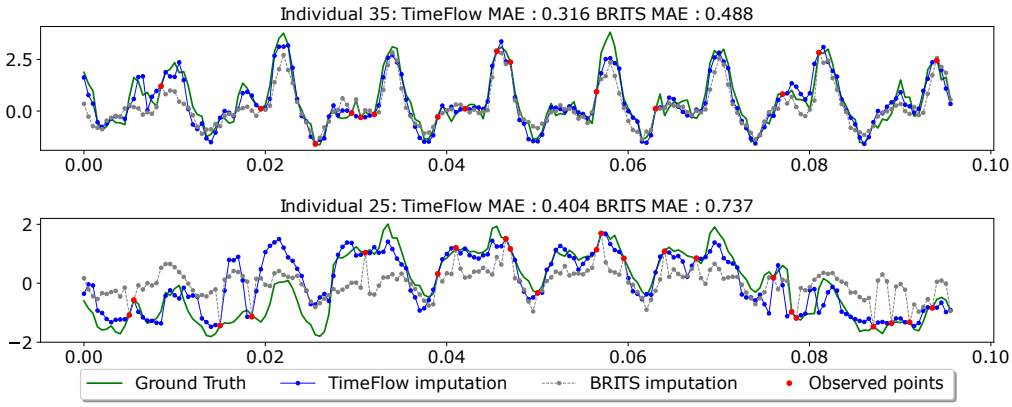


Figure 4.3: *Electricity dataset.* TimeFlow imputation (blue line) and BRITS imputation (gray line) with 10% of known point (red points) on the eight first days of samples 35 (top) and 25 (bottom).

Imputation against non deep learning methods. In addition to the deep learning methods presented in Table 4.2, we evalute TimeFlow against two classic machine learning baselines, K-Nearest Neighbours (KNN) and linear interpolation, which are valuable for getting an idea of the complexity of the problem.

Table 4.3: Mean MAE imputation results on the missing grid only over five different time window. τ stands for the subsampling rate. Bold results are best, underline results are second best.

	τ	TimeFlow	Linear interpolation	KNN (k=3)
Electricity	0.05	**0.324 ± 0.013**	0.828 ± 0.045	0.531 ± 0.033
	0.10	**0.250 ± 0.010**	0.716 ± 0.039	0.416 ± 0.020
	0.20	**0.225 ± 0.008**	0.518 ± 0.029	0.363 ± 0.019
	0.30	**0.212 ± 0.007**	0.396 ± 0.022	0.342 ± 0.017
	0.50	**0.194 ± 0.007**	0.275 ± 0.015	0.323 ± 0.016
Solar	0.05	**0.095 ± 0.015**	0.339 ± 0.031	0.151 ± 0.017
	0.10	**0.083 ± 0.015**	0.170 ± 0.014	0.128 ± 0.017
	0.20	**0.072 ± 0.015**	0.088 ± 0.010	0.110 ± 0.016
	0.30	**0.061 ± 0.012**	0.063 ± 0.009	0.103 ± 0.017
	0.50	0.054 ± 0.013	**0.044 ± 0.008**	0.096 ± 0.016
Traffic	0.05	**0.283 ± 0.016**	0.813 ± 0.027	0.387 ± 0.014
	0.10	**0.211 ± 0.012**	0.701 ± 0.026	0.293 ± 0.012
	0.20	**0.168 ± 0.006**	0.508 ± 0.022	0.249 ± 0.010
	0.30	**0.151 ± 0.007**	0.387 ± 0.018	0.228 ± 0.009
	0.50	**0.139 ± 0.007**	0.263 ± 0.013	0.204 ± 0.009
TimeFlow improvement		/	49.06 %	35.95 %

Results. KNN imputation uses information from other individuals and gives satisfactory results at all sampling rates. On the other hand, the purely univariate approach of linear interpolation struggles at low sampling rates but performs well at high sampling rates. TimeFlow significantly outperforms both baselines by a large margin.

4.4.1.2 Imputation on previously unseen time series.

In more practical scenarios, such as cases involving the installation of new sensors, we often encounter new time series originating from the same underlying phenomenon. In such instances, it becomes crucial to make inferences for these previously unseen time series. Thanks to efficient adaptation in latent space, our model can easily be applied to these new time series, contrasting with SOTA methods like SAITS and BRITS, which require full model retraining on the whole set of time series.

Setting. In this section we analyze in details the imputations results for previously unseen time series described in Section 4.4.1. Specifically, TimeFlow is trained on a given set of time series within a defined time window and then used for inference on new time series. We train TimeFlow on 50 % of the samples and consider the remaining 50 % as the new time series. We compare in Table 4.4 observed grid fit scores and missing grid inference scores for time series known at training and time series unknown at training.

Table 4.4: TimeFlow MAE imputation errors results for imputation previsouly unseen time series.

		Known time series		New time series	
	τ	Fit	Inference	Fit	Inference
Electricity	0.05	0.060 ± 0.010	0.402 ± 0.021	0.142 ± 0.083	0.413 ± 0.026
	0.10	0.046 ± 0.006	0.302 ± 0.010	0.144 ± 0.098	0.309 ± 0.016
	0.20	0.067 ± 0.015	0.285 ± 0.014	0.154 ± 0.089	0.291 ± 0.022
	0.30	0.093 ± 0.022	0.266 ± 0.010	0.163 ± 0.073	0.271 ± 0.017
	0.50	0.108 ± 0.012	0.236 ± 0.010	0.167 ± 0.061	0.245 ± 0.017
Solar	0.05	0.014 ± 0.002	0.104 ± 0.015	0.050 ± 0.037	0.109 ± 0.016
	0.10	0.017 ± 0.002	0.092 ± 0.015	0.052 ± 0.036	0.099 ± 0.017
	0.20	0.028 ± 0.008	0.078 ± 0.014	0.058 ± 0.031	0.089 ± 0.017
	0.30	0.038 ± 0.009	0.072 ± 0.013	0.063 ± 0.028	0.084 ± 0.018
	0.50	0.045 ± 0.011	0.066 ± 0.013	0.067 ± 0.025	0.080 ± 0.019
Traffic	0.05	0.044 ± 0.003	0.291 ± 0.013	094 ± 0.051	0.291 ± 0.012
	0.10	0.033 ± 0.001	0.209 ± 0.010	0.093 ± 0.060	0.216 ± 0.012
	0.20	0.037 ± 0.006	0.175 ± 0.008	0.095 ± 0.058	0.186 ± 0.013
	0.30	0.048 ± 0.005	0.164 ± 0.006	0.098 ± 0.051	0.175 ± 0.013
	0.50	0.068 ± 0.004	0.159 ± 0.007	0.110 ± 0.042	0.169 ± 0.012

Results. The results presented in Table 4.4 indicate that the inference MAE for missing grids shows consistency between known and new samples, regardless of the

data or sampling rate. However, it is worth noting that there is a slight drop in performance compared to the results in table Table 4.2. This decrease is because in Table 4.4, the shared architecture is trained on only half the samples, affecting its overall performance.

4.4.2 Forecasting

4.4.2.1 Forecasting for known time series

In this section, we are interested in the conventional long-term forecasting scenario. It consists in predicting the phenomenon in a specific future period, the horizon, based on the history of a limited past period, the look-back window. The forecaster is trained on a set of n observed time series for a given time window (train period) and tested on new distinct time windows.

Setting. For a given time series $\boldsymbol{x}^{(j)}$, $\mathcal{T}_{in}^{(j)}$ denotes the look-back window and $\mathcal{T}_{out}^{(j)}$ the horizon of H points. During training, at each epoch, we train $f_{\boldsymbol{\theta},h_{\boldsymbol{w}}(\boldsymbol{z}^{(j)})}$ following Algorithm 1 with randomly drawn pairs of look-back window and horizon $(\mathcal{T}_{in}^{(j)} \cup \mathcal{T}_{out}^{(j)})_{j\in\mathcal{B}}$ within the observed train period. Then, for a distinct new time window $\mathcal{T}^{*(j)}$, given a look-back window $\mathcal{T}_{in}^{*(j)}$ we forecast future values any $t \in \mathcal{T}^{*(j)}$, the horizon interval, following Algorithm 2. We illustrate the training and inference of TimeFlow for the forecasting task in Figure 4.4. For further insight into the training window and inference periods, as well as additional experiments conducted under different inference scenarios, see Appendix C.4.1.

Baselines. To evaluate the quality of our model in long-term forecasting, we compare it to the discrete baselines PatchTST (Nie et al., 2022), DLinear (Zeng et al., 2022), AutoFormer (Wu et al., 2021), and Informer (Zhou et al., 2021). We also include continuous baselines DeepTime and Neural Process (NP). See Appendix C.4.3.1 for the baseline training procedure and hyperparameter selection. In Table 4.5, we present the forecasting results for standard horizons in long-term forecasting: $H \in \{96, 192, 336, 720\}$. The look-back window length is fixed to 512.

Results. The results in Table 4.5 show that our approach ranks in the top two across all datasets and horizons and is the overall best continuous method. TimeFlow's performance is comparable to the current SOTA model PatchTST, with only 2% relative difference. Moreover, TimeFlow shows consistent results across the three datasets, whereas the other best discrete and continuous baselines, i.e. PatchTST and DeepTime, performance drops for some datasets. We also note that, despite the great performance of the SOTA PatchTST, other transformer-based baselines (discrete methods in Table 4.5) perform poorly. We provide a detailed insight on

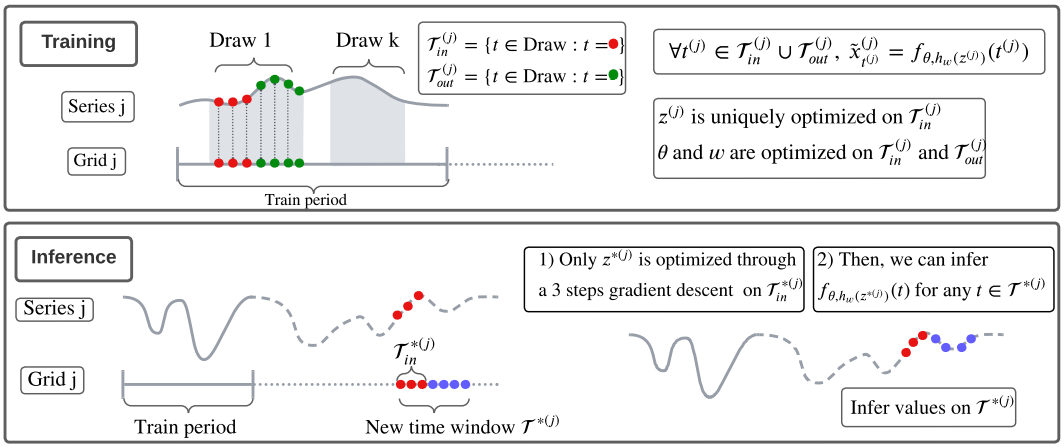


Figure 4.4: Training and inference procedure of TimeFlow for forecasting. (i) During training (top-figure), for each time series $x^{(j)}$, we observe some look-back window/horizon drawing pairs in the trained period. TimeFlow is trained with Algorithm 1 to predict all observed timestamps in this drawing pairs while being conditioned by the observed look-back window. (ii) Once TimeFlow is optimized, the objective during inference (bottom-figure) is to infer the horizon over new time windows (blue dots •) while being conditioned by the newly observed look-back window (red dots •).

these results in Appendix C.4.1. Overall, although this evaluation setting favors discrete methods because the time series are observed at evenly distributed time steps, TimeFlow consistently performs as well as PatchTST and outperforms all the other methods, whether discrete or continuous. It is the first time that a continuous model has achieved the same level of performance as discrete methods within their specific setting.

4.4.2.2 Forecasting on previously unseen time series

This section discusses how TimeFlow adapts to unseen time series, which is critical in forecasting. Indeed, in many real-world applications, forecasters are trained on a limited subset of available samples and applied to a wider range of samples during inference. Informer, AutoFormer, or DLinear original architectures directly model the relationships between time series (channel-dependence), limiting their adaptability to new samples. In contrast, TimeFlow takes a different approach by considering the observed series at different locations as distinct samples, similar to PatchTST, Neural Process, and DeepTime. This independence allows TimeFlow to effectively generalize to previously unseen time series of the same phenomenon.

Setting. In this setting, we propose to evaluate how TimeFlow performs on previously unseen time series. We compare it to the best forecaster, PatchTST. We

Table 4.5: Mean MAE forecast results averaged over different time windows. Each time, the model is trained on one time window and tested on the others (there are 2 windows for *SolarH* and 5 for *Electricity* and *Traffic*). H stands for the horizon. Bold results are best, and underlined results are second best. TimeFlow improvement represents the overall percentage improvement achieved by TimeFlow compared to the specific method being considered.

		Continuous methods			Discrete methods			
	H	TimeFlow	DeepTime	Neural Process	Patch-TST	DLinear	AutoFormer	Informer
Electricity	96	0.228 ± 0.028	0.244 ± 0.026	0.392 ± 0.045	**0.221 ± 0.023**	0.241 ± 0.030	0.546 ± 0.277	0.603 ± 0.255
	192	0.238 ± 0.020	0.252 ± 0.019	0.401 ± 0.046	**0.229 ± 0.020**	0.252 ± 0.025	0.500 ± 0.190	0.690 ± 0.291
	336	0.270 ± 0.031	0.284 ± 0.034	0.434 ± 0.076	**0.251 ± 0.027**	0.288 ± 0.038	0.523 ± 0.188	0.736 ± 0.271
	720	0.316 ± 0.055	0.359 ± 0.051	0.607 ± 0.150	**0.297 ± 0.039**	0.365 ± 0.059	0.631 ± 0.237	0.746 ± 0.265
SolarH	96	**0.190 ± 0.013**	**0.190 ± 0.020**	0.221 ± 0.048	0.262 ± 0.070	0.208 ± 0.014	0.245 ± 0.045	0.248 ± 0.022
	192	**0.202 ± 0.020**	0.204 ± 0.028	0.244 ± 0.048	0.253 ± 0.051	0.217 ± 0.022	0.333 ± 0.107	0.270 ± 0.031
	336	0.209 ± 0.017	**0.199 ± 0.026**	0.240 ± 0.006	0.259 ± 0.071	0.217 ± 0.026	0.334 ± 0.079	0.328 ± 0.048
	720	**0.218 ± 0.041**	0.229 ± 0.024	0.403 ± 0.147	0.267 ± 0.064	0.249 ± 0.034	0.351 ± 0.055	0.337 ± 0.037
Traffic	96	0.217 ± 0.032	0.228 ± 0.032	0.283 ± 0.027	**0.203 ± 0.037**	0.228 ± 0.033	0.319 ± 0.059	0.372 ± 0.078
	192	0.212 ± 0.028	0.220 ± 0.022	0.292 ± 0.024	**0.197 ± 0.030**	0.221 ± 0.023	0.368 ± 0.057	0.511 ± 0.247
	336	0.238 ± 0.034	0.245 ± 0.038	0.305 ± 0.039	**0.222 ± 0.039**	0.250 ± 0.040	0.434 ± 0.061	0.561 ± 0.263
	720	0.279 ± 0.050	0.290 ± 0.052	0.339 ± 0.038	**0.269 ± 0.057**	0.300 ± 0.057	0.462 ± 0.062	0.638 ± 0.067
TimeFlow improvement		/	3.74 %	29.06 %	3.23 %	6.92 %	42.09 %	48.57 %

train TimeFlow and PatchTST on 50 % of the samples and consider the remaining 50 % as the new time series. The training procedure is the same as described in Figure 4.4. In Figure 4.5, we present the results of TimeFlow and PatchTST for both known and new samples (for periods outside the training window).

Results. The results in Figure 4.5 highlight two key observations. First, both approaches show robust adaptability to new samples, as evidenced by the minimal difference in mean absolute error between known and new samples at inference. Second, TimeFlow and PatchTST exhibit comparable performance in this context, with negligible differences across horizons and datasets.

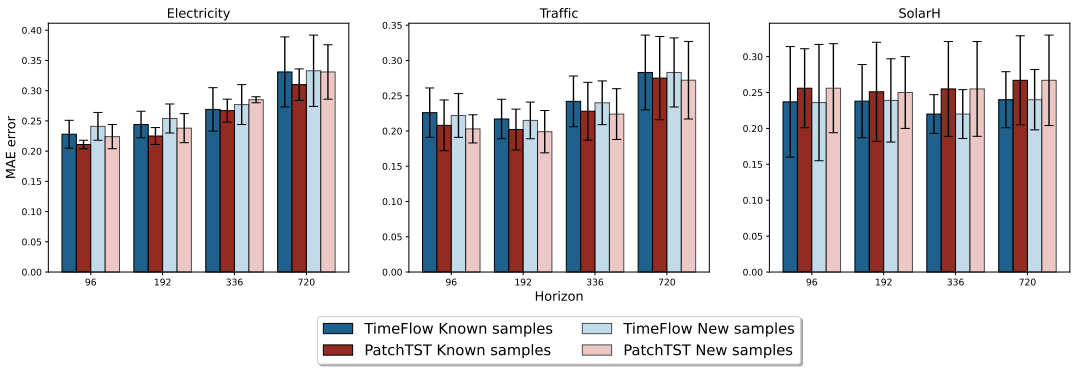


Figure 4.5: Mean MAE forecasting task results over different horizons in the context of generalization to new time series. Comparison of TimeFlow and PatchTST performances on the *Electricity*, *Traffic* and *SolarH* datasets.

4.4.3 Challenging task: Forecast while imputing incomplete look-back windows

In real-world scenarios, it is common to encounter missing or irregularly sampled series when making predictions on new time windows (Cinar et al., 2018; Tang et al., 2020). Continuous methods can handle these cases, as they are designed to accommodate irregular sampling within the look-back window. In this section, we formulate a task to simulate these real-world scenarios. It's worth noting that this task is often encountered in practice but is rarely considered in the DL literature.

Setting and baselines. This scenario is similar to the forecast setting in Section 4.4.2 and illustrated in Figure 4.4. The difference is that during inference, the look-back window is subsampled at a rate τ smaller than the one used for the training phase. This simulates a situation with missing observations in the look back window. Consequently, two distinct tasks emerge during the inference phase: imputing missing points within the sparsely observed look-back window, and forecasting over the horizon with this degraded context. In Table 4.6, we compare to the two other continuous baselines, DeepTime and NP on *Electricity* and *Traffic* for different τ's and horizons.

Table 4.6: MAE results for forecasting with missing values in the look-back window. τ stands for the percentage of observed values in the look-back window. Best results are in bold. TimeFlow improvement represents the overall percentage improvement (for each task) achieved by TimeFlow compared to the specific method being considered.

			TimeFlow		DeepTime		Neural Process	
	H	τ	Imputation error	Forecast error	Imputation error	Forecast error	Imputation error	Forecast error
Electricity	96	0.5	**0.151 ± 0.003**	**0.239 ± 0.013**	0.209 ± 0.004	0.270 ± 0.019	0.460 ± 0.048	0.486 ± 0.078
		0.2	**0.208 ± 0.006**	**0.260 ± 0.015**	0.249 ± 0.006	0.296 ± 0.023	0.644 ± 0.079	0.650 ± 0.095
		0.1	**0.272 ± 0.006**	**0.295 ± 0.016**	0.284 ± 0.007	0.324 ± 0.026	0.740 ± 0.083	0.737 ± 0.106
	192	0.5	**0.149 ± 0.004**	**0.235 ± 0.011**	0.204 ± 0.004	0.265 ± 0.018	0.461 ± 0.045	0.498 ± 0.070
		0.2	**0.209 ± 0.006**	**0.257 ± 0.013**	0.244 ± 0.007	0.290 ± 0.023	0.601 ± 0.075	0.626 ± 0.101
		0.1	**0.274 ± 0.010**	**0.289 ± 0.016**	0.282 ± 0.007	0.315 ± 0.025	0.461 ± 0.045	0.724 ± 0.090
Traffic	96	0.5	**0.180 ± 0.016**	**0.219 ± 0.026**	0.272 ± 0.028	0.243 ± 0.030	0.436 ± 0.025	0.444 ± 0.047
		0.2	**0.239 ± 0.019**	**0.243 ± 0.027**	0.335 ± 0.026	0.293 ± 0.027	0.596 ± 0.049	0.597 ± 0.075
		0.1	**0.312 ± 0.020**	**0.290 ± 0.027**	0.385 ± 0.025	0.344 ± 0.027	0.734 ± 0.102	0.731 ± 0.132
	192	0.5	**0.176 ± 0.014**	**0.217 ± 0.017**	0.241 ± 0.027	0.234 ± 0.021	0.477 ± 0.042	0.476 ± 0.043
		0.2	**0.233 ± 0.017**	**0.236 ± 0.021**	0.286 ± 0.027	0.276 ± 0.020	0.685 ± 0.109	0.678 ± 0.108
		0.1	**0.304 ± 0.019**	**0.277 ± 0.021**	0.331 ± 0.025	0.324 ± 0.021	0.888 ± 0.178	0.877 ± 0.174
TimeFlow improvement			/	/	18.97 %	11.87 %	61.88 %	58.41 %

Results. In Table 4.6, the results show that TimeFlow consistently outperforms other methods in imputation and forecasting for every scenarios. When comparing with the complete look-back windows observations scenario from Table 4.5, one

observes that at a 0.5 sampling rate, TimeFlow presents only a slight reduction in performance, whereas other baseline methods experience more significant drops. For instance, when we compare forecast results between a complete window and a $\tau = 0.5$ subsampled window for *Electricity* with a forecasting horizon of $H = 96$, TimeFlow's error increases by a mere 4.6% (from 0.228 to 0.239). In contrast, DeepTime's error grows by over 10% (from 0.244 to 0.270), and NP experiences a rise of around 25% (from 0.392 to 0.486).

For lower sampling rates, TimeFlow still delivers correct predictions. Qualitatively, we see on the series example in Figure 4.6 that despite observing only 10% of the look-back window, the model can correctly infer both the complete look-back window and the horizon. Both quantitative and qualitative results show the robustness and efficiency of TimeFlow on this particularly challenging setting.

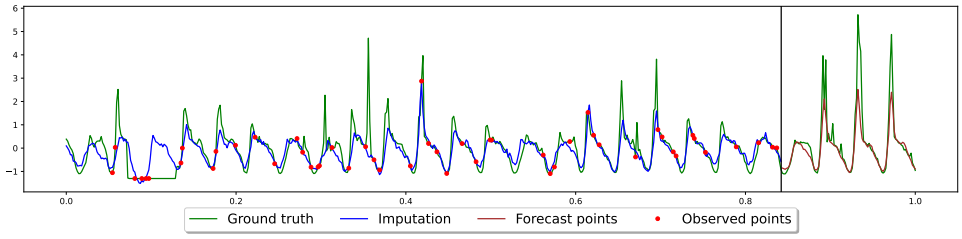


Figure 4.6: *Traffic dataset, sample 95.* In this figure, TimeFlow simultaneously imputes and forecasts at horizon 96 with a 10% partially observed look-back window of length 512.

4.4.4 Quantify uncertainty with TimeFlow: experiment on block imputation

In this experimental section, we investigate TimeFlow's ability to be trained using the pinball loss function. This quantile loss, coupled with TimeFlow, enables the continuous modeling of uncertainty over time, offering a more robust and comprehensive understanding of the temporal dynamics and variability within the time series.

Motivations. Quantifying uncertainty is paramount for effective decision-making in time series modeling. Time series data are often subject to various forms of noise and anomalies. As a result, predictions based solely on point estimates can be misleading, leading to suboptimal or even detrimental decisions. Therefore, incorporating uncertainty quantification into the forecasting and imputation models allows one to better understand the range of possible outcomes, thereby making more informed and robust decisions.

TimeFlow addresses this critical need by being able to train with loss functions specifically designed to quantify uncertainty. A prominent approach to uncertainty

estimation is quantile regression. Quantile regression provides a way to predict not only the central tendency of the time series, but also its distribution, providing insight into the probable range of values.

Methodology. To implement this, we utilize the pinball loss. The pinball loss, also known as quantile loss, is designed to penalize overestimations and underestimations asymmetrically, thereby capturing the inherent uncertainty in the predictions. The pinball loss for a given quantile q is defined as:

$$\begin{cases} q(x - \hat{x}) & \text{if } x > \hat{x} \\ (1 - q)(\hat{x} - x) & \text{if } x \leq \hat{x} \end{cases} \tag{4.1}$$

Where x is the ground truth value and $\hat{x}$ is the predicted value. The parameter q (between 0 and 1) indicates the specific quantile being estimated. For example, $(q = 0.5)$ corresponds to the median, $(q = 0.9)$ corresponds to the 90th percentile, and so on. By minimizing this loss function during training (Algorithm 1), TimeFlow learns to effectively predict different quantiles of the target distribution.

4.4.4.1 Experiments

We propose to qualitatively evaluate the uncertainty estimation for the block imputation task.

Setting. We use the *Electricity* dataset and train TimeFlow with pinball losses to fit time series spanning two weeks with hourly time steps (T=336). During inference, for time series not seen during training, we fill in a missing block of two days (48 points). We estimate the $5\%, 25\%, 75\%$, and 95% quantiles. Figure 4.7 illustrates the uncertainty estimation results for six samples.

Results. As shown in Figure 4.7, TimeFlow effectively produces fairly narrow uncertainty bands, most of which encompass the ground truth. For more challenging imputation cases, the uncertainty bands tend to be wider, indicating the increased difficulty of the task. This highlights TimeFlow's capability to adaptively quantify uncertainty based on the complexity of the block imputation scenario.

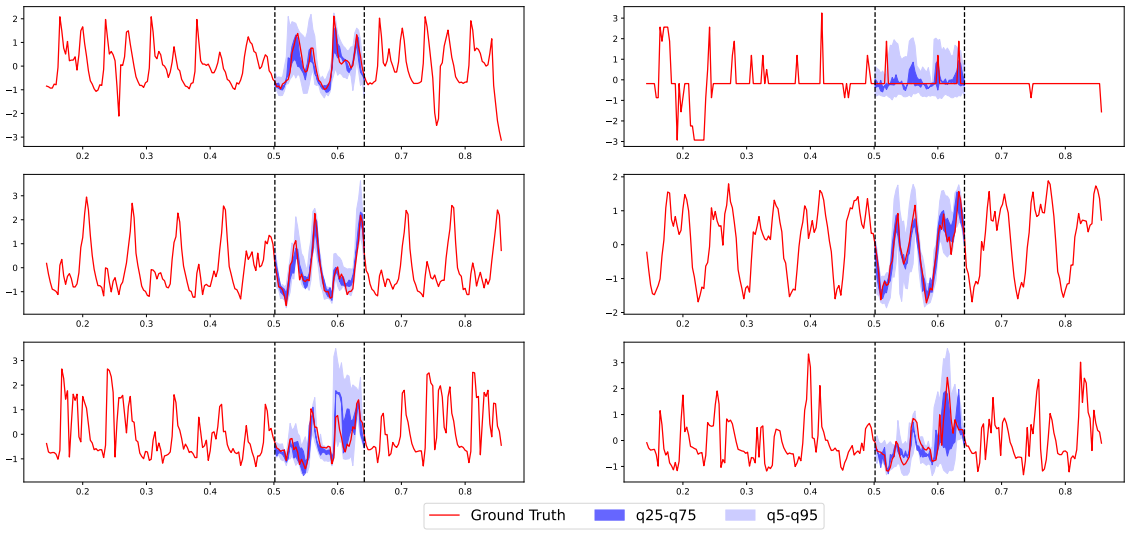


Figure 4.7: Quantifying uncertainty in block imputation of two missing days in the *Electricity* dataset.

4.5 Limitations

While TimeFlow shows promising performance across various tasks and settings, it is important to recognize some limitations.

General remarks. First, due to its auto-decoding process, TimeFlow tends to be significantly slower at inference time compared to other baselines by one to two orders of magnitude (Table C.17). Then, it should be noted that effective training of TimeFlow requires a relatively large number of samples (typically ≥ 100) to allow the model to distinguish between individual patterns and shared information accurately.

Exploring drastic distribution shifts with TimeFlow. Although TimeFlow effectively handles sets of homogeneous time series, additional mechanisms are required to handle heterogeneous time series with different frequencies effectively. The per-context shift modulation mechanism does not allow TimeFlow to fit time series with drastically different structures.

We performed an additional experiment where we explored its zero-shot capabilities by training TimeFlow to forecast on a given dataset and then applying the trained model to a new dataset without retraining. We present the experiment results in Table 4.7.

Table 4.7: Zero-shot forecasting experiments. Comparison of forecasting performances of TimeFlow supervised, TimeFlow trained on *Electricity* and TimeFlow trained on *SolarH*.

	H	Supervised TimeFlow	TimeFlow trained on Electricity	TimeFlow trained on SolarH
Electricity	96	0.225	/	0.576
	192	0.243	/	0.681
	336	0.275	/	0.852
Traffic	96	0.193	0.213	0.523
	192	0.207	0.241	0.526
	336	0.228	0.246	0.567
SolarH	96	0.191	0.193	/
	192	0.181	0.186	/
	336	0.186	0.185	/

As shown in Table 4.7, the zero-shot performance of TimeFlow, when pre-trained on datasets with similar frequencies, is quite promising compared to the fully supervised approach. For example, TimeFlow trained on the *Electricity* dataset performs well on the *Traffic* dataset. However, when TimeFlow trained on the *SolarH* dataset is applied in zero-shot to the *Electricity* and *Traffic* datasets, the results degrade significantly compared to the fully supervised model. Future research could focus on training TimeFlow jointly on a diverse range of datasets to evaluate its zero-shot generalization capabilities.

Incorporating context variables. So far, our experiments with TimeFlow have been limited to univariate time series datasets of the form $\{\boldsymbol{x}^{(j)}\}_{j=1}^{n}$, where $\boldsymbol{x}^{(j)} \in \mathbb{R}^T$. However, context variables are crucial for accurate time series modelisation in industrial applications. To make TimeFlow suitable for real-world problems, we need to develop a model variant that incorporates these additional context variables and allows it to condition its behavior on temporal and static context factors.

4.6 Conclusion

We have introduced a unified framework for continuous time series modeling leveraging conditional INR and meta-learning. Our experiments have demonstrated superior performance compared to other continuous methods, and better or comparable results to SOTA discrete methods. One of the standout features of our framework is its inherent continuity and the ability to modulate the INR parameters. This unique flexibility lets TimeFlow effectively tackle a wide array of challenges, including forecasting in the presence of missing values, accommodating irregular time steps, and extending the trained model's applicability to previously unseen time

series and new time windows. Our empirical results have shown TimeFlow's effectiveness in handling homogeneous multivariate time series. As a logical next step, extending TimeFlow's capabilities to address heterogeneous multivariate phenomena represents a promising direction for future research.

Chapter 5

Exploring TimeFlow Representations

In the previous chapter, we demonstrated that TimeFlow can capture neural representations from unaligned and irregular time series. This chapter aims to explore the quality of TimeFlow's learned representations and demonstrate their usefulness for downstream tasks.

5.1 Introduction

In Chapter 4 we introduced a model capable of extracting representations from potentially unaligned and irregular time series. This chapter delves into the representation learning capabilities of TimeFlow, focusing on both the structure of the latent space and the practical utility of the learned representations.

(i) First, we explore the latent space structure between two learned codes. By interpolating between these codes, we can visualize how intermediate representations behave in the time series domain. This helps us understand the

smoothness and structure of the latent space, providing insights into how TimeFlow captures and encodes meaningful features of time series data.

(ii) Next, we investigate TimeFlow's sensitivity to modulation perturbations. We observe how these perturbations affect the decoded time series by adding Gaussian noise to specific modulation dimensions for a given layer.

(iii) Furthermore, we examine the distributions of latent codes for times series observed at different time periods. Using principal component analysis, we visualize these distributions and analyze the impact of temporal distribution shifts on the latent space.

(iv) In the final section, we explore the potential of unconditional time series generation using the learned latent space representations. We can generate new samples and decode them into time series data by learning a probabilistic distribution over the latent space.

Through these explorations, we aim to validate TimeFlow's ability to extract semantically rich and practically useful representations, showcasing its potential for downstream tasks.

5.2 Latent space exploration

In this section, we explore the structure of TimeFlow's latent space through several post-hoc experiments.

5.2.1 Latent space interpolation between representations

First, we aim to explore the structure of the latent space between two encoded time series. We construct an interpolation path in the latent space between the two learned codes and examine the resulting interpolations in the decoded time series space.

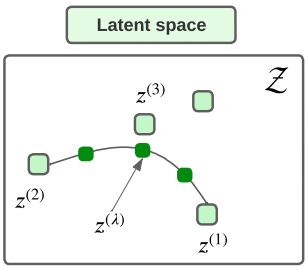


Figure 5.1: Latent space interpolation path (Bezier curve).

Setting. Consider two latent codes $\boldsymbol{z}^{(1)}$ and $\boldsymbol{z}^{(2)}$. We choose the interpolation path to be a Bézier curve with a control point $\boldsymbol{z}^{(3)}$ defined by $\boldsymbol{z}^{(\lambda)} = (1-\lambda)^2\boldsymbol{z}^{(1)} + 2\lambda(1-\lambda)\boldsymbol{z}^{(3)} + \lambda^2\boldsymbol{z}^{(2)}$, shown in Figure 5.1. We interpolate between the two latent codes $\boldsymbol{z}^{(1)}$ and $\boldsymbol{z}^{(2)}$ using regularly spaced values of $\lambda \in [0, 1]$. Figure 5.2 shows a comparison between the decoded values $f_{\theta,h_{\boldsymbol{w}}(\boldsymbol{z}^{(\lambda)})}(t)$ and the Bézier interpolation between the ground truth time series $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ on the *Electricity* dataset.

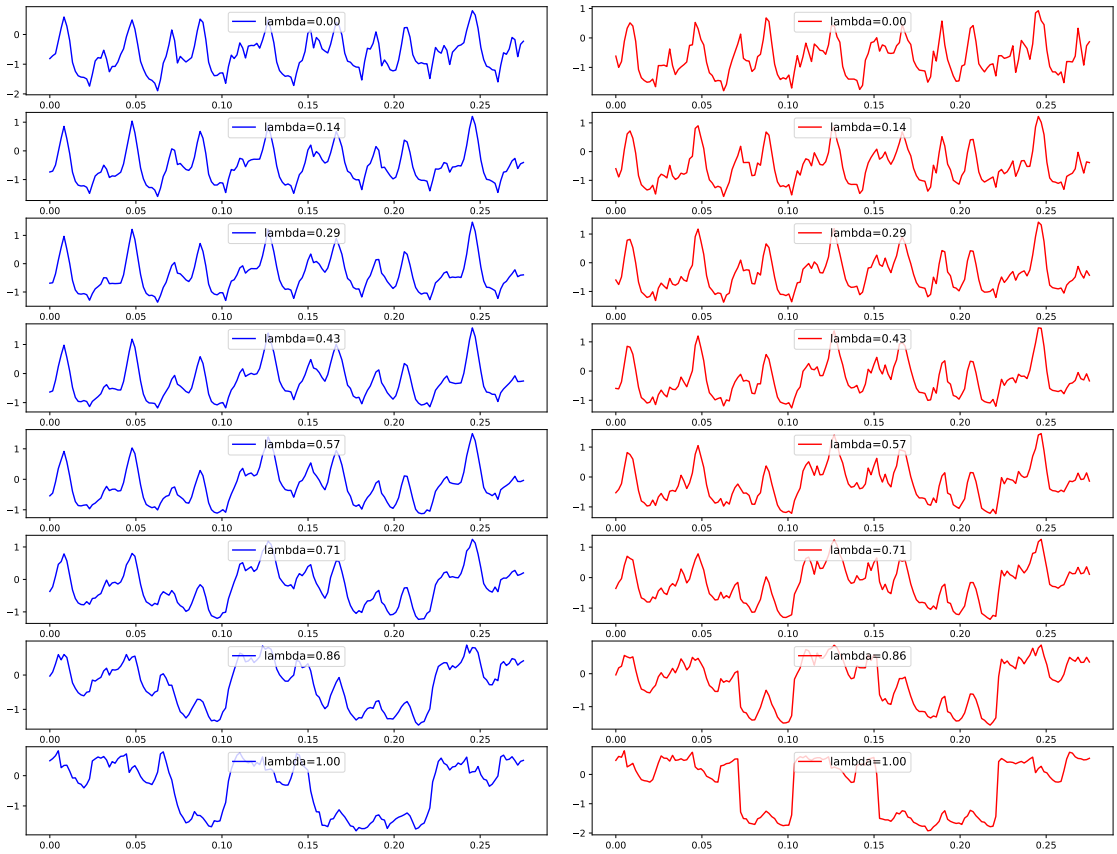


Figure 5.2: *Electricity* dataset. Visualization of the decoded Bézier interpolations between latent codes $\boldsymbol{z}^{(1)}$ and $\boldsymbol{z}^{(2)}$ (left plot), compared with Bézier interpolations of ground truth time series $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ (right plot).

Results. In Figure 5.2, we observe that the interpolation path between two codes yields a smooth transition in the time series domain. These transitions are similar to the ones between the ground truth time series. This suggests that the latent space is smooth and well-structured and that the learned representations captured meaningful features of the time series, which could explain TimeFlow's generalization property.

5.2.2 TimeFlow sensitivity to modulations perturbation

In the previous section, we observed that the latent space is well structured. Another question arises: *can we interpret the dimension of the modulation mechanism?*

Setting. In this experiment, we perturb the modulation (by adding Gaussian noise) for only one layer and a given channel of the INR (see Appendix D.1) for the *Electricity* dataset. Then we observe the difference in the time domain between the non-perturbed and the perturbed TimeFlow. For example, in Figure 5.3, we

add noise only for the third layer of the INR and the 50th channel.

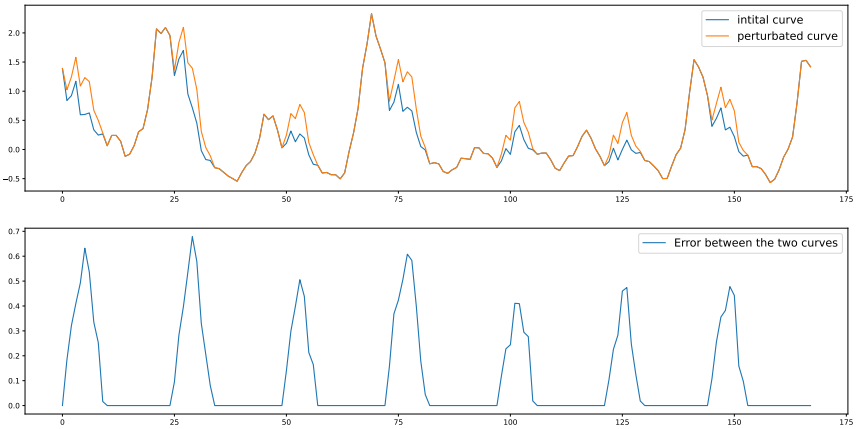


Figure 5.3: Small perturbation to the modulation of the third layer, 50th channel.

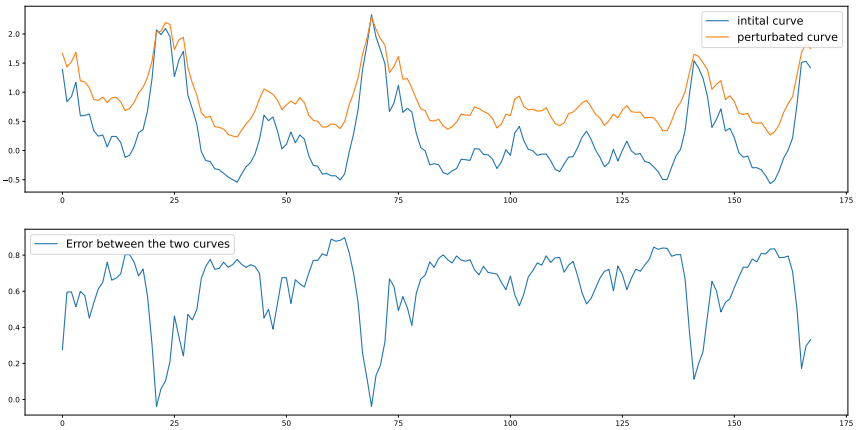


Figure 5.4: Small perturbation to the modulation of the third layer, 51th channel.

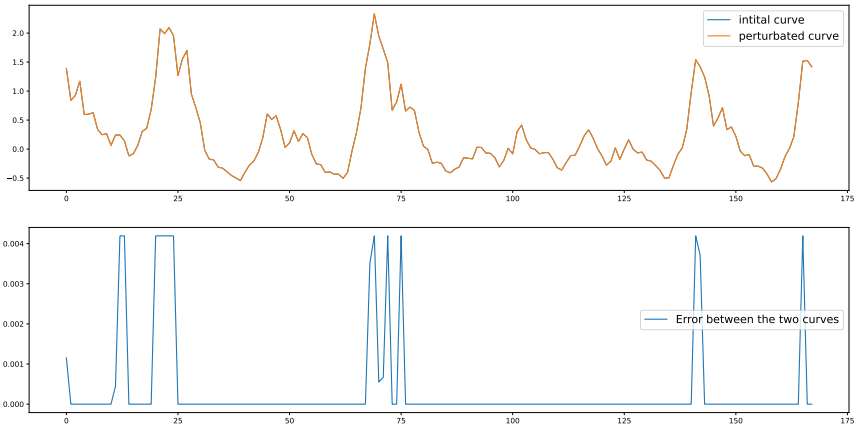


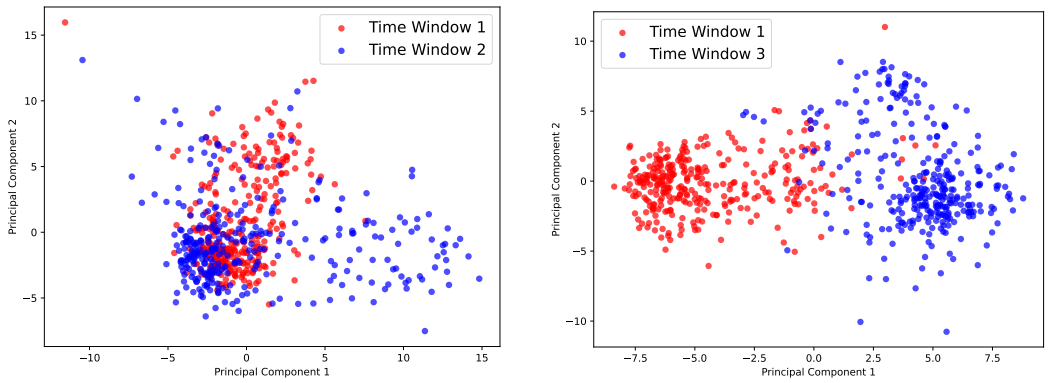
Figure 5.5: Small perturbation to the modulation of the fourth layer, 50th channel.

Results. In Figure 5.3, we observed that adding a small perturbation added a smooth daily frequency pattern. In Figure 5.4, we observed that adding a small perturbation induces a bias that impacts the high frequencies but does not affect the low frequencies. Finally, in Figure 5.5, adding a small perturbation induces a very local and slight bias (the effect is almost null). In conclusion, the impact of the small perturbation depends on the channel and the layer, but it is hard to interpret each dimension independently.

5.2.3 Visualization of code distributions in the latent space

Examining the behavior of the latent space at the sample level is insightful, as it reveals how individual time series behave within the latent space. However, exploring the latent space between two different time series distributions is another critical aspect that provides a broader understanding of the model's representational capabilities.

Setting. In this experiment, we encode all samples (321 samples) from the *Electricity* dataset for two distinct time periods (each period is about 25 days $\approx$ 600 timestamps). This results in two distributions of latent codes, each representing different temporal support. Then, we employ Principal Component Analysis (PCA) to visualize these two latent code distributions in a two-dimensional space, as illustrated in Figure 5.6. This visualization allows us to observe the encoded time series distribution for different temporal distribution shift. In Figure 5.6a, the two compared time period are separated by approximately 3 months ($\approx$ 2000 timestamps). In Figure 5.6b, the two compared time period are separated by approximately 6 months ($\approx$ 4000 timestamps).



(a) The temporal shift between the two codes distribution is 3 months.

(b) The temporal shift between the two codes distribution is 6 months.

Figure 5.6: *Electricity* dataset. Visualization of the two first PCA axes for different temporal distributions of latent codes.

Results. Figure 5.6a shows that when the temporal periods are not too far from each other, the distributions of codes largely overlap in the 2D visualization from the PCA. Conversely, as illustrated in Figure 5.6b, when the temporal periods are far from each other, the distributions of codes between the two periods become more distinct in the 2D visualization. This observation suggests that for the *Electricity* dataset, the proximity or disparity in temporal distribution shift influences the separability of latent representations in the 2D PCA space. However, this presumed separability in the latent space does not seem to significantly impact the generalization performance of TimeFlow across time, as evidenced by the results presented in Table 4.5 and Table C.15. The results suggest that TimeFlow can preserve the time series distribution shifts in latent space while performing well for the different tasks.

5.3 Unconditional generation over the latent space

In the previous section, we observed that the latent space induced by TimeFlow is well-structure and is able to capture meaningful features. In this section, we want to explore if we can learn a probabilistic distribution over the latent space and then generate new samples in the latent space. Afterward, we can pass the generated codes $\boldsymbol{z}$ to TimeFlow decoder and observed the new generated time series.

5.3.1 Motivations

Synthetic time series generation has recently gained attention in the time series community.

- In particular, there has been an increased interest in time series foundation models, i.e., training reusable time series models (e.g. forecasters) on large datasets from multiple sources. To build interesting/effective time series foundation models, the amount of collected time series should be prominent and these time series should be diverse. Since many time series come from the industrial sector (e.g., wind power generation, electricity load consumption, road traffic measurements, air quality sensors, etc.), companies cannot share these data due to privacy or proprietary restrictions. However, they may have an interest in sharing generated data (Emami et al., 2023).

- In addition, using synthetic time series to augment training datasets is a promising way to improve the generalization properties of time series models (Fawaz et al., 2018).

From a technical perspective, using a generative model over TimeFlow latent space induce practical advantages.

(i) Training of the generative model over the latent space of TimeFlow inherits the capability to handle unaligned/irregular time series.

(ii) The generated time series are continuous and can be queried at any timestamp, which is very convenient for creating a large dataset with aligned timestamps.

(iii) As demonstrated in latent diffusion (Rombach et al., 2022), learning a generative model over the latent space allows for generating high-quality data by effectively capturing complex patterns in the latent space, leading to more realistic and diverse outputs.

5.3.2 Method: a two stages approach

Problem setting. Let us consider a time series dataset $\{\boldsymbol{x}^{(j)}\}_{j=1}^{n}$ which are sampled from an unknown probability distribution $p(\mathbf{x})$. At training we aim to approximate the true distribution $p(\mathbf{x})$ with the estimated distribution $\hat{p}_\nu(\mathbf{x})$. At generation we aim to draw new time series from the estimated distribution $x^{(gen)} \sim \hat{p}_\nu(\mathbf{x})$. To achieve this, we use a two-stage approach that couples TimeFlow's representation capabilities with latent space generative modeling.

Model overview. At training we propose to rely on a two stages approach.

1. First, we use TimeFlow to fit the time series $\{\boldsymbol{x}^{(j)}\}_{j=1}^{n}$. After obtaining good reconstruction, we can extract the corresponding representation dataset $\{\boldsymbol{z}^{(j)}\}_{j=1}^{n}$ of fixed dimension ($\forall j \in \{1, ..., n\},\ \boldsymbol{z}^{(j)} \in \mathbb{R}^{\boldsymbol{d}}$).

2. Second, we learn a generative model over $p(\mathbf{z})$ based on the observed codes $\{\boldsymbol{z}^{(j)}\}_{j=1}^{n}$. We choose to fit a Denoising Diffusion Probabilistic Model (DDPM) (Ho et al., 2020) regarding the impressive capabilities of these models compared to Generative Adversarial Networks (GANs) (Goodfellow et al., 2020) and Variational Auto Encoders (VAEs) (Kingma and Welling, 2013). At the end of the training process, we have learned a distribution $\hat{p}_\nu(\mathbf{z})$ that approximates $p(\mathbf{z})$.

At inference, we can then generate $\mathbf{z}^{(\mathbf{gen})} \sim \hat{p}_\nu(\mathbf{z})$, pass it through the trained TimeFlow decoder and get $\mathbf{x}^{(\mathbf{gen})}$ for arbitrary timestamps (include in the temporal support). Each stage is explained in detail in the following section.

5.3.2.1 Stage 1. Fit TimeFlow

First, we use TimeFlow (see Algorithm 1) to fit the potential unaligned/irregular time series $\{\boldsymbol{x}^{(j)}\}_{j=1}^{n}$ thanks to the model $f_{\theta,h_{\mathbf{w}}(.)}$. After obtaining good reconstruction, we can extract the corresponding representation dataset $\{\boldsymbol{z}^{(j)}\}_{j=1}^{n}$ of fixed dimension ($\forall j \in \{1, ..., n\},\ \boldsymbol{z}^{(j)} \in \mathbb{R}^{\boldsymbol{d}}$). The procedure is depicted in Figure 5.7.

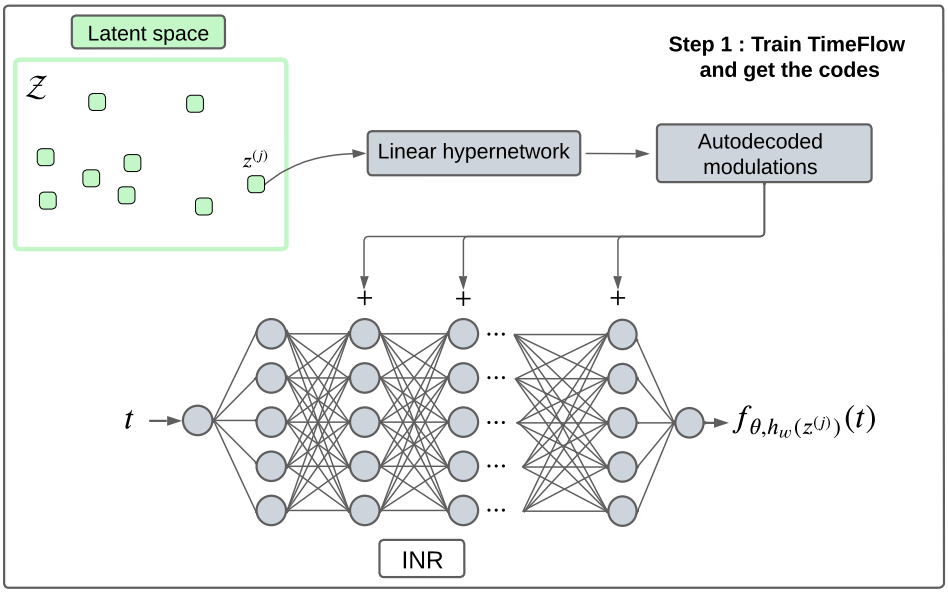


Figure 5.7: Step one. Fit TimeFlow and get codes from the latent space.

5.3.2.2 Stage 2. Fit a DDPM on TimeFlow latent space

A DDPM is a generative model used to model data distributions. The main idea is to gradually corrupt a data point with noise over several steps and then learn how to reverse this process to generate new data points. Below is a step-by-step, high-level explanation.

The forward process. In the forward process, we gradually add noise to the input code $\mathbf{z_0}$ over a series of K steps to produce noisy versions of the data $\mathbf{z_1}, \mathbf{z_2}, \ldots, \mathbf{z_K}$.

At each step k, Gaussian noise is added:

$$q(\mathbf{z_k}|\mathbf{z_{k-1}}) = \mathcal{N}(\mathbf{z_k}; \sqrt{\alpha_k}\mathbf{z_{k-1}}, (1-\alpha_k)I)$$

Here:

- $\mathcal{N}(\cdot; \mu, \mathbf{\Sigma})$ is the normal distribution with mean μ and variance $\mathbf{\Sigma}$.
- α_k is a positive constant, slightly less than one, that controls the amount of noise added at step k.

The full forward process from $\mathbf{z_0}$ to $\mathbf{z_K}$ is:

$$q(\mathbf{z_{1:K}}|\mathbf{z_0}) = \prod_{k=1}^{K} q(\mathbf{z_k}|\mathbf{z_{k-1}})$$

At the end of the forward diffusion process, we want the noisy input to follow a standard normal distribution, $q(\mathbf{z_K}|\mathbf{z_0})$ close to a $\mathcal{N}(0, I)$.

Reverse diffusion process. The objective is to learn how to reverse the forward noising process. The reverse process attempts to remove the noise step-by-step, starting from $\mathbf{z_K}$and proceeding back to $\mathbf{z_0}$:

$$\hat{p}_\nu(\mathbf{z_{k-1}}|\mathbf{z_k}) = \mathcal{N}(\mathbf{z_{k-1}}; \mu_\nu(\mathbf{z_k}, k), \sigma^2(k))$$

Here, μ_ν represents the estimated mean of the denoising distribution, which is estimated by a neural network. The combination of the forward process and the reverse diffusion process in the latent space is illustrated in Figure 5.8.

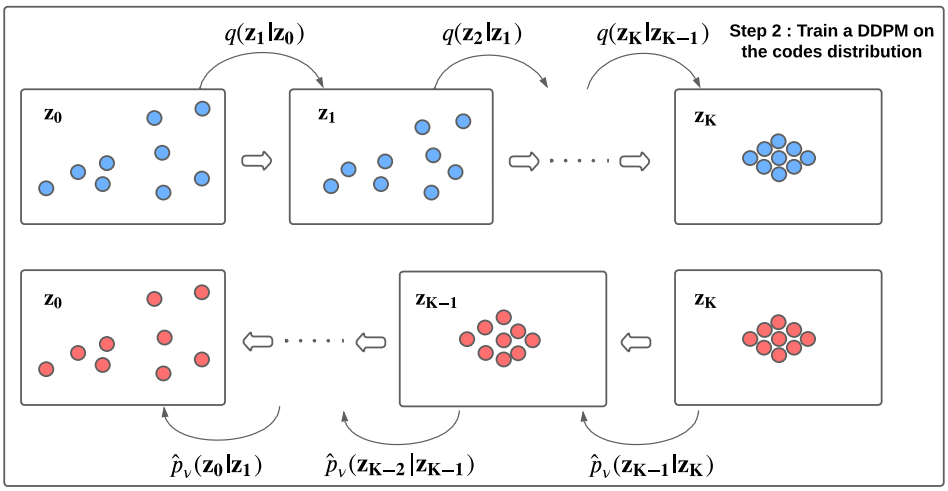


Figure 5.8: Step two. train a DDPM on the latent code distributions

Evidence Lower Bound (ELBO). Similar to training the VAEs, DDPMs use the ELBO on the data likelihood to train the model. The objective is to maximize this bound in ν, which can be written as minimizing the following loss function:

$$\mathcal{L}_\nu = \mathcal{L}_K + \sum_{k=2}^{K} \mathcal{L}_{k-1} + \mathcal{L}_0 \tag{5.1}$$

Where:

- **Prior matching term.** $\mathcal{L}_K = D_{KL}(q(\mathbf{z_K}|\mathbf{z_0})\|p(\mathbf{z_K}))$ measures how close the final noisy distribution $q(\mathbf{z_K}|\mathbf{z_0})$ is to a standard Gaussian prior $p(\mathbf{z_K})$. D_{KL} stands for the Kullback–Leibler divergence which is a divergence between two distributions.
- **Consistency term.** $\mathcal{L}_{k-1} = D_{KL}(q(\mathbf{z_{k-1}}|\mathbf{z_k}, \mathbf{z_0})\|\hat{p}_\nu(\mathbf{z_{k-1}}|\mathbf{z_k}))$ measures how close the reverse process is to the true posterior.
- **Reconstruction term.** $\mathcal{L}_0 = -\log \hat{p}_\nu(\mathbf{z_0}|\mathbf{z_1})$ measures the reconstruction error.

Training. Instead of estimating the neural network $\mu_\nu(.)$, we estimate another related neural network, $\epsilon_\nu(.)$, known as the denoiser. By rewriting, substituting, and deriving appropriately the ELBO, we just have to minimize the quantity in Equation (5.2) (we ignore the constant terms and expectations for simplicity). For a comprehensive understanding of the mathematics and details of the DDPM, please refer to Chan (2024).

$$\mathcal{L}_\nu \propto \sum_{k=1}^{K} ||\epsilon_\nu(\mathbf{z_k}) - \mathbf{z_0}||^2. \tag{5.2}$$

The terms in the equation are defined as follows:

- $\epsilon_\nu(.)$: The denoiser neural network to train. Its objective is to denoise $\mathbf{z_k}$ for any $k \in \{1, .., K\}$ such that the denoised signal $\epsilon_\nu(\mathbf{z_k})$ is close to the ground truth $\mathbf{z_0}$.
- $\mathbf{z_k}$: Sampled from the Gaussian distribution below:

$$\begin{aligned}
\mathbf{z_k} &\sim q(\mathbf{z_k}|\mathbf{z_0}) \\
&= \mathcal{N}(\mathbf{z_k}|\sqrt{\bar{\alpha_k}}\mathbf{z_0}, (1-\bar{\alpha_k})\mathbf{I}), \quad \bar{\alpha_k} = \sum_{i=1}^{k} \alpha_i \\
&= \sqrt{\bar{\alpha_k}}\mathbf{z_0} + \sqrt{(1-\bar{\alpha_k})}\rho, \quad \rho \sim \mathcal{N}(0, \mathbf{I}).
\end{aligned}$$

- $\mathbf{z_0}$: The ground truth code.

The loss in Equation (5.2) is more convenient for training than the one in Equation (5.1) because the denoiser can be trained without doing all the diffusion steps, which is much less time consuming. In practice, we simply follow the procedure described in Algorithm 3 to train the denoiser. For more details on DDPM, please refer to Chan (2024).

Algorithm 3: DDPM Training

while *no convergence* **do**

Sample batch $\mathcal{B}$ of data $(\mathbf{z}^{(j)})_{j\in\mathcal{B}}$;

$\mathbf{z_0^{(j)}} = \mathbf{z}^{(j)}\ \forall j \in \mathcal{B}$;

Pick a random diffusion step $k^{(j)}$ in Uniform$[1, K]$ $\quad \forall j \in \mathcal{B}$;

Draw $\rho^{(j)} \sim \mathcal{N}(0, \mathbf{I})$ $\quad \forall j \in \mathcal{B}$;

$\mathbf{z_k^{(j)}} = \sqrt{\bar{\alpha}_k}\mathbf{z_0^{(j)}} + \sqrt{(1-\bar{\alpha}_k)}\rho^{(j)}$ $\quad \forall j \in \mathcal{B}$;

Update ν according to $\nabla_\nu \sum_{j\in\mathcal{B}} ||\epsilon_\nu(\mathbf{z_k^{(j)}} - \mathbf{z_0^{(j)}})||^2$

end

5.3.2.3 Generation

Generate new codes $\mathbf{z}^{(\mathbf{gen})}$. To generate new time series data, we start with a sample from the standard Gaussian distribution $\mathbf{z_K} \sim \mathcal{N}(0, I)$ and apply the learned reverse process step-by-step to obtain $\mathbf{z_{K-1}}, \mathbf{z_{K-2}}, \ldots, \mathbf{z_0}$. The generated code with the desirable statistical property is $\mathbf{z_0}$. For additional details on the DDPM generation process using the trained denoiser $\epsilon_\nu(.)$, please refer to Appendix D.2.

Pass the generated codes through the TimeFlow decoder. The generated code $\mathbf{z}^{(\mathbf{gen})}$ can now modulate the TimeFlow architecture, which can be expressed as $f_{\theta,h_w(\mathbf{z}^{(\mathbf{gen})})}$. Thus, we can generate the new time series $\mathbf{x}^{(\mathbf{gen})}$ for any timestamp $t \in \mathcal{T}$ by querying $f_{\theta,h_w(\mathbf{z}^{(\mathbf{gen})})}(t)$. The whole procedure is illustrated in Figure 5.9.

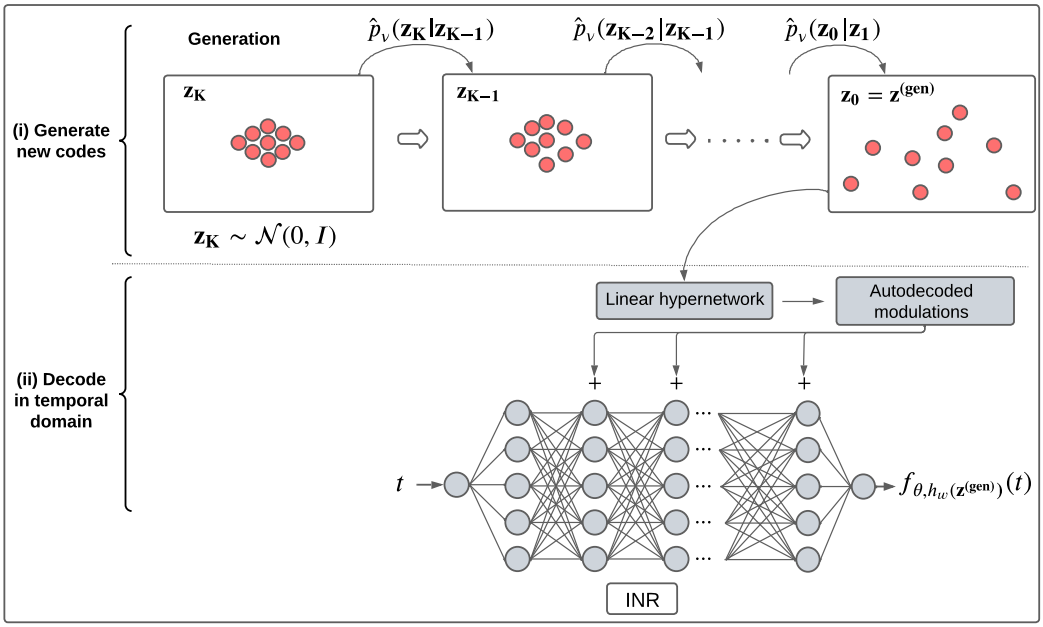


Figure 5.9: Overview of the generation procedure.

5.3.3 Experiments

Setting. We train the proposed model on 8,000 time series from the *Electricity* dataset, each representing hourly sampled load curves over two weeks (336 timestamps). We reserve 2,000 time series for testing and generate an additional 2,000 time series using the procedure described in Section 5.3.2. Our goals for the generated data are twofold: they should be indistinguishable from real data and exhibit diversity, capturing the full range of variability present in the real data. Following the existing literature on time series generation (Yoon et al., 2019; Coletta et al., 2024), we compute a discriminative score and conduct a qualitative evaluation of the generated time series.

- **The discriminative score** measures the fidelity of the generated data. We take 1,000 samples each from the test and generated time series to create a "train" dataset for a 1-NN classifier. We then classify another 1,000 test and 1,000 generated time series, assigning labels based on the nearest neighbor in the "train" dataset. The score is the classification accuracy minus 0.5, where a lower score indicates the generated data are indistinguishable from the test data by the 1-NN classifier.

- **Qualitatively**, we analyze the distributions of the generated and test time series in a 2-dimensional space using t-Distributed Stochastic Neighbor Embedding (t-SNE) visualizations (Van der Maaten and Hinton, 2008). t-SNE preserves local structures, making it effective for identifying clusters and patterns. We aim for the generated and test time series to be indistinguishable in this low-dimensional space, indicating good diversity.

Model implementation and baselines. • To implement our method, we couple TimeFlow with an existing DDPM implementation[1]. • The architecture of the denoiser is detailed in Appendix D.3. The first baseline we compare with, applies the same DDPM directly to the time series. This comparison highlights the benefits of learning generation in the latent space of TimeFlow rather than directly on the time series. The DDPM architecture is identical to that used with TimeFlow and is trained under the same conditions (batch size, number of epochs, hyperparameters, etc.). • We also compare the proposed method with TimeGan (Yoon et al., 2019), one of the most popular time series generation models.

Quantitative experiment. In Table 5.1, we showcase the discriminative score for the three considered methods.

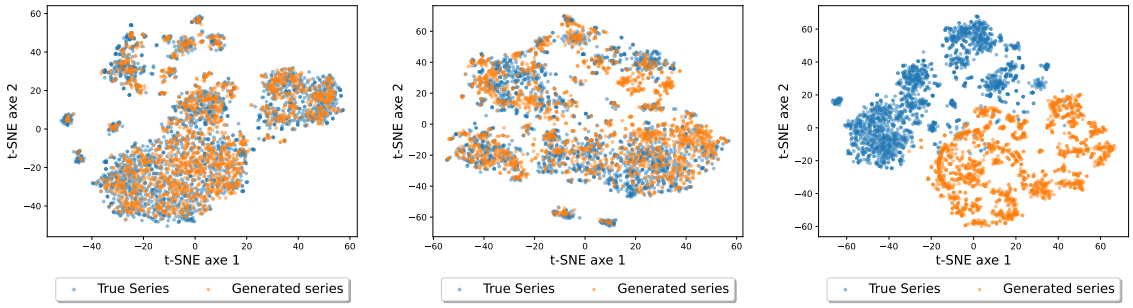
[1] https://github.com/lucidrains/denoising-diffusion-pytorch

Table 5.1: Discriminative score on the *Electricity* dataset.

	TimeFlow + DDPM	**DDPM only**	**TimeGAN**	**Fully separable generation**
Discriminative score	**0.1388**	0.1704	0.4890	0.5000

Results. Table 5.1 demonstrates that distinguishing between time series from the test set and those generated by TimeFlow + DDPM or DDPM only is challenging. The discriminative scores suggest that the series generated by these diffusion-based methods closely resemble the test series. TimeFlow + DDPM slightly outperforms DDPM alone, likely due to the enhanced generation stability in the latent space, as seen in latent diffusion models (Rombach et al., 2022). In contrast, with TimeGAN, it is easy to distinguish between the test and the generated series.

Qualitative experiments. We compare the three methods using t-SNE visualizations in 2D in Figure 5.10.



(a) TimeFlow + DDPM. (b) DDPM. (c) TimeGan.

Figure 5.10: t-SNE visualization comparing generated and test time series for the three considered methods.

Results. As shown in Figure 5.10, it is nearly impossible to distinguish between the test series and those generated by TimeFlow + DDPM on the 2D t-SNE projection. The t-SNE plots indicate that the generated series from TimeFlow + DDPM capture a diverse range of patterns, mirroring the variability present in the test data. For the DDPM-only method, it is also challenging to differentiate the generated series from the test series, although there are still a few regions in the 2D plots where they do not overlap. In contrast, the TimeGAN model fails to generate data that is convincingly similar to the test series. We also propose a 2D PCA visualization in Appendix D.4.

Some visual plots. Figure 5.11 shows three samples generated using the proposed generation method alongside three test series that were not used to train the model. We generate series of two weeks length and hourly timestamps. Visually, we show that we can generate diverse series that respect the statics of the initial distribution (such as the hourly and weekly seasonality).

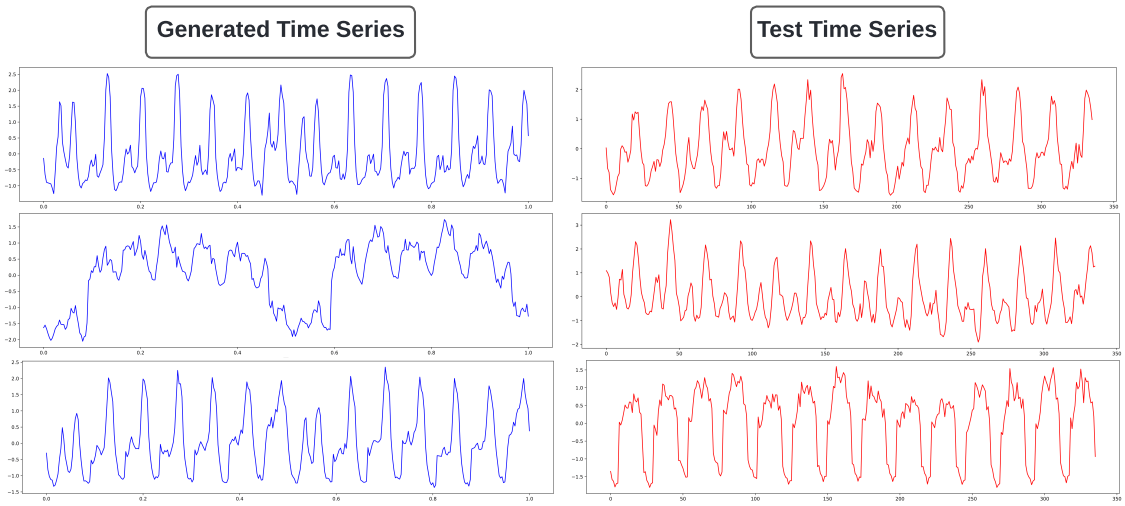


Figure 5.11: Visualization of three generated and three real test time series from the *Electricity* dataset.

5.4 Limitations and conclusion

Limitations. Despite the promising results, several limitations need to be addressed. First, the interpretability of individual latent dimensions remains a challenge, making it difficult to fully understand their specific contributions to TimeFlow performance. Second, while our unconditional generation experiments were successful on the *Electricity* dataset, further studies are needed to validate TimeFlow's scalability and effectiveness on different time series datasets. In addition, the current form of TimeFlow's latent space inherits certain limitations of TimeFlow itself. For example, it does not support conditional generation, such as including meteorological variables as a condition. Finally, while effective, the two-step generation mechanism can be time consuming compared to direct generation approaches.

Conclusion. In this chapter, we conducted an in-depth investigation of the latent space induced by TimeFlow and its representation learning capabilities. Our analysis of the latent space interpolation showed that the transitions between learned codes are smooth and well structured, capturing meaningful features of the time series data. In addition, we showed that the latent space preserves temporal distribution shifts of the encoded time series. Finally, we demonstrated the practical utility of

TimeFlow's latent space for generating realistic and diverse synthetic time series using a DDPM.

The ability to extract meaningful representations from unaligned/irregular time series can be useful in many real-world applications. Future work may explore TimeFlow's latent representations for other downstream tasks such as classification or anomaly detection.

Part III

Conclusion

Chapter 6

Conclusion

6.1 Synthesis

Neural representation learning for time series is a rapidly growing area of research. In this thesis, we delve into the topic through the following key points.

1. **Historical context and the emergence of time series neural representation.** In Chapter 2, we begin by reviewing the history of time series modeling, highlighting the importance of representation learning in the domain. With the rise of deep learning, we highlight the emergence of neural representation learning for time series and show the flexibility these representations offer across different paradigms, such as contrastive learning and reconstruction learning. We then identify several open challenges in representation learning for time series and highlight their relevance to EDF.

2. **Interpretable neural representations.** Chapter 3 focuses on achieving interpretable neural representations for time series, specifically applied to downstream classification tasks. Despite the flexibility of neural networks, it is critical for decision-makers to understand the decision made by the model. We begin by establishing the requirements for an interpretable neural representation. We identify five essential requirements for making time series representations interpretable and propose a novel architecture that utilizes a vector quantization mechanism in the latent space to meet these criteria. The obtained representations are then quantitatively and qualitatively evaluated on downstream classification tasks on the UCR archive.

3. **Continuous time series modeling.** Chapter 4 addresses the challenges posed by irregular and unaligned time series through continuous time series modeling. We introduce a new model called TimeFlow, which is based on modulated implicit neural representation (INR) and meta-learning optimization. TimeFlow's modulated INR architecture effectively fits irregular time series

and captures the underlying continuity within a representation. The meta-learning component allows TimeFlow to adapt to new samples and contexts by modifying only the representation part of the architecture. We demonstrate TimeFlow's capability through imputation and forecasting experiments.

4. **Evaluation of TimeFlow representations.** In Chapter 5, we examine the representations learned by TimeFlow. First, we evaluate the quality of these representations through various post-hoc experiments, noting that the latent space is structured, distribution shifts between time series are reflected within the latent space, and different dimensions of the modulation have semantic significance. Next, we assess the utility of TimeFlow for downstream tasks by evaluating its performance in generation task. This involves generating new representations via a generative model learned in the latent space, and subsequently decoding these representations into the time series space. We demonstrate that this method can efficiently generate continuous time series from potentially irregular and non-aligned data.

From an industrial perspective. The work conducted in this thesis has not only contributed to academic advancement, but has also addressed key needs of EDF. Specifically:

- Improving the interpretability of model output increases the acceptance of deep learning models by decision makers. While the models currently used in the industry for time series classification are simple and interpretable, our first work provides a powerful alternative by proposing an interpretable time series neural representation.
- Our second proposed model, TimeFlow, demonstrates robustness and adaptability, allowing it to be reused across different samples and time periods without retraining, even in the presence of distribution shifts.
- Efficient modeling of large numbers of unaligned time series is crucial for many applications in the energy sector. TimeFlow facilitates the joint and effective handling of these complex datasets.

While this work has contributed to the field of time series neural representation, we have outlined some limitations in our contributions that need to be addressed in future work. We also discuss the constraint identified with the currently available public datasets in Appendix E.

In the next section, we outline some perspectives for future work aimed at addressing these limitations.

6.2 Perspectives

This section outlines some technical perspectives for future work to address limitations identified in this thesis.

Improving classification with the symbolic sequence learning algorithm. The main limitation for the interpretability and performance of classification in Chapter 3 lies in the way we apply logistic regression to the learned discrete representation. Currently, we fail to explore the full space of subsequences for discriminative patterns by using only symbolic subsequences of limited length.

A solution to this issue was proposed in Ifrim and Wiuf (2011), where the authors introduced the Symbolic Sequence Learning (SEQL) method for classification. The SEQL method searches the entire subsequence space for discrete symbolic sequences using a gradient-bounded coordinate-descent algorithm. This approach efficiently selects discriminative subsequences without exhaustively exploring the whole space following these steps. (i) Based on the gradient of a small subsequence, they can possibly discard all subsequences beginning with this small subsequence. (ii) After a few branch-and-bound iterations, they investigate all candidates with the greatest gradient magnitudes. (iii) After identifying the most discriminative subsequence, they apply a gradient step only to the parameter corresponding to that subsequence (coordinate descent).

This promising method has already been tested with the SAX-SEQL method (Nguyen et al., 2017), which supports both ℓ_1 and ℓ_2 regularizations and offers several advantages such as an effective search in the entire symbolic subsequence space, performances gains, and enhance interpretability by identifying the most discriminative subsequences.

Applying this classification method on top of our proposed neural discrete representation appears to be a promising avenue for future research.

Exploring the transferability of our proposed neural discrete representation learning model. In preliminary experiments, we observed that the vector quantization auto-encoder can be effectively transferred across different time series datasets. Specifically, it can be trained on one or multiple datasets and still perform well in reconstructing data from an unseen dataset. Additionally, the convolutional mechanism allows our architecture to handle time series of varying lengths, positioning our model as a versatile feature extractor. This capability is particularly advantageous for time series datasets with limited samples. We intend to further explore this promising direction in future work.

Develop a conditioning TimeFlow for multivariate time series and context variables. Although TimeFlow demonstrates versatility by being applicable to new samples, different temporal contexts, and irregularly sampled time series, one missing component is the ability to incorporate context variables. In many real-world scenarios, accurate characterization of time series requires external context variables, which can be either static or temporal. In future work, we plan to develop a version of TimeFlow that can accept both static and temporal external variables. One way to achieve this is a modulation mechanism conditioned on context variables as illustrated in Figure 6.1.

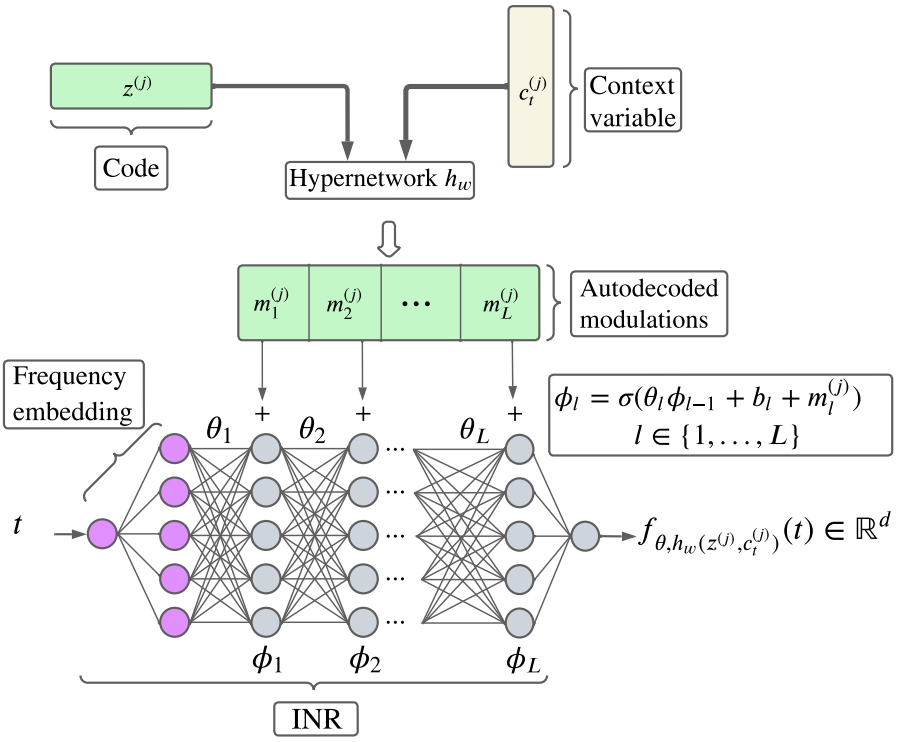


Figure 6.1: TimeFlow architecture conditioned on context variable.

Bibliography

A. Araujo, W. Norris, and J. Sim. Computing receptive fields of convolutional neural networks. *Distill*, 4(11):e21, 2019.

K. Bandara, R. J. Hyndman, and C. Bergmeir. Mstl: A seasonal-trend decomposition algorithm for time series with multiple seasonal patterns. *arXiv preprint arXiv:2107.13462*, 2021.

Y. Bengio, A. C. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, 2013. doi: 10.1109/TPAMI.2013.50.

L. Beyer, X. Zhai, and A. Kolesnikov. Better plain vit baselines for imagenet-1k. *arXiv preprint arXiv:2205.01580*, 2022.

M. Bilos, K. Rasul, A. Schneider, Y. Nevmyvaka, and S. Günnemann. Modeling temporal data as continuous functions with stochastic process diffusion. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *International Conference on Machine Learning, ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 2452–2470. PMLR, 2023.

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

E. D. Brouwer, J. Simm, A. Arany, and Y. Moreau. Gru-ode-bayes: continuous modeling of sporadically-observed time series. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 7379–7390, 2019.

R. Brown. *Statistical Forecasting for Inventory Control*. McGraw-Hill, 1959. URL https://books.google.fr/books?id=oKI8AAAAIAAJ.

J. A. Cadzow, B. Baseghi, and T. Hsu. Singular-value decomposition approach to time series modelling. In *IEE Proceedings F (Communications, Radar and Signal Processing)*, volume 130, pages 202–210. IET, 1983.

W. Cao, D. Wang, J. Li, H. Zhou, Y. Li, and L. Li. Brits: bidirectional recurrent imputation for time series. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6776–6786, 2018.

A. Chaman and I. Dokmanic. Truly shift-equivariant convolutional neural networks with adaptive polyphase upsampling. In *55th Asilomar Conference on Signals, Systems, and Computers, ACSSC*, pages 1113–1120. IEEE, 2021. doi: 10.1109/IEEECONF53345.2021.9723377.

S. H. Chan. Tutorial on diffusion models for imaging and vision. *arXiv preprint arXiv:2403.18103*, 2024.

H. Chen, S. Grant-Muller, L. Mussone, and F. Montgomery. A study of hybrid neural network approaches and the effects of missing data on traffic forecasting. *Neural Computing & Applications*, 10:277–286, 2001.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

R. R. Chowdhury, J. Li, X. Zhang, D. Hong, R. K. Gupta, and J. Shang. Primenet: Pre-training for irregular multivariate time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 7184–7192, 2023.

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

Y. G. Cinar, H. Mirisaee, P. Goswami, É. Gaussier, and A. Aït-Bachir. Period-aware content attention rnns for time series forecasting with missing values. *Neurocomputing*, 312:177–186, 2018.

J. S. Clark and O. N. Bjørnstad. Population time series: process variability, observation errors, missing values, lags, and hidden states. *Ecology*, 85(11):3140–3150, 2004.

T. Cohen and M. Welling. Group equivariant convolutional networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2990–2999, 2016.

A. Coletta, S. Gopalakrishnan, D. Borrajo, and S. Vyetrenko. On the constrained time-series generation problem. *Advances in Neural Information Processing Systems*, 36, 2024.

G. Corani, A. Benavoli, and M. Zaffalon. Time series forecasting with gaussian processes needs priors. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part IV 21*, pages 103–117. Springer, 2021.

C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2:303–314, 1989.

A. Das, W. Kong, R. Sen, and Y. Zhou. A decoder-only foundation model for time-series forecasting. *arXiv preprint arXiv:2310.10688*, 2023.

H. A. Dau, A. J. Bagnall, K. Kamgar, C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. J. Keogh. The UCR time series archive. *IEEE CAA J. Autom. Sinica*, 6(6):1293–1305, 2019. doi: 10.1109/jas.2019.1911747.

A. Dempster, F. Petitjean, and G. I. Webb. ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Min. Knowl. Discov.*, 34(5):1454–1495, 2020. doi: 10.1007/s10618-020-00701-z.

A. Dempster, D. F. Schmidt, and G. I. Webb. Hydra: Competing convolutional kernels for fast and accurate time series classification. *Data Mining and Knowledge Discovery*, 37(5):1779–1805, 2023.

D. Deng, F. Karl, F. Hutter, B. Bischl, and M. Lindauer. Efficient automated deep learning for time series forecasting, 2022.

D. Dennis, D. A. E. Acar, V. Mandikal, V. S. Sadasivan, V. Saligrama, H. V. Simhadri, and P. Jain. Shallow rnn: accurate time-series classification on resource constrained devices. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

J. Dong, H. Wu, H. Zhang, L. Zhang, J. Wang, and M. Long. Simmtm: A simple pre-training framework for masked time-series modeling. *Advances in Neural Information Processing Systems*, 36, 2024.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth

16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

W. Du, D. Côté, and Y. Liu. Saits: Self-attention-based imputation for time series. *Expert Systems with Applications*, 219:119619, 2023.

E. Dupont, H. Kim, S. M. A. Eslami, D. J. Rezende, and D. Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 5694–5725. PMLR, 2022.

P. Emami, A. Sahu, and P. Graf. Buildingsbench: A large-scale dataset of 900k buildings and benchmark for short-term load forecasting. *Advances in Neural Information Processing Systems*, 36:19823–19857, 2023.

R. Fathony, A. K. Sahu, D. Willmott, and J. Z. Kolter. Multiplicative filter networks. In *International Conference on Learning Representations*, 2021.

H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Data augmentation using synthetic data for time series classification with deep residual networks. *arXiv preprint arXiv:1808.02455*, 2018.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

E. Fix. *Discriminatory analysis: nonparametric discrimination, consistency properties*, volume 1. USAF school of Aviation Medicine, 1985.

E. Fons, A. Sztrajman, Y. El-Laham, A. Iosifidis, and S. Vyetrenko. Hypertime: Implicit neural representation for time series. *CoRR*, abs/2208.05836, 2022.

V. Fortuin, M. Hüser, F. Locatello, H. Strathmann, and G. Rätsch. SOM-VAE: interpretable discrete representation learning on time series. In *7th International Conference on Learning Representations, ICLR*, 2019.

V. Fortuin, D. Baranchuk, G. Rätsch, and S. Mandt. Gp-vae: Deep probabilistic time series imputation. In *International conference on artificial intelligence and statistics*, pages 1651–1661. PMLR, 2020.

J. Franceschi, A. Dieuleveut, and M. Jaggi. Unsupervised scalable representation learning for multivariate time series. In *Advances in Neural Information Processing Systems 32*, pages 4652–4663, 2019.

M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. J. Rezende, and S. M. A. Eslami. Conditional neural processes. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, volume 80, pages 1690–1699. PMLR, 2018.

A. Gersho and R. M. Gray. *Vector quantization and signal compression*, volume 159 of *The Kluwer international series in engineering and computer science*. Kluwer, 1991. doi: 10.1007/978-1-4615-3626-0.

P. Geurts. Pattern extraction for time series classification. In *European conference on principles of data mining and knowledge discovery*, pages 115–127. Springer, 2001.

G. Giebel and G. Kariniotakis. Wind power forecasting—a review of the state of the art. *Renewable energy forecasting*, pages 59–109, 2017.

R. Godahewa, C. Bergmeir, G. I. Webb, R. J. Hyndman, and P. Montero-Manso. Monash time series forecasting archive. In *Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

J. Grabocka, A. Nanopoulos, and L. Schmidt-Thieme. Classification of sparse time series via supervised matrix factorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 929–934, 2012.

J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme. Learning time-series shapelets. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, pages 392–401. ACM, 2014. doi: 10.1145/2623330.2623613.

T. J. Hastie. Generalized additive models. In *Statistical models in S*, pages 249–307. Routledge, 2017.

S. Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

K. Higashiyama, Y. Fujimoto, and Y. Hayashi. Feature extraction of nwp data for wind power forecasting using 3d-convolutional neural networks. *Energy Procedia*, 155:350–358, 2018.

J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

C. C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International journal of forecasting*, 20(1):5–10, 2004.

L. Huang and T. Hoefler. Compressing multidimensional weather and climate data into neural networks. In *International Conference on Learning Representations, ICLR*, 2023.

M. Hüsken and P. Stagge. Recurrent neural networks for time series classification. *Neurocomputing*, 50:223–235, 2003. doi: 10.1016/S0925-2312(01)00706-8.

F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

R. J. Hyndman, E. Wang, and N. Laptev. Large-scale unusual time series detection. In *2015 IEEE international conference on data mining workshop (ICDMW)*, pages 1616–1619. IEEE, 2015.

G. Ifrim and C. Wiuf. Bounded coordinate-descent for biological sequence classification in high dimensional predictor space. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 708–716. ACM, 2011. doi: 10.1145/2020408.2020519.

International Energy Agency (IEA). Wind power generation, 2023. URL https://www.iea.org/energy-system/renewables/wind. IEA, Paris.

H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019.

H. Ismail Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P.-A. Muller, and F. Petitjean. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 34 (6):1936–1962, 2020.

B. K. Iwana, V. Frinken, and S. Uchida. A robust dissimilarity-based neural network for temporal pattern recognition. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 265–270, 2016. doi: 10.1109/ICFHR.2016.0058.

B. K. Iwana, V. Frinken, and S. Uchida. DTW-NN: A novel neural network for time series recognition using dynamic alignment between inputs and weights. *Knowledge-Based Systems*, 188:104971, 2020. doi: 10.1016/j.knosys.2019.104971.

K. Jeong and Y. Shin. Time-series anomaly detection with implicit neural representation. *CoRR*, abs/2201.11950, 2022.

T. W. Joo and S. B. Kim. Time series forecasting based on wavelet filtering. *Expert Systems with Applications*, 42(8):3868–3874, 2015.

A. Kampouraki, G. Manis, and C. Nikou. Heartbeat time series classification with support vector machines. *IEEE transactions on information technology in biomedicine*, 13(4):512–518, 2008.

J. Keisler, E.-G. Talbi, S. Claudel, and G. Cabriel. An algorithmic framework for the optimization of deep neural networks architectures and hyperparameters. *arXiv preprint arXiv:2303.12797*, 2023.

E. J. Keogh and T. Rakthanmanon. Fast shapelets: A scalable algorithm for discovering time series shapelets. In *Proceedings of the 13th SIAM International Conference on Data Mining*, pages 668–676. SIAM, 2013. doi: 10.1137/1.9781611972832.74.

T. Kim, W. Ko, and J. Kim. Analysis and impact evaluation of missing data imputation in day-ahead pv generation forecasting. *Applied Sciences*, 9(1):204, 2019.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

T. Kohonen. The self-organizing map. *Proc. IEEE*, 78(9):1464–1480, 1990. doi: 10.1109/5.58325.

M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

M. Lange and U. Focken. *Physical approach to short-term wind power prediction*, volume 208. Springer, 2006.

R. J. Larsen and M. L. Marx. *An introduction to mathematical statistics*. Prentice Hall Hoboken, NJ, 2005.

V. Le Guen. Deep learning for spatio-temporal forecasting–application to solar energy. *arXiv e-prints*, pages arXiv–2205, 2022.

Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Y. Li, S. Lin, B. Zhang, J. Liu, D. S. Doermann, Y. Wu, F. Huang, and R. Ji. Exploiting kernel sparsity and entropy for interpretable CNN compression. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 2800–2809. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00291.

Y. Li, Z. Chen, D. Zha, M. Du, J. Ni, D. Zhang, H. Chen, and X. Hu. Towards learning disentangled representations for time series. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3270–3278. ACM, 2022. doi: 10.1145/3534678.3539140.

Z. Li, Z. Rao, L. Pan, P. Wang, and Z. Xu. Ti-mae: Self-supervised masked time series autoencoders. *arXiv preprint arXiv:2301.08871*, 2023.

Y. Liang, H. Wen, Y. Nie, Y. Jiang, M. Jin, D. Song, S. Pan, and Q. Wen. Foundation models for time series analysis: A tutorial and survey. *arXiv preprint arXiv:2403.14735*, 2024.

J. Lin, E. J. Keogh, S. Lonardi, and B. Y. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11, 2003. doi: 10.1145/882082.882086.

J. Lin, E. J. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 15(2):107–144, 2007. doi: 10.1007/s10618-007-0064-z.

Z. C. Lipton. The mythos of model interpretability. *Commun. ACM*, 61(10):36–43, 2018. doi: 10.1145/3233231.

H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search, 2018.

H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search, 2019a.

S. Liu, H. Yu, C. Liao, J. Li, W. Lin, A. X. Liu, and S. Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022.

S. Liu, X. Li, G. Cong, Y. Chen, and Y. Jiang. Multivariate time-series imputation with disentangled temporal representations. In *The Eleventh International Conference on Learning Representations, ICLR*, 2023.

Y. Liu, R. Yu, S. Zheng, E. Zhan, and Y. Yue. Naomi: Non-autoregressive multiresolution sequence imputation. *Advances in neural information processing systems*, 32, 2019b.

Y. Luo, X. Cai, Y. Zhang, J. Xu, et al. Multivariate time series imputation with generative adversarial networks. *Advances in neural information processing systems*, 31, 2018.

Y. Luo, Y. Zhang, X. Cai, and X. Yuan. E2gan: End-to-end generative adversarial network for multivariate time series imputation. In *Proceedings of the 28th international joint conference on artificial intelligence*, pages 3094–3100. AAAI Press, 2019.

Y. Luo, C. Xu, Y. Liu, W. Liu, S. Zheng, and J. Bian. Learning differential operators for interpretable time series modeling. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1192–1201. ACM, 2022. doi: 10.1145/3534678.3539245.

Q. Ma, J. Zheng, S. Li, and G. W. Cottrell. Learning representations for time series clustering. In *Advances in Neural Information Processing Systems 32*, pages 3776–3786, 2019.

Q. Ma, Z. Liu, Z. Zheng, Z. Huang, S. Zhu, Z. Yu, and J. T. Kwok. A survey on time-series pre-trained models. *arXiv preprint arXiv:2305.10716*, 2023.

P. Malhotra, V. TV, L. Vig, P. Agarwal, and G. Shroff. Timenet: Pre-trained deep recurrent neural network for time series classification. In *25th European Symposium on Artificial Neural Networks, ESANN 2017, Bruges, Belgium, April 26-28, 2017*, 2017.

J. Mei, Y. de Castro, Y. Goude, and G. Hébrail. Nonnegative matrix factorization for time series recovery from a few temporal aggregates. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2382–2390, 2017.

M. Middlehurst, J. Large, M. Flynn, J. Lines, A. Bostrom, and A. Bagnall. Hivecote 2.0: a new meta ensemble for time series classification. *Machine Learning*, 110(11):3211–3243, 2021.

M. Middlehurst, P. Schäfer, and A. Bagnall. Bake off redux: a review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery*, pages 1–74, 2024.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR*, 2013.

B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

T. L. Nguyen, S. Gsponer, and G. Ifrim. Time series classification by sequence learning in all-subsequence space. In *33rd IEEE International Conference on Data Engineering, ICDE*. IEEE Computer Society, 2017. doi: 10.1109/ICDE.2017.142.

T. L. Nguyen, S. Gsponer, I. Ilie, M. O'Reilly, and G. Ifrim. Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations. *Data Min. Knowl. Discov.*, 33(4):1183–1222, 2019. doi: 10.1007/s10618-019-00633-3.

A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *CoRR*, abs/2211.14730, 2022.

M. I. Owis, A. H. Abou-Zied, A.-B. Youssef, and Y. M. Kadah. Study of features based on nonlinear dynamical modeling in ecg arrhythmia detection and classification. *IEEE transactions on Biomedical Engineering*, 49(7):733–736, 2002.

C. Panigutti, R. Hamon, I. Hupont, D. Fernandez Llorca, D. Fano Yela, H. Junklewitz, S. Scalzo, G. Mazzini, I. Sanchez, J. Soler Garrido, et al. The role of explainable ai in the context of the ai act. In *Proceedings of the 2023 ACM conference on fairness, accountability, and transparency*, pages 1139–1150, 2023.

J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011. doi: 10.5555/1953048.2078195.

D. B. Percival and A. T. Walden. *Wavelet methods for time series analysis*, volume 4. Cambridge university press, 2000.

P. Piotrowski, D. Baczyński, M. Kopyt, and T. Gulczyński. Advanced ensemble methods using machine learning and deep learning for one-day-ahead forecasts of electric energy production in wind farms. *Energies*, 15(4):1252, 2022.

T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, pages 262–270. ACM, 2012. doi: 10.1145/2339530.2339576.

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006.

G. Riahy and M. Abedi. Short term wind speed forecasting for wind turbine applications using linear prediction method. *Renewable energy*, 33(1):35–41, 2008.

R. M. Rifkin and A. Klautau. In defense of one-vs-all classification. *J. Mach. Learn. Res.*, 5:101–141, 2004.

G. Rilling, P. Flandrin, P. Goncalves, et al. On empirical mode decomposition and its algorithms. In *IEEE-EURASIP workshop on nonlinear signal and image processing*, volume 3, pages 8–11, 2003.

R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

Y. Rubanova, R. T. Q. Chen, and D. Duvenaud. Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907, 2019.

C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.*, 1(5):206–215, 2019. doi: 10.1038/s42256-019-0048-x.

H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, 1984.

P. Schäfer. The BOSS is concerned with time series classification in the presence of noise. *Data Min. Knowl. Discov.*, 29(6):1505–1530, 2015. doi: 10.1007/s10618-014-0377-7.

P. Schäfer and M. Högqvist. SFA: a symbolic fourier approximation and index for similarity search in high dimensional datasets. In *15th International Conference on Extending Database Technology, EDBT*, pages 516–527. ACM, 2012. doi: 10.1145/2247596.2247656.

M. Schulz and K. Stattegger. Spectrum: Spectral analysis of unevenly spaced paleoclimatic time series. *Computers & Geosciences*, 23(9):929–945, 1997.

M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

P. Senin and S. Malinchik. SAX-VSM: interpretable time series classification using SAX and vector space model. In *2013 IEEE 13th International Conference on Data Mining*, pages 1175–1180. IEEE Computer Society, 2013. doi: 10.1109/ICDM.2013.52.

L. Serrano, L. Le Boudec, A. Kassaï Koupaï, T. X. Wang, Y. Yin, J.-N. Vittaut, and P. Gallinari. Operator learning with neural fields: Tackling pdes on general geometries. *Advances in Neural Information Processing Systems*, 36, 2024.

J. Shi, J. Guo, and S. Zheng. Evaluation of hybrid forecasting approaches for wind speed and power generation time series. *Renewable and Sustainable Energy Reviews*, 16(5):3471–3480, 2012.

S. N. Shukla and B. M. Marlin. Multi-time attention networks for irregularly sampled time series. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Statista Research Department. Europe: Electricity demand per capita 2022. https://www.statista.com/statistics/1262471/per-capita-electricity-consumption-europe/, 2022.

M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020.

X. Tang, H. Yao, Y. Sun, C. C. Aggarwal, P. Mitra, and S. Wang. Joint modeling of local and global temporal dynamics for multivariate time series forecasting with missing values. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI*, pages 5956–5963. AAAI Press, 2020.

Y. Tashiro, J. Song, Y. Song, and S. Ermon. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. *Advances in Neural Information Processing Systems*, 34:24804–24816, 2021.

R. Tavenard. An introduction to dynamic time warping. https://rtavenar.github.io/blog/dtw.html, 2021.

R. Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. Rußwurm, K. Kolar, et al. Tslearn, a machine learning toolkit for time series data. *Journal of machine learning research*, 21(118):1–6, 2020.

R. Tawn and J. Browell. A review of very short-term wind and solar power forecasting. *Renewable and Sustainable Energy Reviews*, 153:111758, 2022. ISSN 1364-0321. doi: https://doi.org/10.1016/j.rser.2021.111758. URL https://www.sciencedirect.com/science/article/pii/S1364032121010285.

S. J. Taylor and B. Letham. Forecasting at scale. *The American Statistician*, 72(1): 37–45, 2018.

S. Tonekaboni, D. Eytan, and A. Goldenberg. Unsupervised representation learning for time series with temporal neighborhood coding. *arXiv preprint arXiv:2106.00750*, 2021.

P. Trirat, Y. Shin, J. Kang, Y. Nam, J. Na, M. Bae, J. Kim, B. Kim, and J.-G. Lee. Universal time-series representation learning: A survey. *arXiv preprint arXiv:2401.03717*, 2024.

United Nations Convention on Climate Change. Paris Agreement. Climate Change Conference (COP21), Dec. 2015. URL https://unfccc.int/sites/default/files/english_paris_agreement.pdf.

A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, et al. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 12, 2016.

A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems 30*, pages 6306–6315, 2017.

L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008, 2017.

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

T. K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4 (1):52–57, 1968.

R. Wan, S. Mei, J. Wang, M. Liu, and F. Yang. Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting. *Electronics*, 8(8):876, 2019.

Y. Wang. *Interpretable time series classification. (Classification interprétable de séries temporelles)*. PhD thesis, University of Rennes 1, France, 2021.

Y. Wang, R. Emonet, É. Fromont, S. Malinowski, and R. Tavenard. Adversarial regularization for explainable-by-design time series classification. In *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI*, pages 1079–1087. IEEE, 2020. doi: 10.1109/ICTAI50040.2020.00165.

Y. Wang, R. Zou, F. Liu, L. Zhang, and Q. Liu. A review of wind speed and wind power forecasting with deep neural networks. *Applied Energy*, 304:117766, 2021. ISSN 0306-2619. doi: https://doi.org/10.1016/j.apenergy.2021.117766. URL https://www.sciencedirect.com/science/article/pii/S0306261921011053.

X. Weng and J. Shen. Classification of multivariate time series using two-dimensional singular value decomposition. *Knowledge-Based Systems*, 21:535–539, 2008.

G. Woo, C. Liu, D. Sahoo, A. Kumar, and S. C. H. Hoi. Deeptime: Deep time-index meta-learning for non-stationary time-series forecasting. *CoRR*, abs/2207.06046, 2022.

H. Wu, J. Xu, J. Wang, and M. Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 22419–22430, 2021.

M. Wu, M. C. Hughes, S. Parbhoo, M. Zazzi, V. Roth, and F. Doshi-Velez. Beyond sparsity: Tree regularization of deep models for interpretability. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence,*, pages 1670–1678. AAAI Press, 2018.

K. Yang and C. Shahabi. A pca-based similarity measure for multivariate time series. In *Proceedings of the 2nd ACM international workshop on Multimedia databases*, pages 65–74, 2004.

L. Ye and E. J. Keogh. Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *Data Min. Knowl. Discov.*, 22(1-2):149–182, 2011. doi: 10.1007/s10618-010-0179-5.

Y. Yin, M. Kirchmeyer, J.-Y. Franceschi, A. Rakotomamonjy, and P. Gallinari. Continuous pde dynamics forecasting with implicit neural representations. In *International Conference on Learning Representations, ICLR*, 2023.

J. Yoon, D. Jarrett, and M. Van der Schaar. Time-series generative adversarial networks. *Advances in neural information processing systems*, 32, 2019.

Z. Yue, Y. Wang, J. Duan, T. Yang, C. Huang, Y. Tong, and B. Xu. Ts2vec: Towards universal representation of time series. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI*, pages 8980–8987. AAAI Press, 2022.

A. Zeng, M. Chen, L. Zhang, and Q. Xu. Are transformers effective for time series forecasting? *CoRR*, abs/2205.13504, 2022.

G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff. A transformer-based framework for multivariate time series representation learning. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2114–2124. ACM, 2021. doi: 10.1145/3447548.3467401.

H. Zhang, T. B. Ho, Y. Zhang, and M.-S. Lin. Unsupervised feature extraction for time series clustering using orthogonal wavelet transform. *Informatica*, 30(3), 2006.

K. Zhang, Q. Wen, C. Zhang, R. Cai, M. Jin, Y. Liu, J. Y. Zhang, Y. Liang, G. Pang, D. Song, et al. Self-supervised learning for time series analysis: Taxonomy, progress, and prospects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

R. Zhang. Making convolutional networks shift-invariant again. In *International conference on machine learning*, pages 7324–7334. PMLR, 2019.

X. Zhang, Z. Zhao, T. Tsiligkaridis, and M. Zitnik. Self-supervised contrastive pre-training for time series via time-frequency consistency. *CoRR*, abs/2206.08496, 2022a. doi: 10.48550/arXiv.2206.08496.

X. Zhang, Z. Zhao, T. Tsiligkaridis, and M. Zitnik. Self-supervised contrastive pre-training for time series via time-frequency consistency. *Advances in Neural Information Processing Systems*, 35:3988–4003, 2022b.

X. Zheng, T. Wang, W. Cheng, A. Ma, H. Chen, M. Sha, and D. Luo. Parametric augmentation for time series contrastive learning. *arXiv preprint arXiv:2402.10434*, 2024.

H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, pages 11106–11115, 2021.

T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 27268–27286. PMLR, 2022.

L. Zimmer, M. Lindauer, and F. Hutter. Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl. *CoRR*, abs/2006.13799, 2020. URL https://arxiv.org/abs/2006.13799.

L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pages 7693–7702. PMLR, 2019.

Appendix A

WindDragon: Enhancing Wind Power Forecasting with Automated Deep Learning

Keisler, J., Le Naour, E. **WindDragon: Enhancing Wind Power Forecasting with Automated Deep Learning.** *International Conference on Learning Representations (ICLR), Tackling Climate Change with Machine Learning Workshop*, 2024.

Achieving net zero carbon emissions by 2050 requires the integration of increasing amounts of wind power into power grids. This energy source poses a challenge to system operators due to its variability and uncertainty. Therefore, accurate forecasting of wind power is critical for grid operation and system balancing. This paper presents an innovative approach to short-term (1 to 6 hour horizon) wind power forecasting at a national level. The method leverages Automated Deep Learning combined with Numerical Weather Predictions wind speed maps to accurately forecast wind power.

A.1 Introduction

To meet the 2050 net zero scenario envisaged by the Paris Agreement (United Nations Convention on Climate Change, 2015), wind power stands out as a critical energy source for the future. Remarkable progress has been made since 2010, when global electricity generation from wind power was 342 TWh, rising to 2,100 TWh in 2022 (International Energy Agency (IEA), 2023). The IEA targets approximately 7,400 TWh of wind-generated electricity by 2030 to meet the zero-emissions scenario. However, to realize the full potential of this intermittent energy source, accurate forecasts of wind power generation are needed to efficiently integrate it into the power grid.

Research in wind power forecasting has developed a wide range of methods (Giebel and Kariniotakis, 2017; Tawn and Browell, 2022), including statistical (Riahy and Abedi, 2008), physical (Lange and Focken, 2006), hybrid (Shi et al., 2012), and deep learning (DL) (Wang et al., 2021) approaches. These methods use a variety of data sources, including historical wind power records, geospatial satellite data, on-site camera imagery, and numerical weather prediction (NWP) forecasts. Among these, typical NWP-based methods primarily focus on using local time series of wind speed forecasts for local wind power prediction (Piotrowski et al., 2022). However, NWP forecasts produce richer outputs, notably spatial predictions of physical quantities such as wind speed and direction over large scale grids (e.g. national or regional). Predicting aggregated (e.g national or regional) wind power from such fine-grained spatial information appears promising and is largely unexplored in the literature (Higashiyama et al., 2018). Thus, we propose to explore how wind speed maps combined with suitable machine learning models can capture complex patterns, improving large scale wind power predictions.

In this work, we propose to leverage the spatial information in NWP wind speed maps for national wind power forecasting by exploiting the capabilities of DL models. The overall methodology is illustrated in Figure A.1. To fully exploit DL mechanisms potential, we introduce WindDragon, an adaptation of the DRAGON[1] (Keisler et al., 2023) framework. WindDragon is an Automated Deep Learning (AutoDL) framework for short-term wind power forecasting using NWP wind speed maps. WindDragon's performances are benchmarked against conventional computer vision models, such as Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs), as well as standard baselines in wind power forecasting. The experimental results highlight two findings:

- The use of full NWP wind speed maps coupled with DL regressors significantly outperforms other baselines.

[1] https://dragon-tutorial.readthedocs.io/en/latest/index.html

- WindDragon demonstrates superior performance compared to traditional computer vision DL models.

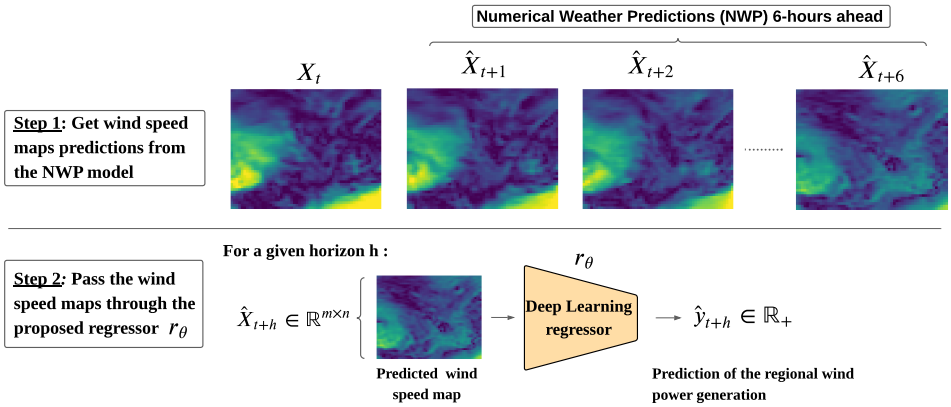


Figure A.1: Global scheme for wind power forecasting. Every 6 hours, the NWP model produces hourly forecasts. Each map is processed independently by the regressor which maps the grid to the wind power corresponding to the same timestamp.

A.2 WindDragon: a framework for regression on wind speed maps

Deep Learning models have the ability to capture complex spatial patterns, which makes them well suited for modeling non-linear relationships between meteorological features and wind energy production. These models are especially useful when wind farms are scattered across the map (see Figure A.3) and wind speed has significant variance across locations.

CNNs and ViTs, both prominent in computer vision, might under-perform in the context of wind speed map regression for global wind power forecasting. By learning local and spatial patterns, CNNs efficiently map structured inputs to numerical values. However, CNN's shift-invariant property (Zhang, 2019) can hinder wind power forecasting because identical wind speeds at different map locations do not equate to the same power generation due to the uneven distribution of wind farms. Conversely, ViTs excel at image classification by segmenting images into patches and applying self-attention mechanisms, but the size of the considered datasets (less than 20000 points for the training dataset) might limit their effectiveness. Given these concerns, the use of AutoDL frameworks to automatically identify the most appropriate DL architecture is a promising solution.

The DRAGON framework. DRAGON (Keisler et al., 2023) is an AutoDL framework which automatically generates well-performing deep learning models for a given task. Compare to other AutoDL frameworks (Liu et al., 2019a; Hutter et al., 2019; Zimmer et al., 2020; Deng et al., 2022), DRAGON provides a flexible search space, which can be used on any task. It allows the extension of the possibilities in terms of architectures and is adapted when the type of architecture to use is unclear or when high performance is sought by tuning hyperparameters. We used several tools from the generic framework to adapt it for wind power forecasting from wind speed maps.

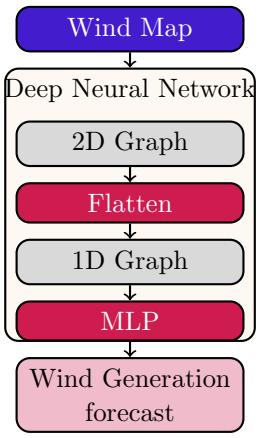


Figure A.2: WindDragon's meta model for wind power forecasting.

WindDragon: adapting the DRAGON framework for wind power forecasting. The neural networks in DRAGON are represented as directed acyclic graphs, with nodes representing the layers and edges representing the connections between them. In our case, a value $\hat{y}_t \in \mathbb{R}$ is predicted from a 2D map $X_t \in \mathbb{R}^{m \times n}$. The search space is then restricted to a specific family of constrained architectures, as represented in Figure A.2. A first graph processes 2D data and can be composed by convolutions, pooling, normalization, dropout, and attention layers. Then, a flatten layer and a second graph follow. This one is composed by MLPs, self-attention, convolutions and pooling layers. A final MLP layer is added at the end of the model to convert the latent vector to the desired output format. We optimized the solutions from our search space using an evolutionary algorithm, as detailed Appendix A.5.1.

A.3 Experiments

Datasets. The wind speed maps used are 100-meter high forecasts at a 9 km resolution provided by the HRES [2] model from the European Centre for Medium-Range Weather Forecasts (ECMWF). The maps are provided at an hourly time step and there are 4 forecast runs per day (every 6 hours). Only the six more recent forecasts are used here as the forecasting horizon of interest is six hours. The hourly french regional and national wind power generation data came from the french TSO[3].

Data preparation. The national forecast of wind power generation is obtained by summing the forecasts of the 12 administrative regions of Metropolitan France. According to our first experiments, this bottom-up technique produced better results

[2] https://www.ecmwf.int/en/forecasts/datasets/set-i
[3] https://www.rte-france.com/eco2mix

than predicting national production directly. The division of a national map into regions is a challenge, as shown in Figure A.3 as wind turbines are not evenly distributed across the regions. Therefore, we selected areas around each wind farm in the region and took the convex hull of all the considered points. The result is a seamless map that includes local wind turbines with no gaps to disrupt the models. Installed capacity data for each region - corresponding to the maximum wind power a region can produce - is available and updated every three months. It was collected and used to scale the wind power target. Years from 2018 to 2019 are used to train the models, and data from 2020 is used to evaluate how the models perform.

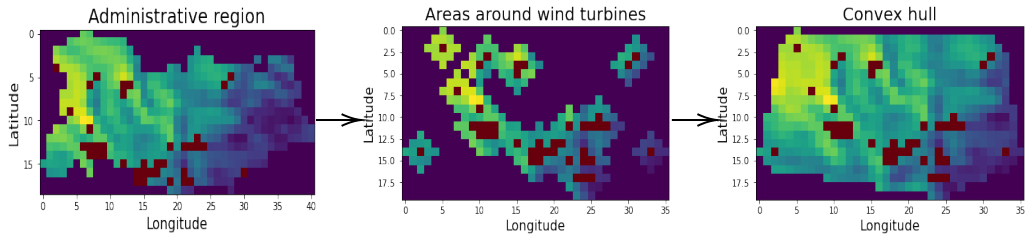


Figure A.3: Data preparation for the region Auvergne-Rhône-Alpes. The wind farms are represented in red. The first image shows the distribution of wind farms across the administrative region.

We use the following baselines to compare hourly forecasts for an horizon h ($h \in \{1, ..., 6\}$):

- **Persistence.** Predicts wind power generation at future time $t+h$ as equal to the observed generation at current time t.
- **XGB on Wind Speed Mean.** Forecasts wind power at $t+h$ using a two-step approach: (i) Compute the mean wind speed for the considered region at $t+h$ using NWP forecasts. (ii) Apply an XGBoost regressor (Chen and Guestrin, 2016) to predict power generation based on the computed mean wind speed.
- **Convolutional Neural Networks (CNNs).** Forecasts wind power at $t+h$ using the NWP predicted wind speed map. CNNs can efficiently regress a structured map on a numerical value by learning local and spatial patterns (LeCun et al., 1995).
- **Vision Transformers (ViTs).** Forecasts wind power at $t+h$ using the NWP predicted wind speed map. The map is segmented into patches and a self-attention mechanism is used to capture the dependencies between these patches (Dosovitskiy et al., 2020).
- **Implicit Neural Representation (INR) + XGB.** Forecasts wind power at $t+h$ using a two-step approach: (i) Compute a representation $\boldsymbol{z}^{(j)}$ for

wind speed map $\boldsymbol{X}^{(j)}$ for the considered region at $t+h$ using NWP forecasts. (ii) Apply an XGBoost regressor (Chen and Guestrin, 2016) to predict power generation based on the representation code $\boldsymbol{z}^{(j)}$. See Appendix A.5.2 for more details on this baseline.

The implementation details of the baselines are described in Appendix A.5.2.

We compute two scores: **Mean Absolute Error (MAE)** in Megawatts (MW), showing the absolute difference between ground truth and forecast, and **Normalized Mean Absolute Error (NMAE)**, a percentage obtained by dividing the MAE by the average wind power generation for the test year.

Results. We run experiments for each of the 12 French metropolitan regions and then aggregate the predictions to derive national results. The national prediction results are presented in Table A.1, while detailed regional results can be found in Table A.2 (Appendix A.5.3).

Table A.1: National results: sum of the regional forecasts for each models. The best results are highlighted in bold and the best second results are underlined.

	WindDragon		INR + XGB		CNN		ViT		XGB on mean		Persistence	
	MAE (MW)	NMAE	MAE (MW)	NMAE	MAE (MW)	NMAE	MAE (MW)	NMAE	MAE (MW)	NMAE	MAE (MW)	NMAE
France	**346.7**	**7.7 %**	<u>358.1</u>	<u>7.9 %</u>	369.0	<u>8.1 %</u>	385.7	8.5 %	416.7	9.2 %	779.7	17.3 %

The results in Table A.1 highlight three key findings:

(i) **Improved performance with aggregated NWP statistics.** Using the average of NWP-predicted wind speed maps coupled with an XGB regressor significantly outperforms the naive persistence baseline.

(ii) **Gains from full NWP map utilization**. More complex patterns can be captured by using the full predicted wind speed map, as opposed to just the average, thereby improving forecast accuracy. In this context, the ViT, the CNN and the INR based regressors applied to full maps yielded gains of 31 MW (7.4%), 47 MW (12.8%) and 58 MW (16.2%) respectively, over the mean-based XGB.

(iii) **WindDragon's superior performances**. WindDragon outperforms all baselines, showing an improvement of 22 MW (6%) over the CNN (the second best fully supervised method). On an annual basis, this corresponds to approximately 193 GWh, which is equivalent to the annual consumption of a French town of 32,000 inhabitants [4]. Refer to Appendix A.5.1 for WindDragon's architecture example.

[4]based on the average European per capita consumption (Statista Research Department, 2022)

In Figure A.4, we present the aggregated national wind power forecasts using both WindDragon and the CNN baseline during a given week. While both models deliver highly accurate forecasts, it's important to highlight that DRAGON demonstrates superior accuracy, particularly in predicting high peak values.

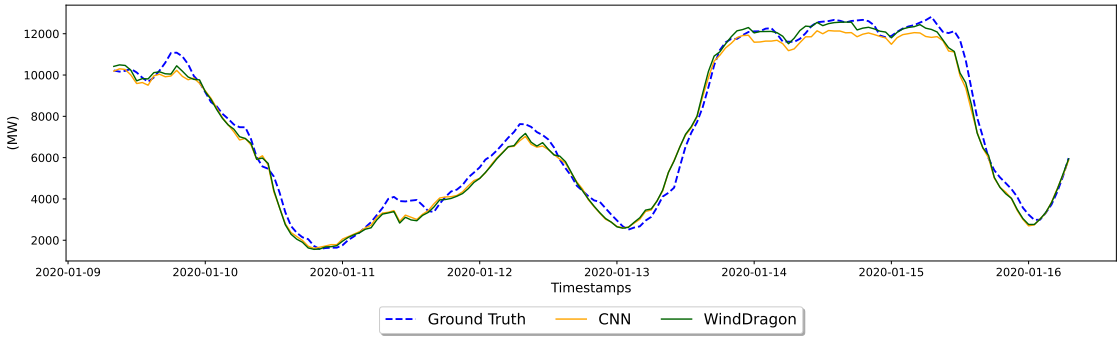


Figure A.4: Wind power forecasts for a week in January 2020. The figure displays the ground truth as dotted lines, and the forecasts from the two top-performing models, WindDragon and the CNN.

A.4 Conclusion and impact statement

In this paper, we have presented two key findings that show great promise. First, using NWP wind speed forecasts as a map significantly improves forecast accuracy compared to using only aggregated values. Second, our framework, WindDragon, shows superior performance to all other baseline models. The significant improvement provided by WindDragon is particularly critical in light of the increasing reliance on wind energy, driven by the pursuit of the net-zero scenario.

Future work could adapt our methodology for photovoltaic (PV) systems, applying it to solar radiation maps generated by NWP models. While current deep learning research in PV primarily focuses on short-term nowcasting (Le Guen, 2022), our method holds promise for extending the forecasting horizon, potentially improving the efficiency and reliability of solar power predictions.

A.5 Additional results

A.5.1 WindDragon details

Search algorithm. The DRAGON framework contains operators, namely mutation and crossover, which are commonly used in meta-heuristics such as the evolutionary algorithm and the simulated annealing, to optimize graphs. The mutation operators are used to add, remove, or modify nodes and connections in the graph, as well as to modify the operations and their hyperparameters within the nodes. Crossover involves exchanging parts of two graphs. The mutation and crossover operators were utilised to construct a steady-state (asynchronous) evolutionary algorithm. Compare to the original algorithm, this version enhances efficiency on HPC by producing two offsprings from the population as soon as a free process is available, rather than waiting for the entire population to be evaluated (Liu et al., 2018).

With the division by region, we slightly modified the generic evolutionary algorithm in WindDragon to avoid having to run an optimisation by region, which would be very costly. In this context, a deep neural network f from our search space Ω is parametrized by its architecture α and its hyperparemeters λ. Once α and λ have been settled, the model is trained on the data to optimize the weights θ. We assumed that the architecture α and the hyperparameters λ would be broadly similar across regions. Therefore, we modified our evolutionary algorithm to process all regions at the same time. We create and evolve α and λ independently of the region, and, to optimize the weights θ, we randomly select the region on which the model would be train and evaluate. In order not to penalize models that have been evaluated on regions that are difficult to predict, we use a global loss function, which consists in dividing the loss obtained on the region ℓ_{region} by the loss of our baseline CNN model on that region, $\mathcal{L}_{\text{region}}$. During the optimisation, for each region, we progressively save the best model evaluated on it.

Results. The outputs of WindDragon would be by region the best model found during the optimisation and the prediction of this model. The found architectures vary a bit from a region to another. An example of the best model for the region Grand Est can be found Figure A.5. This architecture uses self-attention just like in the Transformer (Vaswani et al., 2017), but without the patches that can be found in the ViT architecture. The model is also a lot smaller than a Transformer, which can explain why it outperforms the other baselines on this region

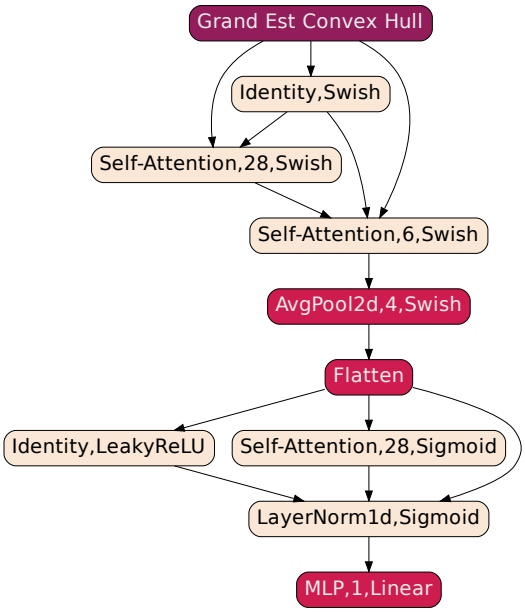


Figure A.5: Dragon automatically found architecture applied on the Grand Est region.

A.5.2 Baselines details

The baselines used in Appendix A.3 are explained in more detail below.

Convolutional Neural Network (CNN). Figure A.6 shows the architecture of the CNN baseline that we implemented. We used a simple grid search to optimize the hyperparameters (e.g. the number of layers, the kernel sizes, the activation functions)

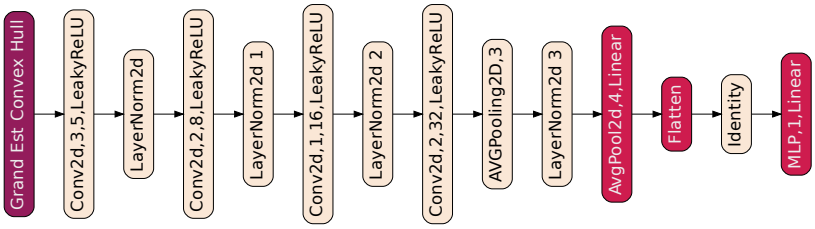


Figure A.6: CNN architecture applied on the Grand Est region.

Vision Transformer (ViT). The Vision Transformer used in this paper is based on SimpleViT's (Beyer et al., 2022) architecture. We reused the implementation

from lucidrains package [5].

XGboost on the mean of the NWP wind speed map. Figure A.7 shows the two-steps procedure of the XGboost baseline.

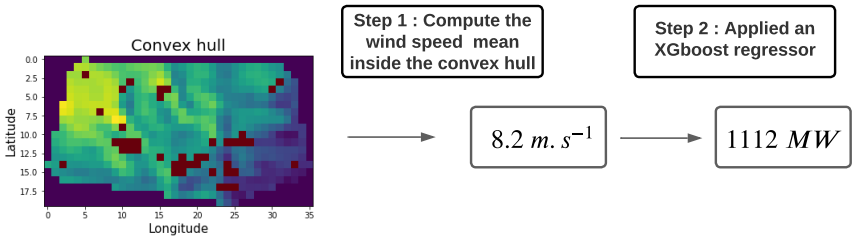


Figure A.7: Visual illustration of the XGB two-steps approach on the Auvergne-Rhône-Alpes region.

XGBoost on the INR's representation of the NWP Wind Speed Map This baseline follows a two-step procedure:

(i) Fit the wind speed map using a modulated INR (Dupont et al., 2022; Serrano et al., 2024) and obtain the associated vector representation (for more details on fitting a modulated INR, please refer to Chapter 4).

(ii) Fit an XGBoost model on top of the learned representations.

A.5.3 Regional results

Table A.2: Regional results. The best results are highlighted in bold and the second best results are underlined.

	WindDragon		INR + XGB		CNN		ViT		XGB on mean		Persistence	
Region	MAE (MW)	NMAE	MAE (MW)	NMAE	MAE (MW)	NMAE	MAE (MW)	NMAE	MAE (MW)	NMAE	MAE (MW)	NMAE
Auvergne-Rhône-Alpes	**19.5**	**14.9 %**	20.2	15.5 %	19.6	15.0 %	21.6	16.5 %	29.2	22.4 %	28.7	22.0 %
Bourgogne-Franche-Comté	**32.9**	**14.8 %**	33.8	15.2 %	34.1	15.4 %	37.2	16.8 %	42.3	19.1 %	58.7	26.6 %
Bretagne	**36.1**	**14.1** %	36.4	14.2 %	38.0	14.9 %	39.9	15.6 %	47.1	18.4 %	67.2	26.3 %
Centre-Val de Loire	**53.3**	**44.0 %**	55.7	15.7 %	57.3	16.1 %	59.0	16.6 %	61.9	17.5 %	96.7	27.3 %
Grand Est	**125.6**	**12.5 %**	127.9	12.8%	130.5	13.1 %	161.0	16.1 %	148.8	14.9 %	251.2	25.1 %
Hauts-de-France	**159.7**	**12.1 %**	163.6	12.3 %	167.6	12.7 %	177.0	13.4 %	178.8	13.5 %	320.1	24.2 %
Île-de-France	**6.8**	**22.6 %**	7.4	24.8 %	7.2	23.7 %	7.4	24.3 %	7.5	24.9 %	9.5	31.5 %
Normandie	**29.6**	**12.7 %**	29.8	12.8 %	30.8	13.2 %	31.2	13.4 %	36.8	15.8 %	55.9	24.0 %
Nouvelle-Aquitaine	**43.1**	**15.7 %**	44.9	16.3 %	44.0	16.4 %	48.4	17.6 %	53.7	19.6 %	77.9	28.4 %
Occitanie	**51.2**	**12.3 %**	55.4	13.3 %	55.8	13.5 %	64.1	15.5 %	91.6	22.1 %	96.3	23.2 %
PACA	**3.5**	**32.4 %**	3.6	33.9 %	**3.5**	**32.4** %	4.0	37.2 %	4.5	41.4 %	4.3	39.5 %
Pays de la Loire	**37.1**	**13.6 %**	38.1	14 %	39.0	14.3 %	39.9	14.7 %	41.9	15.4 %	74.9	27.5 %

[5] https://github.com/lucidrains/vit-pytorch

Appendix B

Appendix of Chapter 3

B.1 Reproductiblity statement

Our work is entirely reproducible.

Code. The code for all our experiments is available at this link.

Data. The data are available at this link.

GPU. We used NVIDIA TITAN RTX 24Go single GPU to conduct all the experiments for our method, which is coded in PyTorch (Python 3.9.2).

B.2 How to compute receptive fields regions

With our model, it is very useful to be able to calculate the receptive fields and in particular the receptive field regions relative to an element (or a region) of the representation. To do this, we just need to adapt the following formulas computed in Araujo et al. (2019) to our architecture:

$$v_0 = v_L \prod_{i=1}^{L} s_i - \sum_{l=1}^{L} (1 + p_l - k_l) \prod_{i=1}^{l-1} s_i$$

$$u_0 = u_L \prod_{i=1}^{L} s_i - \sum_{l=1}^{L} p_l \prod_{i=1}^{l-1} s_i$$

Where:

- v_0 stands for the left-most coordinates of the receptive field in the intial time series

- u_0 stands for the right-most coordinates of the receptive field in the intial time series
- v_L stands for the left-most coordinates in the representation
- u_L stands for the left-right coordinates in the representation
- k_l stands for the kernel size at depth l
- s_l stands for the stride at depth l
- p_l stands for the padding at depth l
- L stands for the depth of the network

Thanks to these operations, we can highlight the receptive fields of the elements of the representation in Figure 3.12 and Figure 3.10.

B.3 Impact on accuracy results of the number of available centroids

The quantitive experiments in Subsection 3.6.1 were performed for 32 centroids available during the vector quantization (K=32). It is interesting to see how the number of centroids affects the classification performance. Table B.1 shows the accuracy results averaged over the 25 UCR datasets on which the experiments were conducted.

Table B.1: Mean accuracy on the previous 25 UCR datasets for different k ($k \in \{8, 16, 32\}$). The best result are in bold and the second best result are underlined.

	$k = 8$	$k = 16$	$k = 32$
Mean accuracy	0.753	<u>0.778</u>	**0.793**

We observed that increasing the number of available centroids improves accuracy. However, it harms the centroids' expressiveness and, thus, the interpretability of the representation.

B.4 Proof of the shift equivariance property

In this appendix, we propose to describe how the adaptive polyphase downsampling/upsampling mechanism preserves the shift equivariance property despite downsampling (or upsampling). To do so, we pick up the proof given in Chaman and Dokmanic (2021). We first present the proof of shift equivariance for the adaptive polyphase downsampling mechanism and then the adaptive polyphase upsampling case is straightforward.

Definition: We define the shift equivariance property as follows. Let S be an element of a sequence (a single element or a subsequence), and G_T the group of discrete translations along the temporal axis. If we take τ to be any discrete translation in G_T and f to be a function equivariant by discrete translation for G_T, then there exists $\tau' \in G_{T'}$ such that: $f(\tau(S)) = \tau'(f(S))$.

Case 1: Adapative polyphase downsampling (APS-D) for downsampling by half : To explain the APS-D operation, let us introduce $s(t)$ the convolutions output sequence of length T_0. We can then define the two sub-sequences s_0 and s_1 such that $s_0(t) = s(2t)$ and $s_1(t) = s(2t+1)$. The APS-D operation D_2^A consists in sub-sampling the sub-sequence s_l such that:

$$D_2^A(s) = s_l \quad \text{where} \quad l = \arg\max ||s_j||_1 \text{ (for } j \in \{0, 1\})$$

We introduce the following notations:

- τ_k is a discrete translation by k
- $s_{APS} = D_2^A(s)$ is the non shift sequence output
- $s_{APS}^{(k)} = D_2^A(\tau_k(s))$ is the shifted sequence output

Then we have:

$$s_{APS}^{(k)} = \begin{cases} \tau_{\frac{k}{2}}(s_{APS}), \text{when } k \text{ is even} \\ \tau_{\frac{k+2i-1}{2}}(s_{APS}), \text{when } k \text{ is odd} \end{cases}$$

Case 2: Adapative polyphase usampling (APS-U) for upsampling by two: For the APS-U, the s signal is upsampled by two. We insert 0 elements in each even or odd position according to the phase chosen in the block of the corresponding encoder. By construction, this operation is shift equivariant. For more details on this operation, see the original paper (Chaman and Dokmanic, 2021).

Appendix C

Appendix of Chapter 4

C.1 Reproductiblity statement

Our work is entirely reproducible, and all the references to the information in order to reproduce it are in this section.

Code. The code for all our experiments is available at this link.

Data. A subset of the processed data is available with the code at this link.

Model. The model and the training details are presented in Appendix A.2 and the hyperparameter selection is available in Appendix C.2.1.

GPU. We used NVIDIA TITAN RTX 24Go single GPU to conduct all the experiments for our method, which is coded in PyTorch (Python 3.9.2).

C.2 Architecture details and ablation studies

C.2.1 Architecture details

For all imputation and forecasting experiments we choose the following hyperparameters :

- $\mathbf{z}$ dimension: 128
- Number of layers: 5
- Hidden layers dimension: 256
- $\gamma(\mathbf{t}) \in \mathbb{R}^{2\times 64}$
- z code learning rate (α in Algorithm 1): 10^{-2}

- Hypernetwork and INR learning rate: 5×10^{-4}
- Number of steps in inner loop: $K = 3$
- Number of epochs: 4×10^4
- Batch size: 64

It is worth noting that the hyperparameters mentioned above remain consistent across all experiments conducted in the paper. We chose to maintain a fixed set of hyperparameters for our model, while other imputation and forecasting approaches commonly fine-tune hyperparameters based on a validation dataset. The obtained results exhibit high robustness across various settings, suggesting that the selected hyperparameters are already effective in achieving reliable outcomes.

C.2.2 Ablation studies

C.2.2.1 Fourier features vs SIREN on imputation task

Baseline. The SIREN network differs from the Fourier features network because it does not explicitly incorporate frequencies as input. Instead, it is a multi-layer perceptron network that utilizes sine activation functions. An adjustable parameter, denoted ω_0, is multiplied with the input matrices of the preceding layers to capture a broader range of frequencies. For this comparison, we adopt the same hyperparameters described in Appendix C.2.1, selecting $\omega_0 = 30$ to align with Sitzmann et al. (2020). Furthermore, we set the learning rate of both the hypernetwork and the INR to 5×10^{-5} to enhance training stability. In Table C.1, we compare the imputation results obtained by the Fourier features network and the SIREN network, specifically focusing on the first time window from the *Electricity*, *Traffic* and *Solar* datasets.

Table C.1: MAE imputation errors on the first time window of each dataset. Best results are bold.

	τ	TimeFlow	TimeFlow w SIREN
Electricity	0.05	**0.323**	0.466
	0.10	**0.252**	0.350
	0.20	**0.224**	0.242
	0.30	**0.211**	0.222
	0.50	**0.194**	0.209
Solar	0.05	**0.105**	0.114
	0.10	**0.083**	0.094
	0.20	**0.065**	0.079
	0.30	**0.061**	0.072
	0.50	**0.056**	0.066
Traffic	0.05	**0.292**	0.333
	0.10	**0.220**	0.252
	0.20	**0.168**	0.191
	0.30	**0.152**	0.163
	0.50	**0.141**	0.154

Results According to the results presented in Table C.1, the Fourier features network outperforms the SIREN network in the imputation task on these datasets. Notably, the performance gap between the two network architectures are more pronounced at low sampling rates. This disparity can be attributed to the SIREN network's difficulty in accurately capturing high frequencies when the time series is sparsely observed. We hypothesize that the MLP with ReLU activations correctly learns the different frequencies of time series with multi-temporal patterns by switching on or off the Fourier embedding frequencies.

C.2.2.2 Influence of the latent code dimension

The dimension of the latent code $\mathbf{z}$ is a crucial parameter in our architecture. If it is too small, it underfits the time series. Consequently, this adversely affects the performance of both the imputation and forecasting tasks. On the other hand, if the dimension of z is too large, it can lead to overfitting, hindering the model's ability to generalize to new data points.

Baselines To investigate the impact of z dimensionality on the performance of TimeFlow, we conducted experiments on the three considered datasets, specifically focusing on the forecasting task. We varied the sizes of $\mathbf{z}$ within $\{32, 64, 128, 256\}$. The other hyperparameters are set as presented in Appendix C.2.1. The obtained results for each $\mathbf{z}$ dimension are summarized in Table C.2.

Results The results presented in Table C.2 suggest that a $\mathbf{z}$ dimension of 128 is a reasonable compromise but only optimal for some settings. Moreover, even though

Table C.2: MAE error for different z dimension.

	H	32	64	128	256
Electricity	96	0.232 ± 0.016	0.222 ± 0.017	0.222 ± 0.018	**0.215 ± 0.019**
	192	0.245 ± 0.020	0.239 ± 0.018	**0.230 ± 0.026**	0.233 ± 0.017
	336	0.254 ± 0.029	0.244 ± 0.028	0.262 ± 0.031	**0.243 ± 0.032**
	720	0.295 ± 0.027	0.284 ± 0.028	0.303 ± 0.041	**0.283 ± 0.029**
SolarH	96	0.182 ± 0.009	0.181 ± 0.012	**0.179 ± 0.003**	0.225 ± 0.047
	192	0.195 ± 0.014	0.195 ± 0.016	**0.193 ± 0.015**	0.197 ± 0.029
	336	**0.181 ± 0.011**	0.182 ± 0.011	0.189 ± 0.013	0.183 ± 0.012
	720	0.201 ± 0.027	**0.199 ± 0.025**	0.209 ± 0.029	0.200 ± 0.030
Traffic	96	0.223 ± 0.024	0.215 ± 0.028	0.215 ± 0.037	**0.210 ± 0.033**
	192	0.214 ± 0.018	0.217 ± 0.025	0.206 ± 0.023	**0.203 ± 0.024**
	336	0.238 ± 0.029	0.231 ± 0.029	**0.226 ± 0.030**	0.229 ± 0.029
	720	0.272 ± 0.040	0.269 ± 0.035	**0.259 ± 0.038**	0.262 ± 0.040

the choice of **z** dimension seems important, it doesn't critically impact the MAE error for the forecasting task.

C.2.2.3 Influence of the number of gradient steps

As can be seen in Table C.3, using three gradient steps at inference yield an inference time of less than 0.2 seconds. The latter can still be reduced by doing only one step at the cost of an increase in the forecasting error. As observed in Table C.3, increasing the number of gradient steps above 3 steps during inference does not improve forecasting performance.

Table C.3: Inference time (in seconds) and MAE error on the forecasting task on the *Electricity* dataset for a horizon of length 720, a look-back window of length 512, and a varying number of adaptation gradient steps. The statistics are computed over 10 runs using an NVIDIA TITAN RTX GPU.

Gradient descent steps	1	3	10	50	500	5000
Inference time (s)	0.109 ± 0.003	0.176 ± 0.009	0.427 ± 0.031	3.547 ± 0.135	17.722 ± 0.536	189.487 ± 8.060
MAE	0.351 ± 0.038	0.303 ± 0.041	0.300 ± 0.040	0.299 ± 0.039	0.302 ± 0.038	0.308 ± 0.037

Results We conduct more extensive experiments in Table C.4, Table C.5, Table C.6 to quantify the MAE score variation according to different number of gradient steps during training and inference. The tables show that using the same number of steps in training and inference leads to better results. This is expected since using different gradient steps for training and inference makes the inference model slightly different from the training model. In addition, using 3 gradient steps instead of 1 clearly improves the performances, but using 10 instead of 3 does not. Indeed, it usually leads to overall better results for longer horizon, but the gain is not clear for smaller horizons. Hence using 3 gradient steps is a suitable choice.

Table C.4: MAE error on the forecasting task using 1 inner-step during training and a varying number of adaptation gradient steps at inference. Best results are in bold and / symbol means that the MAE score is very high (≥ 1).

	H	1	3	10	50
Electricity	96	**0.244 ± 0.017**	0.246 ± 0.017	0.261 ± 0.016	/
	192	**0.253 ± 0.024**	**0.253 ± 0.022**	0.261 ± 0.020	0.265 ± 0.019
	336	**0.267 ± 0.032**	0.268 ± 0.030	0.277 ± 0.028	0.281 ± 0.027
	720	0.302 ± 0.030	0.306 ± 0.029	0.310 ± 0.028	**0.301 ± 0.029**
SolarH	96	**0.192 ± 0.023**	0.623 ± 0.397	/	/
	192	**0.175 ± 0.006**	0.252 ± 0.068	/	/
	336	**0.192 ± 0.016**	0.471 ± 0.029	/	/
	720	**0.216 ± 0.034**	0.465 ± 0.063	/	0.550 ± 0.187
Traffic	96	**0.215 ± 0.029**	0.329 ± 0.039	/	/
	192	**0.208 ± 0.019**	0.310 ± 0.033	0.312 ± 0.032	/
	336	**0.237 ± 0.028**	0.307 ± 0.038	/	/
	720	**0.263 ± 0.038**	0.320 ± 0.040	/	/

Table C.5: MAE error on the forecasting task using 10 inner-steps during training and a varying number of adaptation gradient steps at inference. Best results are in bold.

	H	1	3	10	50
Electricity	96	0.259 ± 0.020	**0.222 ± 0.018**	**0.222 ± 0.017**	0.228 ± 0.019
	192	0.269 ± 0.020	**0.230 ± 0.026**	0.232 ± 0.020	0.233 ± 0.026
	336	0.273 ± 0.033	**0.262 ± 0.031**	0.264 ± 0.032	0.268 ± 0.032
	720	0.351 ± 0.038	0.303 ± 0.041	0.300 ± 0.040	**0.299 ± 0.039**
SolarH	96	0.487 ± 0.196	**0.179 ± 0.003**	0.181 ± 0.003	0.186 ± 0.003
	192	0.411 ± 0.088	**0.193 ± 0.015**	0.195 ± 0.014	0.199 ± 0.013
	336	0.435 ± 0.153	**0.189 ± 0.013**	0.203 ± 0.006	0.223 ± 0.012
	720	0.394 ± 0.173	0.209 ± 0.029	**0.203 ± 0.006**	0.209 ± 0.027
Traffic	96	0.320 ± 0.038	**0.215 ± 0.037**	0.219 ± 0.043	0.226 ± 0.046
	192	0.299 ± 0.023	**0.206 ± 0.023**	0.209 ± 0.026	0.214 ± 0.027
	336	0.345 ± 0.038	**0.226 ± 0.030**	0.228 ± 0.031	0.233 ± 0.032
	720	0.321 ± 0.034	**0.259 ± 0.038**	0.260 ± 0.038	0.266 ± 0.039

Table C.6: MAE error on the forecasting task using 10 inner-steps during training and a varying number of adaptation gradient steps at inference. Best results are in bold.

	H	1	3	10	50
Electricity	96	0.381 ± 0.030	0.249 ± 0.024	**0.236 ± 0.024**	0.238 ± 0.024
	192	0.448 ± 0.045	0.273 ± 0.019	**0.244 ± 0.014**	**0.244 ± 0.013**
	336	0.514 ± 0.053	0.283 ± 0.033	**0.241 ± 0.025**	0.242 ± 0.024
	720	0.647 ± 0.068	0.400 ± 0.051	**0.286 ± 0.023**	0.287 ± 0.021
SolarH	96	0.605 ± 0.029	0.380 ± 0.018	**0.188 ± 0.012**	0.199 ± 0.015
	192	0.382 ± 0.072	0.250 ± 0.012	**0.202 ± 0.034**	0.204 ± 0.035
	336	0.745 ± 0.105	0.431 ± 0.221	**0.201 ± 0.033**	0.208 ± 0.032
	720	0.745 ± 0.082	0.477 ± 0.039	**0.205 ± 0.030**	**0.205 ± 0.029**
Traffic	96	0.450 ± 0.023	0.273 ± 0.026	**0.225 ± 0.028**	0.230 ± 0.034
	192	0.506 ± 0.028	0.318 ± 0.021	**0.233 ± 0.022**	0.236 ± 0.026
	336	0.500 ± 0.042	0.320 ± 0.021	**0.247 ± 0.028**	0.249 ± 0.031
	720	0.511 ± 0.035	0.323 ± 0.022	**0.266 ± 0.027**	0.272 ± 0.024

C.2.2.4 TimeFlow variants with other meta-learning techniques

Baselines Before converging to the current architecture and optimization of TimeFlow, we explored different options to condition the INR with the observations. The first one was inspired by the neural process architecture, which uses a set encoder to transform a set of observations $(t_i, x_{t_i})_{i \in \mathcal{I}}$ into a latent code $\mathbf{z}$ by applying a pooling layer after a feed forward network. We observed that this encoder in combination with the modulated fourier features network was able to achieve relatively good results on the forecasting task but suffered of underfitting on more complex datasets such as *Electricity*.

This led us to consider auto-decoding methods instead, i.e. encoder-less, architectures for conditioning the weights of the coordinate-based network. We trained TimeFlow with the REPTILE algorithm (Nichol et al., 2018), which is a first-order meta-learning technique that adapts the code in a few steps of gradient descent. In contrast with a second-order method, we observed that REPTILE was less costly to train but struggled to escape sub optimal minima, which led to unstable training and underfitting.

From an implementation point of view, the only difference between second order and first order, is that in the latter the code is detached from the computation graph before taking the outer-loop parameter update. When the code is not detached, it remains a function of the common parameters $\mathbf{z} = z(\theta, \mathbf{w})$, which means that the computation graph for the outer-loop also includes the inner-loop updates to the codes. Therefore the outer-loop gradient update involves a gradient through a gradient and requires an additional backward pass through the INR to compute the Hessian. Please refer to Finn et al. (2017) for more technical details.

Table C.7: Comparison of second-order and first-order (REPTILE) meta learning for TimeFlow on the imputation task. Mean MAE results on the missing grid over five different time windows. τ stands for the subsampling rate. Bold results are best.

	τ	TimeFlow	TimeFlow w REPTILE
Electricity	0.05	**0.324 ± 0.013**	0.363 ± 0.062
	0.10	**0.250 ± 0.010**	0.343 ± 0.036
	0.20	**0.225 ± 0.008**	0.312 ± 0.043
	0.30	**0.212 ± 0.007**	0.308 ± 0.035
	0.50	**0.194 ± 0.007**	0.305 ± 0.046
Solar	0.05	**0.095 ± 0.015**	0.125 ± 0.025
	0.10	**0.083 ± 0.015**	0.123 ± 0.032
	0.20	**0.072 ± 0.015**	0.108 ± 0.021
	0.30	**0.061 ± 0.012**	0.105 ± 0.027
	0.50	**0.054 ± 0.013**	0.102 ± 0.021
Traffic	0.05	**0.283 ± 0.016**	0.304 ± 0.026
	0.10	**0.211 ± 0.012**	0.264 ± 0.009
	0.20	**0.168 ± 0.006**	0.242 ± 0.019
	0.30	**0.151 ± 0.007**	0.218 ± 0.020
	0.50	**0.139 ± 0.007**	0.216 ± 0.017

Results In Table C.7, we show the performance of first-order TimeFlow on the imputation task. In low sampling regimes the difference with TimeFlow is less perceptive, but its performance plateaus when the number of points increases. This is not surprising. Indeed, as though the task is actually simpler when τ increases, the optimization is made more difficult with the increased number of observations. We provide the performance of TimeFlow with a set encoder on the Forecasting task in Table C.8. We observed that this version failed to generalize well for complex datasets.

Table C.8: Comparison of optimization-based and set-encoder-based meta learning for TimeFlow on the forecasting task. Mean MAE forecast results over different time windows. H stands for the horizon. Bold results are best.

	H	TimeFlow	TimeFlow w set encoder
Electricity	96	**0.228 ± 0.026**	0.362 ± 0.032
	192	**0.238 ± 0.020**	0.360 ± 0.028
	336	**0.270 ± 0.031**	0.382 ± 0.038
	720	**0.316 ± 0.055**	0.431 ± 0.059
SolarH	96	**0.190 ± 0.013**	0.251 ± 0.071
	192	**0.202 ± 0.020**	0.239 ± 0.058
	336	**0.209 ± 0.017**	0.235 ± 0.040
	720	**0.218 ± 0.048**	0.231 ± 0.032
Traffic	96	**0.217 ± 0.036**	0.276 ± 0.031
	192	**0.212 ± 0.028**	0.281 ± 0.034
	336	**0.238 ± 0.034**	0.297 ± 0.042
	720	**0.279 ± 0.050**	0.333 ± 0.048

C.2.2.5 Influence of the modulation

In TimeFlow, we apply shift modulations to the parameters of the INR, i.e. for each layer l we only modify the biases of the network with an extra bias term $\phi_{\mathbf{l}}^{(\mathbf{j})}$. We generate these bias terms with a linear hypernetwork that maps the code $\mathbf{z}^{(\mathbf{j})}$ to the modulations. The output of the l-th layer of the modulated INR is thus given by $\phi_{\mathbf{l+1}} = \text{ReLU}(\theta_{\mathbf{l}}\phi_{\mathbf{l-1}} + \mathbf{b_l} + \psi_{\mathbf{l}}^{(\mathbf{j})})$, where $\psi_{\mathbf{l}}^{(\mathbf{j})} = \mathbf{W_l z}^{(\mathbf{j})}$ and $(\mathbf{W_l})_{l=1}^{L}$ are parameters of the hypernetwork. However, another common modulation is the combination of the scale and shift modulation, which leads to the output of the l-th layer of the modulated INR being given by $\phi_{\mathbf{l+1}} = \text{ReLU}((\mathbf{S_l z}^{(\mathbf{j})}) \circ (\theta_{\mathbf{l}}\phi_{\mathbf{l-1}} + \mathbf{b_l}) + \psi_{\mathbf{l}}^{(\mathbf{j})})$, where $\psi_{\mathbf{l}}^{(\mathbf{j})} = \mathbf{W_l z}^{(\mathbf{j})}$, and $(\mathbf{W_l})_{l=1}^{L}$ and $(\mathbf{S_l})_{l=1}^{L}$ are parameters of the hypernetwork and $\circ$ is the Hadamard product.

In Table C.9, we conduct additional experiments on the *Electricity* dataset in the forecasting setting with different time horizons. In these experiments, we compare two scenarios: one where the INR is modulated only by a shift factor and the other where the INR is modulated by both a shift and a scale factor. We kept the architecture and hyperparameters consistent with those described in Appendix C.2.1. The experiments shown in Table C.9 indicate that the INR is longer to train with shift and scale modulations due to the increased number of parameters involved. Furthermore, we observe that the shift and scale modulated INR performed similarly or even worse than the INR with only shift modulation. These two drawbacks, namely an increased computational time and similar or worse performances, motivate modulating the INR only by a shift factor.

Table C.9: Ablation on modulations for the forecasting task on *Electricity* dataset for different horizons. Models are trained on a given time window and tested on four new time windows. Models are trained on a single NVIDIA TITAN RTX GPU.

	96		192		336		720	
	MAE	Training time	MAE	Training time	MAE	Training time	MAE	Training time
Shift	0.233 ± 0.014	2h30	0.245 ± 0.016	2h31	0.264 ± 0.020	2h33	0.303 ± 0.041	2h46
Shift and scale	0.257 ± 0.019	3h29	0.263 ± 0.014	3h32	0.268 ± 0.025	3h45	0.308 ± 0.037	4h14

C.2.2.6 Discussion on other hyperparameters

While the dimension of z is indeed a crucial hyperparameter, it is important to note that other hyperparameters also play a significant role in the performance of the INR. For example, the number of layers in the FFN directly affects the ability of the model to fit the time series. In our experiments, we have observed that using five or more layers yields good performance, and including additional layers can lead to slight improvements in the generalization settings.

Similarly, the number of frequencies used in the frequency embedding is another important hyperparameter. Using too few frequencies can limit the network's ability to capture patterns, while using too many frequencies can hinder its ability to generalize accurately.

The choice of learning rate is critical for achieving stable convergence during training. Therefore, in practice, we use a low learning rate combined with a cosine annealing scheduler to ensure stable and effective training.

C.3 Imputation experiments

C.3.1 Baselines details

C.3.1.1 Baselines training and hyperparameters

The baselines underwent meticulous training and extensive testing, involving thorough exploration of hyperparameters. We used SAITS repository (code) for BRITS and SAITS. The adopted setting results from a hyperparameters search, which yields marginally superior results for both methods compared to the default settings. The marginal difference in scores underscores the robustness of BRITS and SAITS. In addition, mTAN and TIDER did not perform optimally with the recommended configurations, requiring an extensive search. Details of the parameters explored are provided in Table C.10 and Table C.11. While the hyperparameter search led to performance improvements, overall results remained sub-optimal. For CSDI, the recommended settings proved inadequate for the considered datasets, prompting a comprehensive search. Among various parameters, the number of diffusion steps emerged as crucial, significantly enhancing performance, particularly at higher draw ratios. However, superior performance was attained with more diffusion steps, albeit at increased computational cost. The chosen parameters are detailed in Table C.12. In addition, the original DeepTime implementation did not perform well on imputation. Hence, we adapted the DeepTime training procedure (see Appendix C.3.2). Lastly, the vanilla Neural Process baseline underperformed, so we customized its architecture to conduct a fair comparison with TimeFlow. We used the INR and hypernetwork from TimeFlow to align the Neural Process with our temporal frequency bias and shift modulation technique.

C.3.1.2 Models complexity

We can see in Table C.13 that our method has 10 times less parameters than BRITS and 20 times less than SAITS. It is mainly due to their modelisation of interaction between samples. SAITS, which is based on transformers has the highest number of parameters when mTAN has the lowest number of parameters.

Table C.10: mTAN hyperparameter search.

Dimension size	γ linear scheduler	lr	NumRefPoints	k-iwae	Target ratio
50	1	1×10^{-5}	32	5	0.2
100	0.95	0.0001	64	10	0.8
-	0.5	0.001	128	-	-
-	0.1	0.005	-	-	-

Table C.11: TIDER hyperparameter search.

Dimension size	λ_{ar}	λ_{trend}	lr	Season number
50	0.1	0.01	0.0001	2
100	0.2	0.1	0.001	10
-	-	-	0.005	15
-	-	-	-	20

Table C.12: CSDI chosen hyperparameters.

Epochs	lr	Layers	Channels	Nheads	Embedding dimension	NSteps	Schedule	Time embedding
5000	0.001	4	64	8	128	100	Quad	128

Table C.13: Number of parameters for each DL methods on the imputation task on the *Electricity* dataset.

	TimeFlow	DeepTime	NeuralProcess	mTAN	SAITS	BRITS	TIDER
Number of parameters	602k	1315k	248k	113k	11 137k	6 220k	1 034k

C.3.2 Details on DeepTime adaptation for imputation

As DeepTime was proposed to address the forecasting task with a deeptime-index model, the authors did not tackle the task of imputation and left it out for future work. Given the success of this method and the motivation of our work, we wanted to explore its capabilities to impute time series with several subsampling rates. Following our current framework, we first tried to train the model in a self-supervised way, i.e. trying to reconstruct observations $x^{(j)} \in \mathcal{T}^{(j)}$ after the INR has been conditioned with the Ridge Regressor on the same set of observations, but discovered failure cases for $\tau \leq 0.20$. To be faithful to the original supervised training of DeepTime, we therefore randomly mask out 50% of the observations that we use as context for the Ridge Regressor and try to infer the other 50% (the targets) to train the INR.

We provide a qualitative comparison of the model's performance with these two different training procedures in Figure C.1. We can notice that the model that results from the self-supervised training perfectly fits the observations but completely misses the important patterns of the series. On the other hand, when DeepTime is trained to infer target values based on observations, it is able to capture the general trends. We think that in the small subsampling regime ($\tau \leq 0.20$), the Ridge Regressor easily fits very well all the observations which hinders the training of the INR's basis.

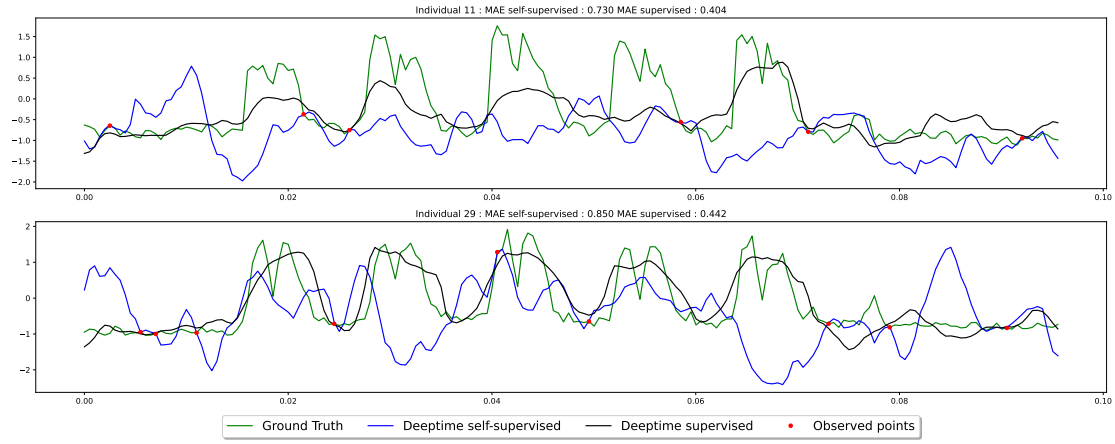


Figure C.1: *Electricity dataset*. Self supervised DeepTime imputation (blue line) and supervised DeepTime imputation (black line) with 5% of known point (red points) on the eight first days of samples 11 (top) and 29 (bottom).

C.4 Forecasting experiments

C.4.1 Distinction between adjacent time windows and new time windows during inference

In Section 4.4.2, we presented the forecasting results for periods outside the training period. These periods can be classified into two types: adjacent to or disjoint from the training period. Figure C.2 illustrates these distinct test periods for the *Electricity* dataset. The same principle applies to the *Traffic* and *SolarH* datasets, with one notable difference: the number of test periods is smaller in these datasets compared to *Electricity* dataset due to the fewer time steps available.

In Table 4.5, we presented the results indistinctly for the two types of test periods: adjacent to and disjoint from the training window. Here, we aim to differentiate the results for these two types of window and emphasize their significant impact on Informer and AutoFormer results. Specifically, Table C.14 showcases the results for

the test periods adjacent to the training window. In contrast, Table C.15 displays the results for the test periods disjointed from the training window

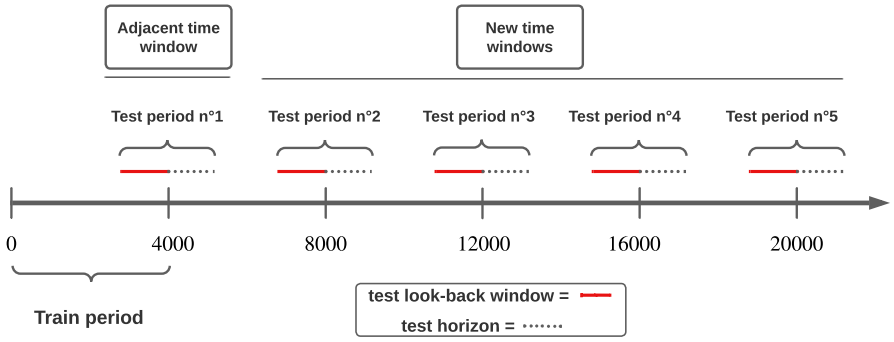


Figure C.2: Distinction between adjacent time windows and new time windows during inference for the *Electricity* dataset.

Results TimeFlow, PatchTST, DLinear and DeepTime maintain consistent forecasting results whether tested on the period adjacent to the training period or on a disjoint period. However, AutoFormer and Informer show a significant drop in performance when tested on new disjoint periods.

Table C.14: Mean MAE forecast results for adjacent time windows. H stands for the horizon. Bold results are best, underline results are second best.

		Continuous methods			Discrete methods			
	H	TimeFlow	DeepTime	Neural Process	Patch-TST	DLinear	AutoFormer	Informer
Electricity	96	0.218 ± 0.017	0.240 ± 0.027	0.392 ± 0.045	**0.214 ± 0.020**	0.236 ± 0.035	0.310 ± 0.031	0.293 ± 0.0184
	192	0.238 ± 0.012	0.251 ± 0.023	0.401 ± 0.046	**0.225 ± 0.017**	0.248 ± 0.032	0.322 ± 0.046	0.336 ± 0.032
	336	0.265 ± 0.036	0.290 ± 0.034	0.434 ± 0.075	**0.242 ± 0.024**	0.284 ± 0.043	0.330 ± 0.019	0.405 ± 0.044
	720	0.318 ± 0.073	0.356 ± 0.060	0.605 ± 0.149	**0.291 ± 0.040**	0.370 ± 0.086	0.456 ± 0.052	0.489 ± 0.072
SolarH	96	**0.172 ± 0.017**	0.197 ± 0.002	0.221 ± 0.048	0.232 ± 0.008	0.204 ± 0.002	0.261 ± 0.053	0.273 ± 0.023
	192	**0.198 ± 0.010**	0.202 ± 0.014	0.244 ± 0.048	0.231 ± 0.027	0.211 ± 0.012	0.312 ± 0.085	0.256 ± 0.026
	336	0.207 ± 0.019	**0.200 ± 0.012**	0.241 ± 0.005	0.254 ± 0.048	0.212 ± 0.019	0.341 ± 0.107	0.287 ± 0.006
	720	**0.215 ± 0.016**	0.240 ± 0.011	0.403 ± 0.147	0.271 ± 0.036	0.246 ± 0.015	0.368 ± 0.006	0.341 ± 0.049
Traffic	96	0.216 ± 0.033	0.229 ± 0.032	0.283 ± 0.028	**0.201 ± 0.031**	0.225 ± 0.034	0.299 ± 0.080	0.324 ± 0.113
	192	0.208 ± 0.021	0.220 ± 0.020	0.292 ± 0.023	**0.195 ± 0.024**	0.215 ± 0.022	0.320 ± 0.036	0.321 ± 0.052
	336	0.237 ± 0.040	0.247 ± 0.033	0.305 ± 0.039	**0.220 ± 0.036**	0.244 ± 0.035	0.450 ± 0.127	0.394 ± 0.066
	720	**0.266 ± 0.048**	0.290 ± 0.045	0.339 ± 0.037	0.268 ± 0.050	0.290 ± 0.047	0.630 ± 0.043	0.441 ± 0.055
TimeFlow improvement		/	6.56 %	30.79 %	2.64 %	7.30 %	35.43 %	33.07 %

Table C.15: Mean MAE forecast results for new time windows. H stands for the horizon. Bold results are best, underline results are second best.

		Continuous methods			Discrete methods			
	H	TimeFlow	DeepTime	Neural Process	Patch-TST	DLinear	AutoFormer	Informer
Electricity	96	0.230 ± 0.012	0.245 ± 0.026	0.392 ± 0.045	**0.222 ± 0.023**	0.240 ± 0.025	0.606 ± 0.281	0.605 ± 0.227
	192	0.246 ± 0.025	0.252 ± 0.018	0.401 ± 0.046	**0.231 ± 0.020**	0.257 ± 0.027	0.545 ± 0.186	0.776 ± 0.257
	336	0.271 ± 0.029	0.285 ± 0.034	0.434 ± 0.076	**0.253 ± 0.027**	0.298 ± 0.051	0.571 ± 0.181	0.823 ± 0.241
	720	0.316 ± 0.051	0.359 ± 0.048	0.607 ± 0.15	**0.299 ± 0.038**	0.373 ± 0.075	0.674 ± 0.245	0.811 ± 0.257
SolarH	96	0.208 ± 0.005	**0.206 ± 0.026**	0.221 ± 0.048	0.293 ± 0.089	0.212 ± 0.019	0.228 ± 0.027	0.234 ± 0.011
	192	**0.206 ± 0.012**	0.207 ± 0.037	0.244 ± 0.048	0.274 ± 0.060	0.223 ± 0.029	0.356 ± 0.122	0.280 ± 0.033
	336	0.211 ± 0.005	**0.199 ± 0.035**	0.240 ± 0.006	0.264 ± 0.088	0.223 ± 0.032	0.327 ± 0.029	0.366 ± 0.039
	720	**0.222 ± 0.020**	0.217 ± 0.028	0.403 ± 0.147	0.262 ± 0.083	0.251 ± 0.047	0.335 ± 0.075	0.333 ± 0.012
Traffic	96	0.218 ± 0.042	0.229 ± 0.032	0.283, 0.0275	**0.204 ± 0.039**	0.229 ± 0.032	0.326 ± 0.049	0.388 ± 0.055
	192	0.213 ± 0.028	0.220 ± 0.023	0.292, 0.0236	**0.198 ± 0.031**	0.223 ± 0.023	0.575 ± 0.254	0.381 ± 0.049
	336	0.239 ± 0.035	0.244 ± 0.040	0.305, 0.0392	**0.223 ± 0.040**	0.252 ± 0.042	0.598 ± 0.286	0.448 ± 0.055
	720	0.280 ± 0.047	0.290 ± 0.055	0.339, 0.0375	**0.270 ± 0.059**	0.304 ± 0.061	0.641 ± 0.072	0.468 ± 0.064
TimeFlow improvement		/	2.50 %	27.75 %	3.41 %	6.80 %	46.26 %	45.53 %

C.4.2 Plots comparison: TimeFlow vs PatchTST

Table 4.5 demonstrates the similar forecasting performance of TimeFlow and PatchTST across all horizons. To visually represent their predictions, the figures below showcase the forecasted outcomes of these methods for two samples (24 and 38) and two horizons (96 and 192) on the *Electricity*, *SolarH*, and *Traffic* datasets.

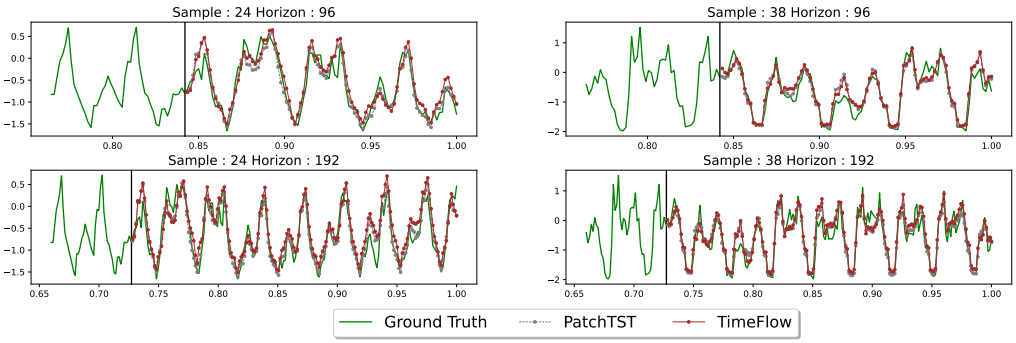


Figure C.3: Qualitative comparisons of TimeFlow vs PatchTST on the *Electricity* dataset for new time windows.

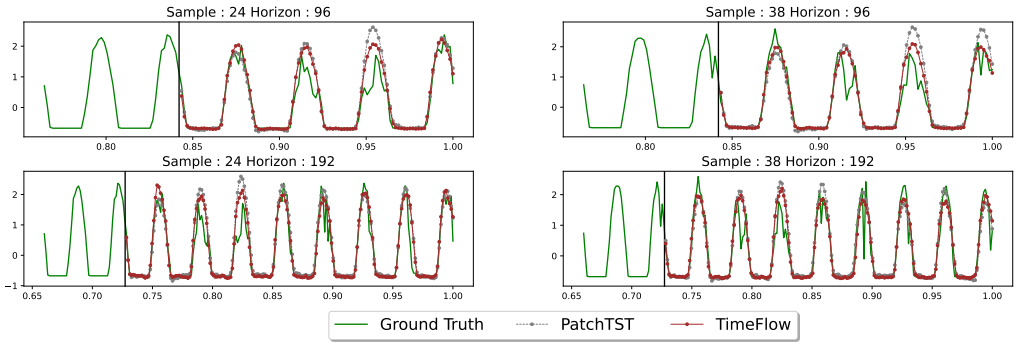


Figure C.4: Qualitative comparisons of TimeFlow vs PatchTST on the *SolarH* dataset for new time windows.

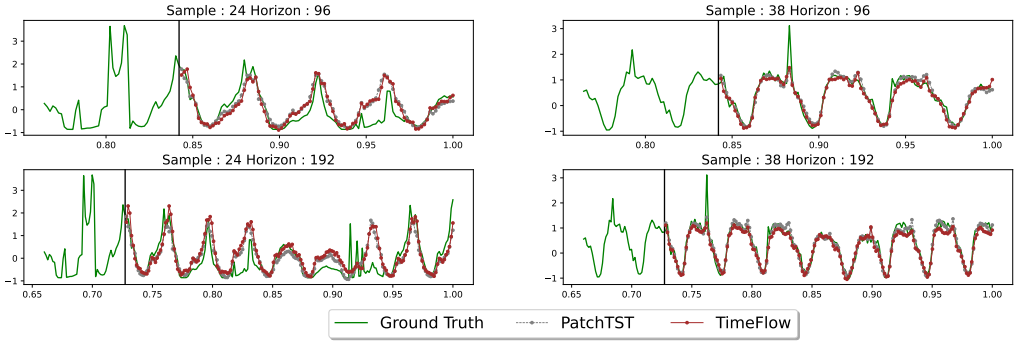


Figure C.5: Qualitative comparisons of TimeFlow vs PatchTST on the *Traffic* dataset for new time windows.

Results The visual analysis of the figures above reveals that the predictions of TimeFlow and PatchTST are remarkably similar. For instance, when examining sample 24 and horizon 192 of the *Traffic* dataset, both forecasters exhibit similar error patterns. The only noticeable distinction emerges in the *SolarH* dataset, where PatchTST tends to overestimate certain peaks.

C.4.3 Baseline details

C.4.3.1 Baselines training and hyperparameters

We provide a detailed breakdown of the hyperparameters and our training approach for the forecasting baselines. We took an in-depth approach, testing each method under a range of configurations to ensure they were well-suited to the unique characteristics of the datasets and tasks at hand. For DLinear and transformer baselines, including PatchTST, AutoFormer, and Informer, we utilized the implementations detailed in the PatchTST baselines (code) and adhered to the best practices recommended for our particular tasks. Notably, our implementation of PatchTST was

combined with ReVIN, enhancing the robustness of the results. Regarding DeepTime, we followed the recommended hyperparameters, opting for a structure with 5 layers, each 256 units wide, and using 4096 Fourier features spanning a diverse set of scales. As for the Neural Process, the standard model did not train as expected. So, we customized its architecture to conduct a fair comparison with TimeFlow. We used the INR and hypernetwork from TimeFlow to align the Neural Process with our temporal frequency bias and shift modulation technique. We also meticulously searched for the optimal hyperparameters, like the Kullback Leibler (KL) divergence weight and learning rate. Moreover, we extended the training duration to ensure thorough convergence.

C.4.3.2 Models complexity

In this section, we present the parameter counts and the inference time for the main forecasting baselines. Except for TimeFlow and DeepTime, the number of parameters varies with the number of samples, the look-back window, and the horizon. Thus, we report the number of parameters for two specific configurations, including a fixed dataset, a fixed look-back window, and a fixed horizon. In Table C.16, we see that for PatchTST and DLinear, the larger the horizon, the more the number of parameters increases. In Table C.17, it is shown that all methods' computational time increases with the horizon, which is expected. Moreover, TimeFlow is slower than the baselines that use forward computations only. Still, on the Electricity dataset, for example, the method can infer for 321 samples a horizon of 720 values with a look-back window of 512 timestamps in less than 0.2s, which does not look prohibitive for many real-world usages. This is mainly due to the small number of gradient steps at inference.

Table C.16: The number of parameters for main baselines on the forecasting task on the *Electricity* dataset for horizons 96 and 720. The look-back window size is 512.

	TimeFlow	DeepTime	Neural Process	Patch-TST	DLinear	Informer	Autoformer
96	602k	1 315k	480k	1 194k	98k	984k	1 005k
720	602k	1 315k	480k	6 306k	739k	984k	1 005k

Table C.17: Inference time (in seconds) for the forecasting task on the *Electricity* dataset with horizons 96 and 720 and a look-back window of length 512. The statistics are computed over 10 runs using an NVIDIA TITAN RTX GPU.

	TimeFlow	Patch-TST	DLinear	DeepTime	AutoFormer	Informer
96	0.147 ± 0.007	0.016 ± 0.002	0.007 ± 0.003	0.006 ± 0.002	0.027 ± 0.001	0.0191 ± 0.002
720	0.176 ± 0.009	0.020 ± 0.001	0.009 ± 0.001	0.010 ± 0.002	0.034± 0.001	0.0251 ± 0.002

C.4.4 Sparsely observed look-back window: comparison with Patch-TST

Setting and baseline. Let's consider a setting where at inference time, the look-back window is sparsely observed. Models such as PatchTST must proceed in two steps: (i) completing the look-back window on a dense regular grid using imputation; (ii) apply the model on the completed window to predict the future. We compared TimeFlow with the following two-step processing baseline: linear interpolation handling the missing values within the partially observed look-back window, and PatchTST handling the forecasting task. We conducted experiments on the *Traffic* and *Electricity* datasets, focusing on the 96 and 192 horizons. In Table C.18, we present the results at different sampling rates $\tau \in \{0.5, 0.2, 0.1\}$ within the look-back window.

Table C.18: MAE results for forecasting on new samples and new period with missing values in the look-back window. Best results are in bold.

			TimeFlow		Linear interpo + PatchTST	
	H	τ	Imputation error	Forecast error	Imputation error	Forecast error
Electricity	96	1.	0.000 ± 0.000	0.228 ± 0.028	0.000 ± 0.000	**0.221 ± 0.023**
		0.5	**0.151 ± 0.003**	**0.239 ± 0.013**	0.257 ± 0.008	0.279 ± 0.026
		0.2	**0.208 ± 0.006**	**0.260 ± 0.015**	0.482 ± 0.019	0.451 ± 0.042
		0.1	**0.272 ± 0.006**	**0.295 ± 0.016**	0.663 ± 0.029	0.634 ± 0.053
	192	1.	0.000 ± 0.000	0.238 ± 0.020	0.000 ± 0.000	**0.229 ± 0.020**
		0.5	**0.149 ± 0.004**	**0.235 ± 0.011**	0.258 ± 0.006	0.280 ± 0.032
		0.2	**0.209 ± 0.006**	**0.257 ± 0.013**	0.481 ± 0.021	0.450 ± 0.054
		0.1	**0.274 ± 0.010**	**0.289 ± 0.016**	0.669 ± 0.030	0.650 ± 0.060
Traffic	96	1.	0.000 ± 0.000	0.217 ± 0.032	0.000 ± 0.000	**0.203 ± 0.037**
		0.5	**0.219 ± 0.017**	**0.224 ± 0.033**	0.276 ± 0.012	0.255 ± 0.041
		0.2	**0.278 ± 0.017**	**0.252 ± 0.029**	0.532 ± 0.017	0.483 ± 0.040
		0.1	**0.418 ± 0.019**	**0.382 ± 0.014**	0.738 ± 0.023	0.721 ± 0.073
	192	1.	0.000 ± 0.000	0.212 ± 0.028	0.000 ± 0.000	**0.197 ± 0.030**
		0.5	**0.176 ± 0.014**	**0.217 ± 0.017**	0.276 ± 0.011	0.245 ± 0.029
		0.2	**0.233 ± 0.017**	**0.236 ± 0.021**	0.532 ± 0.020	0.480 ± 0.050
		0.1	**0.304 ± 0.019**	**0.277 ± 0.021**	0.734 ± 0.022	0.787 ± 0.172

Results. Although PatchTST performs slightly better with a dense look-back window, its performance significantly deteriorates as the value of τ decreases. In contrast, the performance of TimeFlow is only minimally affected by the reduction in the sampling rate.

C.4.5 Influence of the look-back window for forecasting

In Figure C.6, it is shown that both excessively short and overly long look-back windows can harm TimeFlow forecasting performance. More precisely, the performances increases with the look-back window size up to a certain size, where the performances then drop slowly.

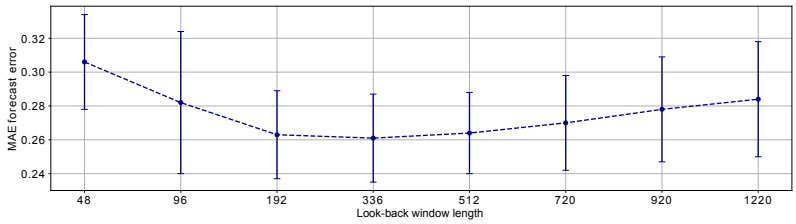


Figure C.6: MAE forecast error per look-back windows length for the *Electricity* dataset (horizon window length is 336). The model is trained on a given time window and tested on four new time windows.

C.4.6 Influence of the horizon length for forecasting

In Figure C.7, it is shown that the performances decrease with the length of the horizon. This is to be expected, since the longer the horizon, the harder the task.

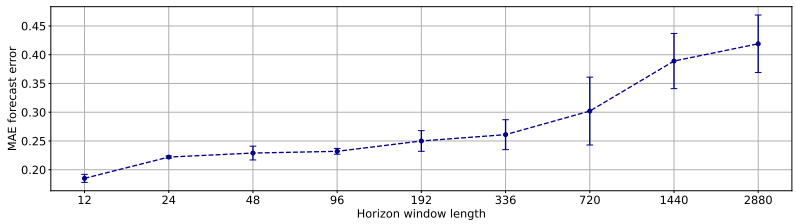


Figure C.7: MAE forecast error per horizons length for the *Electricity* dataset (look-back window length is 512). The model is trained on a given time window and tested on four new time windows.

C.5 Discussion on using frequency embedding as input to regression models

C.5.1 Related work

In forecasting univariate time series, several models have explored the integration of frequency embedding of timestamps for regression purposes. Taylor and Letham (2018) approached the time series as a continuous function of time using a general additive model (Hastie, 2017). They represented seasonality components as learnable Fourier series while explicitly specifying the ground truth seasonalities (e.g., weekly, monthly). Similarly, Hyndman and Athanasopoulos (2018) proposed embedding the timestamp t with the ground truth frequencies and applying a regression model to predict the series value at the timestamp t. Both methods rely on the explicit specification of seasonalities and are tailored for purely univariate time series, where information is not shared between samples.

In contrast, other models, such as TimeFlow or DeepTime (Woo et al., 2022), based on deep learning techniques, offer more flexibility. These approaches can autonomously learn relevant frequencies and effectively share information between samples through backpropagation. This enables a more dynamic and adaptable approach to time series forecasting, particularly in scenarios with complex temporal patterns and inter-sample dependencies.

C.5.2 Experiments

Given the seasonal patterns observed in the *Solar*, *Electricity*, and *Traffic* datasets, an alternative to deep learning forecasting methods is to individually regress a timestamp embedding on the corresponding value using a robust regressor. This approach exploits the inherent periodicity in the data, using timestamp embeddings to capture temporal dependencies and accurately predict the target values.

Baselines. We compare TimeFlow against two regression baselines:

- **TimeFlow frequencies embedding + XGBoost**: This baseline uses the same frequency embedding as TimeFlow and applies an XGBoost regressor (Chen and Guestrin, 2016) on top. The aim is to assess whether the XGBoost regressor can effectively identify the correct frequencies, filter out irrelevant ones, and establish the appropriate mapping between timestamps and values.
- **XGB Explicit Seasonal encoding + XGboost**: in this baseline, we give *explicitly* the right frequencies of each datasets to the model. It is important to highlight that **this method uses information that other baselines**

don't. For instance for the *Traffic* dataset, the Explicit Seasonal encoding is $\gamma(t) = \left(t, \cos\left(\frac{2\pi t}{24}\right), \sin\left(\frac{2\pi t}{24}\right), \cos\left(\frac{2\pi t}{24\times 7}\right), \sin\left(\frac{2\pi t}{24\times 7}\right)\right)$. It allows to explicitly integers trend, daily frequencies and weekly frequencies. We apply the same type of frequencies embedding for *Solar* and *Electricity* with the appropriate seasonalities.

Experimental Setup. For each dataset and sample, we applied the frequency embedding individually and then trained an XGBoost regressor on each observed timestamp (the look-back window in forecasting and the observed grid in imputation). This approach results in one model per sample. The XGBoost regressor is configured with the following hyperparameters:

- n estimators: 500
- max depth: 4
- learning rate: 0.1
- lambda: [0.1, 1, 10]

The optimal regularization parameter lambda is determined through cross-validation. The imputation and forecasting results are presented in detail in Table C.19 and Table C.20, respectively.

Table C.19: Mean MAE imputation results on the missing grid only. In the table, τ stands for the subsampling rate, i.e. the proportion of observed points considered for each samples. Bold results are best, underlined results are second best.

	τ	TimeFlow	TimeFlow frequencies embedding+ XGB	Explicit Seasonal encoding + XGB
Electricity	0.05	**0.324 ± 0.013**	0.834 ± 0.092	0.365 ± 0.051
	0.10	**0.250 ± 0.010**	0.761 ± 0.074	0.318 ± 0.049
	0.20	**0.225 ± 0.008**	0.632 ± 0.066	0.278 ± 0.044
	0.30	**0.212 ± 0.007**	0.536 ± 0.041	0.259 ± 0.048
	0.50	**0.194 ± 0.007**	0.418 ± 0.042	0.238 ± 0.022
Solar	0.05	**0.095 ± 0.015**	0.603 ± 0.035	0.234 ± 0.021
	0.10	**0.083 ± 0.015**	0.478 ± 0.024	0.190 ± 0.022
	0.20	**0.072 ± 0.015**	0.350 ± 0.022	0.150 ± 0.019
	0.30	**0.061 ± 0.012**	0.286 ± 0.018	0.134 ± 0.011
	0.50	**0.054 ± 0.013**	0.227 ± 0.015	0.123 ± 0.015
Traffic	0.05	**0.283 ± 0.016**	0.739 ± 0.140	0.344 ± 0.036
	0.10	**0.211 ± 0.012**	0.676 ± 0.129	0.290 ± 0.029
	0.20	**0.168 ± 0.006**	0.562 ± 0.108	0.245 ± 0.027
	0.30	**0.151 ± 0.007**	0.487 ± 0.095	0.223 ± 0.015
	0.50	**0.139 ± 0.007**	0.393 ± 0.083	0.198 ± 0.021
TimeFlow improvement		/	69.5 %	33.7 %

Imputation results. TimeFlow performs better than the other two methods. Although the second baseline explicitly incorporates ground truth frequencies and provides decent results, its inability to share information between samples leads to the loss of valuable insights that TimeFlow effectively exploits. In addition, the first baseline struggles to learn the correct frequencies and overfits observed data points, resulting in excessive high-frequency noise. As a result, its performance degrades significantly compared to the second baseline, where frequencies are explicitly provided. These findings underscore TimeFlow's ability to identify the underlying frequencies, and leverage shared information across samples to improve accuracy during imputation.

Table C.20: Mean MAE forecast results for adjacent time windows averaged over different time windows. Each time, the model is trained on one time window and tested on the others (there are 2 windows for SolarH and 5 for Electricity and Traffic). H stands for the horizon. Bold results are best, and underlined results are second best

	H	TimeFlow	TimeFlow frequencies embedding + XGB	Explicit Seasonnal encoding + XGB
Electricity	96	**0.218 ± 0.017**	0.662 ± 0.102	<u>0.282 ± 0.020</u>
	192	**0.238 ± 0.012**	0.750 ± 0.128	<u>0.279 ± 0.021</u>
	336	**0.265 ± 0.036**	0.809 ± 0.136	<u>0.294 ± 0.041</u>
	720	**0.318 ± 0.073**	0.852 ± 0.144	<u>0.357 ± 0.092</u>
SolarH	96	**0.172 ± 0.017**	0.792 ± 0.062	<u>0.244 ± 0.023</u>
	192	**0.198 ± 0.010**	0.933 ± 0.055	<u>0.236 ± 0.018</u>
	336	**0.207 ± 0.019**	1.033 ± 0.052	<u>0.229 ± 0.022</u>
	720	**0.215 ± 0.016**	1.116 ± 0.057	<u>0.262 ± 0.021</u>
Traffic	96	**0.216 ± 0.033**	0.655 ± 0.156	<u>0.288 ± 0.052</u>
	192	**0.208 ± 0.021**	0.678 ± 0.139	<u>0.246 ± 0.033</u>
	336	**0.237 ± 0.040**	0.719 ± 0.143	<u>0.262 ± 0.044</u>
	720	**0.266 ± 0.048**	0.741 ± 0.140	<u>0.288 ± 0.063</u>
TimeFlow improvement		/	70.8 %	15.7 %

Forecasting results. TimeFlow also outperforms the other two baselines in forecasting. However, the improvement over the second baseline, where the correct frequencies are explicitly provided, is more modest compared to the gains observed in imputation. Nevertheless, the relative improvement achieved by TimeFlow remains significant (exceeding 15 %). Similar to the imputation scenario, the XGBoost baseline with TimeFlow timestamps encoding, which attempts to learn the correct frequencies, fails to discern them accurately and introduces excessive high-frequency components.

C.6 Discussion: meta-learning optimization in time series forecasting

The concept of inner and outer loops is to be reformulated within the broader general framework of model-agnostic meta-learning (Finn et al., 2017), where the authors seek to enable rapid adaptation of the model to unseen tasks. In TimeFlow, we adapt the general idea of agnostic meta-learning to our tasks. We propose an efficient way to achieve this goal by splitting the parameters into two parts: context parameters (learned in the inner loop, responsible for the adaptive part of the model) and meta-parameters (or "parameters shared across tasks") (learned in the outer loop, responsible for the generic part of the model).

In the context of time series forecasting. TimeFlow aims to have a subset of parameters that adapts to specific contextual factors (e.g., the look-back window of a particular sample) and another subset that performs the forecasting task according to this learned context (e.g., forecasting any point within the forecast horizon as well as within the look-back window). To achieve this, we seek to adjust the codes $\mathbf{z}^{(\mathbf{j})}$ exclusively on the contexts (e.g., the look-back window) for each sample j, while the shared parameters between samples θ and $\mathbf{w}$ characterize a shared function capable of forecasting based on a given $\mathbf{z}^{(\mathbf{j})}$. This function could be represented as $f_{\theta,\mathbf{w}}(t, z(\mathbf{x}^{(\mathbf{j})}))$. This concept entails adapting $\mathbf{z}^{(\mathbf{j})}$ by sample j and training θ, $\mathbf{w}$ by batch.

Appendix D

Appendix of Chapter 5

D.1 Noisy modulation visualization

In the experiment in Section 5.2.2, we perturb the modulation (by adding Gaussian noise) for only one layer and a given channel of the INR and observe the difference in the time domain between the non-perturbed and the perturbed TimeFlow. We provide in Figure D.1 a visualization of the modulation perturbation.

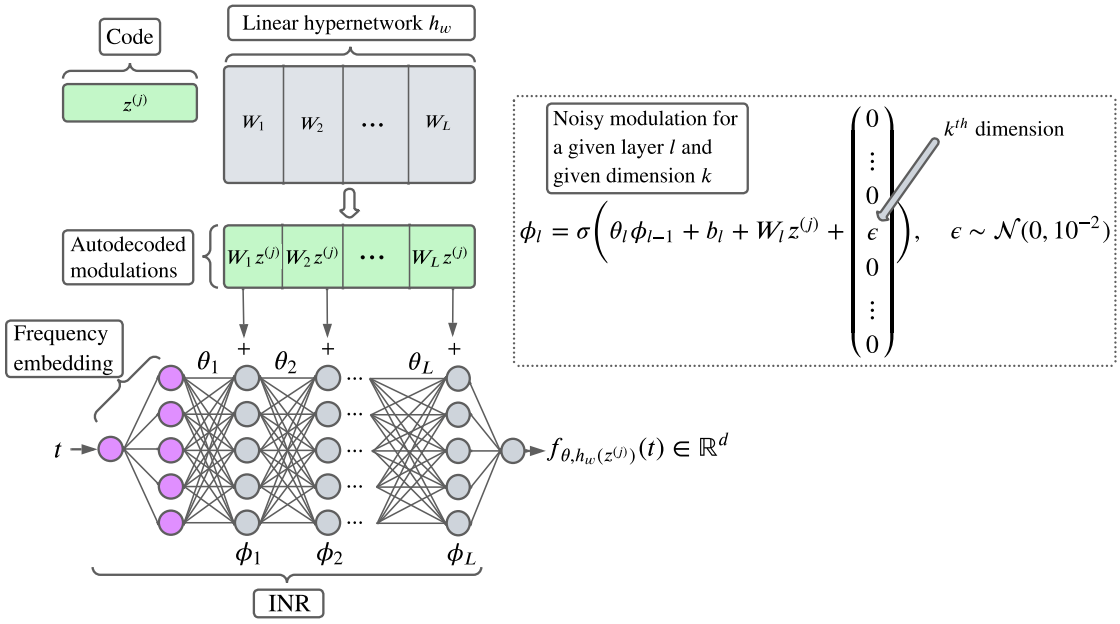


Figure D.1: Noisy modulation visualization.

D.2 DDPM inference process.

In Algorithm 4 we present the procedure to generate new $\mathbf{z}^{(\mathbf{gen})}$ from a trained $\epsilon_\nu(.)$ denoiser.

Algorithm 4: DDPM Inference

Sample $\mathbf{z_K} \sim \mathcal{N}(0, \mathbf{I})$;
for $k \in \{K, ..., 1\}$ **do**
  $\mathbf{z_{k-1}} \leftarrow \frac{1-\bar{\alpha}_{k-1}\sqrt{\alpha_k}}{1-\bar{\alpha}_k}\mathbf{z_k} + \frac{(1-\alpha_k)\sqrt{\alpha_{k-1}}}{1-\bar{\alpha}_k}\epsilon_\nu(\mathbf{z_k}) + \sigma(k)\rho, \quad \rho \sim \mathcal{N}(0, \mathbf{I})$;
end
$\mathbf{z}^{(\mathbf{gen})} \leftarrow \mathbf{z_0}$;

D.3 Denoiser architecture in the DDPM implementation

In this section, we provide a detailed description of the denoiser architecture used in the DDPM-only and TimeFlow + DDPM models, as mentioned in Section 5.3.3. The denoiser is based on a 1D UNet architecture, and we present a simplified overview of the architecture in Figure D.2.

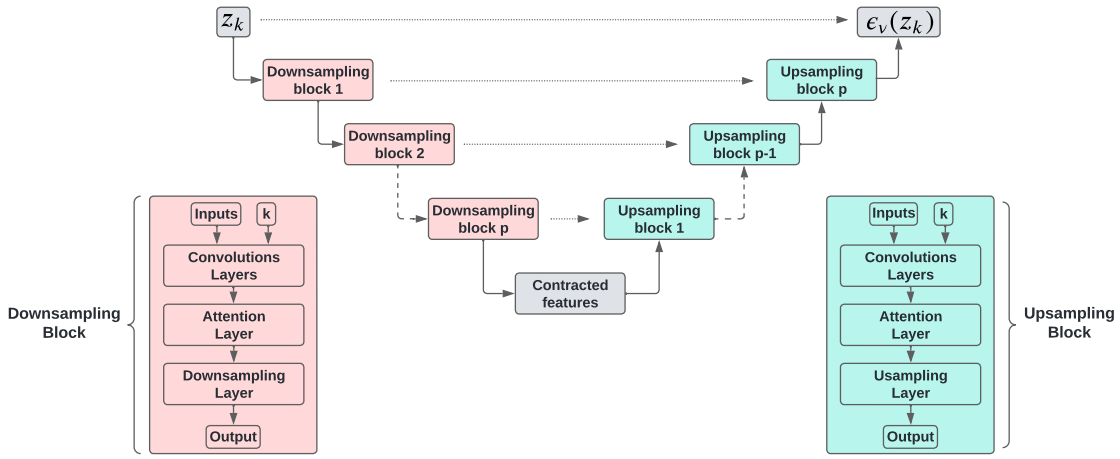


Figure D.2: Simplified overview of the denoiser UNet 1D architecture.

For a more comprehensive understanding of the architecture, we recommend referring to the official implementation[1]. It is important to note that we did not focus on optimizing the hyperparameters of the denoiser for these experiments.

[1] https://github.com/lucidrains/denoising-diffusion-pytorch

D.4 Principal component analysis visualization

In addition to the quantitative analysis proposed in Section 5.3.2.3, we propose to visualize the generated data and the test data using principal component analysis (PCA) for the three methods under consideration. We maintain an identical setting as the one proposed with t-SNE visualization (Figure 5.10) to ensure consistency and comparability. Figure D.3 shows the 2D PCA results for the three generation methods.

(a) TimeFlow + DDPM. (b) DDPM. (c) TimeGan.

Figure D.3: PCA visualization comparing generated and test time series for the three considered methods.

Results. As shown in Figure D.3, the test series and those generated by TimeFlow + DDPM and the DPPM-only method are indistinguishable. However, while some TimeGan-generated series overlap with the test series, the generated and test distributions appear to differ significantly.

Appendix E

Discussion on public time series datasets

In this section, we would like to discuss the limitation identified with the currently available public datasets for time series.

The availability of public datasets like the UCR archive (Dau et al., 2019) and Monash archive (Godahewa et al., 2021), among others, has significantly advanced machine learning for time series. However, several limitations have been identified that hinder progress in deep learning. These are subjective observations:

- **Datasets volume.** Public time series datasets are relatively small, unlike natural language processing or computer vision. There is no equivalent of Wikipedia or ImageNet for time series. This limitation may hinder the performance of deep learning methods for time series tasks. For example, in the forecasting task, there are very few public datasets with a considerable number of samples and timestamps. The largest datasets we know of are *Traffic* and *Electricity*, each with several hundred samples and tens of thousands of timestamps. These volumes are relatively small for training deep learning models, making their advantages over traditional machine learning models less pronounced (see Appendix C.5). Similarly, in classification, most of the time, series classification datasets in the UCR archive are small, sometimes with only a few dozen training examples, making it challenging to apply deep learning models effectively.

- **Multivariate time series datasets.** In this manuscript, we have chosen to define multivariate time series datasets as $\{\boldsymbol{x}^{(j)}\}_{j=1}^{n}$, where $\boldsymbol{x}^{(j)} \in \mathbb{R}^{c \times T}$, with n referring to the number of samples, T the number of time steps, and c the number of sensors/channels considered. We defined a series as multivariate when $c > 1$. This definition seems well-established in time series classification but not in forecasting. Many methods, for example, consider datasets like

SolarH, *Electricity*, and *Traffic* as multivariate datasets with samples corresponding to different channels. They treat these datasets similarly to datasets like *Weather*, which measures various underlying phenomena such as temperature, humidity, and precipitation at a single location.

We believe that effective development of multivariate deep learning models for time series requires access to large public datasets where multiple variables are measured. For example, for $c = 2$, this could include a target variable (such as wind power generation) and a context variable (such as measured wind speed) at numerous locations (e.g., thousands of locations). While it may seem idealistic to have such public datasets, this type of data is often available in private industrial datasets. We believe that publicly sharing time series generated from these datasets would be a great opportunity for the time series research community.

Appendix F

Résumé étendu en français

Cette annexe a pour objectif de fournir un résumé détaillé du manuscrit de thèse. Dans une première section, Appendix F.1, nous définissons le concept de représentation de séries temporelles et présentons ses avantages par rapport aux méthodes purement supervisées. Nous introduisons ensuite les méthodes neuronales pour l'apprentissage de représentations, en soulignant les avancées qu'elles apportent pour les tâches en aval. Nous concluons cette section en identifiant les problèmes ouverts en apprentissage de représentations neuronales pour les séries temporelles. Dans la section Appendix F.2, nous exposons succinctement nos contributions visant à répondre aux problèmes ouverts mentionnés précédemment. Nous y discutons apprentissage de représentations neuronales : (i) discrètes et interprétables, (ii) capables de gérer les changement de distributions, (iii) capables de capturer la continuité. Pour conclure, nous proposons une synthèse de nos travaux dans la section Appendix F.3.

F.1 Motiver les représentations neuronales pour les séries temporelles

F.1.1 Les séries temporelles : une introduction

Les séries temporelles sont omniprésentes dans divers domaines tels que l'industrie, la météorologie, la finance ou encore la santé. Ces données, collectées au fil du temps, peuvent représenter des phénomènes liés à des activités humaines, des événements physiques ou encore des phénomènes hybrides. Historiquement, l'analyse des séries temporelles est associée à la *prévision* d'une série unique de manière autoregressive, l'objectif étant de prédire le futur à partir des tendances et motifs observés dans le passé. Les premiers modèles développés, tels que les méthodes d'exponential smoothing (Brown, 1959; Holt, 2004) et d'autoregressive integrate moving average (Box et al., 2015), s'inscrivent dans ce cadre.

Avec la prolifération des données, l'attention s'est tournée vers l'analyse des jeux de données de séries temporelles. Ces jeux de données peuvent être définis comme une collection de séries temporelles mesurant des phénomènes similaires, par exemple, la consommation électrique horaire sur deux semaines de plusieurs milliers de foyers. La modélisation conjointe de ces données a permis l'essor de nouvelles méthodes, améliorant les performances en *prévision* et en *imputation*. En effet, dans de nombreux cas d'application, les relations entre les séries permettent de modéliser plus finement les séries temporelles que les modèles basés uniquement sur des informations individuelles. La prolifération des jeux de données a également fait émerger de nouvelles tâches telles que : • la *classification*, qui consiste à discriminer des séries en fonction de leur label (par exemple, classifier un électrocardiogramme selon la présence d'une anomalie cardiaque). • La *génération*, qui consiste à simuler des séries temporelles présentant des caractéristiques similaires à celles d'un jeu de données d'entraînement.

Contrairement aux modèles purement supervisés, un nouveau paradigme est apparu avec les modèles *d'apprentissage de représentation*. Ces modèles visent à apprendre des représentations des données dans un nouvel espace (nommé espace latent) sans résoudre explicitement une tâche supervisée particulière. Ces représentations capturent les caractéristiques sous-jacentes aux données, projetant les séries temporelles dans un espace structuré où elles sont plus simples à manipuler. Par la suite, les représentations apprises peuvent être réutilisées pour résoudre des tâches spécifiques en aval, comme illustré dans la Figure F.1.

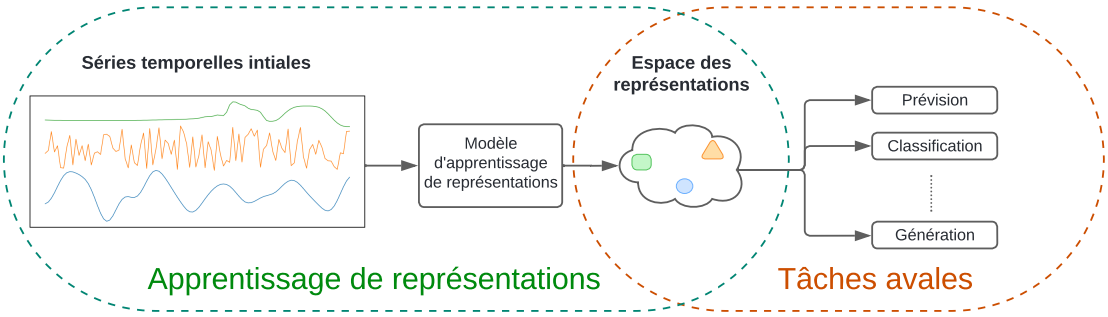


Figure F.1: Visualisation de la pipeline d'apprentissage de représentations pour les des séries temporelles. Étape 1 : apprendre la représentation de la série temporelle par apprentissage non supervisé. Étape 2 : application d'un modèle supervisé par dessus la représentation apprise pour traiter une tâche en aval. Ce graphique est inspiré de Trirat et al. (2024).

F.1.2 L'apprentissage de représentations pour les séries temporelle

L'apprentissage de représentations pour les séries temporelles vise à projeter les données originales dans un espace latent distinct de leur domaine temporel d'origine. L'objectif principal est d'extraire des caractéristiques pertinentes pour des tâches d'apprentissage automatique ultérieures. Cette approche peut offrir plusieurs avantages, tels que : (i) des performances améliorées, notamment avec peu d'exemples, (ii) un temps de calcul réduit, (iii) une meilleure interprétabilité, (iv) la capacité à capturer la structure continue des données.

La représentation d'une série temporelle est définie comme le résultat d'une fonction d'encodage appliquée à la série temporelle d'origine. La dimension latente de la représentation peut varier en fonction de la méthode d'encodage utilisée. Par exemple, la dimension temporelle peut être réduite ou même complètement contractée, entraînant une perte de la temporalité. Un exemple simple de représentation de série temporelle consiste à extraire des statistiques descriptives telles que la moyenne, la variance, le minimum et le maximum de chaque série (voir Figure F.2).

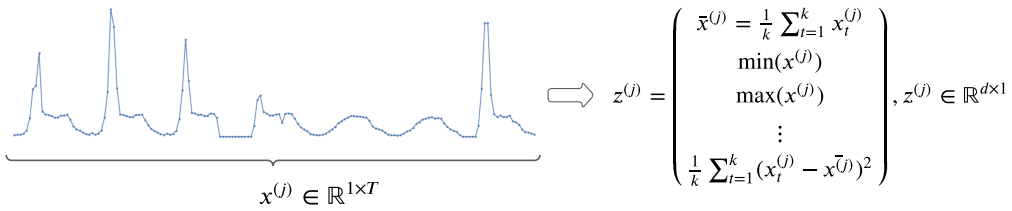


Figure F.2: Vecteur de représentation d'une série temporelle composé de simples statistiques descriptives.

Ce vecteur de caractéristiques peut ensuite être utilisé pour des tâches de classification, de détection d'anomalies, etc. Il existe un grand nombre de méthodes d'apprentissage de représentations pour les séries temporelles. Parmi les plus populaires en apprentissage automatique, on trouve : (i) la décomposition saisonnalités/tendance/résidu (STR), et (ii) la représentation d'Approximation par Agrégat Symbolique (SAX) (Lin et al., 2003, 2007).

(i) *La décomposition STR* permet de décomposer une série temporelle en ses composantes saisonnières, tendance et résiduelle. Cette décomposition facilite la représentation de la structure sous-jacente des séries temporelles périodiques. Elle est couramment employée pour des tâches en aval telles que la prévision ou la détection d'anomalies (lorsque les données s'y prêtent).

(ii) *La représentation SAX*, quant à elle, discrétise une série temporelle en une séquence de symboles de longueur réduite. Les représentations SAX sont principalement réutilisées en aval pour des tâches de classification.

Ces deux méthodes d'apprentissage de représentation offrent différents paradigmes pour représenter les séries temporelles selon les tâches en aval. Elles sont très utilisées dans diverses applications de l'apprentissage automatique.

Il existe de nombreuses autres méthodes d'apprentissage de représentations de séries temporelles, telles que l'analyse en composantes principales (PCA) (Yang and Shahabi, 2004), l'approximation symbolique de Fourier (SFA) (Schäfer and Högqvist, 2012), et encore les décompositions en ondelettes (Percival and Walden, 2000), pour n'en citer que quelques-unes. Cependant, avec l'essor de *l'apprentissage profond*, de nouvelles possibilités de représentation de séries temporelles ont émergé. Contrairement à l'apprentissage automatique traditionnel, qui repose sur des modèles préconçus et nécessite souvent un travail fastidieux de sélection et de transformation manuelle des caractéristiques, l'apprentissage profond utilise des réseaux de neurones capables d'apprendre directement de manière efficace à partir des données brutes. Ces réseaux de neurones peuvent capturer des motifs et relations complexes dans les données, offrant ainsi une grande flexibilité.

Dans la prochaine section, nous nous intéresserons aux représentations neuronales de séries temporelles en raison de leur capacité à capturer avec précision la structure sous-jacente des données, leur aptitude à construire des représentations réutilisables pour plusieurs tâches en aval, ainsi que leur flexibilité pour construire des espaces latents structurés.

F.1.3 L'apprentissage de représentations neuronales pour les séries temporelles

Les représentations neuronales offrent une alternative aux représentations classiques des séries temporelles, où la projection de l'espace temporel original vers l'espace latent est effectuée par un réseau de neurones. Ces réseaux sont constitués de couches interconnectées, chacune ajustée par des paramètres apprenables et séparées par des fonctions d'activation. Les couches apprenables peuvent prendre différentes formes, telles que les couches denses, récurrentes, convolutives ou encore d'attention. Leur utilisation en séries temporelles a notamment permis de grandes avancées pour des tâches supervisées telles que l'imputation (Cao et al., 2018), la prévision (Nie et al., 2022) et la classification (Ismail Fawaz et al., 2020). Dans le cadre de l'apprentissage de représentations, les mécanismes d'apprentissage profond sont souvent combinés dans la même architecture pour tirer parti de leurs forces complémentaires. Le choix des couches est important mais le choix de la structure de l'architecture est également cruciale pour apprendre des représentations neuronales efficaces. On distingue plusieurs types de structures pour apprendre des représentations neuronales.

(i) *Encodeur-Décodeur.* Une structure avec un encodeur et un décodeur pour apprendre une représentation latente en sortie de l'encodeur et reconstruire le signal original à l'aide du décodeur.

(ii) *Encodeur unique.* Une structure avec seulement un encodeur pour projeter les données dans l'espace latent. Ces modèles sont généralement optimisés avec une perte contrastive.

(iii) *Auto-décodeur.* Une structure avec seulement un décodeur qui prend une représentation latente apprenable initialisée aléatoirement. À l'entraînement, cette représentation est optimisée pour reconstruire le signal original.

Ces dernières années, l'apprentissage de représentations neuronales pour les séries temporelles est devenu un champ de recherche actif, donnant naissance à de nombreuses méthodes. Parmi celles-ci, nous illustrons ci-dessous les modèles d'apprentissage de représentation avec deux modèles : T-Loss et PatchTST.

- *T-Loss.* Cette méthode est basée sur des réseaux de neurones convolutifs avec une architecture composée uniquement d'un encodeur. La série temporelle est projetée sur un vecteur de représentation latente qui compresse totalement la dimension temporelle. Le modèle est entraîné grâce à une fonction de perte contrastive pour produire des représentations ayant des propriétés géométriques intéressantes, rapprochant dans l'espace latent les séries temporelles similaires et éloignant les séries statistiquement différentes. Ce modèle est utilisé pour des tâches de classification (notamment avec peu de labels),

de visualisation et de clustering. De plus, il possède de bonnes propriétés de transfert, c'est-à-dire l'application du modèle pré-entraîné à des nouveaux jeux de données.

- *PatchTST.* Cette méthode est basée sur une architecture encodeur-décodeur avec des mécanismes d'attention. Le modèle est entraîné grâce à une stratégie de masquage des patches (sous-segments) de la série en entrée, dont le but est de les reconstruire en sortie. Une représentation latente de la série est extraite juste avant la couche de sortie du modèle. Les auteurs ont démontré que la représentation latente de PatchTST était utile pour la tâche de prévision, y compris pour des jeux de données non vus pendant l'entraînement.

Ces deux modèles ne sont qu'un échantillon des avancées rendues possibles par les nombreuses contributions en apprentissage de représentations neuronales pour les séries temporelles ces dernières années. Toutefois, des problèmes ouverts subsistent dans ce domaine de recherche, et nous en aborderons certains dans la section suivante.

F.1.4 Problèmes ouverts

Malgré l'essor des méthodes de représentations neuronales pour les séries temporelles et les nombreuses améliorations par rapport à l'apprentissage traditionnel de représentations, certaines questions restent en suspens. Dans cette thèse, réalisée en collaboration avec Électricité de France (EDF), nous avons identifié plusieurs problèmes ouverts qui représentent des enjeux de recherche ayant un fort impact pour EDF.

- *Interprétabilité des représentations neuronales.* Les approches de représentations non supervisées visent à construire des représentations des séries temporelles en capturant les caractéristiques sous-jacente aux données sans connaissance d'expert ni supervision humaine. Elles ont démontré de bonnes performances pour le clustering, la classification, l'imputation des valeurs manquantes et encore la prévision. Cependant, les modèles proposés dans la littérature manquent d'interprétabilité, ce qui pose des problèmes pour la prise de décision. Dans la revue d'apprentissage de représentations de Bengio et al. (2013), il est indiqué que de bonnes représentations doivent extraire des *facteurs explicatifs* et garantir une *cohérence temporelle*. Nous constatons que, dans la pratique, les approches neuronales actuelles ne répondent pas à ces critères. En effet, les représentations neuronales sont souvent des vecteurs latents sans cohérence temporelle, et de surcroît, les valeurs de ces représentations n'ont pas de signification interprétable.

- *Modèles adaptables pour les changements de distribution.* L'un des principaux problèmes ouverts dans l'apprentissage des représentations de séries temporelles est la capacité à créer des modèles adaptatifs qui peuvent gérer efficacement les changements de distribution. Ces défis sont présents lorsque de nouveaux échantillons de données ou des contextes temporels différents sont introduits, exigeant des modèles qu'ils s'adaptent aux changements de distribution, sans nécessiter un ré-apprentissage approfondi. Dans ce contexte, il est intéressant de concevoir des modèles basés sur des représentations, capables de s'adapter dynamiquement à de nouvelles données tout en conservant des performances élevées. Le besoin de modèles robustes et flexibles qui répondent aux changements temporels et aux nouveaux échantillons est très présent dans les applications industrielles.

- *Capture d'une représentation neuronale à partir de séries temporelles non alignées et irrégulières.* Dans l'analyse des séries temporelles, la variété, l'hétérogénéité et le nombre croissant de capteurs déployés présentent de nouveaux défis que les méthodes de représentations actuelles ne parviennent souvent pas à relever. En effet, en industrie, les séries temporelles proviennent souvent de sources diverses, présentant potentiellement un échantillonnage irrégulier et pouvant contenir des valeurs manquantes. De plus, les capteurs peuvent éventuellement ne pas être alignés entre eux. Pour résoudre cela, les modèles existants continus en temps semblent être une solution pour traiter de telles données. Cependant, en pratique, ils peinent à extraire des représentations efficaces des séries temporelles. Nous observons notamment que leurs performances sont nettement inférieures aux modèles conçus pour les grilles temporelles discrètes et régulières. Il y a donc un besoin de créer des approches neuronales capables de construire des représentations à partir de séries temporelles non alignées et irrégulières tout en garantissant des performances robustes pour divers scénarios en aval.

Dans la section suivante, nous résumons les modèles que nous avons proposés pour répondre à ces questions ouvertes.

F.2 Contributions

F.2.1 Représentations neuronales interprétables de séries temporelles : application en classification

Qu'est ce que l'inteprétabilité ? L'interprétabilité est un concept qui ne fait pas l'unanimité (Lipton, 2018), la confusion résultant des différentes significations de l'interprétabilité et de l'explicabilité. Pour les modèles de séries temporelles, Wang

(2021) propose une taxonomie claire de l'interprétabilité présentée dans la figure F.3.

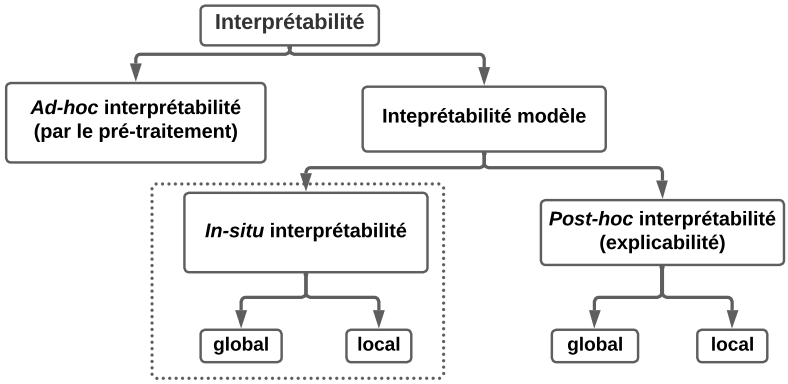


Figure F.3: Taxonomie des modèles de séries temporelles interprétables introduite dans Wang (2021).

L'interprétabilité post-hoc fait référence aux méthodes qui analysent le modèle après l'apprentissage et qui sont généralement indépendantes du modèle. Elle est souvent liée au domaine de recherche en intelligence artificielle explicable (XAI). Toutefois, les méthodes post-hoc n'expliquent la décision que pour une instance en particulier, et des méthodes supplémentaires sont nécessaires pour comprendre le modèle dans son ensemble. À contrario, les modèles interprétables in-situ sont interprétables par conception. Pour ces modèles, l'interprétabilité découle directement du modèle sans qu'aucun autre processus ne soit appliqué après la phase d'apprentissage (Rudin, 2019; Wang, 2021). Le niveau de cette interprétabilité peut être local ou global. L'interprétabilité globale est définie comme un modèle facile à comprendre pour un humain et nécessitant une faible complexité de calcul. En revanche, l'interprétabilité locale est un moyen d'interpréter la décision d'un modèle pour un cas particulier (Lipton, 2018). L'interprétabilité globale implique souvent l'interprétabilité locale, mais l'inverse n'est pas vrai.

Contexte. Dans ce travail, nous nous concentrons sur l'interprétabilité globale in-situ et développons une méthode neuronale interprétable discrète pour la représentation des séries temporelles. En série temporelle, les modèles issus de l'apprentissage de représentation symbolique sont souvent décrits comme interprétables. Cependant, les informations capturées par les symboles sont limitées et ne permettent pas une interprétation globale de la représentation. D'autre part, l'essor des méthodes d'apprentissage de représentation neuronale permet d'atteindre des performances impressionnantes pour les tâches en aval, mais ces méthodes ne sont pas interprétables. Notre contribution vise à combiner le meilleur des deux approches : une représentation neuronale interprétable.

Définir des pré-requis pour construire une représentation neuronale interprétable. La première contribution de ce travail est de définir des pré-requis pour établir un pont entre les représentations symboliques et les représentation neuronales afin de construire des représentations symboliques neuronales interprétables pour les séries temporelles. La représentation doit : (i) Être discrète. (ii) Respecter la consistence temporelle. (iii) Être décodable. (iv) avoir un encodeur et un décodeur qui respectent l'équivariance par translation. (v) Être ajustable en fonction des fréquences.

Construction d'un modèle neuronal respectant les pré-requis. Dans un deuxième temps, nous proposons un nouveau modèle neuronal non supervisé qui répond aux pré-requis définis en amont. Le modèle neuronal proposé est basé sur une architecture d'auto-encodeur avec un mécanisme de quantification vectorielle dans l'espace latent (Gersho and Gray, 1991; van den Oord et al., 2017).

Expériences et application en classification. Pour démontrer les qualités de l'architecture proposée, nous réalisons une tâche de classification par dessus la représentation neuronale symbolique apprise. Nous démontrerons qu'un simple classifier linéaire par dessus les représentations permet de discriminer efficacement tout en préservant l'interprétabilité. Nous démontrons également par des expériences qualitatives que la décision de classification est à la fois interprétable au niveau global et local.

Pour résumer. Nos principales contributions sont résumées ci-dessous :

- Nous définissons et formalisons les pré-requis fondamentaux pour construire des représentations neuronales symboliques interprétables pour les séries temporelles.
- Nous proposons une architecture de réseau de neurones non supervisé qui satisfait ces pré-requis.
- Nous utilisons ces représentations pour des tâches de classification tout en préservant l'interprétabilité de la représentation et nous nous évaluons par des expériences quantitatives et qualitatives.

Le Naour, E., Agoua, G., Baskiotis, N., and Guigue, V. **Interpretable time series neural representation for classification purposes.** *IEEE 10th International Conference on Data Science and Advanced Analytics (IEEE DSAA)* 2023.

F.2.2 TimeFlow : Modélisation continue des séries temporelles pour l'imputation et la prévision avec des représentations neuronales implicites

Contexte. De nombreux modèles à l'état de l'art, tels que les transformers, ont été conçus pour des grilles denses et régulières (Wu et al., 2021; Nie et al., 2022; Du et al., 2023). Cependant, ces modèles ont du mal à gérer les données irrégulières et souffrent souvent d'une dégradation significative des performances en leur présence (Chen et al., 2001; Kim et al., 2019).

Motivations. Notre objectif est d'explorer des alternatives aux modèles basés sur des grilles régulières et être capable de traiter, dans un cadre unifié, les tâches d'imputation et de prévision pour des séries temporelles irrégulières, arbitrairement échantillonnées et non alignées. Les modèles continus dépendants du temps (Rasmussen and Williams, 2006; Garnelo et al., 2018; Rubanova et al., 2019) offrent une telle alternative. Toutefois, jusqu'à présent, leurs performances ont été nettement inférieures à celles des modèles conçus pour des grilles discrètes régulières. Il y a quelques années, les représentations neuronales implicites (INR) sont apparues comme un outil puissant pour représenter les images en tant que fonctions continues de coordonnées spatiales (Sitzmann et al., 2020; Tancik et al., 2020) avec de nouvelles applications récentes pour la génération d'images (Dupont et al., 2022) ou la modélisation de systèmes dynamiques (Yin et al., 2023).

Contributions. Dans ce travail, nous exploitons le potentiel des modèles INR conditionnels dans le cadre d'une approche de méta-apprentissage en proposant TimeFlow (Figure F.4). TimeFlow est modèle unifié conçu pour modéliser des séries temporelles continues et pour traiter les tâches d'imputation et de prévision avec des observations irrégulières et non alignées. Nos principales contributions sont les suivantes :

(i) Nous proposons un nouveau modèle qui excelle dans la modélisation des séries temporelles en tant que fonctions continues du temps, acceptant en entré des pas de temps arbitraires, permettant ainsi le traitement de séries temporelles irrégulières et non alignées pour les tâches d'imputation et de prévision. Il s'agit de l'une des toutes premières tentatives d'adaptation des INR qui permet de traiter efficacement les tâches d'imputation et de prévision dans un cadre unifié.

(ii) Nous avons effectué des comparaison quantitatives approfondies avec les modèles continus et discrets les plus récents. Nous démontrons que notre approche est plus performante que les approches d'apprentissage profond à l'état de l'art (SOTA) continues et discrètes pour l'imputation. En ce qui concerne

les prévisions à long terme, TimeFlow surpasse les modèles continus existants à la fois sur des échantillons réguliers et irréguliers. De plus, il performe de manière similaire que les modèles discrets SOTA sur des séries temporelles régulièrement échantillonnées tout en permettant une plus grande flexibilité pour les échantillonnages irréguliers, ce qui lui permet de faire face à des situations où les modèles discrets échouent. En outre, nous prouvons que notre méthode traite sans effort de nouvelles séries temporelles et des nouvelles fenêtres de contexte, ce qui rend TimeFlow adapté aux applications du monde réel.

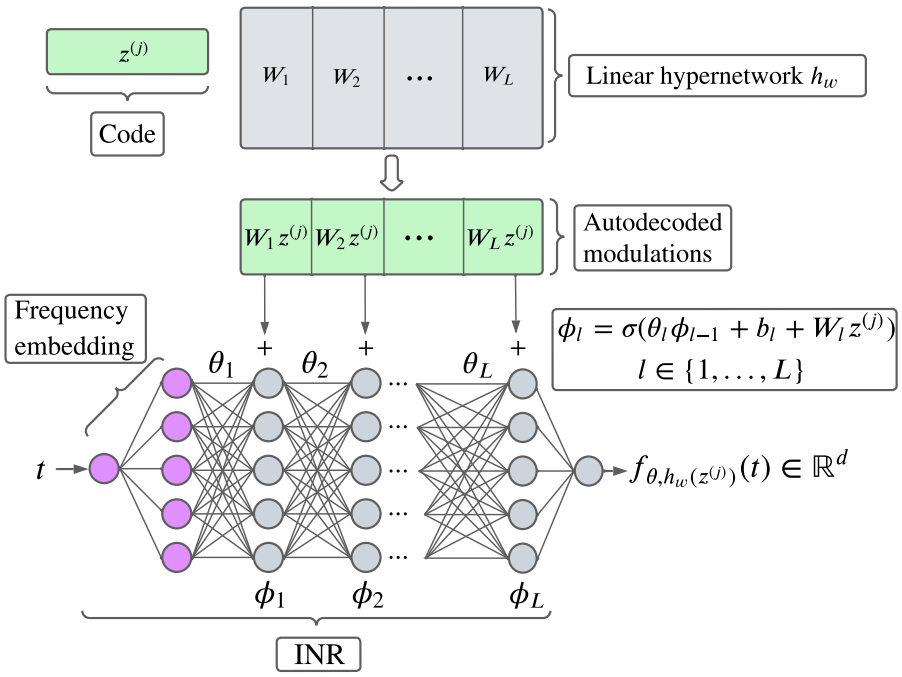


Figure F.4: Vue d'ensemble de l'architecture TimeFlow. Forward pass pour approximer la série temporelle $\mathbf{x}^{(j)}$ au pas de temps t. $\mathbf{z}^{(j)}$ représente la représentation apprise, σ la fonction d'activation ReLU, ϕ les couches de l'INR et les autres symboles non définis sont des paramètres apprenables.

Le Naour, E., Serrano, L., Migus, L., Yin, Y., Agoua, G., Baskiotis, N., Gallinari, P., and Guigue, V. **Time Series Continuous Modeling for Imputation and Forecasting with Implicit Neural Representations.** *Transactions on Machine Learning Research (TMLR)* 2024.

F.2.3 Exploration des capacités des représentations apprises par TimeFlow

Contexte. Dans la contribution précédente, nous avons démontré que TimeFlow peut capturer des représentations neuronales à partir de séries temporelles non alignées et irrégulières.

Motivations. Cette dernière contribution vise à explorer la qualité des représentations apprises par TimeFlow et à démontrer leur utilité pour des tâches en aval. Nous approfondissons l'étude des capacités d'apprentissage de représentations de TimeFlow en nous concentrant d'abord sur l'interprétabilité de l'espace latent, la compréhension de sa structure et son adaptabilité aux changements de distribution. Ensuite, nous examinons comment les représentations apprises peuvent être utiles pour des tâches en aval, notamment pour la génération de données synthétiques.

Contributions. Les différentes contributions de ce travail sont les suivantes :

(i) Nous commençons par explorer le comportement de l'espace latent entre deux représentations apprises. En interpolant, nous pouvons observer les transitions entre les représentations en visualisant des représentations intermédiaires dans le domaine initial des séries temporelles (à travers le décodeur). Cette expérience nous aide à comprendre comment l'espace latent est structuré, ce qui nous donne un aperçu de la manière dont TimeFlow capture et encode les caractéristiques importantes des séries temporelles.

(ii) Ensuite, nous étudions la sensibilité de TimeFlow aux perturbations des représentations. Nous observons comment ces perturbations affectent les séries temporelles décodées en ajoutant un bruit gaussien à des couches cachées de l'INR.

(iii) Puis, nous examinons la distribution des représentations latentes pour des séries temporelles observées à différentes périodes. En utilisant l'analyse en composantes principales, nous visualisons ces distributions et analysons l'impact des changements de distribution temporelle sur l'espace latent.

(iv) Dans la dernière section, nous explorons le potentiel de la génération de séries temporelles à l'aide des représentations apprises dans l'espace latent. Pour cela, nous utilisons un modèle génératif entraîné sur l'espace de représentations pour créer de nouveaux échantillons. Ces échantillons peuvent être ensuite décodés pour n'importe quel pas de temps à travers le décodeur de TimeFlow pour obtenir de nouvelles séries temporelles. Nous démontrons que les séries temporelles générés sont diverses et fidèles aux "vraies" séries.

Grâce aux explorations menées dans ce travail, nous validons les capacités de TimeFlow à extraire des représentations sémantiquement riches et structurées. De plus, nous démontrons leur utilité pratique pour la génération de séries temporelles synthétiques.

F.3 Conclusion

En conclusion, cette thèse explore en profondeur l'apprentissage de représentations neuronales pour les séries temporelles, un domaine en plein essor. Nous avons d'abord présenté l'émergence de ces représentations à travers divers paradigmes tels que l'apprentissage contrastif et l'apprentissage par reconstruction. Nous avons mis en lumière les avancées majeures permises par ces nouveaux modèles, tout en soulignant les questions de recherche encore ouvertes.

Nos contributions sont les suivantes :

- Nous avons d'abord abordé la nécessité d'obtenir des représentations neuronales interprétables, particulièrement pour la tâche de classification. À cet effet, nous avons proposé une nouvelle architecture utilisant une quantification vectorielle dans l'espace latent, répondant à cinq prérequis essentiels pour l'interprétabilité des représentations neuronales. Les qualités de cette représentation ont été évaluées quantitativement et qualitativement.

- Pour traiter les séries temporelles irrégulières et non alignées, nous avons introduit le modèle TimeFlow, basé sur des représentations neuronales implicites optimisées par méta-apprentissage. Ce modèle s'adapte efficacement à de nouveaux échantillons et contextes, démontrant sa capacité à réaliser des tâches d'imputation et de prévision dans divers scénarios.

- Enfin, nous avons évalué les représentations apprises par TimeFlow, démontrant une bonne structuration dans l'espace latent et leur utilité pour la génération de séries temporelles.

D'un point de vue industriel, les travaux de cette thèse répondent à des besoins clés d'EDF en améliorant l'interprétabilité des modèles de deep learning, en proposant des représentations adaptées aux séries irrégulières et en permettant de construire des modèles adaptables dans le temps et pour de nouveaux contextes. Toutes ces avancées répondent à des besoins pratiques dans le secteur de l'énergie. Cependant, il est important de noter certaines limitations à ces travaux. Le principal obstacle à l'utilisation pratique des modèles proposés est qu'ils ne sont pas encore conçus pour accepter des séries temporelles multivariées. Ainsi, le développement de modèles de représentations conditionnables par des variables externes constitue une piste de recherche prometteuse pour l'avenir.