Recurrent Neural Networks for Time Series Prediction

Bernardo Pérez Orozco

Balliol College

University of Oxford

A thesis submitted in partial fulfilment for the degree of

Doctor of Philosophy

Trinity Term 2019

To Alicia, Bernardo, Stéphanie and Alex:

Home is wherever you are.

Para Alicia, Bernardo, Stéphanie y Alex:

Mi casa son ustedes.

Abstract

Attempting to predict the future long precedes the time where we could first quantify much of our present. But nowadays, performing long-term and reliable time series forecasting remains as relevant as ever across a vast number of application domains, including astronomy, healthcare, meteorology, physiology, energy systems, econometrics, finance and sociology, amongst many others. Practitioners act upon these forecasts, and it is thus mandatory that such forecasts are accompanied by well-calibrated uncertainty assessments so as to enable a better-informed decision-making process.

This thesis proposes a novel time series forecasting framework, namely a scalable and general-purpose recurrent neural network approach which provides probabilistic predictions. Our method recasts time series forecasting as a symbolic sequence learning task by introducing a discrete encoding scheme over measurements. Such symbol sequence tasks have been most successfully dealt with by Long Short-Term Memory (LSTM) models and extensions, such as the sequence-to-sequence model, which form the core of our methodology. This presentation is accompanied by a large-scale experiment where we show the effectiveness of our method over 45 different datasets in comparison with state-of-the-art baselines, such as Gaussian Process Regression (GPR).

Crucially, our framework also offers a number of extensions that address a broad range of use cases beyond univariate time series prediction. For example, we show how predictive densities over the occurrence of critical events within a predictive horizon can be inferred from our forecasts. Whereas this requires access to labelled data, we provide a complementary method for recognising and labelling prototypical time series segments on the basis of the feature embedding learnt by our neural networks. Such motifs can subsequently be used to build predictive densities and characterise time series. Lastly, we extend our framework to allow for multivariate forecasting, enabling efficient inference of joint predictive densities over multiple correlated time series.

Finally, time series data scarcity constrains the choice of a forecasting model, especially as over-parameterised models such as recurrent neural networks cannot generalise from limited data. In this thesis, we propose a novel approach to infer accurate and reliable predictive distributions from scarce time series data. We achieve this through a recurrent neural network approach that infers a joint feature representation over the space of quantised time series by means of a compilation of auxiliary datasets. Cross-task knowledge transfer is readily enabled by assuming an ordinal regression approach, as quantised time series can then be characterised in terms of their ordinal bin sequences.

Acknowledgements

Pursuing a doctorate is as much an intellectual journey as one of personal growth. These years at Oxford have been truly life-changing, in great part due to all the special people I have been fortunate enough to cross paths with, and to whom I would like to dedicate a few words.

First and foremost, I cannot emphasise enough how tremendously lucky I was to have Prof. Steve Roberts as both a guide and a companion throughout this journey. Steve, thanks so much for being so welcoming and patient from the very first time we met, and all the way through the very end, walking by my side through the good and the rough. It has been an unforgettable journey, full of lessons that I will carry with me for the rest of my life, and this is in great part thanks to you.

I would also like to thank Prof Niki Trigoni and Prof Wenwu Wang for their comprehensive feedback in making this work better.

Thanks are also due to everyone at the Machine Learning Research Group at Eagle House: Adam Cobb, Alessandra Tosi, Davide Zilli, Elmarie van Heerden, Gabriele Abbati, Henry Kenlay, Ivan Kiskin, Jaleh Zand, Jonny Downing, Justin Bewsher, Kyriakos Polymenakos, Mike Osborne, Nafisa Sharif, Olga Isupova, Samuel Kessler, Stefan Zohren, Tom Pretty, Xiaowen Dong, Yunpeng Li, Zihao Zhang. Thank you all for the extremely valuable discussions, ideas and support throughout these years. Special thanks are due to Gabriele, Henry, María and Tom for proofreading this thesis.

Lastly, I would also like to acknowledge the National Council for Science and Technology in Mexico (CONACYT) and EDUCAFIN-Guanajuato for their financial support. I am also very grateful to Prof Carlos Ramírez for his advice during the early stages of my graduate studies.

****

A huge thank you is due to Dr Davide Zilli. Davide, I really don't have enough words of gratitude for all your support throughout these years, in so many aspects and even prior to the start of this adventure. None of this would have been possible without all your advice and sincere friendship.

I would also like to thank Andrea Gerwer and Ernesto Ocampo, who have truly become family to me – now I've even become an uncle! Thanks for all the family Sundays, for all the love and for being such a foundational support since the beginning of this journey.

I am also very grateful to Paula Pérez and Léonard Berrada: Oxford was a whole different city thanks to you, I'm too lucky to have found you. Thanks for all the great times. The best is yet to come!

Massive thanks are also due to Gabriele Abbati, Kyriakos Polymenakos, Henry Kenlay and Tom Pretty: having you in my daily life has been an absolute pleasure. The lab is *the lab* thanks to you, fam.

I would also like to thank María del Río, Marcela Mendoza and Leila Walker, who were a fundamental source of support. Balliol was at its most Mexican thanks to you. Thanks for all the love, for the great times, for the scolding and the coffee breaks!

I am also extremely grateful to Cristina Bueno, Rodrigo Domínguez, Angélica Martínez and Rebeca Gutiérrez: you were here way before this, and you stayed all the way throughout. Thanks for always being a message away.

When leaving Mexico, I never imagined I would find a second family an ocean away. Thank you all for your sincere friendship and for the great times. Europe is now home thanks to all of you: Ignacio Funke and Antonio Lombardo, for so many adventures; my dearest Greeks, Ada Alevizaki, Nikitas Rontsis and Peny Zacharopoulou; my *tía* Victoria Dittmar; Carlos Pérez Ricart and Leonore Lukcshy; Javier Amate; Marta Bautista; Pepe Manzano; Álvaro Silva; Wendy Poole, you've been great; Julia Wiedmann and Maximilian Schleich; Alice Zannin, Miriam Pedrós, Cris Silgo, Peter Fernández, Juliá Camps, Núria Sánchez, Teresa Vilanova and Cindy Barrantes; all my great friends at Balliol: Sofía Castelló, Manos Alexis, Jerome Jochems, Tomislav Vladisavljevic, Yayoi Teramoto; and Federico López, Cirenia Chávez, David Pérez and Caro Ramos for the great times we had at strengthening the Mexican community in the UK.

Last, but very certainly not least, I would like to express my most heartfelt gratitude to the most important people in my life: my parents and my sister. No achievement has any meaning without your support and your love. Thank you for always being here for me, even if we are an ocean away. Thanks for absolutely everything.

A mis padres y mi hermana: ninguno de mis logros tiene sentido alguno sin ustedes. Son lo más importante que tengo, aún si estamos a un océano de distancia. Gracias por absolutamente todo, todos los días desde hace tantos años.

Contents

List of Figures

List of Tables

1 Introduction

Predicting the future is arguably one of the most challenging tasks of humanity. Indeed, the origins of prediction are as fundamental as when humanity first understood the notion of time itself. Admittedly, attempting to predict the future long precedes the time where we could first quantify much of our present. Take temperature, for instance. It was not until the late 16^{th} and early 17^{th} centuries that the first thermometers were built. Yet, humanity's need for *forecasting* temperature dates back at least to the earliest agricultural civilisations, who attributed the behaviour of weather to deities. Temperature is an example of a quantity that varies over time, also known as a *time series* (Box et al., 2015).

Nowadays, the task of performing long-term, reliable time series forecasting remains as relevant as ever in both academia and industry. Application domains are great in number, including meteorology, energy systems, astronomy, finance, dynamical systems, physiology and many others. In these fields, practitioners use forecasts to make decisions. It is therefore mandatory to offer any and all uncertainty measurements alongside our predictions. Indeed, we argue that decision-makers benefit from assessing such uncertainty, since point-forecasts, or otherwise overconfident predictions that underestimate the predictive variance, could lead them to make erroneous decisions.

In this thesis, we propose a novel time series forecasting framework based

on recurrent neural networks that is able to infer accurate and robust long-term predictions and their associated measures of uncertainty. Our method and its extensions address a number of key relevant challenges in the time series forecasting literature, which we outline below.

1.1 Challenges in time series forecasting

Time series forecasting is an arduous task that goes well beyond providing a real number for various time instants. When developing time series forecasting methodologies, there also exist predictive and computational performance challenges that must be addressed for the framework to be purposeful. We list some of these challenges below.

Well-calibrated uncertainty quantification. When we offer time series forecasts, it is crucial that they are accompanied by adequate measures of uncertainty that correspond with real-world probabilities. This uncertainty arises not only from noise in the observed measurements, but also as a result of our ignorance about the correctness of our modelling decisions. Uncertainty assessments allow practitioners to incorporate model honesty in their decision-making process, which further makes a case for these assessments to correspond with real-world probabilities. Indeed, we argue that models that confess their ignorance are less harmful than overconfident models, which can lead analysts to make erroneous decisions. Probability theory offers a principled framework to assess such uncertainty in the form of *predictive probability distributions*, as we will discuss in Section 2.2.

Flexible predictive descriptions. As we will also see in Chapter 2, several time series forecasting methods make assumptions that lead to unimodal Gaussian predictive distributions. One problem with these is that they may be unfit to describe long-term behaviour, especially in systems where small changes

in the short-term may induce vastly different outcomes in the long-term. We thus require frameworks to be able to adapt their forecast descriptions over time, adjusting their shape aspects as predictive horizons grow.

Scalability with dataset size. Time series datasets vary widely in size. For some, only a handful of measurements may be available. For others, thousands of observations may be at hand. We require methods that can both, generalise from scarce data, but also be computationally efficient to infer when provided with large datasets. In the main, Bayesian nonparametric methods, such as the Gaussian Process (GP) (Rasmussen and Williams, 2006), have become state-of-the-art thanks to their ability to infer exact full predictive distributions, even from little data. In their original formulation, nevertheless, these methods scale poorly as the dataset size grows, in which case practitioners recur to approximations, such as the Sparse GP (Snelson and Ghahramani, 2006), or other families of methods, such as neural networks.

Query-based forecasting. It is often the case that decision-makers are interested in resolving queries about forecasts. For example, they might wonder about the time of occurrence of an event, its duration, or the number of times it will occur within a given time horizon. How to resolve such queries can often be an open-ended problem, especially in the case where the queries of interest have little-to-no labelled data at hand.

Learning long-term, non-linear relationships. One further desirable aspect of time series models is that they should be capable of learning long-term temporal dependencies. This refers to the ability of detecting the effect of measurements that are seemingly "far apart". In a similar fashion, these models should be able to capture any non-linear relationships between their inputs and the observed time series measurements.

Multivariate time series. In a given system, several quantities of interest may be observed simultaneously and possibly be correlated. When forecasting

such types of multivariate systems, the simplest scenario assumes different time series channels are independent, which fails to capture any potential information that is of mutual relevance between channels. It is thus often convenient to model the correlation between targets, but admittedly it may also come at the expense of greater computational cost.

1.2 Outline of contributions

The core of this thesis develops a novel, recurrent neural network-based framework for probabilistic time series forecasting. Deep and recurrent neural network models, such as the Long Short-Term Memory (LSTM) (Schmidhuber et al., 2007), have recently gained significant attention in the domain of sequential learning tasks, for example, those that arise in the Natural Language Processing community, e.g. (Graves, Mohamed, and Hinton, 2013) and (Bahdanau, Cho, and Bengio, 2014). This could be related to their comparatively good scalability with respect to dataset size, in addition to the increasing amount of software and hardware resources available for their development.

To the best of our knowledge, however, these models have not seen the same success in regression tasks. A potential reason is that exact inference of fully probabilistic versions, such as Bayesian neural networks, is intractable, and even approximate inference can be too computationally expensive to justify their usage in comparison with other methods, such as the Gaussian Process (GP).

In light of this, the contributions that we offer in this thesis are as follows:

- **A novel time series forecasting framework.** We develop a novel recurrent neural network framework for time series prediction that is capable of producing fully (though approximate) probabilistic forecasts. Our approach introduces a time series quantisation step that recasts the time series forecasting task as an ordinal (auto-)regression one, i.e. transform-

ing the prediction task into a sequential classification problem. Ordinal regression, also known as ordinal classification in some literature, tasks lie between pure classification and regression by requiring data labels drawn from an ordered set. Such an ordinal approach enables our model's predictive distributions to flexibly adapt its shape parameters for long-term forecasting. Furthermore, by incorporating recent developments in the Machine Learning literature, we also assess uncertainty in a grounded, though approximate, manner (Gal and Ghahramani, 2016b). We then provide an exhaustive evaluation against other state-of-the-art time series forecasting methods across 45 different datasets, and encompassing a variety of metrics that assess our models in terms of predictive accuracy and well-calibrated uncertainty quantification.

- **Open source implementation.** We also offer an open source implementation for Python 3.7. Our library is an extensible software project that centralises a wide range of time series methods and frequently-used data evaluation and preprocessing routines. It can easily be expanded to support other time series methods, and it currently supports GPs and linear-Gaussian AR(p) models. Ordinal time series forecasting (TSF) for Python 3.7 is available here: `https://github.com/bperezorozco/ordinal_tsf` and we offer full documentation in Appendix C.

- **A framework for querying and interpreting time series forecasts.** We directly tackle the task of giving answers to open-ended queries that can be derived from probabilistic forecasts. We propose a set of routines to construct predictive probability distributions over critical events, assuming access to labelled instances of such events; when such data is unavailable, we also provide a set of routines to extract and visualise salient time series structural information, such as prototypical motifs, that can serve to

describe critical events.

- **A multivariate time series framework.** As motivated in the previous section, time series may often interact or possess some degree of correlation. We thus extend our univariate time series forecasting framework to enable the inference of full joint predictive distributions for multivariate time series.

- **The General Unified Model (GUM): a zero-shot and few-shot approach to time series forecasting.** Time series datasets vary widely in size, with some only possessing a handful of observations. Inferring a neural network model from scratch for this type of datasets may result in poor predictive performance. In this thesis, we directly address this issue by proposing a novel GUM for time series forecasting. Our approach draws inspiration from the multi-task learning literature and consists in extending our ordinal regression approach to learn a joint latent representation of an ample catalogue of auxiliary time series. By assuming an ordinal approach, different time series can be characterised in a single joint feature space, which then acts as a basis for cross-task knowledge transfer. We assess this framework in two different settings: a *zero-shot* evaluation, where we show the learnt models can infer accurate predictive distributions for new, unseen time series, even before any further learning takes place; and a *few-shot* setting, where our models act as the initial state of the model before further learning takes place.

1.3 Thesis layout

The rest of this thesis is structured as follows. In Chapter 2, we review foundational concepts from the time series, Bayesian statistics and neural network corpora that will reappear abundantly throughout this work. Setting out this theoretical framework paves the way to Chapter 3, where we introduce our novel

time series forecasting methodology: the Memory-endowed Ordinal Regression Deep neural network (MOrdReD). Our method combines an ordinal regression approach (as introduced in Section 3.1) with state-of-the-art recurrent neural network models (which we introduce in Sections 3.2 and 3.3) to achieve robust and probabilistic long-term forecasting. We then offer an exhaustive evaluation in Section 3.5, where we compare MOrdReD's performance against state-of-the-art time series forecasting baselines, across an ample range of datasets and metrics associated with predictive accuracy and uncertainty quantification.

Whereas Chapter 3 presents our framework in the context of one-dimensional time series forecasting, in Chapter 4 we show how it can be extended beyond univariate prediction with two separate contributions. The first part of this chapter shows how predictive densities of structured queries can be derived from MOrdReD's forecasts. We showcase this in two scenarios: first, in Section 4.1, we show how to construct predictive densities over the timing of occurrence of critical events when we have access to labelled data. Then, in Section 4.2, we build upon this by demonstrating how characteristic prototypical patterns can be extracted by analysing the feature space learnt by MOrdReD. This analysis also provides the reader with a set of tools that serve to visualise and shed light on the feature embeddings of our models. Finally, we conclude Chapter 4 by showing how our framework can be extended to multivariate time series forecasting.

Neural networks are known to suffer from poor generalisation in the absence of sufficient data. This problem is of great relevance in the context of time series forecasting, as we often encounter time series where only a handful of measurements are available, thus potentially limiting the applicability of our method. In Chapter 5, we directly address this by proposing a General Unified Model (GUM) for time series forecasting. By combining concepts from the multi-task learning literature with an ordinal regression approach, we show how a single model that learns to predict a joint dataset of auxiliary time series can extrapolate this

knowledge to novel, unseen tasks. In an extreme case, this enables MOrdReD to forecast other time series for which there is no training data at hand, in a zero-shot fashion, as we show in Section 5.3.

Finally, we conclude this thesis in Chapter 6, with a summary of our contributions and a discussion of future work avenues.

2 Neural networks for time series forecasting

2.1 Time series forecasting

A time series is a list of measurements taken sequentially in time (Box et al., 2015). Examples of time series include: offshore tide height, electrocardiograms, encephalograms, air temperature and humidity at a given location, a country's population size, audio recordings, the number of sunspots per year, foreign currency exchange rates, interest rates, and chaotic systems such as the Lorenz attractor, to name a few.

In these varied settings, inferring models that explain the observed data well, whilst also generalising to *out-of-sample* measurements (such as those in the *test* dataset), is therefore of great relevance. Indeed, the interest in time series forecasting may arguably date back to the very dawn of human civilisation, with attempts at predicting the weather for agricultural purposes. The field also encompasses a rich history of statistical methods for formal analysis, including the development of least-square regression methods by Legendre and Gauss more than two centuries ago.

2.1.1 The task

Consider a finite univariate time series with T real-valued and evenly-sampled measurements, $\mathbf{x} = [x_1, \ldots, x_t, \ldots x_T]$. As is often the case for regression tasks, let us start by regarding each observation $x_t \in \mathbb{R}$ as the output response to M input regressors $\mathbf{z}_t \in \mathbb{R}^M$ subject to random additive disturbances ε_t:

$$x_t = f(\mathbf{z}_t; \theta) + \varepsilon_t,$$

where f is the model and θ the model parameters.

Choosing the regressors (or covariates) $\mathbf{z}_t$ is largely problem-dependent, but they may include observations in the recent past, e.g. $x_{t-1}, \ldots, x_{t-P}$ for a look-back horizon P, in addition to relevant *exogenous* measurements. If the time series possesses any sort of (quasi-)seasonality, the regressors can also include further *endogenous* observations from the distant past, e.g. $x_{t'}, t' \ll t$.

To illustrate, consider the task of predicting the net daily solar energy output x_t in the city of Oxford at day t. Relevant information can be found in the recently available observations $x_{t-1}, \ldots, x_{t-P}$, and in any other pertinent hardware (number and type of solar panels) and weather variables (temperature, cloud coverage, among others) available at time t. Some information could also be gained by examining the annual lookback for the last few K years, e.g. $x_{t-365}, \ldots, x_{t-365K}$[1].

2.1.2 The model

The model f encapsulates the temporal evolution of x_t and its relationship to the input regressors $\mathbf{z}_t$. Our aim is to propose a hypothesis f that captures complex, non-linear relations between $\mathbf{z}_t$ and x_t, whilst also remaining computationally efficient to infer from the observed measurements. Furthermore, since inputs and outputs are observed sequentially, it is crucial that the model f also be able to

[1]Subject to leap year adjustments.

characterise the temporal dependencies between observations.

For a single set of observations, we can infer multiple hypotheses that explain the measurements under some task-dependent criterion. It must be noted, however, that the goal of inferring the model f is not only to explain the observed measurements, but also crucially, to *generalise* to out-of-sample predictive horizons. To this end, scholars often engage with a principle of parsimony, recognised as *Occam's razor*. This principle states that, given competing hypotheses that explain the data well, one should choose the simplest one. Indeed, models that fit the data with a substantial noise component ε_t "too well" may be *overfitting*, or failing to capture the underlying relationship between regressors and time series. Guarding models against overfitting is hence a crucial requirement for time series forecasting tasks.

We shall also distinguish between parametric and nonparametric models. We refer to f as a *parametric* model if the vector $\theta \in \mathbb{R}^{|\theta|}$ is finite-dimensional (Murphy, 2012). An example is when f is assumed to be a linear function of $\mathbf{z}_t$, as in ordinary least-squares linear regression. In contrast, we shall refer to f as a *nonparametric* model if θ can be infinite-dimensional, i.e. when the number of parameters in the model grows as we observe more data (Ghahramani, 2013). This is the case, for example, of the Gaussian Process (Rasmussen and Williams, 2006), whose predictions rely on the inverse of a constructed covariance matrix whose rank grows with the number of points in the dataset.

In general, nonparametric models are more flexible, but exact inference may become computationally intractable for very large datasets. On the other hand, prediction with parametric models is usually more efficient, especially in the presence of big datasets – at the expense of making stronger assumptions about the nature of the data through the structure and number of parameters of the model class of f. Such assumptions are collectively known as the **inductive bias** (Murphy, 2012).

2.1.3 Predictive uncertainty

In addition to being parsimonious and capturing non-linear relationships and temporal dependencies, it is essential that our forecasts incorporate all associated measures of predictive uncertainty, as this is a fundamental aspect in the decision-making process. Being limited to point forecasts can be troublesome and, indeed, lead to ill-informed decisions. Practitioners thus benefit from model honesty, i.e. better decisions can be made when models confess their uncertainty.

In the main, quantifying uncertainty for long-term predictions has been a fundamental challenge. This uncertainty may arise from a number of sources, such as the noise in the observed regressors and measurements, in addition to our ignorance about the structure and number of parameters θ of the model $f(\mathbf{z}_t; \theta)$ (Bishop, 1995). When considering a long-term forecast for P_h steps ahead after observing T measurements in a time series $\mathbf{x}$, uncertainty in earlier predictions must also be propagated to avoid future overconfident forecasts.

Indeed, constructing rich *predictive probability distributions* has been extensively researched in Bayesian statistics (Bernardo and Smith, 2007; Jaynes, 1996; Rasmussen and Williams, 2006; Barber, 2012), signal processing (Durbin and Koopman, 2012), machine learning (Bishop, 2006; Mattos et al., 2016; Damianou and Lawrence, 2013) and other fields. This includes the literature on parametric models, such as polynomial regressors and the family of Markovian state-space models (Durbin and Koopman, 2012).

Over the last decade, nonparametric probabilistic approaches such as the Gaussian Process (GP) (Rasmussen and Williams, 2006), and extensions such as the Recurrent Gaussian Process (RGP) (Mattos et al., 2016) and the Gaussian Process State Space Models (GPSSM) (Turner, Deisenroth, and Rasmussen, 2010), have increasingly dominated the time series modelling literature over the last decade, especially as classic Markov models (including autoregressive pro-

cesses and the like) can be seen as special cases of the GP.

Let us note simply that, powerful though the Gaussian Process (GP) approach is, the standard GP model makes the tacit assumption that the additive disturbances ε_t are normally distributed, leading to a Gaussian conditional predictive posterior distribution. This assumption can be restrictive in the long-term forecasting scenario, where short-term forecast uncertainty may induce multiple plausible forecasts in the long-term.

Furthermore, the original formulation of the GP, in addition to extensions such as the RGP and GPSSM, suffer from poor scalability, which can hinder the use of the large amounts of data readily available in many time series forecasting tasks. Lastly, to obtain best performance, expert knowledge, or extensive optimisation may be needed to refine the kernel function at the heart of the GP.

2.1.4 Motivation for our framework

In parallel to these developments, deep and Recurrent Neural Network (RNN) models have gained significant attention in the domain of sequential classification tasks, such as those that lie at the core of the Natural Language Processing (NLP) corpus. Neural networks, despite their parametric complexity, scale well with dataset size. Furthermore, the increasing availability and development of algorithms, data, hardware and software have enabled these models to become the state-of-the-art in sequential classification tasks such as speech recognition (Graves, Mohamed, and Hinton, 2013) and machine translation (Bahdanau, Cho, and Bengio, 2014).

In the context of time series forecasting, however, neural networks have not achieved the same success as GPs. We argue thhat this is the case because, to the author's best knowledge, the vast majority of work has only dealt with point forecasts or provided a limited account of uncertainty, e.g. by performing quantile regression, instead of inferring a full predictive distribution (Pradeepkumar

and Ravi, 2017; Ardalani-Farsa and Zolfaghari, 2010; Khashei and Bijari, 2011; Zhang, 2003; Paoli et al., 2010; Faruk, 2010). A potential cause is that exact inference of fully probabilistic models, such as Bayesian neural networks, is intractable or can only be done expensively through sampling approaches. In a similar fashion, approximate Bayesian inference methods can be also computationally expensive even for relatively small networks. This leaves little room to justify their usage in comparison with other probabilistic methods, such as GPs.

This motivates the core of this thesis, where we develop a Recurrent Neural Network (RNN) framework for time series forecasting that is capable of producing fully (though approximate) probabilistic forecasts. We achieve this by recasting the time series forecasting task as an ordinal (auto-)regression one. Ordinal regression uses a discrete encoding mechanism to quantise time series measurements, effectively transforming the prediction task into a sequential classification problem. State-of-the-art developments from the ML and NLP corpora can then be readily incorporated into our framework.

Ordinal forecasts can further describe rich multi-modal, non-Gaussian behaviour, akin to that which can describe long-term forecasts more faithfully. The number of modes and other shape features of the predictive posterior distribution can then be adapted dynamically for different predictive horizons. Furthermore, our ordinal approach is a wholly general-purpose framework that is scalable with dataset size and requires little-to-no human intervention or expert craftsmanship for model specification.

In order to provide a full specification of our model in Chapter 3, we will dedicate the rest of this chapter to review all required background terminology. We start by reviewing foundational concepts from Probability theory and Decision theory in Section 2.2, which will be essential to formally assess uncertainty in our predictions. We then give a gentle review of neural network models and how to learn them in Section 2.3, which paves the way to the introduction of our time

series forecasting framework in Chapter 3.

2.2 Making decisions under uncertainty

Probability is the language of uncertainty. How to appropriately quantify it and propagate it to produce long-term forecasts thus lies at the very heart of probability theory, as it can be used to encapsulate and manipulate our beliefs about the state of the world through a rigorous and powerful framework. In fact, it is not an *ad hoc* choice, as shown by Cox (1946), who was the first one to provide a formal proof that probability theory can be regarded as an extension of Boolean logic to situations involving uncertainty.

2.2.1 Probability theory

We start by reviewing the most foundational terminology in probability theory that will reappear systematically throughout this thesis. The following synthesis is largely based on the canonical resources by Bishop (2006), Murphy (2012), MacKay (2003), Ghahramani (2013) and Barber (2012).

Discrete random variables

Let us start with two random variables A, B that admit values from the discrete sets $\mathcal{A} = \{a_1, \ldots, a_{N_1}\}, \mathcal{B} = \{b_1, \ldots, b_{N_2}\}$, respectively. We will denote $p(A = a_i), p(B = b_j)$ as the probability that the random variables A, B take the values a_i, b_j respectively, and $p(A, B)$ as the *joint probability* that A, B occur *simultaneously*. For discrete random variables, p is called a probability mass function (pmf), which satisfies:

$$0 \leq p(A) \leq 1 \qquad \text{and} \qquad \sum_{a_i \in \mathcal{A}} p(A = a_i) = 1.$$

Two types of discrete variables that we will model in the development of this thesis are binary and multinomial variables. We can describe the case of a binary variable $x \in \{0, 1\}$ that has probability $\hat{\theta} \in [0, 1]$ of taking state $x = 1$ with the Bernoulli distribution, i.e. $x \sim \text{Ber}(x \mid \hat{\theta})$ as given by:

$$\text{Ber}(x \mid \hat{\theta}) = \hat{\theta}^x (1 - \hat{\theta})^{1-x}.$$

In a similar fashion, if the variable x is drawn from a set of M mutually exclusive outcomes $\{A_1, \ldots, A_M\}$ with respective occurrence probabilities $\hat{\boldsymbol{\theta}} = \{\hat{\theta}_1, \ldots, \hat{\theta}_M\}$ such that $\sum_{m=1}^{M} \hat{\theta}_m = 1$, then we can describe it with a **categorical** distribution, i.e. $x \sim \text{Cat}(x \mid \hat{\boldsymbol{\theta}})$, a generalisation of the Bernoulli distribution. To this end, let us encode the variable x using a one-hot encoding (a term we will use interchangeably with the 1-of-K encoding) given by $x_m = \mathbb{I}(x = A_m)$, where $\mathbb{I}$ is the indicator function. The categorical distribution is readily given by:

$$\text{Cat}(x \mid \hat{\boldsymbol{\theta}}) = \prod_{m=1}^{M} \hat{\theta}_m^{x_m}.$$

We can manipulate probabilities through two fundamental rules that form the core of probability theory: the **product rule** and the **sum rule** (Bishop, 2006). The product rule allows us to rewrite joint probability as:

$$p(A, B) = p(A \mid B)p(B),$$

where $p(A \mid B)$ is the *conditional probability* of A given B. The conditional probability is defined only if $p(B) > 0$ and is given by:

$$p(A \mid B) = \frac{p(A, B)}{p(B)} \qquad \text{if } p(B) > 0,$$

For a larger set of N random variables $A_1, \ldots, A_N$, successive application of the product rule yields the **chain rule** of probability:

$$p(A_1, \ldots, A_N) = p(A_1) \prod_{i=0}^{N-2} p(A_{N-i} \mid A_{N-i-1}, \ldots, A_1).$$

The second core rule of probability theory is the sum rule, which states that variables can be *marginalised* or *summed out* as follows:

$$p(A) = \sum_{b_j \in \mathcal{B}} p(A, B) = \sum_{b_j \in \mathcal{B}} p(A \mid B = b_j) p(B = b_j).$$

Consequently from these rules, we can derive **Bayes' theorem**:

$$p(A \mid B) = \frac{p(A, B)}{p(B)} = \frac{p(B \mid A) p(A)}{\sum_{a_i \in \mathcal{A}} p(B \mid A) p(A)},$$

which is a foundational pillar in *Bayesian inference*, the rigorous framework we will use to quantify uncertainty in time series prediction. We will return to this in Section 2.2.2.

Continuous random variables

We can readily extend this framework to real-valued random variables $x, y \in \mathbb{R}$. Let us consider the probability $p(x)\Delta x$ of the event $x \in (x, x + \Delta x)$. As $\Delta x \to 0$, we refer to $p(x)$ as a probability density function (pdf) over x, and more generally define the probability of $x \in (l, r)$ for two scalars $l, r \in \mathbb{R}$ as:

$$p(x \in (l, r)) = \int_l^r p(x) dx,$$

which must satisfy:

$$p(x) \geq 0, \qquad \int_{-\infty}^{\infty} p(x) dx = 1.$$

We now define the cumulative density function (cdf) $P(r)$, which describes the probability that $x \in (-\infty, r), r \in \mathbb{R}$, as:

$$P(r) = \int_{-\infty}^{r} p(x)dx,$$

and satisfies $\frac{d}{dx}P(x) = p(x)$ (we assume the derivative exists). In a similar fashion, we can characterise a probability density function through its *quantile function* $P^{-1}(\alpha)$, given by:

$$P^{-1}(\alpha) = \min\{x_\alpha \in \mathbb{R} \mid \alpha \leq P(x_\alpha)\},\ \alpha \in (0, 1).$$

Intuitively, the quantile x_α of order α is the first point in the pdf's support where a fraction α of the probability mass has been integrated, i.e. the minimum x_α that satisfies:

$$P^{-1}(\alpha) = \min\left\{x_\alpha \middle| \int_{-\infty}^{x_\alpha} p(x)dx \leq \alpha\right\}.$$

Let us remark that both quantile functions and cdfs are only defined for random variables drawn from an ordered set, e.g. in the case of discrete random variables $A \in \mathcal{A}$, these functions are only defined for cases when $\mathcal{A}$ is an ordered set. Quantiles regularly appear in methods for measuring the similarity between pdfs, such as quantile-quantile plots, of which we make extensive in Section 3.5, where we assess the predictive probability distributions of different time series forecasting frameworks.

Two examples of pdfs that are commonplace in the literature, and that will appear exhaustively in this thesis, are the uniform distribution $\mathcal{U}(x \mid a, b)$, given by:

$$\mathcal{U}(x \mid a, b) = \frac{1}{b-a}\mathbb{I}(a \leq x \leq b), \qquad a, b \in \mathbb{R},$$

and the Gaussian or normal distribution $\mathcal{N}(x \mid \mu, \sigma^2)$ with mean μ and variance σ^2:

$$\mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \qquad \mu \in \mathbb{R}, \sigma > 0.$$

Alternatively, the Gaussian distribution can also be parameterised with a precision parameter $\tau = \frac{1}{\sigma^2}$:

$$\mathcal{N}(x \mid \mu, \tau) = \sqrt{\frac{\tau}{2\pi}} \exp\left(-\tau\frac{(x-\mu)^2}{2}\right).$$

Summarising distributions

When we discuss how to make decisions in Section 2.2.3, these will often come in the shape of a summary statistic of a probability distribution. Three such summary statistics are the median, the mean and the mode. These describe characteristic locations (or values of the pdf's support) where certain criteria are met. We start by reviewing the median x_{med}, which is given by $x_{\text{med}} = P^{-1}(0.5)$, i.e. the first point in the pdf's support where exactly half of the probability mass has been integrated.

A second relevant summary statistic is the expected value, mean or average $\mathbb{E}[x] = \mu_x$ of a probability distribution, given by:

$$\mu_x = \sum_x xp(x) \qquad \mu_x = \int xp(x)dx.$$

Functions $g(x)$ of the random variable $x \sim p(x)$ can also be averaged with the definition above:

$$\mathbb{E}[g] = \sum_x p(x)g(x) \qquad \mathbb{E}[g] = \int p(x)g(x)dx,$$

for the case of discrete and continuous random variables, respectively. Often, we

will not have access to $p(x)$ directly, but rather to a set of N_s samples $\{\hat{x}_s\}_{s=1}^{N_s}$ drawn from it. The expectation can then be approximated through the arithmetic mean of the samples:

$$\mathbb{E}[x] \simeq \frac{1}{N_s} \sum_{s=1}^{N_s} \hat{x}_s,$$

which converges to the true value in the limit as $N_s \to \infty$. We note that this estimator is more sensitive to outliers (values that have very low probability of occurrence) than the sample median, making the latter a more robust choice.

A third measure of central tendency is the mode, given by the value $x_{\max}$ at which the distribution $p(x)$ reaches its maximum, i.e. $x_{\max} = \arg\max_x \{p(x)\}$. Throughout this thesis, *multimodal distributions*, i.e. those that possess multiple local maxima or modes, will play a key role. In time series forecasting, these arise naturally in order to describe long-term predictions where there exist multiple plausible forecasts.

An example of a pdf that can describe multimodal behaviour is the Gaussian Mixture Model (GMM), which is a convex combination of K Gaussians $\mathcal{N}(x \mid \mu_k, \sigma_k^2), 1 \leq k \leq K$ with mixing weights $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)$ such that $\sum_{k=1}^{K} \pi_k = 1$. The GMM is given by:

$$\text{GMM}(x \mid \boldsymbol{\mu}, \boldsymbol{\sigma}^2, \boldsymbol{\pi}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, \sigma_k^2),$$

where $\mu_k \in \mathbb{R}\, \sigma_k > 0$, $0 < \pi_k < 1$.

2.2.2 Bayesian statistics

There exist two broad interpretations of probability: the frequentist and the Bayesian views. In the former case, quantified probabilities are interpreted as the fraction of times a certain event occurs in the limit of repetitions, e.g. the

fraction of times a fair coin would land tails in a potentially infinite repetition of tosses.

In the *subjective* or *Bayesian* view, the rules of probability are regarded as a framework to quantify, as a real number, our degree of belief in a proposition, doing so in a manner that is consistent with our intuition about probability (Barber, 2012). Such propositions may not necessarily be the outcome of a random event – indeed, we can use a real number to describe our degree of belief about a deterministic proposition whose truth value we ignore (MacKay, 2003). As we receive relevant evidence about the proposition, our ignorance fades away and we update our degrees of belief accordingly. Such updates must be consistent with the rules of probability, and therefore two independent decision-makers that make the same assumptions and receive the same evidence must come to the same conclusion.

Although both paradigms have their own relative merits and criticisms, the core of this thesis will tackle the time series forecasting task from a Bayesian perspective. As we will develop, the Bayesian approach requires formally setting out all our assumptions and prior knowledge about the task in the form of a *prior distribution*. This encapsulation of our beliefs is then updated in light of new evidence, which is the available data.

Let us relate this discussion to the task requirements we described in Section 2.1. We wish to forecast the next time series entry x_{T+1} from the covariates $\mathbf{z}_{T+1}$ following:

$$x_{T+1} = f(\mathbf{z}_{T+1}; \theta) + \varepsilon_{T+1},$$

where $x_{T+1} \in \mathbb{R}$, $\mathbf{z}_{T+1} \in \mathbb{R}^M$, $\varepsilon_{T+1} \sim \mathcal{N}(\varepsilon \mid 0, \sigma_\varepsilon^2)$ and θ are the model parameters. For the time being, let us assume f is a parametric model, i.e. that $\theta \in \mathbb{R}^{|\theta|}$ is finite-dimensional, and we will address any additional details for nonparametic

models in Section 3.5.2.

When assessing the uncertainty of the model parameters θ, we should incorporate both our *prior* assumptions and knowledge about the task, in addition to all available evidence, i.e. the time series data and additional inputs $\mathbf{x} = [x_1, \ldots, x_T], \mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_T]$. Noting that the regressors $\mathbf{Z}$ are generated independently of θ, this *parameter inference* or *parameter learning* task can be performed through Bayes' theorem in the following manner:

$$p(\theta \mid \mathbf{x}, \mathbf{Z}) = \frac{p(\theta, \mathbf{x}, \mathbf{Z})}{p(\mathbf{x}, \mathbf{Z})} = \frac{p(\mathbf{x} \mid \theta, \mathbf{Z})p(\mathbf{Z})p(\theta)}{\int p(\theta, \mathbf{x} \mid \mathbf{Z})p(\mathbf{Z})d\theta}.$$

One more application of the product rule and simplifying yields:

$$p(\theta \mid \mathbf{x}, \mathbf{Z}) = \frac{p(\mathbf{x} \mid \theta, \mathbf{Z})p(\theta)}{\int p(\mathbf{x} \mid \theta, \mathbf{Z})p(\theta)d\theta},$$

where:

- $p(\theta \mid \mathbf{x}, \mathbf{Z})$ is the *posterior distribution* of the parameters, which describes the model parameters θ once the data $\mathbf{x}, \mathbf{Z}$ have been observed.
- $p(\mathbf{x} \mid \theta, \mathbf{Z})$ is the *likelihood function*, which describes the likelihood that a set of parameters explains the observed measurements well.
- $p(\theta)$ is the *prior distribution*, which encapsulates our beliefs about the model parameters θ before any data has been observed by the model.
- the denominator:

  $$p(\mathbf{x} \mid \mathbf{Z}) = \int p(\mathbf{x} \mid \theta, \mathbf{Z})p(\theta)d\theta,$$

  is the *marginal likelihood* or *model evidence*.

With this knowledge and the next vector of covariates $\mathbf{z}_{T+1}$ in hand, we can now assess the *posterior predictive distribution* for the next entry in the time series x_{T+1}:

$$\begin{aligned} p(x_{T+1} \mid \mathbf{z}_{T+1}, \mathbf{x}, \mathbf{Z}) &= \int p(x_{T+1}, \theta \mid \mathbf{z}_{T+1}, \mathbf{x}, \mathbf{Z}) d\theta, \\ &= \int p(x_{T+1} \mid \mathbf{z}_{T+1}, \theta) p(\theta \mid \mathbf{x}, \mathbf{Z}) d\theta. \end{aligned}$$

At this point, it is worth noting a fundamental insight in the frequentist and Bayesian interpretations. For a frequentist, there exists a set of optimal parameters θ that explain the observed data. Predictions are computed using this optimal choice, and these can further be endowed with error bars through resampling schemes such as the boostrap (Bishop, 2006). An example of such frequentist estimation methods is *Maximum Likelihood Estimation (MLE)*, given by:

$$\theta_{\text{MLE}} = \arg\max_{\theta} p(\mathbf{x} \mid \theta, \mathbf{Z}).$$

In this setting, learning the parameters θ is thus regarded as an *optimisation* problem.

On the contrary, whereas frequentist analysis regards data as a random variable of fixed parameters, Bayesians express degrees of belief in the model parameters through a probability density (Durbin and Koopman, 2012). This assessment is then propagated into our forecasts in accordance with the same foundational rules of probability theory. In a Bayesian's view, the manner to quantify predictive uncertainty is by *integrating out* the possible values of θ (MacKay, 2003). Indeed, Bayesians regard parameter learning and out-of-sample predicting as an *inference* problem.

Nevertheless, let us remark that performing **exact Bayesian inference** en-

tails solving the integrals over the parameter space that appear in the definitions of the posterior and predictive posterior distributions. In models such as deep neural networks, which can easily have millions of parameters, solving these integrals may be intractable. In those cases, we recur to the *approximate Bayesian inference* literature, which covers sampling methods such as Markov Chain Monte Carlo (MCMC) and deterministic approximations, such as variational Bayes (Bishop, 2006). We will return to this when we discuss neural network parameter inference for our framework, in Section 2.3.3.

2.2.3 Bayesian decision theory

In this subsection, we describe how to make optimal decisions from our inferred distributions. In the Bayesian setting, we wish to choose an action $\hat{x}_t$ from an action space $\hat{\mathcal{X}}$ that optimises the *posterior expected loss* for a task with inputs $\mathbf{z}_t \in \mathcal{Z}$ and outputs $x_t \in \mathcal{X}$. The posterior expected loss is given respectively for discrete and continuous actions by:

$$\rho(\hat{x}_t \mid \mathbf{z}_t) = \mathbb{E}_{p(x_t|\mathbf{z}_t)}[\mathcal{L}(x_t, \hat{x}_t)] = \sum_{x_t \in \mathcal{X}} \mathcal{L}(x_t, \hat{x}_t) p(x_t \mid \mathbf{z}_t),$$
$$\rho(\hat{x}_t \mid \mathbf{z}_t) = \mathbb{E}_{p(x_t|\mathbf{z}_t)}[\mathcal{L}(x_t, \hat{x}_t)] = \int \mathcal{L}(x_t, \hat{x}_t) p(x_t \mid \mathbf{z}_t) dx_t,$$

where $\mathcal{L}(x_t, \hat{x}_t)$ is a task-dependent loss incurred by taking action $\hat{x}_t$. The discrete case is related, for instance, to the optimal class label $\hat{x}_t \in \mathbb{Z}^M$ in classification tasks with M distinct categories (in this case, $\mathcal{X} = \mathbb{Z}^M$) , whereas the continuous case corresponds to the optimal prediction in a regression task, e.g. $\hat{x}_t \in \mathbb{R}$ and $\mathcal{X} = \mathbb{R}$.

The choice of loss function $\mathcal{L}(x_t, \hat{x}_t)$ largely depends on the task of interest. For classification, a common choice is the 0-1 loss, which penalises all *misclassification*

errors equally:

$$\mathcal{L}(x_t, \hat{x}_t) = \mathbb{I}(x_t \neq \hat{x}_t),$$

and for which the posterior expected loss becomes:

$$\begin{aligned}
\rho(\hat{x}_t \mid \mathbf{z}_t) &= \sum_{x_t \in \mathcal{X}} \mathbb{I}(x_t \neq \hat{x}_t) p(x_t \mid \mathbf{z}_t), \\
&= 1 - \sum_{x_t \in \mathcal{X}} \mathbb{I}(x_t = \hat{x}_t) p(x_t \mid \mathbf{z}_t), \\
&= 1 - p(x_t \mid \mathbf{z}_t),
\end{aligned}$$

which is minimised when the second term is largest, i.e.:

$$\begin{aligned}
\hat{x}_t^* &= \arg\min_{x_t \in \mathcal{X}} \left(1 - p(x_t \mid \mathbf{z}_t)\right), \\
&= \arg\max_{x_t \in \mathcal{X}} \left(p(x_t \mid \mathbf{z}_t)\right).
\end{aligned}$$

And therefore, the optimal class to assign to input $\mathbf{z}_t$ is the mode of the posterior class probabilities.

In the case of regression, $\hat{x}_t \in \mathbb{R}$ and a common loss function is the squared error $\mathcal{L}(x_t, \hat{x}_t) = (x_t - \hat{x}_t)^2$. Substituting in a similar fashion as the above:

$$\begin{aligned}
\rho(\hat{x}_t \mid \mathbf{z}_t) &= \int (x_t - \hat{x}_t)^2 p(x_t \mid \mathbf{z}_t) dx_t, \\
&= \int x_t^2 p(x_t \mid \mathbf{z}_t) dx_t - 2 \int x_t \hat{x}_t^2 p(x_t \mid \mathbf{z}_t) dx_t + \int \hat{x}_t^2 p(x_t \mid \mathbf{z}_t) dx_t, \\
&= \int x_t^2 p(x_t \mid \mathbf{z}_t) dx_t - 2\hat{x}_t \int x_t p(x_t \mid \mathbf{z}_t) dx_t + \hat{x}_t^2,
\end{aligned}$$

which is optimised when:

$$\frac{d}{d\hat{x}_t}\rho(\hat{x}_t \mid \mathbf{z}_t) = -2\int x_t p(x_t \mid \mathbf{z}_t)dx_t + 2\hat{x}_t = 0,$$
$$\rightarrow \hat{x}_t^* = \int x_t p(x_t \mid \mathbf{z}_t)dx_t = \mathbb{E}[x_t \mid \mathbf{z}_t],$$

and therefore the optimal decision for the squared error loss is to choose the posterior mean.

2.2.4 Remarks

In this section, we have reviewed foundational probability theory concepts, and introduced two of the main frameworks that will help us make probabilistic time series forecasting with neural networks: Bayesian inference and decision theory. Both of these will come up extensively in the rest of the thesis, and particularly when we present how to infer our time series forecasting model and make fully probabilistic predictions with it.

2.3 Neural network preliminaries

Having introduced the probabilistic framework we will use to forecast time series, let us now summarise relevant foundational concepts in the neural network literature. These preliminaries pave the way for introducing the Long Short-Term Memory (LSTM) neural network layer, which is a central component of the time series forecasting framework developed in this thesis: the Memory-endowed Ordinal Regression Deep neural network (MOrdReD).

A neural network is a layered *parametric* model between vector inputs $\mathbf{z}_t \in \mathbb{R}^M$ and outputs $\mathbf{x}_t \in \mathbb{R}^{M'}$, where $M, M' \in \mathbb{Z}^+$ are the dimensions of the inputs and outputs, respectively. A single neural network model comprises a collection of interconnected layers that successively perform parameterised non-linear trans-

formations. Such intermediate representations, also referred to as *activations* in the literature, were originally inspired by biological neurons whose activity spikes in the presence of certain features.

Although time series forecasting has been introduced as a regression framework in Section 2.1, the core framework of this thesis relies heavily on a number of concepts from the neural network-based *classification* literature. We will therefore cover concepts relevant to both tasks. Due to the breadth of this discussion, and to keep presentation simple, let us review the notation we will use in this section:

- $\mathbf{x}_t \in \mathbb{R}^{M'}$ is the t-th target. For univariate regression, $M' = 1$, and $\mathbf{x}_t$ is a scalar time series measurement as introduced in Section 2.1. For classification tasks, M' equals the number of classes in the dataset.

- $\mathbf{z}_t \in \mathbb{R}^{M}$ is the model input. In Section 2.1, we referred to this as the regressors or covariates. In autoregressive time series tasks, $\mathbf{z}_t$ may include, for example, the last P previous observations on the time series $\mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-P}$, in addition to any other *exogenous* inputs to the model.

- f is the model. For this section, f will be a neural network with parameters θ.

- $\hat{\mathbf{x}}_t \in \mathbb{R}^{M}$ is the model output, given by $\hat{\mathbf{x}}_t = f(\mathbf{z}_t; \theta)$.

- $\mathbf{Z} \in \mathbb{R}^{T \times M}, \mathbf{X} \in \mathbb{R}^{T \times M'}$ are the design and target matrices, obtained by concatenating all inputs and targets respectively over the T observed timesteps.

In the context of Machine Learning and Pattern Recognition, *learning* a neural network refers to *inferring* all parameters θ that explain the observed data $\mathbf{Z} \in \mathbb{R}^{T \times M}, \mathbf{X} \in \mathbb{R}^{T \times M'}$. As per the desiderata discussed in Section 2.1 for time series forecasting, we shall seek to quantify the uncertainty in the model parameters and incorporate it into our forecasts over future predictive horizons. This can

be achieved through Bayesian inference, which provides a powerful and rigorous framework to achieve this task, as reviewed in Section 2.2.2.

One critical challenge for Bayesian methods is the large amount of computation required to perform exact inference of the posterior and predictive posterior distributions:

$$p(\theta \mid \mathbf{X}, \mathbf{Z}) = \frac{p(\mathbf{X} \mid \theta, \mathbf{Z})p(\theta)}{\int p(\mathbf{X} \mid \theta, \mathbf{Z})p(\theta)d\theta},$$
$$p(\mathbf{x}_{T+1} \mid \mathbf{z}_{T+1}, \mathbf{X}, \mathbf{Z}) = \int p(\mathbf{x}_{T+1} \mid \mathbf{z}_{T+1}, \theta)p(\theta \mid \mathbf{X}, \mathbf{Z})d\theta,$$

respectively. Assessing these distributions requires integrating out the parameter θ, which for the case of many modern neural networks may have millions of dimensions, making exact inference intractable.

Furthermore, although approximate Bayesian inference methods for neural networks have been in the literature for a few decades, including Hamiltonian (hybrid) Monte Carlo (Neal, 1996a) methods and deterministic approximations such as Laplace's method and Variational Bayes (Bishop, 2006), these remained efficient only for small-to-medium sized networks. During this time, point estimates of the parameters were commonly found by optimising a task-dependent *loss function*. Two examples of such estimators are the Maximum Likelihood Estimation (MLE) and the Maximum A Posteriori (MAP), given by:

$$\theta_{\text{MLE}} = \arg\max_{\theta} p(\mathbf{X} \mid \theta, \mathbf{Z}) \qquad \theta_{\text{MAP}} = \arg\max_{\theta} p(\mathbf{X} \mid \theta, \mathbf{Z})p(\theta).$$

Other than the fact that neither of them assesses the uncertainty of the parameters, both introduce further difficulties. As a frequentist estimator, the MLE is known to overfit when there is not enough data available (Bishop, 2006; Murphy, 2012). In a similar fashion, the MAP, which chooses the posterior mode as estimator, is often less representative of the distribution than other summary

statistics, such as the mean.

Nevertheless, one of the reasons that both approaches were popular is likely due to their recasting of the learning problem as an optimisation one. In particular, neural networks are particularly well-suited to gradient-based optimisation methods, since error signals can be computed and propagated efficiently through backpropagation (Rumelhart, Hinton, Williams, et al., 1988), an algorithm based on successive applications of the chain rule of calculus that exploits the composite or sequential structure of neural networks.

Recent developments in the neural network community, however, have shown how to perform efficient, though approximate, Bayesian inference (Srivastava et al., 2014; Gal and Ghahramani, 2016b). These showed the link between the dropout regularisation technique, which we introduce in Section 2.3.3, and the construction of a *variational* objective function, thereby also recasting approximate Bayesian inference as an optimisation problem. This is the approach we shall pursue to assess uncertainty in the context of time series forecasting.

In the rest of this section, we will build upon these concepts. We shall start by describing foundational terminology in the neural networks literature, starting with the Multilayer Perceptron (MLP) in Section 2.3.1 and other topologies for structured input in Section 2.3.2. We will then discuss parameter inference in Section 2.3.3 and neural network optimisation methods in Section 2.3.4.

2.3.1 The Multilayer Perceptron

A neural network layer is a collection of neurons, which are the most basic computational units in a neural network. Neurons between layers are connected through weighted directed edges, and the set of all such weights is collectively listed as the set of parameters, θ, of the neural network. In the simplest case, layers are connected sequentially, i.e. in a feedforward fashion. In this illustrative case, every layer performs a parameterised non-linear transformation of the previous

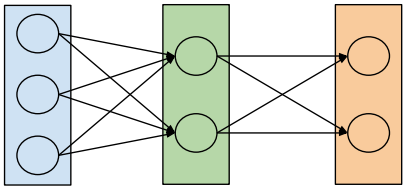


Figure 2.1: Multilayer Perceptron (MLP) with three layers: an input layer (blue) with three units, a fully-connected (FC) hidden layer (green) with two hidden units and a second FC output layer (orange) with two units. Fully-connected units have incoming edges from all units in their input layer.

layer's output, i.e. the output at layer l with n_l units, $\mathbf{h}^{(l)} \in \mathbb{R}^{n_l}$, is given by

$$\mathbf{h}^{(l)} = f^{(l)}(\mathbf{h}^{(l-1)}; \theta^{(l)}) = \phi^{(l)}(g^{(l)}(\mathbf{h}^{(l-1)}; \theta^{(l)})),$$

where $1 \leq l \leq L$; L is the number of layers in the network; $\phi^{(l)}(\cdot)$ is an element-wise **activation function** or non-linearity for layer l; $\theta^{(l)} \subset \theta$ is the subset of parameters pertaining to the l-th layer; and layers $l = 1, l = L$ are the model's input and output layers respectively. For all other $1 < l < L$, the vector $\mathbf{h}^{(l)} \in \mathbb{R}^{n_l}$ is the set of activations at hidden layer l.

The parametric shape of $g^{(l)}$ is given by the *class* of the l-th layer. From a historical perspective, the development of classes of neural network layers dates back to the Perceptron model in the 1960s (Minsky and Papert, 1969). One milestone in the development of these models was the Multilayer Perceptron (MLP), which is a feedforward model with at least three layers: an input layer, one hidden (or intermediate) fully-connected layer, and an output fully-connected layer, as shown in Figure 2.1.

Such fully-connected (FC) layers possess units with a weighted edge coming in from every unit in their input layers. For a feedforward model, their parametric form $f_{FC} : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}$, with learnable parameters $\theta^{(l)} = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}$, is given

by:

$$\mathbf{h}^{(l)} = f_{FC}(\mathbf{h}^{(l-1)}; \mathbf{W}^{(l)}, \mathbf{b}^{(l)}) = \phi^{(l)}(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}),$$

where $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}, \mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$ are the layer **weights** and **bias**, respectively, and ϕ is the layer's activation function.

The activation function, ϕ controls the amount of activity after neurons are exposed to stimuli propagated from previous layers. Although ϕ can be chosen to be linear, this case is usually reserved only for the final layer of networks modelling real-valued outputs (as would be the case for regression tasks) and contrary to binary or multinomial outputs. Indeed, consecutive hidden layers with linear activation functions can otherwise be straightforwardly combined into one. For this reason, we henceforth focus only on non-linear activation functions and use the terms activation function and non-linearity interchangeably throughout the rest of this thesis.

Some common examples of element-wise non-linearities in the literature are the sigmoid activation function, the hyperbolic tangent and the rectified linear unit (ReLU). For $x \in \mathbb{R}$, they are given respectively by:

$$\begin{aligned}
\sigma(x) &= \frac{1}{1 + \exp(-x)}, \\
\tanh(x) &= \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}, \\
\mathrm{ReLU}(x) &= \max(0, x).
\end{aligned}$$

Another common choice of non-linearity, often used in the context of multiclass classification to describe categorical variables, is the softmax function, given by:

$$\mathrm{softmax}(\mathbf{x}) = \left(\frac{\exp(\mathbf{x}_1)}{\sum_{k=1}^{n} \exp(\mathbf{x}_k)}, \cdots, \frac{\exp(\mathbf{x}_n)}{\sum_{k=1}^{n} \exp(\mathbf{x}_k)} \right), \qquad \mathbf{x} \in \mathbb{R}^n.$$

As a side note, the softmax function is of special importance beyond its application as a non-linearity. In particular, it arises naturally as the class posterior distribution $p(\mathcal{C}_k \mid \mathbf{z})$ given an input $\mathbf{z} \in \mathbb{R}^M$ for a wide range of generative models (Bishop, 2006). It also arises as the multiclass generalisation of logistic regression, i.e. a classifier with linear decision boundaries over the input features. In the context of neural networks, such boundaries are drawn instead through a linear combination of non-linear features, as given by the activations of previous layers.

Let us also note that another desirable property of activation functions is that they shall be continuously-differentiable. This crucially enables efficient learning via gradient-based optimisers. We discuss this further in Subsection 2.3.4.

We remark that the usage of non-linearities is vital for allowing neural networks to learn complex relations in data. As first shown by Cybenko (1993) in his universal approximation theorem, an MLP with a single hidden layer with a non-linear activation can approximate any continuous function to arbitrary precision on a closed and bounded subspace of $\mathbb{R}^n$. As each successive layer computes linear combinations of non-linear basis functions learnt from data, neural networks can learn very complex, non-linear relationships.

Although said universal approximation theorem characterises the expressive power of depth-bounded neural networks, it does not describe the learnability of such architectures, and in particular the width that the layers of such depth-bounded models must have. More recent work, however, has shed light upon the case of width-bounded models. For instance, Lu, Pu, et al. (2017) showed that width-$(n + 4)$ ReLU networks with n-dimensional input are universal approximators, highlighting once again the key role of such non-linear activation functions.

A note on output activation functions

Recall that the output of a neural network $\hat{\mathbf{x}}_t$ is given by a transformation of the activations $\mathbf{h}_t^{(L-1)}$ of its second-to-last layer:

$$\hat{\mathbf{x}}_t = f^{(L)}(\mathbf{h}_t^{(L-1)}; \theta^{(L)}),$$

where $\mathbf{h}_t^{(L-1)}$ is the vector of salient higher order features extracted after a sequence of non-linear transformations. If we choose $f^{(L)}(\cdot)$ to be a fully-connected layer, then:

$$\hat{\mathbf{x}}_t = \phi^{(L)}(\mathbf{W}^{(L)}\mathbf{h}_t^{(L-1)} + \mathbf{b}^{(L)}),$$

where $\phi^{(L)}$ is the output layer's activation function, and $\theta^{(L)} = \mathbf{W}^{(L)}, \mathbf{b}^{(L)}$ are the layer parameters. If we assume that the true $\mathbf{x}$'s density is in the exponential family, then the above can be regarded as a Generalised Linear Model (GLM) (Murphy, 2012). From this literature, we know that responses modelled by a GLM under specific likelihood functions have a corresponding *canonical link function* $\phi^{-1}(\cdot)$, which is simply the inverse of the activation function $\phi(\cdot)$.

Although other link function choices exist, choosing the canonical link greatly simplifies the computation of likelihood function gradients, as the adjustments in the latent space of $\mathbf{h}_t^{(L-1)}$ become linear. As we will introduce in Sections 2.3.3 and 2.3.4, gradient computation plays a crucial role in neural network parameter inference, therefore further motivating the usage of the canonical link.

In particular, for a GLM with a Gaussian likelihood, the natural activation function given by the canonical link is simply the identity function; for multiclass classification, the natural choice is the softmax activation (Bishop, 2006). These two likelihood functions will reappear in our presentation when we discuss neural network parameter inference in Section 2.3.3. We refer the reader to Chapter 4

in (Bishop, 2006) and Chapter 9 in (Murphy, 2012) for a deeper discussion of Generalised Linear Models and canonical link functions.

2.3.2 Deep neural networks

We now resume our discussion about layer classes, which so far has been limited to fully-connected layers. Let us recall from Subsection 2.3.1 that layer classes define inter-layer connectivity patterns between neural network units. From a practical perspective, this explicitly incorporates knowledge about input structure into the model. We illustrate this with two layer families that have gained attention in the academic literature during most of this decade: convolutional and recurrent layers.

Despite originating during the 1980s and 1990s (LeCun et al., 1989; Hochreiter and Schmidhuber, 1997), both layer families only achieved major breakthroughs in the early 2010s, especially in Computer Vision and Natural Language Processing (Simonyan and Zisserman, 2015; Sutskever, Vinyals, and Le, 2014; Graves, 2013). In great part, this is a consequence of the increased availability of datasets, which guarded large models against overfitting, in addition to specialised hardware (such as the Graphics Processing Unit (GPU), and more recently, the Tensor Processing Unit (TPU)) and software (Machine Learning frameworks such as Theano (Bergstra et al., 2010), Torch (Collobert, 2011) and Tensorflow (Martín Abadi et al., 2015), in addition to advances in Automatic Differentiation (Bücker et al., 2005) routines.

These early developments gave rise to the adoption of **deep learning** models into the mainstream, where the term "deep" refers to neural network models with "many" layers. Although there is no formal cutoff as to what is the minimum number of layers a deep network must have, this nomenclature rather signals a paradigmatic shift in the context of **feature extraction**. Deep networks aim to achieve *end-to-end* pipelines where salient input features are extracted in a hier-

archical fashion by the lower layers. This is in contrast to earlier approaches that used manually engineered domain-specific features followed by shallow classifiers.

To illustrate this scenario, consider the speech recognition literature. During the late 2000s, a substantial amount of research incorporated feature extraction techniques from the Signal Processing literature, such as Mel-Frequency Cepstral Coefficients (MFCC) and Linear Predictive Coding (LPC) coefficients, followed by sequential learning models such as the Hidden Markov Model (HMM) and the like (Jurafsky and Martin, 2009; Gales, Young, et al., 2008). In contrast, current state-of-the-art models are based on deep neural networks that extract features from the raw audio signal or its spectrum (Amodei et al., 2016).

Let us now have a closer look at convolutional layers, which take inspiration from the animal visual cortex, where neural activity is a function of stimuli from restricted visual fields. In the context of artificial neural networks, this results in neurons that are only locally connected to subgrids in their input layer. This effectively aids in modelling the spatial correlation between individual entries in the input, which occurs frequently in a number of Computer Vision (CV) tasks (e.g. neighbours ("patches") of pixels in structured images are more likely to be correlated than any two distant pixels).

In their simplest form, convolutional layers possess a collection of Q filters $\theta_{\text{Conv}} = \{\mathbf{W}_1, ..., \mathbf{W}_Q\}$. The activations of this layer are then given by the stacked **discrete convolutions** between the layer inputs and each of the filters. Redundancy is removed via pooling layers that aggregate neighbouring activations. Also crucially, as a *striding* operation, the discrete convolution is invariant to translations in its input, a particularly desirable property in many Computer Vision tasks, such as object recognition in images.

In the case of data indexed over time (*sequential* data), recurrent layers update their activations every time the layer is provided with an input token, i.e. the activations $\mathbf{h}_t = f(\mathbf{z}_t, \mathbf{h}_{t-1}; \theta_{\text{Rec}})$. These temporal activations act as an encoded

memory mechanism that is also forwarded to an output layer. Such an output layer is often a fully-connected layer that decodes the state of the memory into the observable output at time t.

Let us remark that recurrent layers, and gated variants such as the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), are now state-of-the-art in several sequential learning tasks, for example: speech recognition (Graves, Mohamed, and Hinton, 2013), machine translation (Bahdanau, Cho, and Bengio, 2014), handwriting recognition (Graves, 2013), image and video caption generation (Vinyals et al., 2015; Venugopalan et al., 2015) and natural language generation (Wen et al., 2015), among others. This extensive literature motivates the core of this thesis, which develops a time series forecasting framework based on recurrent neural networks. Due to the core role these architectures play in our framework, we dedicate a more detailed presentation in Section 3.2.

2.3.3 Parameter inference

Bayesian inference for neural networks

We now focus our attention on the task of inferring a neural network's parameters $\theta = \left\{\theta^{(l)}\right\}_{l=1}^{L}$. Our aim shall be to assess the posterior distribution over the parameters θ after observing the input and output data $\mathbf{Z} \in \mathbb{R}^{T\times M'}, \mathbf{X} \in \mathbb{R}^{T\times M}$. As we described in Section 2.2.2, this is given by Bayes' theorem as:

$$p(\theta \mid \mathbf{X}, \mathbf{Z}) = \frac{p(\mathbf{X} \mid \theta, \mathbf{Z})p(\theta)}{\displaystyle\int p(\mathbf{X} \mid \theta, \mathbf{Z})\, p(\theta)\, d\theta},$$

where, $p(\theta \mid \mathbf{X}, \mathbf{Z})$ is the *posterior distribution*, $p(\mathbf{X} \mid \theta, \mathbf{Z})$ is the *likelihood function*, $p(\theta)$ is the *prior distribution*, and the denominator is the *marginal likelihood* or *model evidence*.

Let us recall that the marginal evidence is only analytically tractable for

a limited class of models, e.g. when the prior and posterior distributions are conjugate (Murphy, 2012). Furthermore, numerical integration may also be computationally infeasible, especially when θ is very high-dimensional. This is often the case for neural networks, which frequently possess millions of parameters (Krizhevsky, Sutskever, and Hinton, 2012). Overcoming this challenge in order to adequately quantify the posterior distribution is of major interest for then computing the *posterior predictive distribution* over the model outputs. In time series forecasting, this is crucial for achieving accurate and well-calibrated error bars for out-of-sample predictions.

Parameter uncertainty, as encapsulated by the posterior distribution, can then be used to assess predictive uncertainty, through the *posterior predictive distribution*, given by:

$$p(\mathbf{x}_{T+1} \mid \mathbf{z}_{T+1}, \mathbf{X}, \mathbf{Z}) = \int p(\mathbf{x}_{T+1} \mid \mathbf{z}_{T+1}, \theta) p(\theta \mid \mathbf{X}, \mathbf{Z}) d\theta,$$

where $\mathbf{x}_{T+1} \in \mathbb{R}^{\mathbb{M}'}$ is the prediction and $\mathbf{z}_{T+1} \in \mathbb{R}^{M}$ are the inputs at time $T+1$. Computing the posterior predictive also requires us to integrate over the parameter space. How to efficiently solve these integrals for *Bayesian neural network* models in particular has thus been an active area of research for decades, and has been directly tackled by methods from the *approximate Bayesian inference* literature.

Such methods can be subdivided into Monte Carlo methods and deterministic approximations (MacKay, 2003). On one hand, Monte Carlo methods include variants of the Metropolis-Hastings algorithm, such as Hamiltonian or Hybrid Monte Carlo (Neal, 1996a). Deterministic approximations propose a tractable *parametric* form for the posterior (e.g. a multivariate Gaussian) and subsequently find optimal values for the distribution parameters. Such approximations include Laplace's method (Bishop, 2006; MacKay, 2003) and *variational Bayes*

approaches, for which there exists an extensive literature (Kingma and Welling, 2013; Titsias and Lázaro-Gredilla, 2014; Barber and Bishop, 1998). One critical issue with both families of methods is to do with their computational efficiency. In the case of Monte Carlo methods, it may be computationally expensive to gather a set of decorrelated samples as these methods suffer from **slow mixing**. This means that samples taken successively may be highly correlated and thus many of them may need to be discarded. Tempering is a strategy to perform faster mixing by constructing an alternative representation of the target distribution that encourages exploration of different modes (Neal, 1996b), however, at this point it is yet to make a strong advance in sampling from complex models (Goodfellow, Bengio, and Courville, 2016).

Several techniques for faster mixing are based on constructing alternative versionsof the target distribution in which the peaks are not as high and the surroundingvalleys are not as low In the case of variational methods, many of these also come with prohibitive computational costs, especially in the light of recent developments that show how inferring neural networks with dropout (Srivastava et al., 2014), which we will introduce in this subsection, can be recast as approximate Bayesian inference in deep Gaussian processes (Gal and Ghahramani, 2016b).

These findings linking the dropout regularisation technique to approximate Bayesian inference (Gal and Ghahramani, 2016a; Gal and Ghahramani, 2016b) have enabled an efficient and principled, though still approximate, quantification of uncertainty in a variety of settings. Indeed, recent work looks at applying this *Monte Carlo dropout* scheme to time series forecasting (Zhu and Laptev, 2017), however still making the tacit assumption that the predictive distribution is unimodal Gaussian. We will directly tackle this issue when we introduce our time series forecasting framework in Chapter 3.

Classical neural network training

Before we show how to perform approximate Bayesian inference with neural networks through Monte Carlo dropout, it must be said that this scheme is supported by methods and terminology that are widespread when inferring optimal model parameter point estimates. To this end, we will now briefly recapitulate what we henceforth call the *classical* view of neural networks. In this framework, the parameters θ are fixed and optimal under a loss function $\mathcal{L}(\mathbf{X}, \hat{\mathbf{X}})$, where $\mathbf{X}, \hat{\mathbf{X}}$ are the observed measurements and neural network predictions over the network inputs $\mathbf{Z}$, respectively. A commonplace estimator in this setting is the Maximum A Posteriori (MAP) estimator, given by:

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\mathbf{X} \mid \theta, \mathbf{Z}) p(\theta).$$

For computational convenience, this is equivalent to minimising the negative log:

$$\theta_{\text{MAP}} = \arg\min_{\theta} \{ -\log p(\mathbf{X} \mid \theta, \mathbf{Z}) - \log p(\theta) \},$$

where $p(\mathbf{X} \mid \theta, \mathbf{Z})$ is the likelihood function, which assesses how well the model (alongside the set of parameters θ) explains the measurements, and $p(\theta)$ is the prior distribution over the parameters. Although based on Bayes' theorem and allowing us to incorporate our beliefs as a prior distribution, MAP estimators are not fully Bayesian as they only produce a point value, and thus do not give any measure of uncertainty. On the other hand, by recasting the learning problem as an optimisation one, MAP estimators can be computed efficiently.

For univariate regression tasks[2], where

$$\mathbf{x}_t = f(\mathbf{z}_t; \theta) + \varepsilon_t, \qquad \mathbf{x}_t \in \mathbb{R}, \mathbf{z}_t \in \mathbb{R}^M,$$

[2]Although this case can also be generalised to multivariate regression following a similar reasoning, we refer the reader to (Murphy, 2012) and (Bishop, 2006) for more details.

let us assume a zero-mean Gaussian prior with a diagonal isotropic covariance over the weights $p(\theta) = \mathcal{N}(\theta \mid \mathbf{0}, \alpha^{-1}\mathbf{I}_{|\theta|})$, in addition to independently and normally distributed additive disturbances $\varepsilon_t \sim \mathcal{N}(\varepsilon_t \mid 0, \beta^{-1}\mathbf{I}_{|\theta|})$, where α^{-1}, β^{-1} are precision hyperparameters and $\mathbf{I}_{|\theta|}$ is the identity matrix of order $|\theta|$. This last assumption gives rise to a Gaussian likelihood:

$$\begin{aligned} p(\mathbf{X} \mid \theta, \mathbf{Z}) &= \prod_{t=1}^{T} p(\mathbf{x}_t \mid \theta, \mathbf{z}_t), \\ &= \prod_{t=1}^{T} \mathcal{N}(\mathbf{x}_t \mid f(\mathbf{z}_t; \theta), \beta). \end{aligned}$$

Substituting the prior and the likelihood function and simplifying, the MAP optimisation objective is thence given by:

$$\theta_{\text{MAP}} = \arg\min_{\theta} \left\{ -\frac{\beta}{2} \sum_{t=1}^{T} (\hat{\mathbf{x}}_t - \mathbf{x}_t)^2 - \frac{\alpha}{2} \|\theta\|^2 \right\},$$

with $\hat{\mathbf{x}}_t = f(\mathbf{z}_t; \theta)$. This is equivalent to the **ridge regression** objective, whose second term explicitly incorporates L_2-regularisation with hyperparameter $\lambda = \alpha/\beta$:

$$\mathcal{L}_{\text{MSE}}(\mathbf{X}, \hat{\mathbf{X}}) = \frac{1}{2T} \sum_{t=1}^{T} \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 + \lambda \|\theta\|^2 \quad \mathbf{x}, \hat{\mathbf{x}} \in \mathbb{R}^{M'}.$$

L_2 regularisation, also known as **weight decay** in the machine learning corpus, improves generalisation error by penalising hypotheses with high curvature variability in favour of simpler generating processes that may be less likely to **overfit** to data. It has also been shown to be intrinsically related to other regularisation techniques, such as noise injection (Sietsma and Dow, 1991; Bishop, 1994).

Let us now turn our attention to 1-of-K multiclass classification tasks. For convenience, let us consider the class labels $\mathbf{x}_t$ in their 1-of-K, also known as one-hot encoded, representations, $\mathbf{x}_t = (\mathbb{I}(\mathbf{x}_t \in \mathcal{C}_1), \ldots, \mathbb{I}(\mathbf{x}_t \in \mathcal{C}_K))$, i.e. $\mathbf{x}_t$ is a binary

vector that's zero everywhere except in index k if its class label $\mathcal{C}_k$. The class label is a discrete multinomial variable, so let us assume a categorical likelihood with outcome probabilities $\hat{\mathbf{x}}_t = f(\mathbf{z}_t)$ such that $\sum_{k=1}^{K} \hat{\mathbf{x}}_{t,k} = 1$:

$$p(\mathbf{x}_t \mid \mathbf{z}_t, \theta) = \prod_{k=1}^{K} \hat{\mathbf{x}}_{t,k}^{\mathbf{x}_{t,k}},$$

which gives rise to the log-likelihood function:

$$\begin{aligned} -\log p(\mathbf{X} \mid \theta, \mathbf{Z}) &= -\log \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{z}_t, \theta), \\ &= -\log \prod_{t=1}^{T} \prod_{k=1}^{K} \hat{\mathbf{x}}_{t,k}^{\mathbf{x}_{t,k}}, \\ &= -\sum_{t=1}^{T} \sum_{k=1}^{K} \mathbf{x}_{t,k} \log \hat{\mathbf{x}}_{t,k}. \end{aligned}$$

In a similar fashion to the univariate regression scenario, by choosing a zero-mean, isotropic Gaussian prior over the model parameters θ:

$$\theta_{\text{MAP}} = \arg\min_{\theta} \left\{ -\sum_{t=1}^{T} \sum_{k=1}^{K} \mathbf{x}_{t,k} \log \hat{\mathbf{x}}_{t,k} - \frac{\alpha}{2} \|\theta\|^2 \right\}.$$

The function above is also known in the literature as the L_2-regularised **cross-entropy** loss $\mathcal{L}_{\text{Xent}}(\mathbf{X}, \hat{\mathbf{X}})$, often abbreviated as *xent*. Indeed, the cross-entropy function also arises frequently in the Information Theory literature, especially as it can be rewritten as the Kullback-Leibler (KL) divergence between the categorical distributions $\mathbf{x}_t, \hat{\mathbf{x}}_t$ up to an additive constant given by the Shannon entropy of $\mathbf{x}_t$ (MacKay, 2003). Since the KL is not symmetric, the cross-entropy is thus regarded as an asymmetric deviation between the distributions $\mathbf{x}_t, \hat{\mathbf{x}}_t$.

In this classical setting for regression and classification tasks, the loss functions described above are typically optimised in the *training phase* through gradient-based iterative algorithms that take advantage of the composite structure of neural networks. Training phases are subdivided into *epochs*, which correspond to a

single scan of the training set with inputs and outputs $\mathbf{Z}, \mathbf{X}$.

During each epoch, neural network parameters θ are updated iteratively, often through a gradient-based rule (this and other approaches will be discussed in Section 2.3.4, when we tackle neural network optimisation). In particular, such gradients can be efficiently computed by *backpropagating* error signals through *backward passes* of the neural network, i.e. starting from the output layer.

This backpropagation algorithm (Rumelhart, Hinton, Williams, et al., 1988) relies on the observation that successive applications of the chain rule of calculus to composite structures, like neural networks, make certain subexpressions appear multiple times in different entries of the gradient $\nabla_\theta \mathcal{L}$. Recomputation can then be minimised by storing intermediate results and accessing them during the *backward passes*. Indeed, this and other Automatic Differentiation (AD) algorithms lie at the heart of several neural network software packages, such as Tensorflow (Martín Abadi et al., 2015) and PyTorch (Paszke et al., 2017).

After the training phase, out-of-sample predictions can then be inferred in the *testing phase* by substituting the optimal parameters θ_{MAP} in the model and performing *forward passes* over any new inputs $\mathbf{z}_{T+1}, \mathbf{z}_{T+2} \ldots$, i.e. $\hat{\mathbf{x}}_{T+P_h} = f(\mathbf{z}; \theta_{\text{MAP}})$, $P_h \in \mathbb{Z}^+$.

Dropout regularisation

The loss functions introduced in the last subsubsection:

$$\mathcal{L}_{\text{MSE}}(\mathbf{X}, \hat{\mathbf{X}}) = \frac{1}{2T} \sum_{t=1}^{T} \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 + \lambda \|\theta\|^2 ,$$
$$\mathcal{L}_{\text{Xent}}(\mathbf{X}, \hat{\mathbf{X}}) = -\sum_{t=1}^{T} \sum_{k=1}^{K} \mathbf{x}_{t,k} \log \hat{\mathbf{x}}_{t,k} - \lambda \|\theta\|^2 .$$

incorporate L_2 regularisation as a result of assuming a zero-mean Gaussian with diagonal covariance prior over the parameters θ. Regularisation schemes, such as L_2, aim to reduce the impact of overfitting, i.e. of keeping a set of parameters

that does not generalise to unseen data.

From the Bayesian point of view, overfitting can be avoided by making predictions that average across all possible explanations, as weighted by the posterior $p(\theta|\mathbf{X}, \mathbf{Z})$ after observing the data $\mathbf{X}, \mathbf{Z}$. From a historical perspective, this posed a challenge to the neural network scholar community, as even approximate Bayesian inference for "medium-sized" networks could not be achieved in a computationally efficient manner. Alternatively, such **model averaging** could be approximated by training separate neural networks over different subsets of the data, and averaging across their predictions. This approach is also computationally expensive, and relies on enough data and resources to train several neural networks.

Dropout (Srivastava et al., 2014; Hinton et al., 2012) is a regularisation technique that addresses these issues by approximately averaging across exponentially many different neural networks. It relies on the observation that a single neural network with N units can be regarded as a collection of 2^N *thinned* subnetworks, each of which can be *sampled* by temporarily dropping out units at random.

In practice, this effectively turns a neural network's forward pass into a *stochastic* pass at training time. To illustrate, consider a fully-connected layer (with n_l units and a non-linearity ϕ) and binary *mask* encoded as a diagonal matrix $\mathbf{M} \in \{0, 1\}^{n_l \times n_l}$ where each element in the diagonal $\mathbf{M}_k \sim \text{Ber}(\mathbf{M}_k \mid 1 - p_{\text{dropout}})$, and $p_{\text{dropout}} \in [0, 1]$ is the probability of dropping a unit in the layer. Then the forward pass of an input $\mathbf{z}_t$ is given by:

$$\mathbf{x}_t = \phi(\mathbf{M}\mathbf{W}\mathbf{z}_t + \mathbf{b}) \qquad \mathbf{W} \in \mathbb{R}^{n_l \times n_l}, \mathbf{b} \in \mathbb{R}^{n_l}.$$

The mask $\mathbf{M}$ is sampled *before* every forward pass, and the gradients of all unused parameters are set to zero during the corresponding backward pass.

Another interpretation of dropout is that it prevents units from learning brittle

complex co-adaptations. These arise when a neuron becomes dependent on a "large" number of neurons in the previous layer. The authors in (Srivastava et al., 2014) motivate the method based on the role of sex in the process of evolution, where a gene's ability to work well with other random sets of genes could be more important than individual fitness. In its original description, dropout is only used at training time. At testing time, no dropout masks are drawn and all weights are scaled by $1/p_{\text{dropout}}$, which ensures the unit's expected output at both training and testing times remains equal.

Dropout as approximate Bayesian inference

Recent developments by Gal and Ghahramani (2016b) have shown a link between dropout regularisation and performing approximate Bayesian inference over a deep Gaussian Process (GP), as we introduce below.

Recall the definition of the predictive posterior distribution for new inputs $\mathbf{z}_{T+1}$ after observing the data $\mathbf{Z}, \mathbf{X}$:

$$p(\mathbf{x}_{T+1} \mid \mathbf{z}_{T+1}, \mathbf{Z}, \mathbf{X}) = \int p(\mathbf{x}_{T+1} \mid \mathbf{z}_{T+1}, \theta) p(\theta \mid \mathbf{Z}, \mathbf{X}) d\theta,$$

where the posterior distribution $p(\theta \mid \mathbf{Z}, \mathbf{X})$ is intractable. Variational Inference (VI) is an approximate Bayesian inference method where we propose a tractable *variational distribution* $q(\theta)$ found by minimising the KL divergence $\text{KL}(q(\theta) \| p(\theta \mid \mathbf{Z}, \mathbf{X}))$ with respect to the true posterior, which is equivalent to maximising the log evidence lower bound (ELBO):

$$\mathcal{L}_{\text{VI}} = \int q(\theta) \log p(\mathbf{X} \mid \mathbf{Z}, \theta) d\theta - \text{KL}(q(\theta) \| p(\theta)),$$

where $p(\theta)$ is our prior distribution over the parameters. This variational posterior

gives rise to the approximate predictive posterior:

$$q(\mathbf{x}_{T+1} \mid \mathbf{z}_{T+1}) = \int p(\mathbf{x}_{T+1} \mid \mathbf{z}_{T+1}, \theta) q(\theta) d\theta.$$

In their work, Gal and Ghahramani (2016b) showed that an arbitrary neural network with dropout applied before every parameterised layer is equivalent to an approximate deep GP. In particular, they show that the loss function $\mathcal{L}_{\text{VI}}$ that arises for inferring a class of variational deep GP can be reduced to the L_2 regularised neural network loss functions $\mathcal{L}_{\text{MSE}}, \mathcal{L}_{\text{Xent}}$, introduced earlier in this subsection, with dropout applied before every parameterised layer during training time.

One crucial difference between the original dropout model averaging presented in (Srivastava et al., 2014) and that of the Monte Carlo (MC) dropout proposed by (Gal and Ghahramani, 2016b) is the latter shows that a single forward pass with dropout enabled *at testing time* is equivalent to drawing a sample from $\hat{\mathbf{x}}_{T+1}^{(s)}$ of the approximate predictive posterior distribution. The *predictive posterior mean* can then be readily computed as follows: let $\hat{\theta}^{(s)}$ be the s-th set of neural network parameters obtained after applying a sample dropout mask; then:

$$\begin{aligned} \hat{\mathbf{x}}_{T+1}^{(s)} &= f(\mathbf{z}_{T+1}; \hat{\theta}^{(s)}), \\ \mathbb{E}_{q(\mathbf{x}_{T+1} | \mathbf{z}_{T+1})}[\mathbf{x}_{T+1}] &\approx \frac{1}{N_s} \sum_{s=1}^{N_s} \hat{\mathbf{x}}_{T+1}^{(s)}, \end{aligned}$$

which is equivalent to performing N_s stochastic forward passes of the model. We also note that, in addition to a theoretical argument, Gal and Ghahramani (2016b) gave an empirical performance analysis of their method with respect to other state-of-the-art approximate Bayesian inference approaches for deep neural networks, achieving or surpassing their performance in most cases.

Final remarks

In this section, we introduced the approximate Bayesian inference framework that we shall use to quantify uncertainty in our long-term time series forecasting framework: MC dropout. One crucial motivation for using this framework is it's computational efficiency at both training and testing times. Training is recast as an optimisation problem of loss functions such as $\mathcal{L}_{\text{MSE}}$ and $\mathcal{L}_{\text{Xent}}$ above, with the addition of a sampling step before every forward pass. At testing time, a single stochastic forward pass of the model can be regarded as a sample drawn from the predictive posterior distribution. Given the relevance of optimisation algorithms in this learning process, we dedicate the next subsection to a review on this matter.

2.3.4 Gradient-based optimisation

At this point, we have presented how we can learn a neural network's parameters through approximate Bayesian inference, which in turn can be recast as an optimisation problem. In this subsection, we direct our attention to optimising the objectives $\mathcal{L}(\mathbf{X}, \hat{\mathbf{X}})$ we introduced in Section 2.3.3.

We start by remarking that global minimisation of $\mathcal{L}(\mathbf{X}, \hat{\mathbf{X}})$ is intractable as it would require solving a very large system of non-linear equations that grows with the number of neural network parameters. As a consequence, iterative optimisation methods are widespread in the neural network community, and have recently gathered substantial attention on par with the increased deployment of deep learning models. This is the case in particular of first-order methods that incorporate curvature information (similar in spirit to second-order methods), such as AdaGrad (Duchi, Hazan, and Singer, 2011).

Mini-batch Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is based on the original steepest descent algorithm, which sequentially performs parameter updates as given by:

$$\theta_{\tau+1} = \theta_\tau - \eta(\nabla\mathcal{L})_{\theta_\tau},\ \eta \in \mathbb{R}^+,$$

where η is the learning rate or step size, θ_τ is the value of model's parameters after $\tau \in \mathbb{N}$ updates, and $(\nabla\mathcal{L})_{\theta_\tau}$ is the gradient of the objective $\mathcal{L}(\mathbf{X}, \hat{\mathbf{X}})$ evaluated at θ_τ.

Note that computing the exact gradient of the summation above requires the calculation of $\hat{\mathbf{x}}_t\ \forall t$. As a result, a single gradient computation for huge datasets may be too computationally expensive, and potentially redundant in cases with repeated or too similar entries in the dataset. In the case of steepest descent, this would additionally result in a single parameter update per epoch (a single *forward* pass of the entire dataset).

In consequence, mini-batch stochastic approaches (Bottou, 2010) are often preferred for computational efficiency. By choosing a random subset of the dataset $\mathbf{Z}' \subset \mathbf{Z}, \mathbf{X}' \subset \mathbf{X}$, we can approximate the gradient by taking the empirical mean over $\mathbf{Z}', \mathbf{X}'$ only. Moreover, let us recall that the sample mean error is $O(n^{-\frac{1}{2}})$, and therefore the returns for each additional point in the sample only grow sub-linearly (Goodfellow, Bengio, and Courville, 2016). Under this scheme, the parameters θ are updated once per mini-batch, and thus multiple times per epoch. Typically, as a rule of thumb minibatches have between 128-256 examples (Murphy, 2012), but this decision is also guided by the amount of data and hardware resources available.

Automatic Differentiation (AD)

We now consider computing the gradient of the stochastic loss $\mathcal{L}(\mathbf{X}, \hat{\mathbf{X}})$. In the literature, this is most often achieved through *backpropagation* (Rumelhart, Hinton, Williams, et al., 1988), an algorithm that efficiently propagates error signals through successive application of the chain rule of calculus.

Additionally, backpropagation is also based on the insight that the hierarchical structure of models like neural networks will induce common subexpressions for different entries of the gradient $\nabla_\theta \mathcal{L}$. Backpropagation minimises recomputation of such subexpressions by storing intermediate results computed in the *forward pass* of the neural network. More generally, backpropagation is also a type of reverse-mode Automatic Differentiation (AD) algorithm, which is itself a general method to compute derivatives of functions specified by computer programs (Kingma and Ba, 2014).

By taking advantage the fact that all such programs can be parsed into (directed) *computational graphs* where each node performs an elemental operation over other nodes, i.e. encapsulating function composition, AD introduces the algorithms that build, store and traverse these computational graphs to efficiently evaluate derivatives. Such elemental operations do not only include additions and products, but also function families such as the trigonometric and exponential whose derivatives can also be expressed in terms of the same set of elemental operations.

Taking the derivative of a node with respect to a set of nodes now requires traversing the computational graph, which is equivalent to the application of the chain rule of calculus for composite functions. Moreover, this graph traversal is coupled with a storage mechanism for intermediate results, which correspond to partial factors in the application of the chain rule (Goodfellow, Bengio, and Courville, 2016). Storing these intermediate results is crucial to maximise computational efficiency, as derivatives with respect to different independent variables

may share common *computational subgraphs*, i.e. a common subsequence in their chain rule expansion. For example, let us consider the case of a shallow MLP, where the derivatives of the final layer activation functions appear in the chain rule expansion for all parameters in the network. Ensuring these intermediate results are stored avoids recomputation every time a different derivative is queried.

We conclude this discussion by highlighting that, as of the time of writing, most state-of-the-art neural network libraries, such as Tensorflow (Martín Abadi et al., 2015) and PyTorch (Paszke et al., 2017), implement some version of AD. This has contributed to accelerate the deployment and availability of deep neural network models, in parallel with the development of specialised hardware such as GPUs. Although a more detailed presentation of this framework exceeds the scope of this thesis, we refer the reader to the comprehensive introductions given in (Goodfellow, Bengio, and Courville, 2016; Bücker et al., 2005; Bishop, 2006).

Learning rates and momentum

We now direct our attention to the learning rate η, introduced in the steepest descent equation:

$$\theta_{\tau+1} = \theta_\tau - \eta(\nabla\mathcal{L})_{\theta_\tau},\ \eta \in \mathbb{R}^+.$$

Subject to the loss function, choosing large values for η may miss the optimum and prevent the algorithm from converging, while too small values may decrease its efficiency by requiring more iterations to converge. Adaptive learning rate methods aim to solve both of these issues by adapting the learning rate according to an update schedule, which in turn is based on an approximation to the Hessian of the loss function constructed from information unveiled by previous gradient computations (Murphy, 2012). Adaptive Gradient (AdaGrad) (Duchi, Hazan, and Singer, 2011), the pioneer in this class of techniques, maintains an individual

step size schedule for each learnable parameter $\theta[k]$. Consider the gradient $g_\tau[k]$ with respect to parameter $\theta[k]$ at time τ. Then the AdaGrad update is given by:

$$\theta_{\tau+1}[k] = \theta_\tau[k] - \eta \frac{g_\tau[k]}{\epsilon_0 + \sqrt{G_\tau[k]}},$$

where

$$G_\tau[k] = G_{\tau-1}[k] + g_\tau[k]^2,$$

and $\epsilon_0 \approx 10^{-8}$ is a smoothing factor to avoid division by zero.

Further information about the curvature of the loss function can be added to the update equations above through a **momentum** term ν_τ. In particular, by keeping a moving average of the gradients:

$$\begin{aligned} \theta_{\tau+1} &= \theta_\tau - \nu_\tau, \\ \nu_\tau &= \gamma \nu_{\tau-1} + \eta (\nabla \mathcal{L})_{\theta_\tau}, \ \gamma \in [0, 1), \end{aligned}$$

we can dampen the effect of very noisy approximations to the true gradient, thus discouraging abrupt changes of direction (also known as the zigzagging effect in some literature (Nocedal and Wright, 2006)).

One recent algorithm that integrates these improvements is Adam (Kingma and Ba, 2014), which stands for Adaptive Moment Estimation, where moment in this case refers to the bias-corrected estimates of the first- and second-order moments of the gradient. Adam keeps an exponentially decaying sum of gradients that emulates the performance of momentum, and has become another typical choice of optimiser in the deep learning community (Goodfellow, Bengio, and Courville, 2016).

2.3.5 Final remarks

In this section, we have presented some of the core terminology in the neural network literature that paves the way to the presentation of the Long Short-Term Memory (LSTM) in Chapter 3, which forms the backbone of our method, after briefly introducing them in Subsection 2.3.2. We will also resume our discussion about the computation of posterior predictive distributions in next chapter.

3 Ordinal Forecasting with Recurrent Neural Networks

In this chapter, we focus our attention on the main framework developed in this thesis: the Memory-endowed Ordinal Regression Deep neural network (MOrdReD), which is a probabilistic, scalable and end-to-end deep learning approach to time series prediction. By incorporating a time series quantisation scheme, MOrdReD is able to recast the time series forecasting task into a symbol sequence learning one. A number of state-of-the-art deep learning models can then be readily incorporated into our framework, such as the Long Short-Term Memory (LSTM). These methods have shown remarkable performance in sequential learning tasks, such as those that frequently arise in the NLP literature (Sutskever, Martens, et al., 2013); however, to the best of the author's knowledge, they have not attracted the same success in the time series literature, as described in Chapter 2.

We firstly detail the major components of our framework, paying special attention to quantisation schemes, the LSTM Recurrent Neural Network architecture, as well as Monte Carlo Dropout approximate Bayesian inference in Sections 3.1, 3.2 and 3.3, respectively. These three components vitally enable MOrdReD to infer long-term predictive distributions.

After, we evaluate our framework with respect to other state-of-the-art methods for time series forecasting (TSF) with a large-scale experiment in Section 3.5. Herein, we measure the accuracy and calibration of our models' predictive distributions for 45 different time series datasets from a variety of application domains, such as dynamical systems, meteorology, physiology and others.

We find that our model is empirically capable of outperforming other state-of-the-art competitors in terms of long-term forecasting uncertainty estimation, whilst also inheriting all the advantages of neural network models, such as scalability with dataset size. As a result, our framework is consistently top-ranked in all evaluation metrics. We additionally provide evidence that it would also be the second-best performing model whenever another baseline achieved the best rank, and in a similar vein, we show that it is unlikely that it will be outperformed by all other baselines simultaneously in any given task, for any metric considered in the assessment.

This study, though comprehensive, still assumes the existence of enough data to fit our model. We directly address this issue in Chapter 5, where we propose a General Unified Model for time series forecasting that can be used as a pre-trained model for time series with scarce data.

3.1 Ordinal regression

The time series forecasting (TSF) methodology introduced in this thesis recasts the problem to an *ordinal regression* task (Harrell Jr, 2015). Ordinal regression, also known as ordinal classification in some literature, lies at the core between plain classification and regression tasks by requiring data labels drawn from an ordered set $\mathcal{C} = \{\mathcal{C}_k\}_{k=1}^{M}$ that satisfies $\mathcal{C}_k \leq \mathcal{C}_{k+1}$, $\forall k$.

To illustrate, let us consider a bounded univariate time series

$$\mathbf{x}^{(\text{real})} = (x_1^{(\text{real})}, \ldots, x_T^{(\text{real})}),\ x_T^{(\text{real})} \in I \subset \mathbb{R},$$

with T measurements and range $I = [\gamma_{\min}, \gamma_{\max}] \subset \mathbb{R}$ and let $\mathcal{C}$ be a partition of I with cardinality $|\mathcal{C}| = M$. Without loss of generality, we assume all the $\mathcal{C}_k$ have the same measure on $\mathbb{R}$, e.g. given $M+1$ equidistant thresholds $\{\gamma_k\}_{k=1}^{M+1}$ such that $\gamma_1 = \gamma_{\min}, \gamma_{M+1} = \gamma_{\max}$ are the minimum and maximum observations in I, then $\mathcal{C}_k = [\gamma_k, \gamma_{k+1})$ are non-overlapping sub-intervals of I with equal step size Δ.

We can then recast time series forecasting as a symbol sequence prediction task by encoding each observation $\mathbf{x}_t^{(\text{real})}$ using a 1-of-M scheme[1]:

$$\mathbf{x}_t = (\mathbb{I}(x_t^{(\text{real})} \in \mathcal{C}_1), \mathbb{I}(x_t^{(\text{real})} \in \mathcal{C}_2), \ldots, \mathbb{I}(x_t^{(\text{real})} \in \mathcal{C}_M)),$$

where $\mathbb{I}(\cdot)$ is the indicator function. To simplify future notation, we will henceforth use $\mathbf{X} \in \{0,1\}^{T \times M}$ to describe a full time series of interest with T one-hot encoded observations $\mathbf{x}_t \in \{0,1\}^M$ subject to the *quantisation* scheme above, in contrast with the original real-valued time series $\mathbf{x}^{(\text{real})} \in \mathbb{R}^T$. We note that the curly bracket notation from formal languages theory denotes the repetition of the integer tokens $0, 1$, i.e. all entries in $\mathbf{x}_t$ are either 0 or 1.

In our ordinal regression task, we will assume that each symbol $\mathbf{x}_t$ is independent of all other symbols observed more than P timesteps before. This induces a dataset of pairs $\mathcal{X} = (\mathbf{X}^{(t-1)}, \mathbf{x}_t)_{t=P+1}^{T}$, where each $\mathbf{X}^{(t-1)} = (\mathbf{x}_{t-P}, \ldots, \mathbf{x}_{t-1})$, $\mathbf{X}^{(t-1)} \in \{0,1\}^{P \times M}$ is an *input sequence* and $\mathbf{x}_t \in \{0,1\}^M$ is the one-hot encoded target symbol. We now aim to infer a model f with parameters θ:

$$\hat{\mathbf{x}}_t = f(\mathbf{X}^{(t-1)}; \theta).$$

[1] We will use the terms 1-of-M and one-hot encoding interchangeably.

As a *sequential learning* problem, in time series forecasting we are interested in characterising the *temporal dependencies* across time series measurements through this model f. Long Short-Term Memory (LSTM)s, which we introduce in the next section, naturally come to light as a befitting methodology due to their empirically proven ability to model sequences over large symbol sets, such as those that frequently arise in the NLP literature (Sutskever, Vinyals, and Le, 2014). With this parametric model in hand, our *ordinal time series forecasting* objective is then to assess the predictive posterior distribution:

$$p(\mathbf{x}_{T+1} \mid \mathbf{X}^{(T)}, \mathcal{X}) = \int p(\mathbf{x}_{T+1} \mid \mathbf{X}^{(T)}, \theta) p(\theta \mid \mathcal{X}) d\theta,$$

where $p(\theta \mid \mathcal{X})$ is the posterior distribution over the model parameters θ. Extensions to the Monte Carlo dropout framework introduced in Section 2.3.3 have shown how to efficiently perform approximate Bayesian inference over such *recurrent neural network* models (Gal and Ghahramani, 2016a). We introduce this extension in Section 3.3. Changing between one-hot encoded representations and the original time series can make the notation cumbersome, so we briefly summarise our notation below:

- $\mathbf{x}^{(\text{real})} \in \mathbb{R}^T$ is the original real-valued univariate time series, with measurements $x_t^{(\text{real})} \in \mathbb{R}$.
- $\mathbf{X} \in \{0, 1\}^{T \times M}$ is the entire one-hot encoded time series, with T one-hot encoded measurements $\mathbf{x}_t \in \{0, 1\}^M$.
- $\mathbf{X}^{(t-1)} \in \{0, 1\}^{P \times M}$ is a time series excerpt of P one-hot encoded consecutive observations finishing at index $t - 1$, i.e. $\mathbf{X}^{(t-1)} = (\mathbf{x}_{t-P}, \ldots, \mathbf{x}_{t-1})$.
- $\mathcal{X} \in \mathbb{R}^{N \times P \times M} = (\mathbf{X}^{(t-1)}, \mathbf{x}_t)_{t=P}^T$ is the time series dataset comprising $N = T - P$ time series excerpts of length P.

We conclude this section with a few remarks about quantisation. Firstly, let

us note that the ordinality of the class labels $\mathcal{C}$ enables us to interpret the one-hot encoded predictions $\hat{\mathbf{x}}_t$ as two different probability distributions: on one hand, as a multinomial over the class labels $\mathcal{C}_k$; on the other hand, as a piece-wise uniform distribution over the original real-valued measurements $x_t^{(\text{real})}$ given by:

$$p(x_t^{(\text{real})} \mid \hat{\mathbf{x}}_t) = \frac{1}{\Delta} \prod_{k=1}^{M} \hat{\mathbf{x}}_t^{\mathbb{I}(x_t^{(\text{real})} \in \mathcal{C}_k)}.$$

This equivalence will play a key role when we evaluate predictive uncertainty in comparison with other models that do not follow an ordinal approach for prediction. Let us also remark that, contrary to other regression approaches whose assumptions lead to unimodal Gaussian predictive distributions, an ordinal take on the task leads to a flexible parametric predictive distribution that can represent, for instance, multimodality. As discussed in Section 2.1.3, this is a desirable property to describe long-term forecasts.

Finally, note that introducing an ordinal regression framework comes at the expense of introducing a quantisation distortion error. Intuitively, letting the number of ordinal bins $M \to \infty$, or $\Delta \to 0$, will minimise the reconstruction error of the signal, but only at greater computational cost. Within reason, the choice of M is therefore problem-dependent and a trade-off between resolution and computational simplicity.

3.2 The Long Short-Term Memory

As motivated in Section 3.1, Recurrent Neural Network models are state-of-the-art in a number of sequential learning tasks, which makes them a suitable methodology for our ordinal regression framework. In this section we now resume our discussion about this family of Recurrent Neural Network (RNN) layers, as briefly introduced in Subsection 2.3.2. In particular, our time series forecasting framework is based on the Long Short-Term Memory (LSTM), which is a recurrent

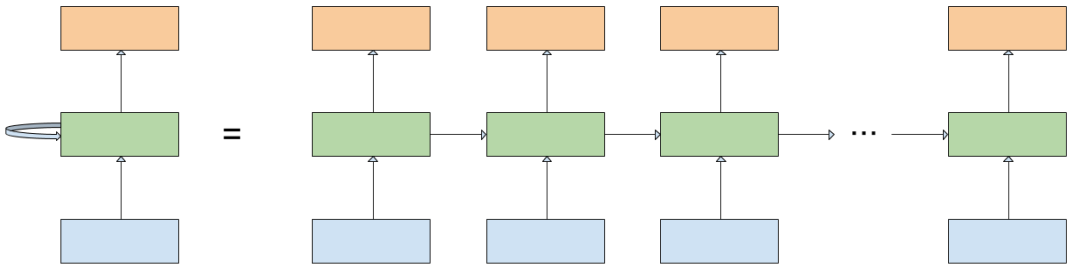


Figure 3.1: Recurrent neural network with three layers: an input layer (blue), a recurrent layer (green) and an output layer (orange). Recurrent layers store their outputs at time $t-1$ and update them every time they are supplied with a new observation $\mathbf{x}_t$. Equivalently, $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta)$, which can be observed more clearly by unrolling the self-edge (left) a finite number of steps (right).

architecture that efficiently propagates error signals in time through a gating mechanism.

To motivate the introduction of LSTMs, we start this section by giving a more detailed presentation of the basic RNN layer and show why it is unable to learn long-term dependencies in Subsection 3.2.1. This paves the way to introduce the LSTM in Subsection 3.2.2 and state-of-the-art extensions such as the sequence-to-sequence model in Subsection 3.2.3.

3.2.1 Recurrent Neural Networks

In Section 2.3, we gave a detailed presentation of feedforward neural networks, i.e. those whose layers are connected sequentially. We now turn our attention to Recurrent Neural Network (RNN) layers, which model dynamical systems through cyclical topologies, i.e. they update their output from time $t-1$ with incoming symbols at time t such that $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta)$, where $\mathbf{x}_t \in \mathbb{R}^M$ is the 1-of-M representation of a time series measurement. To illustrate this mechanism, in Figure 3.1 we show how recurrent layers can be unrolled to map an input sequence to an output sequence.

The simplest type of RNN layer is a direct extension of the fully-connected (FC) layer introduced in Section 2.3.1, where each of the n_h units in the recurrent layer is connected to all other neurons in the same layer, i.e.

$$\mathbf{h}_t = f_{\text{RNN}}(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta_{\text{RNN}}) = \phi(\mathbf{W}_{\text{rec}}\mathbf{h}_{t-1} + \mathbf{W}_{\text{in}}\mathbf{x}_t + \mathbf{b}),$$

where ϕ is the activation function, as discussed in Section 2.3.1, and $\theta_{\text{RNN}} = \{\mathbf{W}_{\text{in}}, \mathbf{W}_{\text{rec}}, \mathbf{b}\}$, $\mathbf{x}_t \in \mathbb{R}^M, \mathbf{W}_{\text{in}} \in \mathbb{R}^{n_h \times M}, \mathbf{W}_{\text{rec}} \in \mathbb{R}^{n_h \times n_h}, \mathbf{b} \in \mathbb{R}^{n_h}$. For future convenience, the expression above can be re-arranged as:

$$\mathbf{h}_t = f_{\text{RNN}}(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta_{\text{RNN}}) = \mathbf{W}_{\text{rec}}\phi(\mathbf{h}_{t-1}) + \mathbf{W}_{\text{in}}\mathbf{x}_t + \mathbf{b}$$

Through recursive substitution, it is now easy to see that:

$$\mathbf{h}_t = f_{\text{RNN}}(\mathbf{x}_t, f_{\text{RNN}}(\mathbf{x}_{t-1}, f_{\text{RNN}}(\dots f_{\text{RNN}}(\mathbf{x}_1, \mathbf{h}_0; \theta_{\text{RNN}}) \dots; \theta_{\text{RNN}}); \theta_{\text{RNN}}),$$

where $\mathbf{h}_0$ is the *initial state* of the recurrent network.

Let us now resume our discussion about gradient-based optimisation from Section 2.3.4. In particular, let us outline the derivatives of $\mathbf{h}_t$ with respect to θ_{RNN}, and more specifically, with respect to the parameters $\mathbf{W}_{\text{rec}}$ associated with the self-edge of the recurrent layer. For simplicity, consider any $w_k \in \mathbf{W}_{\text{rec}}$ and the derivative $\frac{\partial \mathbf{h}_t}{\partial w_k}$. Application of the product rule and the chain rule gives:

$$\begin{aligned}
\frac{\partial \mathbf{h}_t}{\partial w_k} &= \frac{\partial \mathbf{W}_{\text{rec}}}{\partial w_k}\phi(\mathbf{h}_{t-1}) + \mathbf{W}_{\text{rec}}\frac{\partial \phi(\mathbf{h}_{t-1})}{\partial \mathbf{h}_{t-1}}\frac{\partial \mathbf{h}_{t-1}}{\partial w_k} \\
&= \frac{\partial \mathbf{W}_{\text{rec}}}{\partial w_k}\phi(\mathbf{h}_{t-1}) + \mathbf{W}_{\text{rec}}\phi'(\mathbf{h}_{t-1})\frac{\partial \mathbf{h}_{t-1}}{\partial w_k} \\
&= \frac{\partial \mathbf{W}_{\text{rec}}}{\partial w_k}\phi(\mathbf{h}_{t-1}) + \mathbf{W}_{\text{rec}}\phi'(\mathbf{h}_{t-1})\left(\frac{\partial \mathbf{W}_{\text{rec}}}{\partial w_k}\phi(\mathbf{h}_{t-2}) + \mathbf{W}_{\text{rec}}\phi'(\mathbf{h}_{t-2})\frac{\partial \mathbf{h}_{t-2}}{\partial w_k}\right) \\
&= \sum_{\tau=0}^{t-1}\left(\prod_{i=1}^{\tau}\mathbf{W}_{\text{rec}}\phi'(\mathbf{h}_{t-i})\right)\frac{\partial \mathbf{W}_{\text{rec}}}{\partial w_k}\phi(\mathbf{h}_{t-\tau-1})
\end{aligned}$$

The effect of the product factor above plays a key role in gradient-based long-term dependency learning for Recurrent Neural Networks. In particular, consider the stability of the contributions made by the product factor for large values of τ, i.e. as the lag $\tau \to t$, the gradient may shrink or explode, thus making it difficult to update the parameters θ on the basis of variations made at distant lookback horizons. Such instability has also been related to the evolution of the eigenvalues of matrix powers $\mathbf{W}^n, n \in \mathbb{N}, n \to \infty$. We refer the reader to Pascanu, Mikolov, and Bengio (2013) and Goodfellow, Bengio, and Courville (2016) for an in-depth analysis on this matter.

Appropriate regularisation of such recurrent parameters lies at the core of solving the vanishing and exploding gradients problem. In the case of exploding gradients, a scheme that is commonly found in practice is the clipping rule (Pascanu, Mikolov, and Bengio, 2013). From a geometrical perspective, regions with exploding gradients correspond to "cliffs" or walls in the loss function. Taking a step in the direction of the gradient in these regions moves the estimate θ_τ away from the cliff, discouraging the local exploration of areas near the cliff itself. Empirical evidence suggests that this can be mitigated by strategies such as L_2 regularisation in conjunction with re-scaling exploding gradients as per the clipping rule:

$$\nabla_\theta \mathcal{L} \leftarrow \max\left(1, \frac{\epsilon_{\text{clip}}}{\|\nabla_\theta \mathcal{L}\|}\right) \nabla_\theta \mathcal{L}$$

In addition to switching to non-gradient-based methods (Schmidhuber et al., 2007) and introducing novel regularisation schemes (Pascanu, Mikolov, and Bengio, 2013), the case of the vanishing gradient has been dealt with by *gated* recurrent layers in a number of sequence learning tasks (Sutskever, Vinyals, and Le, 2014; Graves and Schmidhuber, 2005; Xu et al., 2014; You et al., 2016). This type of recurrent layer, including the Long Short-Term Memory (LSTM), structurally

addresses the vanishing gradient problem by incorporating gating mechanisms that control the flow of error signals through time.

3.2.2 The Long Short-Term Memory

We now look at the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997), a particular type of Recurrent Neural Network equipped with a gating mechanism that enables efficient propagation of error signals in time. By introducing a Constant Error Carousel (CEC), the LSTM directly tackles the vanishing gradient issue that arises in simple recurrent neural network models, as described in Section 3.2.1. Salient temporal features can thence be efficiently learnt through gradient-based optimisation, using the methods described in Section 2.3.4.

Consider an observed sequence $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_P)$, and let $\odot$ be the element-wise product operator. For each $\mathbf{x}$ with $t = 1, \ldots, P$, the LSTM computes a vector of temporal features $\mathbf{h}_t$, which is given by:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t).$$

Here, $\mathbf{C}_t$ is the LSTM memory cell at time t:

$$\mathbf{C}_t = \mathbf{i}_t \odot \mathbf{S}_t + \mathbf{f}_t \odot \mathbf{C}_{t-1},$$

where

$$\mathbf{S}_t = \tanh\left(\mathbf{W}_{\mathbf{S}}\mathbf{x}'_t + \mathbf{b}_{\mathbf{S}}\right),$$

and $\mathbf{i}_t, \mathbf{o}_t, \mathbf{f}_t$ are the input, output and forget gates respectively:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{\mathbf{i}}\mathbf{x}'_t + \mathbf{b}_{\mathbf{i}}), \qquad \mathbf{o}_t = \sigma(\mathbf{W}_{\mathbf{o}}\mathbf{x}'_t + \mathbf{b}_{\mathbf{o}}), \qquad \mathbf{f}_t = \sigma(\mathbf{W}_{\mathbf{f}}\mathbf{x}'_t + \mathbf{b}_{\mathbf{f}}),$$

where each $\mathbf{W}, \mathbf{b}$ are learnable parameters for each component, respectively. Here σ is the element-wise sigmoid activation function and $\mathbf{x}'_t = (\mathbf{x}_t, \mathbf{h}_{t-1})$ is the concatenation of the observation $\mathbf{x}_t$ and the LSTM activations at the previous time step $\mathbf{h}_{t-1}$.

Let us note that the cell $\mathbf{C}_t$ can be thought of as an encoded memory with dynamic read and update operations, i.e. the LSTM gates control the state of each memory cell by regulating the influx of its two main sources, the input at time t and the storage at time $t-1$:

$$\mathbf{C}_t = \mathbf{i}_t \odot \mathbf{S}_t + \mathbf{f}_t \odot \mathbf{C}_{t-1}$$

$$\text{memory}_t = \text{read} \odot \text{features}_t + \text{retain} \odot \text{memory}_{t-1}.$$

Additionally, we remark that all four components $\mathbf{S}_t, \mathbf{i}_t, \mathbf{o}_t, \mathbf{f}_t$ are simple Recurrent Neural Networks, such as those detailed in Section 3.2.1. This suggests that they may all individually suffer from the gradient instability issues just described. The LSTM architecture avoids this by coupling the gates as given in the definition of the activations and memory $\mathbf{h}_t, \mathbf{C}_t$ above. In particular, this sets out a CEC that permits efficient gradient-based learning and prevents the vanishing gradient problem for the LSTM.

Below, we illustrate the effect of the CEC by sketching out the LSTM activation and memory cell gradients for a parameter $w_k \in \theta$:

$$\frac{\partial \mathbf{h}_t}{\partial w_k} = \frac{\partial \mathbf{o}_t}{\partial w_k} \odot \tanh(\mathbf{C}_t) + \mathbf{o}_t \odot \frac{\partial \tanh(\mathbf{C}_t)}{\partial \mathbf{C}_t} \frac{\partial \mathbf{C}_t}{\partial w_k},$$
$$\frac{\partial \mathbf{C}_t}{\partial w_k} = \frac{\partial \mathbf{i}_t}{\partial w_k} \odot \mathbf{S}_t + \mathbf{i}_t \odot \frac{\partial \mathbf{S}_t}{\partial w_k} + \frac{\partial \mathbf{f}_t}{\partial w_k} \odot \mathbf{C}_{t-1} + \mathbf{f}_t \odot \frac{\partial \mathbf{C}_{t-1}}{\partial w_k}.$$

By using a similar reasoning to the previous derivation, we can see that the factors $\frac{\partial \mathbf{i}_t}{\partial w_k}, \frac{\partial \mathbf{o}_t}{\partial w_k}, \frac{\partial \mathbf{f}_t}{\partial w_k}, \frac{\partial \mathbf{S}_t}{\partial w_k}$ may all (partially) vanish, since they are recursive (dependent on $\mathbf{x}'_t$) and have the shape $\phi(\mathbf{W}\mathbf{x}'_t + \mathbf{b})$.

Nevertheless, this may not be the case for the term $\frac{\partial \mathbf{C}_t}{\partial w_k}$, which is also a recursive on $\frac{\partial \mathbf{C}_{t-1}}{\partial w_k}$. Firstly, remark that $\mathbf{C}_t$ is a function of $\mathbf{C}_{t-1}$ prior to the application of the non-linearity. If we assume that the gradients $\frac{\partial \mathbf{i}_t}{\partial w_k}, \frac{\partial \mathbf{o}_t}{\partial w_k}, \frac{\partial \mathbf{f}_t}{\partial w_k}, \frac{\partial \mathbf{S}_t}{\partial w_k}$ vanish, then:

$$\frac{\partial \mathbf{C}_t}{\partial w_k} \approx (\mathbf{f}_t \odot \mathbf{f}_{t-1} \odot \cdots \odot \mathbf{f}_{t-i}) \odot \frac{\partial \mathbf{C}_{t-i-1}}{\partial w_k}$$

As a result, error signals only dampen between timesteps subject to their deletion by the forget gate $\mathbf{f}_t$, thus enabling LSTMs to learn long-term dependencies.

We conclude this subsection by remarking that the LSTM architecture above only addresses the vanishing gradient problem, but not the exploding gradient case. This is why the LSTM is also used in conjunction with gradient clipping, as described in Section 3.2.1 (Goodfellow, Bengio, and Courville, 2016).

3.2.3 Sequence-to-Sequence Neural Networks

At present, LSTM-based models are state-of-the-art in several sequential learning tasks, such as: speech recognition (Graves, Mohamed, and Hinton, 2013), sequence generation (handwriting: (Graves, 2013), text: (Karpathy, 2015)), financial forecasting (Rutkauskas, Maknickas, and Maknickienė, 2011), reinforcement learning (Hausknecht and Stone, 2015) and models of attention (scene labelling) (You et al., 2016; Vinyals et al., 2015).

In this literature, extensions such as the Bidirectional LSTM (BiLSTM) are commonplace. The BiLSTM extracts temporal characteristics by scanning the input sequence in both left-to-right and right-to-left fashion. This aids in extracting features that capture the relation between lookback and look-ahead observations. This bidirectional model has shown remarkable success in the NLP literature (Graves and Schmidhuber, 2005), including further extensions to LSTM-based models such as the sequence-to-sequence architecture (Sutskever, Vinyals, and

Le, 2014).

We now focus our attention on the sequence-to-sequence model, which consists of two joint Long Short-Term Memory models: an encoder $f^{(\text{enc})}$ and a decoder $f^{(\text{dec})}$. Consider a sequence $\mathbf{X}$ of length P. The encoder then maps the first $P-1$ observations $\mathbf{X}[:P-1]$ into fixed-dimensional summaries, $\mathbf{h}_0^{(\text{dec})}, \mathbf{C}_0^{(\text{dec})}$; and the decoder $f^{(\text{dec})}$, which uses $\mathbf{h}_0^{(\text{dec})}, \mathbf{C}_0^{(\text{dec})}$ as informed initial states in conjunction with the last observation $\mathbf{x}_P$ to predict future observations iteratively. More precisely:

$$
\begin{aligned}
(\mathbf{h}_0^{(\text{dec})}, \mathbf{C}_0^{(\text{dec})}) &= f^{(\text{enc})}(\mathbf{X}[:P-1]), \\
\hat{\mathbf{x}}_{t+1}^{(\text{dec})} &= \text{Softmax}\left(f^{(\text{dec})}(\mathbf{h}_{t-1}^{(\text{dec})}, \mathbf{C}_{t-1}^{(\text{dec})}, \hat{\mathbf{x}}_t^{(\text{dec})})\right),
\end{aligned}
$$

where $\hat{\mathbf{x}}_1^{(\text{dec})} = \mathbf{x}_P$ is the last available observation in the sequence. At test time, direct autoregression is enabled and $\hat{\mathbf{x}}_t^{(\text{dec})}$ is fed back as input to the decoder at time $t+1$ for all $t > 1$.

As their name suggests, sequence-to-sequence architectures are aimed at tasks where we desire to learn a map between sequences. In particular, they address the handling of two widespread issues in sequence learning: variable-length sequences and sequence alignment.

To illustrate, let us consider the task of machine translation from Spanish to English. Several pairs of sentences with the same meaning may possess a different number of tokens (variable-length), and furthermore, their individual words may need to be arranged in different order in each language in order to be grammatical (sentence alignment). These sequence-to-sequence models tackle both issues by decoupling the task in two stages: a sequential feature extractor (the encoder) that finds a fixed-dimensional vector of characteristics for the input sequence, and an iterative decoder that unfolds such characteristics into a target sequence of the required length. We provide a visual summary of this model in Figure 3.2.

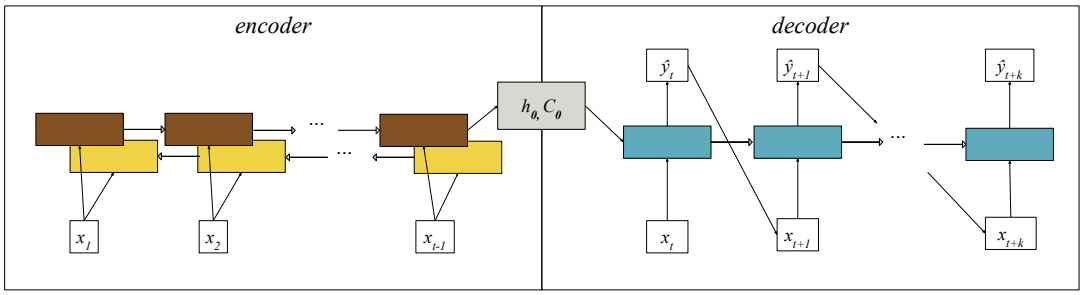


Figure 3.2: A sequence-to-sequence model with a bidirectional encoder. The input sequence is scanned up to time index $t-1$ and subsequently summarised as the decoder's initial states $\mathbf{h}_0, \mathbf{C}_0$. The decoder then produces a one-step-ahead forecast for every new incoming input up to time $t+k$. At prediction time, autoregression is enabled feeding back $\mathbf{x}_{t+i+1}$ as the input to the model.

3.3 Quantifying predictive uncertainty

In this section, we introduce how our framework is capable of assessing and propagating its own predictive uncertainty for long-term forecasting. To this end, let us recall from Section 2.3.3 that our objective now is to compute a *predictive posterior distribution* to describe our forecasts over a given predictive horizon P_h. This crucially provides practitioners with a sharper predictive panorama for their decision-making process.

Picking up from the terminology introduced in Section 2.3.3, recall that the model's posterior predictive distribution over the target $\mathbf{x}_{T+1}$ at a new input excerpt $\mathbf{X}^{(T)}$ after observing all the collection of length P time series excerpts, $\mathcal{X}$, is given by marginalising over the parameters θ:

$$p(\mathbf{x}_{T+1} \mid \mathbf{X}^{(T)}, \mathcal{X}) = \int p(\mathbf{x}_{T+1} \mid \mathbf{X}^{(T)}, \theta) p(\theta \mid \mathcal{X}) d\theta,$$

where $p(\theta \mid \mathcal{X})$ is the posterior distribution as described in Section 2.3.3.

In Section 2.3.3, we also discussed how approximate Bayesian inference of neural networks can be performed efficiently through Monte Carlo dropout (Gal and Ghahramani, 2016b) and still achieve state-of-the-art performance. Specifically, the authors showed that fitting a neural network by optimising the L_2-regularised mean squared error (for regression) and cross-entropy (for classification) losses:

$$\mathcal{L}_{\text{MSE}}(\mathbf{X}, \hat{\mathbf{X}}) = \frac{1}{2T} \sum_{t=1}^{T} \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|^2 + \lambda \|\theta\|^2 ,$$
$$\mathcal{L}_{\text{Xent}}(\mathbf{X}, \hat{\mathbf{X}}) = - \sum_{t=1}^{T} \sum_{k=1}^{K} \mathbf{x}_{t,k} \log \hat{\mathbf{x}}_{t,k} - \lambda \|\theta\|^2 ,$$

with a dropout mask enabled before every parameterised layer is equivalent to performing variational approximate Bayesian inference over a deep Gaussian Process (Damianou and Lawrence, 2013). Variational Bayes relies on minimising the KL divergence between the true posterior distribution $p(\theta \mid \mathcal{X})$ and a tractable variational proposal $q(\theta)$. From Section 2.3.3, we also recapitulate that, after minimising these dropout-enabled loss functions, a single *stochastic* forward pass of the network f with dropout-masked parameters $\hat{\theta}^{(s)}$ could be regarded as drawing a sample from the *variational predictive posterior distribution*:

$$q(\mathbf{x}_{T+1} \mid \mathbf{X}^{(T)}) = \int p(\mathbf{x}_{T+1} \mid \mathbf{X}^{(T)}, \theta)\, q(\theta)\, d\theta,$$

whose first moment can be readily approximated by Monte Carlo integration over N_s samples:

$$\hat{\mathbf{x}}_{T+1}^{(s)} = f(\mathbf{X}^{(T)}; \hat{\theta}^{(s)}),$$
$$\mathbb{E}_{q(\mathbf{x}_{T+1} | \mathbf{X}^{(T)})}[\mathbf{x}_{T+1}] \approx \frac{1}{N_s} \sum_{s=1}^{N_s} \hat{\mathbf{x}}_{T+1}^{(s)}.$$

Although this discussion is grounded and justified for feed-forward neural networks of arbitrary depth, it does not immediately extend to the case of recurrent

neural networks. Indeed, from a regularisation point of view, a naïve application of dropout to Recurrent Neural Networks could be detrimental to their performance, e.g. by dampening the signal of interest in the long-term. This led to the development of schemes on how to adequately apply dropout to recurrent networks (Bluche, Kermorvant, and Louradour, 2015; Zaremba, Sutskever, and Vinyals, 2014; Pachitariu and Sahani, 2013).

One of these schemes is an extension of Monte Carlo dropout, proposed by the same authors in (Gal and Ghahramani, 2016a). Herein, the authors provide a theoretical argument that shows how their own variational framework can be extended to recurrent networks, subject to repeating the *same dropout mask* at each time step for both inputs, outputs, and recurrent layers. This effectively amounts to dropping the same network units at each time step in a single stochastic pass. Inheriting all the computational efficiency advantages from (Gal and Ghahramani, 2016b), this is the approach we follow when performing the long-term time series forecasting experiments illustrated in the remainder of this chapter.

3.4 Memory-endowed Ordinal Regression Deep neural network

Let us now integrate our framework from the core components so far. The core of this thesis develops a Memory-endowed Ordinal Regression Deep neural network (MOrdReD)[2] for time series forecasting, which is based on the core components introduced in Sections 3.1, 3.2 and 3.3. The name "memory-endowed" makes reference to the temporal feature vectors learnt by our recurrent models.

Given a real-valued time series $\mathbf{x}^{(\text{real})}$ observed up to time T, our task is to assess the next P_h measurements, up to time $T + P_h$. In the *quantisation step*,

[2]In Arthurian legend, Mordred was the illegitimate son of King Arthur. Mordred fell in battle against his own father.

we one-hot encode the time series over a set of ordinal bins $\mathcal{C} = \{\mathcal{C}_k\}_{k=1}^{M}$ covering the range of $\mathbf{x}^{(\text{real})}$, which induces the following notation as per our discussion in Section 3.1:

- $\mathbf{x}^{(\text{real})} \in \mathbb{R}^T$ is the full original real-valued univariate time series, with measurements $x_t^{(\text{real})} \in \mathbb{R}$.
- $\mathbf{X} \in \{0,1\}^{T \times M}$ is the entire one-hot encoded time series, with one-hot encoded measurements $\mathbf{x}_t \in \{0,1\}^M$.
- $\mathbf{X}^{(t-1)} \in \{0,1\}^{P \times M}$, $t > P$ is a time series excerpt of P one-hot encoded consecutive observations finishing at index $t-1$, i.e. $\mathbf{X}^{(t-1)} = (\mathbf{x}_{t-P}, \ldots, \mathbf{x}_{t-1})$.

Ordinal forecasting with MOrdReD is divided into two stages. Firstly, in the *parameter inference* phase, we fit a sequence-to-sequence model f with parameters θ over the dataset $\mathcal{X}$, as introduced in Section 3.2. We construct our training dataset as one between sequences $\mathcal{X} = \{(\mathbf{X}^{(t)}, \mathbf{X}^{(t+P)})\}_{t=P}^{T-P}$ where the predicted sequence is given by:

$$\hat{\mathbf{X}}^{(t+P)} = f(\mathbf{X}^{(t)}; \theta),$$

where the encoder firstly scans the input sequence up to time $t-1$ to produce the fixed-dimensional summaries $\mathbf{h}_0^{(\text{dec})}, \mathbf{C}_0^{(\text{dec})}$. These fixed dimensional summaries, alongside the last entry in the input $\mathbf{x}_t$, are used by the decoder to forecast P_h timesteps ahead. The model is learnt in a similar fashion to a classification neural network model, by minimising the dropout-enabled, L_2-regularised cross-entropy loss, as defined in Section 3.3.

After the inference phase, the model can now be used in the *prediction stage* to produce forecasts. In particular, we consider the last $T-P$ observations available from the time series, encoded as $\mathbf{X}^{(T)}$, and fix a predictive horizon P_h. Although P and P_h need not be the same, for simplicity of exposition we assume they share

the same value by default in the rest of this thesis, except where explicitly stated otherwise.

Following our discussion from Section 3.3, we leave dropout enabled and perform N_s stochastic passes, each of which leads to a masked set of parameters $\hat{\theta}^{(s)}$. We pay special attention to ensuring the same masked parameters are used consistently throughout each stochastic forward pass (as described in Section 3.3). In each stochastic pass, we take $\mathbf{h}_0^{(\text{dec})}, \mathbf{C}_0^{(\text{dec})}$ with the masked parameters and feed the last available measurement $\mathbf{x}_T$ as a seed to the decoder. A single recurrent step of the decoder produces a sample forecast $\mathbf{x}_{T+1}^{(s)} \sim q(\mathbf{x}_{T+1} \mid \mathbf{X}^{(T)})$ from the approximate variational predictive posterior distribution $q(\mathbf{x}_{T+1} \mid \mathbf{X}^{(T)})$.

Such a forecast can be fed back into the decoder, in an iterative and *autoregressive* manner, to produce the remaining forecasts until timestep $T + P_h$. We designate this sample forecast as $\hat{\mathbf{X}}_{[s]}^{(T+P_h)}$ of length P_h. Let us remark that the model is used in an open-loop fashion in each stochastic pass, i.e. the output $\hat{\mathbf{x}}_{T+k}$ is fed back in an autoregressive manner as the input at time $T + k + 1$, for all $1 \leq k < P_h$. This enables long-term forecasting for a variable number of timesteps.

After collecting N_s samples, $\hat{\mathbf{X}}_{[1]}^{(T+P_h)}, \ldots, \hat{\mathbf{X}}_{[N_s]}^{(T+P_h)}$, from the variational predictive posterior distribution, we average them to obtain the predictive posterior mean *symbol sequence* forecast $\hat{\mathbf{X}}^{(T+P_h)}$. The predictive posterior mean at each time $\hat{\mathbf{x}}_{T+k}$ is readily given by:

$$\mathbb{E}[\mathbf{x}_{T+k}] \approx \frac{1}{N_s} \sum_{s=1}^{N_s} \hat{\mathbf{x}}_{T+k}^{(s)},$$

Let us recall that a single timestep forecast $\hat{\mathbf{x}}_{T+k}$, $1 \leq k \leq P_h$ has two possible interpretations: on the one hand, it is the categorical distribution over the M ordinal symbols $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_M\}$ at time $T + k$. On the other hand, we note that it is straightforward to reinterpret this ordinal predictive distribution as one

over the original, real-values time series $x_{T+k}^{(real)}$ with range $I = \bigcup_{i=1}^{M} \mathcal{C}_i$. Recalling that each $\mathcal{C}_i = [\gamma_i, \gamma_{i+1}), \gamma \in \mathbb{R}$ with measure $|\mathcal{C}_i| = \gamma_{i+1} - \gamma_i$, we can readily construct a piece-wise uniform probability density function[3] over the range of the time series I. Considering $\pi_{T+k}^{i} = \mathbb{I}(\mathbf{x}_{T+k} \in C_i)$ and $\pi_{T+k}^{\text{out}} = \mathbb{I}(x_{T+k}^{(\text{real})} \in I)$, MOrdReD's predictive distribution $p_{\text{MOR}}(x_{T+k}^{(\text{real})} \mid \hat{\mathbf{x}}_{T+k})$ is given by:

$$\begin{aligned} p_{\text{MOR}}(x_{T+k}^{(\text{real})} \mid \hat{\mathbf{x}}_{T+k}) &= p(x_{T+k}^{(\text{real})} \mid \mathbf{x}_{T+k-1}, \ldots, \mathbf{x}_{T+k-P}) \\ &= \pi_{T+k}^{\text{out}} \prod_{i=1}^{M} \frac{\hat{\mathbf{x}}_{T+k,i}}{|C_i|}^{\pi_{T+k}^{i}}, \\ \log p_{\text{MOR}}(x_{T+k}^{(\text{real})} \mid \hat{\mathbf{x}}_{T+k}) &= \log \pi_{T+k}^{\text{out}} + \sum_{i=1}^{M} \pi_{T+k}^{i} \log \frac{\hat{\mathbf{x}}_{T+k,i}}{|C_i|}, \end{aligned}$$

Quantities such as the negative sequence log-likelihood (NLL) for a sequence of P_h samples drawn from the pre-quantised time series can now be calculated as:

$$\begin{aligned} \text{NLL}_{\text{MOR}} &= -\log \prod_{k=1}^{P_h} p_{\text{MOR}}(x_{T+k}^{(\text{real})} \mid \hat{\mathbf{x}}_{T+k}) \\ &= -\sum_{k=1}^{P_h} \left(\log \pi_{T+k}^{\text{out}} + \sum_{i=1}^{M} \pi_{T+k}^{i} \log \frac{\hat{\mathbf{x}}_{T+k,i}}{|C_i|} \right). \end{aligned}$$

We now conclude this chapter by providing an exhaustive evaluation of the forecasting capabilities of our framework with respect to other state-of-the-art models.

[3]Note that the factor $\pi_{T+k}^{\text{out}} = \mathbb{I}(x_{T+k}^{(\text{real})} \in I)$ in the definition guarantees that the probability density function is zero for measurements outside the observed range.

3.5 Long-term forecasting

We now evaluate our fully end-to-end ordinal time series forecasting method based on Recurrent Neural Networks. In order to assess the performance of our framework, we provide a large-scale benchmark test over 45 different datasets drawn from an ample range of application domains. The datasets, which we describe in Section 3.5.1, were drawn from a variety of sources and include time series with quasi-periodic behaviour and complex shapes that in some cases are only modelled accurately by MOrdReD, as we will illustrate in Section 3.5.5. We compare our method with state-of-the-art baselines (described in Section 3.5.2) in the time series literature from both the Statistics and the Machine Learning perspectives, and evaluate exhaustively over a number of metrics to measure predictive accuracy, uncertainty quantification and forecast reliability.

In this experiment, we perform a large-scale comparison of our MOrdReD framework against the 4 baselines described in Section 3.5.2. Our long-term forecasting task consists in extrapolating P_h observations $(\hat{\mathbf{x}}_{T+1}, \ldots, \hat{\mathbf{x}}_{T+P_h})$ forward from the last P observations $(\mathbf{x}_{T-P}, ..., \mathbf{x}_T)$ in the time series of length T, with $P << P_h$. In order to assess our models' long-term predictive capability, in this task we chose a lookback $P = 100$ and a predictive horizon $P_h = 1000$. We learn a MOrdReD model for each of the 45 datasets described in Section 3.5.1 and infer predictive distributions without looking at real testing data or re-estimating model parameters during the process.

We note that the aim of this forecasting task is to showcase the performance of our method as an "out-of-the-box" methodology, and thus we perform our comparison prior to any application domain crafting takes place (such as tailoring a task-optimal kernel, or appending handcrafted features and exogenous or correlated time series). We now give specific details about data preparation, model selection, model implementation and evaluation metrics.

3.5.1 Data

Our large-scale experiment was performed on 45 datasets compiled from different sources (Fulcher, Little, and Jones, 2013; Rezek and Roberts, 1998; Bramble Bank, 2018). We used the following criteria to obtain this compilation:

- **Length**. All time series must contain at least 10,000 observations. If they contain over 30,000, then they are truncated to this length.
- **Structure**. All time series were drawn from either the provided real-world references, or are a synthetic dynamical system.
- **Domain variety**. Time series were queried and grouped by application domain, and at most two from each resulting group were subsequently drawn at random.

The selected datasets come from a wide range of synthetic datasets, including chaotic maps, dynamical systems, in addition to data obtained from varied application domains that encompass physiology, meteorology, astronomy, amongst others. A list and short descriptions of the selected datasets is given in Appendix B, and full descriptions can be found in the supplemental materials of (Fulcher, Little, and Jones, 2013; Rezek and Roberts, 1998; Bramble Bank, 2018). All the time series in the library were linearly and seasonally detrended as detailed in the corresponding sources. Furthermore, all time series were standardised to zero mean and unit variance.

In the case of MOrdReD, all time series were quantised into a maximum of $M = 300$ bins, largely based on our access to computational resources. For the rest of the models, which perform direct regression, regularising white noise with $\sigma = 1e-3$ was added to the standardised time series. Datasets were further split into training, validation and testing time series. The training sets correspond to

the first 70% of the observations in each time series, validation sets correspond to the subsequent 15%, and test sets to the last 15%.

3.5.2 Baseline models

In this section, we review the baseline models we use to compare our framework with. These largely belong to three different families:

- **RNN regression.** This baseline is a simple counterpart of our model that motivates the transition from direct to ordinal regression with RNNs. We use the same sequence-to-sequence architecture described in Section 3.2.3, but learnt by minimising the mean squared error of the model output with respect to the original time series $\mathbf{x}^{\text{real}}$.

- **State space AR(p) model.** Autoregressive modelling of order p is a well-established statistical method, and widely used as a benchmark for time series forecasting tasks. We simply recapitulate that AR(p) models with time-varying coefficients can be recast as linear Gaussian state-space models. Such systems admit exact parameter inference and prediction in an efficient and principled manner by means of the Kalman filter. We refer the reader to (Durbin and Koopman, 2012) for a more detailed presentation of this model.

- **Gaussian Process autoregression.** GPs, and in particular autoregressive GPs, have dominated the time series literature over the last decade. One potential limitation of this approach is that propagating forecast uncertainty into the future requires handling a *noisy-input* GP, whose exact inference is intractable. Analytical approximations exist in limited cases, such as models with a Squared Exponential kernel and Gaussian input noise (Girard and Murray-Smith, 2005). An alternative approach is to perform approximate inference through simulation methods, which are more extensive but

work for a wider choice of GP kernels. We follow this approach and propose two autoregressive GP baselines: one in which the one-step-ahead predictive posterior distributions are approximated via Monte-Carlo integration, and an alternative approach in which each predictive posterior is approximated with a Variational Bayesian Gaussian Mixture Model (VBGMM). We describe both baselines in Section 3.5.2. The latter approach requires more computational resources, but also enables the model to represent rich, multi-modal behaviour and adapt it over time.

Autoregressive Gaussian Processes

The Gaussian Process (GP) describes distributions over functions (Rasmussen and Williams, 2006). They are defined as a collection of random variables, any finite number of which have a joint Gaussian distribution. A GP is fully specified by a mean function $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$ and a covariance function $k(\mathbf{x}, \mathbf{x}) = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]$. The covariance function (or kernel) describes correlations between data points and can be used to encapsulate prior belief about the generating process of our data, such as its roughness, number of times it can be differentiated, and process length-scale.

For instance, consider the Squared Exponential (SE), Matérn 3/2 and 5/2 and Rational Quadratic (RQ) kernels:

$$
\begin{aligned}
k_{\mathrm{SE}}(r) &= \exp\left(-\frac{r^2}{2\ell^2}\right), \\
k_{\nu=3/2}(r) &= \left(1 + \frac{\sqrt{3}r}{\ell}\right)\exp\left(-\frac{\sqrt{3}r}{\ell}\right), \\
k_{\nu=5/2}(r) &= \left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right)\exp\left(-\frac{\sqrt{5}r}{\ell}\right), \\
k_{\mathrm{RQ}}(r) &= \left(1 + \frac{r^2}{2\alpha\ell^2}\right)^{-\alpha},
\end{aligned}
$$

where ℓ is the *length-scale*, and $\ell, \alpha > 0$. The Matérn is parameterised by $\nu =$

$k + 1/2$, and models functions that are differentiable k times. Two common choices are $k = 1, 2$, giving the kernels above. For $k = 0$, the process becomes very rough, and values $k > 2$ are only suggested in the presence of prior knowledge about higher order derivatives (Rasmussen and Williams, 2006). Furthermore, as $\nu \to \infty$, we also recover the SE covariance.

On the contrary, the SE and RQ model *infinitely differentiable* functions, thereby being suitable choices for smooth systems. The RQ, however, also allows us to model data that varies at multiple scales, as opposed to a characteristic length-scale, like the SE. The relative weighting of large scale and small scale variations is captured by the hyperparameter $\alpha > 0$. Indeed, the RQ can be regarded as an infinite sum of SE covariances with different length-scales. We now refer the reader to Rasmussen and Williams (2006) for a more detailed discussion about the Gaussian Process.

Consider a time series of length T. Long-term forecasting can be done in an autoregressive manner by learning a Gaussian Process f that maps a sequence of length P, $\mathbf{X}^{(t-1)} = (\mathbf{x}_{t-P}, \ldots, \mathbf{x}_{t-1})$, to a normal distribution over its next sample $\mathcal{N}(\mathbf{x}_t \mid \boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2)$. However, sequentially inferring $\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2$ for longer predictive horizons $P_h > 2$ requires adequate forward-propagation of the uncertainty computed in all previous forecasts. This effectively turns the task into one of GP regression with noisy inputs. Exact inference of uncertain, or noisy input, GPs is intractable (Rasmussen and Williams, 2006), and therefore we must recur to approximations, such as those based on Monte Carlo method (Dellaportas and Stephens, 1995).

After fitting the GP to the observed time series, we use it to construct N_{GP} sample trajectories up to time $T + P_h$ for a predictive horizon P_h, i.e. $\hat{\mathbf{X}}^{[s]} = (\hat{\mathbf{x}}_{T+1}^{[s]}, \ldots, \hat{\mathbf{x}}_{T+P_h}^{[s]})$, and used to estimate the moments $\tilde{\boldsymbol{\mu}}_{T+k}$, $\tilde{\boldsymbol{\sigma}}_{T+k}^2$ of the corrected normal distribution $p(\mathbf{x}_{T+k} \mid \tilde{\boldsymbol{\mu}}_{T+k}, \tilde{\boldsymbol{\sigma}}_{T+k}^2)$. Each sample trajectory is built iteratively $\forall k, 2 \leq k \leq P_h$ in the following fashion:

1. at time $T+k-1$, compute $\mathcal{N}(\mathbf{x}_{T+k-1} \mid \boldsymbol{\mu}_{T+k-1}, \boldsymbol{\sigma}_{T+k-1}^2)$ from the sequence

$\hat{\mathbf{X}}^{(T+k-2)}$,

2. draw a sample $\hat{\mathbf{x}}_{T+k-1}^{[s]}$ from $\mathcal{N}(\mathbf{x}_{T+k-1} \mid \boldsymbol{\mu}_{T+k-1}, \boldsymbol{\sigma}_{T+k-1}^2)$, and build the next input seed sequence

$$\hat{\mathbf{X}}^{(T+k-1)} = (\mathbf{x}_{T+k-P-1}, \ldots, \mathbf{x}_{T+k-1}),$$

3. repeat from step 1 and stop when $k > P_h$.

After repeating the procedure above N_{GP} times, the corrected mean and variance $\forall k, 2 \leq k \leq P_h$ are hence given by:

$$\begin{aligned}
\tilde{\boldsymbol{\mu}}_{T+k} &= \mathbb{E}[\mathbf{x}_{T+k}] \approx \frac{1}{N_{\text{GP}}} \sum_{s=1}^{N_{\text{GP}}} \hat{\mathbf{x}}_{T+k}^{[s]}, \\
\tilde{\boldsymbol{\sigma}}_{N+k}^2 &= \mathbb{E}\left[(\mathbf{x}_{T+k})^2\right] - \mathbb{E}\left[\mathbf{x}_{T+k}\right]^2 \\
&\approx \frac{1}{N_{\text{GP}}} \sum_{s=1}^{N_{\text{GP}}} \left(\hat{\mathbf{x}}_{T+k}^{[s]} - \tilde{\boldsymbol{\mu}}_{T+k}\right)^2.
\end{aligned}$$

Bayesian Gaussian Mixture Models

The baseline described above assumes each one-step-ahead predictive distribution is unimodal Gaussian. We propose an alternative baseline in which we characterise the sample trajectories just described as a collection of Variational Bayesian Gaussian Mixture Model (VBGMM), which allows for one-step-ahead predictive distributions with a variable number of modes at each time step. In other words, the prediction $\mathbf{x}_{t+1} \sim \text{GMM}(\mathbf{x}_{t+1} \mid \boldsymbol{\pi}_{t+1}, \boldsymbol{\mu}_{t+1}, \boldsymbol{\sigma}_{t+1}^2)$ with

$$\text{GMM}(\mathbf{x}_{t+1} \mid \boldsymbol{\pi}_{t+1}, \boldsymbol{\mu}_{t+1}, \boldsymbol{\sigma}_{t+1}^2) = \sum_{k=1}^{K} \boldsymbol{\pi}_{k,t+1} \mathcal{N}(\mathbf{x}_{t+1} \mid \boldsymbol{\mu}_{k,t+1}, \boldsymbol{\sigma}_{k,t+1}^2),$$

where each $\boldsymbol{\pi}_{k,t+1}$ is the mixture weight of the individual Gaussians and K is a hyperparameter for the number of mixtures in the model. Indeed this enables the description of richer behaviour than the one given by uni-modal Gaussians,

but comes at the expense of the potentially computationally expensive process of learning up to P_h distinct GMMs.

Although exact Bayesian inference of the parameters $(\boldsymbol{\pi}_{t+1}, \boldsymbol{\mu}_{t+1}, \boldsymbol{\sigma}^2_{t+1})$ is intractable, approximate Bayesian inference can be performed efficiently via Variational Bayes techniques, which is the approach we take in this thesis, as shown by Penny and Roberts (2000), Bernardo, Bayarri, et al. (2003) and Ghahramani (2001). Indeed, Bayesian inference enables a principled prediction framework for other quantities of interest and further guards against overfitting and other well-documented pathologies that occur in point-estimate inference mechanisms, such as Maximum a Posteriori estimation (Bishop, 2006). Furthermore, such Bayesian methods also allow for natural shrinkage of the number of mixtures K, i.e. the weights of less informative mixtures naturally shrink to 0 and thus one need only provide an upper bound on K.

3.5.3 Model learning, selection and implementation

We now provide details for the optimisation of each model's parameters and hyperparameters. The optimal hyperparameter configurations found for our neural network and AR(p) models are given in Appendix D.

- **MOrdReD.** All our MOrdReD models were built in Python 2.7 and using Keras 2.1.2 with the Tensorflow backend (Chollet, 2015). An open source implementation of our models is available at `https://github.com/bperezorozco/ordinal_tsf` and documented in Appendix C. Parameter optimisation was performed by minimising the categorical cross-entropy with Nesterov momentum Adam using $\alpha = 0.002, \beta_1 = 0.9, \beta_2 = 0.999$ and a schedule decay of 0.004, and parameters were initialised using the Glorot Uniform method. At training time, all models incorporated Early Stopping to guard against overfitting, while allowing for a maximum of 50 epochs using mini-batches of 256 training examples. Both the bi-directional

encoder and the decoder share the same number of hidden units $n_u \in \{64, 128, 256, 320\}$, and the output fully-connected layers have M softmax output units corresponding respectively to each time series' ordinal class. All models further incorporate dropout and L_2 regularisation with hyper-parameters $\theta_{\text{dropout}} \in \{0.25, 0.35, 0.5\}$ and $\lambda \in \{10e-6, 10e-7, 10e-8\}$, which were jointly optimised alongside the number of hidden units n_u by grid search on the validation set. MC dropout predictive posteriors were estimated using $N_s = 100$ samples.

- **Direct regression sequence-to-sequence neural network.** These were learnt in a similar fashion as the above, but by minimising the mean squared error loss function of the single output unit (with no activation function) with respect to the ground truth. This baseline serves to motivate the transition into ordinal regression.

- **Autoregressive GP.** All our GP models were implemented using GPy 1.9.2 (GPy, 2012) and using a Matérn 5/2 kernel with additive white noise and Automatic Relevance Determination enabled. Samples drawn from the Matérn 5/2 covariance function are twice-differentiable, which enables the modelling of a wide range of physical systems such as those in our datasets. Time-indexed Gaussian predictive distributions were then estimated following the Monte Carlo-based procedure described in Section 3.5.2, with $N_s = 100$ samples trajectories.

- **ARGP with predictive Gaussian Mixture Model.** This baseline follows the implementation described above, but the time-indexed predictive distributions are estimated via a Gaussian Mixture Model as described in Section 3.5.2. This allows for rich, multi-modal behaviour to be described succinctly at each time step. We performed approximate Bayesian inference to learn these models, with up to $K = 5$ mixtures at each timestep, using

the open source Variational Bayes GMM implementation available in the Scikit-learn toolbox (Pedregosa et al., 2011), v. 0.19.1.

- **State-space AR(p).** The Python library StatsModels 0.9 (Seabold and Perktold, 2010) offers an open source implementation of the state-space formulation of AR(p) modelling. We used this and optimised the lookback hyperparameter $p \in \{16, 32, 64\}$.

Our experiments are further accompanied by a Python library that provides an implementation of our methodology, as well as an interface for the ample number of libraries that implement our baselines. This code and the scripts used to perform these experiments can be found at `https://github.com/bperezorozco/ordinal_tsf`.

We showcase our model's performance by means of an exhaustive comparison over 45 datasets from a wide range of application domains, encompassing a variety of state-of-the-art methods from both the Machine Learning and Statistical modelling communities. Crucially, we evaluate both the accuracy and calibration of our probabilistic forecasts, in addition to the deviation of point forecasts, such as the mean and the median of the predictive distributions. We provide evidence that our method achieves comparative or superior performance in the majority of datasets, and furthermore, we show that even when it does not, on average we still expect it to surpass most of its competitors.

3.5.4 Evaluation metrics

Our evaluation is performed across three axes: predictive accuracy, uncertainty quantification and forecast reliability. This exhaustive comparison enables us to speak not only about the performance of the forecasts predicted by our model, but also about the credibility and reliability of the uncertainty bounds they infer. In terms of forecast accuracy, we measure the Symmetric Mean Absolute Percentage

Error (SMAPE) and the Root Mean Squared Error (RMSE) of the median of the predictive distributions of each method:

$$\text{SMAPE} = \frac{2}{P_h} \sum_{t'=T+1}^{T+P_h} \frac{|\mathbf{x}_{t'} - \hat{\mathbf{x}}_{t'}|}{|\mathbf{x}_{t'}| + |\hat{\mathbf{x}}_{t'}|},$$

$$\text{RMSE} = \sqrt{(\frac{1}{P_h}) \sum_{t'=T+1}^{T+P_h} ||\mathbf{x}_{t'} - \hat{\mathbf{x}}_{t'}||^2}.$$

Crucially, one goal in this experiment is not just to evaluate a model's predictive accuracy, but also to measure its honesty expressed through uncertainty estimations. We prefer models that are able to either produce an accurate confident forecast, or otherwise explicitly confess their ignorance - for instance, through a mean-reversal process. We quantify this via the sequence negative log-likelihood (NLL), which we rewrite using the chain rule for joint distributions:

$$-\log p\left(\mathbf{X}^{(T+P_h)}\right) = -\log p(\mathbf{x}_{T+1}) - \sum_{k=2}^{P_h} \log p\left(\mathbf{x}_{T+k} \middle| \mathbf{x}_{T+1}, \ldots, \mathbf{x}_{T+k-1}\right).$$

For MOrdReD, each term in the summation acquires the piece-wise uniform shape described in Section 3.3; for ARGPs with a GMM predictive distribution, it acquires the shape described in Section 3.5.2; for all the other baselines, it is given by a standard Gaussian density.

In addition to the NLL, we also provide the integrated or cumulative NLL:

$$\text{CNLL} = \sum_{k=1}^{P_h} -\log p\left(\mathbf{X}^{(N+k)}\right).$$

This metric summarises information about how the NLL changes with time, e.g. greater penalties are incurred by models that make inaccurate short-term predictions, whereas erroneous long-term forecasts have a lesser contribution.

Lastly, we benchmark our methodology in terms of the calibration of its uncertainty bound predictions. That is, we measure its ability to produce output

densities that can be interpreted as real-world probabilities. Graphics tools such as quantile-quantile (QQ) plots and reliability diagrams are widespread in the time series literature to measure uncertainty calibration (Oldford, 2016). However, due to the large-scale nature of this task, we instead propose a related summary metric of these which we now introduce.

Consider the time series $\hat{\mathbf{x}}_\alpha$ whose t-th entry is given by the α-quantile of the predictive distribution at time t, i.e.:

$$\hat{\mathbf{x}}_{\alpha,t} = q_t \qquad \text{s.t.} \int_{-\infty}^{q_t} p\left(\mathbf{x}_t \middle| \bigcap_{t'=1}^{t-1} \mathbf{x}_{t'}\right) d\mathbf{x}_t = \alpha.$$

For a given calibrated model and $\alpha, 0 < \alpha < 1$, we expect that exactly $\alpha\%$ of the observations in the ground truth $\mathbf{x}$ will lie below $\hat{\mathbf{x}}_\alpha$. Let r_α be this proportion. Then for better calibrated models, $r_\alpha \to \alpha$ with:

$$r_\alpha = \frac{1}{P_h} \sum_{t=T+1}^{T+P_h} \mathbb{I}(\mathbf{x}_t < \hat{\mathbf{x}}_{\alpha,t}).$$

We thus define the calibration metric QQDist as the (Euclidean) distance between r_α and α:

$$\text{QQDist} = \int_0^1 (r_\alpha - \alpha)^2 d\alpha.$$

We provide two instances of this metric. One in which the r_α for each method are computed for the full predictive horizon $P_h = 1000$, and one up to a truncated horizon $P_h' = 250$. We provide both instances so as to distinguish calibration performance in both the medium and long term.

	AR(p)	Best GP	MOrdReD	Seq2Seq regression
NLL	9	12	**22**	2
Cumulative NLL	6	14	**22**	3
QQ Dist	**18**	6	**18**	3
QQ Dist 250	15	7	**18**	5
Mean RMSE	10	10	**17**	8
Median RMSE	8	**17**	11	9
PredMean SMAPE	7	**15**	**15**	8
PredMed SMAPE	3	14	**22**	6

Table 3.1: Number of datasets in which the model achieves the best performance amongst all competitors. **Bold** indicates best performance. We provide full details of these results in Appendix D.

3.5.5 Results

Performance over this long-term forecasting task is measured over the 8 statistics we presented in Section 3.5.4: SMAPE and RMSE for both the predictive mean and median's accuracy; NLL and Cumulative NLL to evaluate the predictive distributions' accuracy; and QQDist and QQDist-250 for uncertainty calibration.

We summarise our results in Tables 3.1, 3.2, 3.3 and 3.4. We note that our model achieves state-of-the-art performance in an end-to-end fashion. Unlike GP Regression, where deep knowledge of kernels is required and often inaccessible to unfamiliar users, we argue that our framework requires comparatively less user intervention. Additionally, other advantages of neural networks such as scalability are now readily accessible. Our framework can make the most out of big datasets and produce reliable forecasts in the long term that outperform other state-of-the-art techniques.

From Table 3.1 we note that our model is the one that performs best in the largest number of datasets across most of the proposed metrics. Interestingly, we see that the worst performing baseline is the one given by direct regression neural

	AR(p)	Best GP	MOrdReD	Seq2Seq regression
NLL	5	7	**4**	29
Cumulative NLL	5	8	**4**	28
QQ Dist	**4**	10	8	23
QQ Dist 250	**3**	9	11	22
Mean RMSE	10	10	**7**	18
Median RMSE	**9**	**9**	13	14
PredMean SMAPE	18	**8**	**8**	11
PredMed SMAPE	20	**6**	**6**	13

Table 3.2: Number of datasets in which the model achieves the worst performance amongst all competitors. **Bold** indicates best performance. We provide full details of these results in Appendix D.

	GP+GMM	GP	AR(p)	Seq2Seq
NLL	66.67%	66.67%	64.44%	84.44%
CumulNLL	64.44%	64.44%	68.89%	86.67%
QQDist	64.44%	66.67%	42.22%	68.89%
QQDist250	66.67%	62.22%	40.00%	64.44%
Mean RMSE	55.56%	55.56%	62.22%	68.89%
Median RMSE	46.67%	44.44%	53.33%	55.56%
Mean SMAPE	51.11%	51.11%	64.44%	66.67%
Median SMAPE	62.22%	64.44%	82.22%	73.33%

Table 3.3: Percentage of datasets in which MOrdReD outperforms each baseline for each metric.

	MOrdReD	Best GP	AR(p)	Seq2Seq
NLL	**0.84**	1.42	1.36	2.38
CumulNLL	**0.80**	1.29	1.51	2.33
QQ Dist	1.23	1.63	**0.93**	2.2
QQ Dist 250	1.32	1.72	**0.91**	2.04
Mean RMSE	**1.13**	1.49	1.51	1.87
Median RMSE	1.49	**1.27**	1.49	1.76
Mean SMAPE	**1.18**	1.31	1.87	1.64
Median SMAPE	**0.82**	1.29	2.16	1.73

Table 3.4: Average performance rank achieved by each model for each metric across all datasets, where rank 0 is given to the model with *best* performance and 3 to the one with the *worst* performance. Best performance is given in **bold**. We note that MOrdReD consistently performs best in this metric, and in particular that the expected scenario is that it will be either the best or second-best performing model for a given task.

networks. This gives further empirical motivation to transition from direct into ordinal regression for the family of neural network models. We also note that classical statistical models perform very well in the calibration metrics, whereas GPs comparatively underperform. We argue that this could be the case due to the exact sequential inference method of the predictive variance proposed in the state-space formulation of AR(p) models. Finally, GPs are the closest competitors in terms of forecasting accuracy. In Table 3.2 we also enumerate the number of times each model achieves the *worst* performance across all datasets. This makes an even stronger case to use ordinal regression neural networks as an alternative to their least-squares counterparts.

In Table 3.3 we provide the number of datasets in which MOrdReD performs better than each baseline. We observe that our framework achieves better performance across most metrics and baselines in the majority of the datasets, with the exception of the AR(p) baseline in the context of uncertainty calibration.

We now focus on Table 3.4, which provides the mean performance rank for

each baseline and metric. This provides further information about how our method performs when it is not the first in the ranking, and we observe that MOrdReD still achieves the best mean rank for most metrics. In order to produce this table, two considerations were taken into account: on one hand, both GP baselines could propose very similar predictive distributions for some datasets (e.g. those in which multi-modality is not required and therefore both propose a uni-modal Gaussian), and this induced twice the penalty for those methods that under-performed against GPs. We solve this by merging the results of both GPs and retaining the one with better performance. This avoids allocating a double penalty on models that under-perform with respect to GPs if both GPs propose equivalent predictive distributions (e.g. if the optimal number of Gaussian mixtures is 1). This table summarises that whenever our model does not have the best performance, it is still likely to have the second best, etc.

We further observed that in some cases our model is the only one able to learn some complex-shapes, as is the case of electrocardiograms. This complex-shaped signal is consistently predicted in a timely and accurate fashion by MOrdReD in the long-term, whereas no baseline is capable of reproducing the characteristic shape of the QRS complex beyond its first occurrence.

However, we also note that MOrdReD is not always so close to the best-performing model. For instance, we noted that GPs are often the best-performing in chaotic maps and dynamical systems, such as the Lorenz map. Whereas both MOrdReD and GPs are able to model multi-modal behaviour in the long-term, MOrdReD syncs out with respect to the ground truth considerably faster than GPs. Indeed, GPs only make use of their multi-modal predictive capability relatively late in comparison with MOrdReD.

We further provide a visual example of these densities in Figures 3.3 and 3.4. In the former, we observe that MOrdReD accurately predicts the timing of all maxima in the long-term, clearly outperforming all the other baselines in this

task. On the contrary, in the case of the tide height dataset, we observe that the GP baseline is the one that accurately forecasts the timing of all maxima, since MOrdReD syncs out with respect to the ground truth towards the end of the forecast.

We conclude this section by highlighting some key features of our framework. Firstly, by proposing a piece-wise uniform predictive density, our model is able to describe rich, multimodal behaviour in the long-term. Secondly, we argue that, within reason, our proposal requires relatively less expert knowledge to specify the core parts of the model, such as GP kernels.

3.6 Final remarks

In this section, we have introduced the core component of this thesis: the Memory-endowed Ordinal Regression Deep neural network (MOrdReD). Our discussion focused on the univariate time series forecasting setting, and we showed how this problem can be recast as an ordinal regression one. In turn, this enables us to incorporate state-of-the-art models in symbolic sequence learning, such as the LSTM recurrent neural network. Furthermore, we are also able to quantify uncertainty for long-term forecasting thanks to recent developments linking the dropout regularisation technique with approximate Bayesian inference (Gal and Ghahramani, 2016b). Crucially, our framework is accompanied by an open source implementation, available at `https://github.com/bperezorozco/ordinal_tsf` and fully documented in Appendix C.

This setting, however, also opens up new questions, which we tackle in Chapters 4 and 5. For instance, we may wonder how to extend MOrdReD for a multivariate setting, where correlations between channels exist. We dedicate Section 4.3 to tackle this question.

In Section 2.2.3, we also reviewed how to make optimal decisions from in-

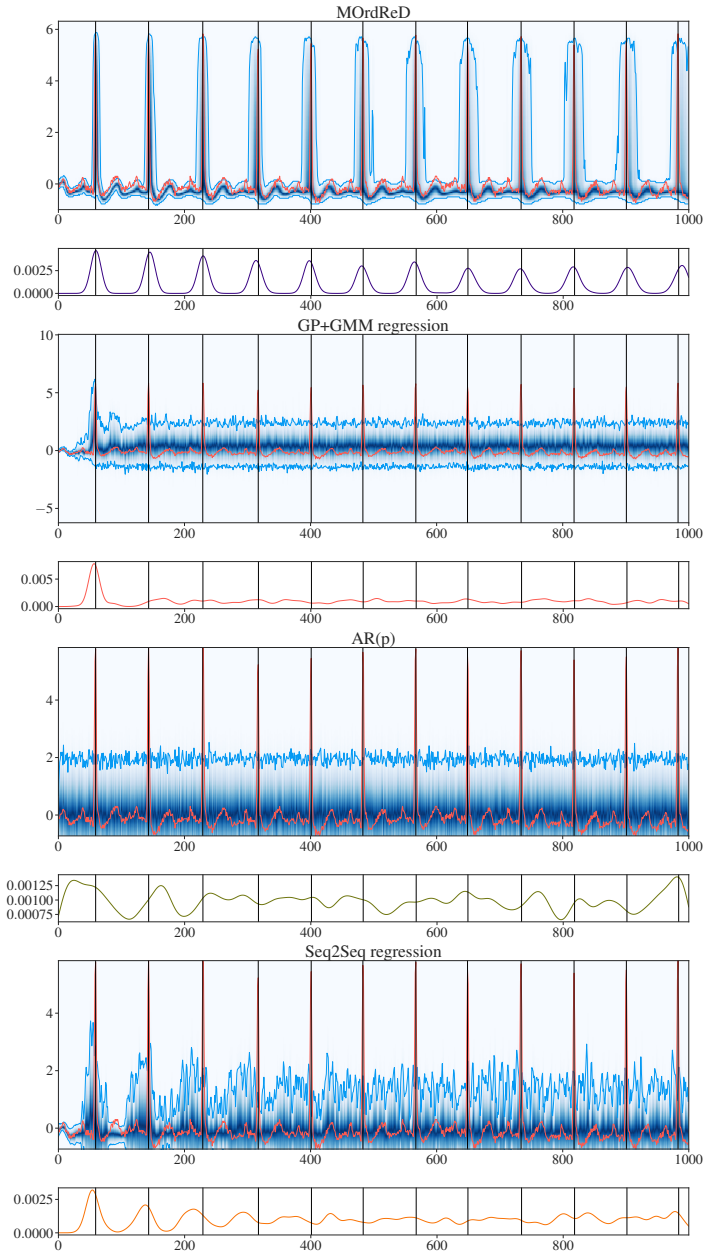


Figure 3.3: Example of out-of-sample forecast with predictive horizon $P_h = 1000$ for an electrocardiogram time series dataset. This is an example of a complex-shaped signal that is consistently predicted in an accurate in timely fashion by our framework only. Ground truth (orange) overlayed on top of the predictive distributions given by each model. 95% confidence bounds are highlighted in light blue. Maxima predictive densities are provided under each predictive distribution. Vertical lines were drawn at the optima of interest.

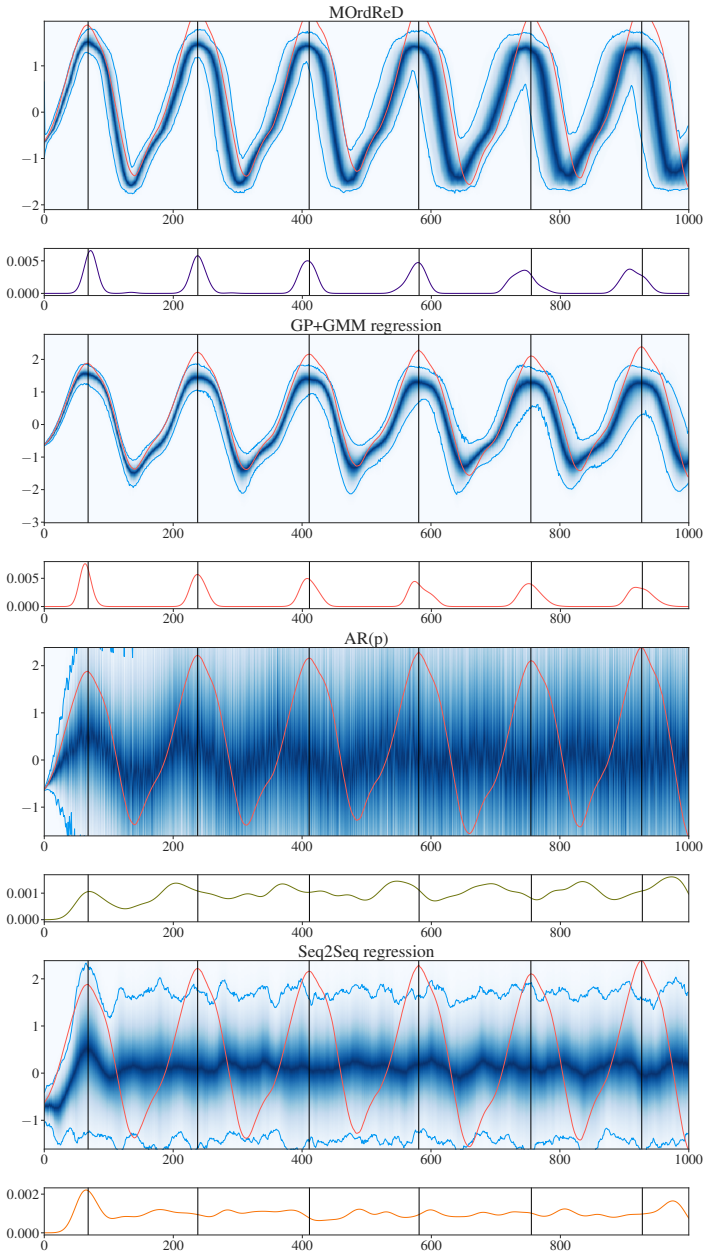


Figure 3.4: Example of out-of-sample forecast with predictive horizon $P_h = 1000$ for the tide height time series dataset. This is an example of a dataset in which our model achieves performance comparable to the best-performing one, which is the GP baseline in this case. Ground truth (orange) overlayed on top of the predictive distributions given by each model. 95% confidence bounds are highlighted in light blue. Maxima predictive densities are provided under each predictive distribution. Vertical lines were drawn at the optima of interest.

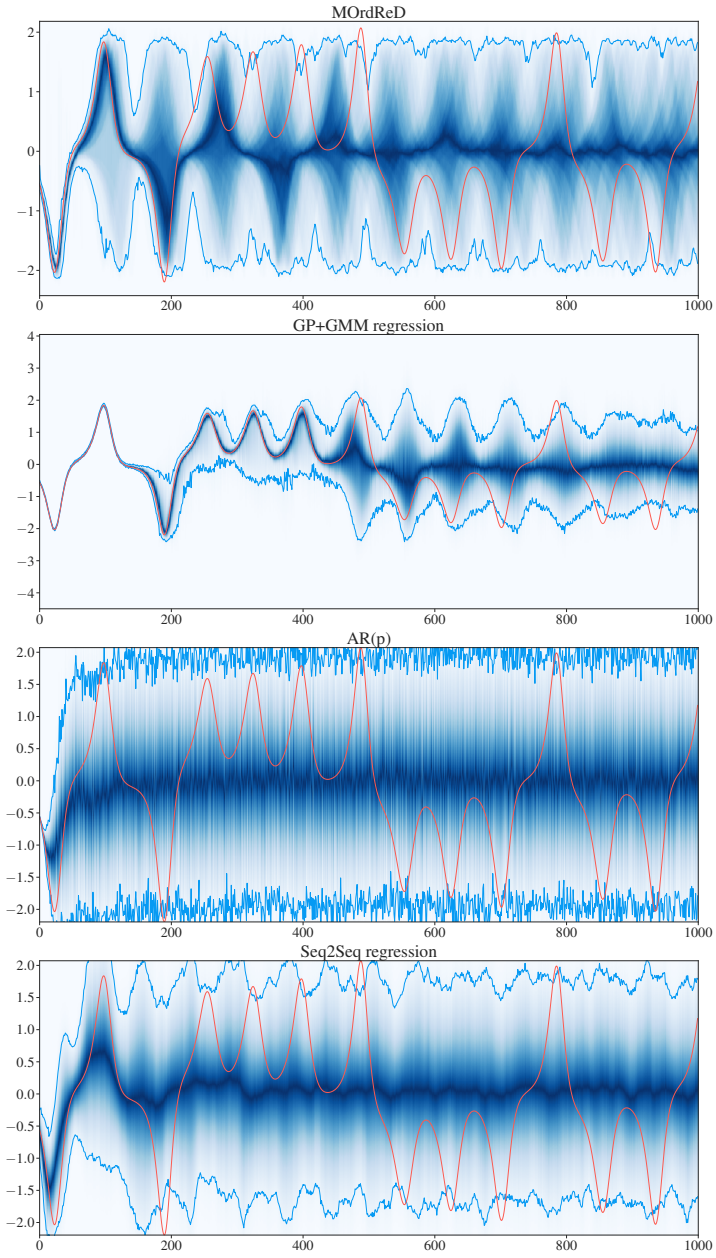


Figure 3.5: Example of out-of-sample forecast with predictive horizon $P_h = 1000$ for the Lorenz chaotic map time series dataset. Both MOrdReD and GP+GMM regression are capable of modelling adaptable multi-modal behaviour through time, but our model phases out with respect to the ground truth before the GP does. This enables the baseline to achieve a clear performance advantage with respect to our model in this case. Ground truth (orange) overlayed on top of the predictive distributions given by each model. 95% confidence bounds are highlighted in light blue.

ferred predictive distributions. Such distributions, however, also hold valuable information that may be of use to query for specific pieces of information that may be of use for practitioners. For example, they may be interested in knowing *when* a certain time series event will happen within a predictive horizon, or the duration or number of occurrences of such an event. This motivates our discussion in Section 4.1, where we show how to build predictive densities for critical events from our MOrdReD forecasts. For some cases, such critical events may be ill-defined, or difficult to label given a time series forecast. Characterising, detecting and segmenting these events is therefore of great relevance, and this motivates Section 4.2, where we propose how to tackle this task by extracting information from MOrdReD's sequence-to-sequence latent feature space.

Finally, one characteristic problem of models such as deep and recurrent neural networks is that, given the large number of parameters they tend to have, inferring them often requires large datasets to avoid overfitting. This is often prohibitive for several tasks, and time series forecasting is an example where, for some datasets, only a handful of observations may be available. This motivates Chapter 5, where we propose a General Unified Model (GUM) recurrent neural network that learns a *joint* latent space over a large number of auxiliary time series where data is available. Enabled by our quantisation step, we show how this knowledge can be transferred to predict time series when only a few measurements are available.

4 Beyond univariate forecasting

In Chapter 3, we showed how long-term predictive probability distributions can be inferred through an ordinal regression approach based on recurrent neural networks. In this chapter, we extend our framework to two different practical settings of interest that the reader can read independently. In the first part, we focus on how to extract information of interest from MOrdReD's forecasts and its learnt temporal features. This "analysis toolkit" covers a broad range of use cases, such as: deriving predictive densities over the occurrence of critical events; analysing and interpreting aspects of the feature space learnt by our recurrent neural networks; and discovering time series motifs and describing their occurrence over time.

Next, in the second part of this chapter, we show how the architecture of our model can be extended to allow for *multivariate* forecasting, which arises naturally, for instance, in systems where different quantities of interest interact and drive each other through time, in addition to others such as multi-sensor networks. We achieve this by inferring joint characteristics of the system dynamics over the different channels of the time series. We introduce this multivariate extension in Section 4.3.

4.1 Constructing densities for critical event occurrence

A crucial component of long-term prediction is quantifying the uncertainty about events of interest. For example, decision-makers may prioritise knowing the time of occurrence of an event over the magnitude of the event itself. To illustrate, consider the tide height dataset of Section 3.5. Experts are often interested in knowing *when* the tide levels will reach their maximum, in addition to the actual height they will reach[1]. For such quasi-periodic time series, one may also wonder about the duration of the next cycle, or the number of times a certain event may occur within a given predictive horizon.

In a similar vein, cardiologists are interested, for instance, in assessing the duration of events such as the QRS complex in electrocardiograms[2]. This further enables them to compute other metrics of interest, such as the RR interval (the time lapse between two R waves) – which has been linked, for example, to Parkinson's disease (Gurevich et al., 2004). Accurately quantifying these metrics is therefore of utmost importance, and fully relies on models that can speak honestly about their forecasts and their uncertainties.

In this section, we develop a methodology to derive predictive densities over such types of event as a byproduct of MOrdReD's forecasts. To illustrate, we shall focus on the case of forecasting the occurrence of *local maxima* in time series. In order to quantify event timings, we construct a non-parametric probability distribution $p(t)$, which describes the probability of an event happening at time t, through a Kernel Density Estimation (KDE) approach. The constructed probability density function $p(t)$ describes the probability that a certain event (in

[1]Despite this being a periodic phenomenon, sensor quantisation leads to representation errors that make the time series quasi-periodic in practice.

[2]Electrocardiograms have three main components that occur sequentially: the P wave, the QRS complex and the T wave, and their alphabetically-ordered naming corresponds to the sequential nature of their occurrence.

this example, reaching a local maximum) will happen at time t.

KDE is a smoothing technique that builds a non-parametric distribution from a labelled sample set $\mathcal{X}$. Inferring a KDE estimator thus assumes access to a dataset with n samples $\mathcal{X} = \{t_1, \ldots, t_n\}$, which are the time indices at which the event of interest occurred. We remark that access to such a dataset may not be readily available for some tasks, which motivates the discussion on pattern discovery in Section 4.2. The KDE estimator is readily given by:

$$p(t) = \frac{1}{n} \sum_{i=1}^{n} K_h(t - t_i),$$

where $t_i \in \mathcal{X}$ is the time index at which an instance of the event of interest occurred, K_h is a *kernel function*, which is a non-negative function that integrates to one, and h is a bandwidth hyperparameter (Silverman, 1986). Smooth densities can be obtained by summing smooth kernels, such as the Gaussian kernel, which gives rise to the estimator:

$$p(t) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{\sqrt{2\pi h^2}} \exp\left\{ -\frac{(t - t_i)^2}{2h^2} \right\}.$$

As a rule of thumb for univariate KDE, Silverman's rule of thumb (Silverman, 1986) can be used to optimise the bandwidth:

$$\hat{h} = \left(\frac{4\hat{\sigma}^5}{3n} \right)^{1/5} \approx 1.06 \hat{\sigma} n^{-1/5},$$

when the true underlying distribution is Gaussian, and where $\hat{\sigma}$ is the sample standard deviation of $\mathcal{X}$.

4.1.1 Implementation and results

Let us now compare MOrdReD with the forecasts obtained from the baselines described in Section 3.5.2. We evaluate our methodology across a representative

subset of the time series from Section 3.5, which correspond to 8 datasets drawn from the literature on meteorology, physiology and dynamical systems. Detail of these datasets is provided in Appendix B.

We briefly describe how to construct the training and evaluation sets $\mathcal{X}, \mathcal{X}^{(\text{true})}$. In the case of $\mathcal{X}$, sample trajectories are drawn from the predictive densities of each framework for each dataset, as described in Sections 2.3.3 and 3.5.2, and the timing of the desired event in each drawn sample is recorded. Notice that this approach propagates our forecast uncertainty onto the constructed event density. In the case of $\mathcal{X}^{(\text{true})}$, the true event timings are recorded directly from the ground truth. Maxima detection was achieved in our experimental setting with the open source PeakUtils Python library.

We subsequently infer $p(t)$ over each baselines's constructed training dataset $\mathcal{X}$. In our work, we used the freely available implementation of KDE included in Scikit-learn (Pedregosa et al., 2011). Now consider the true event timings $\mathcal{X}^{(\text{true})} = (t_1^{(\text{true})}, \ldots, t_n^{(\text{true})})$. The constructed densities are then evaluated in terms of the negative log-likelihood (NLL) for effectively predicting such correct timings:

$$\text{NLL}(\mathcal{X}_{\text{true}}) = -\log \prod_{i=1}^{n} p(t_i^{(\text{true})}) = -\sum_{i=1}^{n} \log p(t_i^{(\text{true})}).$$

In Table 4.1, we provide the NLL results the true timings $\mathcal{X}^{(\text{true})}$ from the constructed densities for each baseline. We observed that our framework does best in more dataset examples than the rest of the baselines, closely followed by the GP baseline, which dominates in the remaining datasets.

In Figures 3.3 and 3.4 from Chapter 3 we showcase how these constructed densities naturally accompany forecast plots. For example, in the case of the electrocardiogram, MOrdReD's predictive density is aligned with all the peaks that occur within the predictive horizon. Furthermore, in both figures we ob-

	MOrdReD	Best GP	AR(p)	Seq2Seq regression
CM_air 1	**19.4725**	21.1712	20.3755	22.1955
CM_air 2	22.7064	**18.3212**	20.2896	21.0814
CM_rhum	**20.5574**	20.6010	20.6663	21.1996
CM_slp 1	14.1069	**13.8725**	14.0731	14.0154
CM_slp 2	**12.1217**	13.2720	13.7170	13.8411
AIRFLOW	**12.1408**	12.5615	14.6095	13.3766
ECG	**68.6775**	81.6095	82.6156	81.2865
TIDE	32.7297	**32.3702**	41.3097	41.0631
# BEST	**5**	3	0	0

Table 4.1: Negative log-likelihood for the event detection task. Figures in **bold** indicate best performance.

serve how the resulting bumps widen as the predictive horizon increases, hence capturing the predictive uncertainty from the original forecasts.

4.2 Inferring time series structure

In this section, we demonstrate how the learnt temporal features of Memory-endowed Ordinal Regression Deep neural network (MOrdReD) models can serve as a basis to derive structural descriptions of time series. Without loss of generality, we argue that providing such descriptions is an open-ended problem, and we therefore develop this section by relating this feature analysis to other tasks of interest in the time series community.

One such task is the extraction of salient time series patterns, also called *motifs*, *primitives*, *frequently-occurring patterns* or *shapelets* in the literature. These serve to many practical applications, such as:

- forming the lexicon in the context of formal grammar induction (Li and Lin, 2010; Senin et al., 2014),
- efficient indexing for searching massive time series databases, as in (Kumar et al., 2005; Lin et al., 2004),

- constructing time series classifiers (Gomes, Jorge, and Azevedo, 2013; Chiu, Keogh, and Lonardi, 2003),
- characterising activity classes in the context of activity recognition (Vahdatpour, Amini, and Sarrafzadeh, 2009),
- detecting anomalous or atypical patterns,

among others. Algorithms for identifying these primitives must incorporate robust similarity metrics that account for variations of different instances of the same motif class. These variations may be explainable by means of morphological filters (Soille, 2013), stretching (variable-length motifs) or the presence of noise. Armed with these motifs, we can readily infer predictive distributions over event timings and their duration, akin to those that we introduced in Section 4.1.

In this section, we propose a novel approach to extract time series motifs based on MOrdReD, which relies on performing a cluster analysis and dimensionality reduction of the temporal characteristics space learnt by our models. In doing so, we also demonstrate how this embedding encapsulates other structural notions in time series, such as chaotic dynamics, quasi-periodicity, and others. We introduce these methods in Sections 4.2.1 and 4.2.2. In Section 4.2.3, we then illustrate this methodology over three datasets: the Mackey-Glass and the Lorenz chaotic attractors, and an electrocardiogram recording, all three of which are detailed in Appendix B.

4.2.1 Clustering temporal features

In Sections 3.2.2 and 3.2.3, we discussed the Long Short-Term Memory (LSTM) and the sequence-to-sequence model, and in Section 3.3, we showed how dropout-enabled forward passes of these models *after* the parameter inference phase can be regarded as draws from an approximate variational predictive posterior distribution, as presented by Gal and Ghahramani (2016a).

In executing these forward passes, we also compute N_s samples of the LSTM recurrent activations, $\mathbf{h}_t^{[1]}, \ldots, \mathbf{h}_t^{[N_s]}$, at each step of the LSTM decoder. We now look at the nature of the groupings that arise from these samples $\mathcal{H} = \{\mathbf{h}_1^{[1]}, \ldots, \mathbf{h}_t^{[n_s]}, \ldots, \mathbf{h}_{P_h}^{[N_s]}\}$ over a predictive horizon P_h.

We construct groups over temporal features through a spectral clustering approach. Although there exist several versions of this algorithm (Shi and Malik, 2000; Ng, Jordan, and Weiss, 2002), without loss of generality, they stem from the core idea of clustering the dataset in a lower-dimensional representation given by a *spectral embedding*. Such an embedding can be derived by considering a similarity graph G over the set $\mathcal{H}$. Luxburg (2007) identifies three approaches to build it:

- The ε-neighbourhood graph: edges are added for every pair of nodes whose pairwise distance is less than $\varepsilon \in \mathbb{R}$. A common choice for this distance metric is the Euclidean distance.

- k-nearest neighbour graphs: vertex i is connected to vertex j only if it is one of its top-k closest neighbours.

- Fully-connected graph: all pairs of nodes with positive similarities are connected with a weighted edge whose value is determined by a similarity function, such as the Gaussian kernel.

Remarks and considerations about each scheme are developed in (Luxburg, 2007).

Given a *similarity graph* G over the set $\mathcal{H}$, the spectral embedding is constructed from the eigenvectors of the Laplacian matrix $L \in \mathbb{R}^{|\mathcal{H}|\times|\mathcal{H}|}$, which is given by:

$$L_{ii} = \text{degree of node } i$$

$$L_{ij} = -1 \text{ if nodes } i, j \text{ are connected, } 0 \text{ otherwise.}$$

We refer the reader to (Luxburg, 2007) for further details of this algorithm and its variations. We also note that, unlike other clustering approaches, such as k-means, spectral clustering has the advantage of not making strong assumptions about the shape of the resulting clusters, and it can be computed efficiently even for large datasets, especially in cases where the similarity matrix is sparse.

We note that, after each $\mathbf{h}_t^{[n_s]}$ has been assigned a cluster label $c_t^{[n_s]} \in \{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$, we can construct predictive densities akin to those we inferred in Section 4.1 over the state of the time series at each timestep within the predictive horizon P_h:

$$p(c_t = \mathcal{D}_k) = \frac{1}{N_s} \sum_{n_s=1}^{N_s} \mathbb{I}\left(c_t^{[n_s]} = \mathcal{D}_k\right).$$

For these experiments, we select the number of clusters through the silhouette score S, which for a cluster assignment over Q objects[3] $\mathcal{H} = \{\mathbf{h}_1, \ldots, \mathbf{h}_Q\}$ that belong to one out of K classes $\mathcal{D}_1, \ldots, \mathcal{D}_K$ is given by:

$$S = \frac{1}{Q} \sum_{q=1}^{Q} S_q, \qquad S_q = \frac{b_q - a_q}{\max\left(a_q, b_q\right)}.$$

If $\mathbf{h}_q \in \mathcal{D}_k$ and $d(\mathbf{h}_q, \mathbf{h}_{q'})$ is the Euclidean distance between the objects $d(\mathbf{h}_q, \mathbf{h}_{q'})$, then:

$$a_q = \frac{1}{|\mathcal{D}_k| - 1} \sum_{\mathbf{h}_{q'} \in \mathcal{D}_k} d(\mathbf{h}_q, \mathbf{h}_{q'}), \qquad b_q = \min_{i \neq k} \frac{1}{|\mathcal{D}_i|} \sum_{\mathbf{h}_{q'} \in \mathcal{D}_i} d(\mathbf{h}_q, \mathbf{h}_{q'}),$$

i.e. a_q is the intra-cluster average distance of object $\mathbf{h}_q$ with respect to all the other objects in its same cluster, and b_q is the average *dissimilarity* of $\mathbf{h}_q$ with respect to its nearest cluster. Note that larger values of s_q are achieved when b_q is much larger than a_q, i.e. $\mathbf{x}_q$ is in a tight cluster that is clearly separated from

[3]Note that $Q = P_h \times N_s$ and that we drop the superscript for the sample index n_s for notational simplicity.

other clusters. We also remark that $S_q = 0$ if $\mathbf{h}_q \in \mathcal{D}_k$ is the only element in $\mathcal{D}_k$.

4.2.2 Visualising feature embeddings

In order to visualise low-dimensional representations of the learnt feature embeddings, we recur to the non-linear dimensionality reduction literature. This rich corpus encompasses methods such as Sammon's mapping, non-linear Principal Component Analysis, Multi-dimensional scaling, in addition to more recent approaches such as Uniform Manifold Approximation and Projection for dimension reduction (UMAP) and state-of-the-art stochastic approaches such as t-distributed Stochastic Neighbour Embedding (t-SNE) (Maaten and Hinton, 2008).

In the experiments of this section, we perform dimensionality reduction with UMAP. As per the comparison presented in (McInnes, Healy, and Melville, 2018), the method is competitive with state-of-the-art t-SNE in terms of visualisation quality, but crucially achieves better performance at preserving aspects of the data's *global* structure. Furthermore, the inferred UMAP embedding supports efficient addition of new data points after inference, as opposed to t-SNE, which needs to be re-optimised every time a new point is added to the dataset of interest.

4.2.3 Interpreting time series features

Having described our clustering and visualisation frameworks, we now illustrate how these can be used to identify structure in our forecasts. Our analysis is centred around three datasets: the Mackey-Glass and the Lorenz chaotic attractors, and an electrocardiogram recording. All three datasets are quasi-periodic, and were selected so that we can illustrate how their individual characteristic traits are reflected in this methodology. Full details of these can be found in Appendix B.

For each dataset, we first compute the spectral cluster labels over the ac-

tivations $\mathcal{H} = \{\mathbf{h}_1^{[1]}, \ldots, \mathbf{h}_t^{[n_s]}, \ldots, \mathbf{h}_{P_h}^{[N_s]}\}$, where P_h is the forecasting horizon. Each subgraphic included in Figures 4.1a, 4.2a and 4.3 is colour-coded according to these results. We then compute a lower-dimensional representation through UMAP, which is also shown in these figures.

In the case of the electrocardiogram, we note that the lower-dimensional representation in Figure 4.1c forms a loop, highlighting the quasi-periodic nature of the time series. Furthermore, we also note that the obtained clusters distinguish the characteristic peak of each quasi-period as a discovered motif in Figure 4.1b. In comparison with our experiments in Section 4.1, this suggests that we can build a density over this characteristic event even without access to a peak detector software package, as we did in previous experiments.

Let us now make a similar analysis for the Lorenz attractor. Note that a characteristic feature of this system is the alternation between two states. In Chapter 3, we noted that an ordinal approach enables us to describe the multimodality of this kind of systems. We now note that, from the learnt embedding's perspective, this is encoded as a loop with a knot, corresponding to the yellow cluster in Figure 4.2c. We observe that activations in this cluster exhibit more variance and form a sort of knot or bridge where the system decides which is state to take. At this point, the system trajectory becomes less uncertain, until it gets back to this knot state.

Finally, in the case of the Mackey-Glass, we first focus our attention on the extracted motifs and their temporal occurrence in Figure 4.3a. We note that motifs such as the dark blue and pink occur throughout the time series with various lengths and sizes, highlighting how our model is able to extract patterns that are robust to these transformations. We can observe a similar behaviour in the light green cluster, which describes motifs with a cup followed by a cap. The depth of the cup varies, e.g. akin to opening and closing transformations from the morphological filtering literature, but still belong in the same cluster.

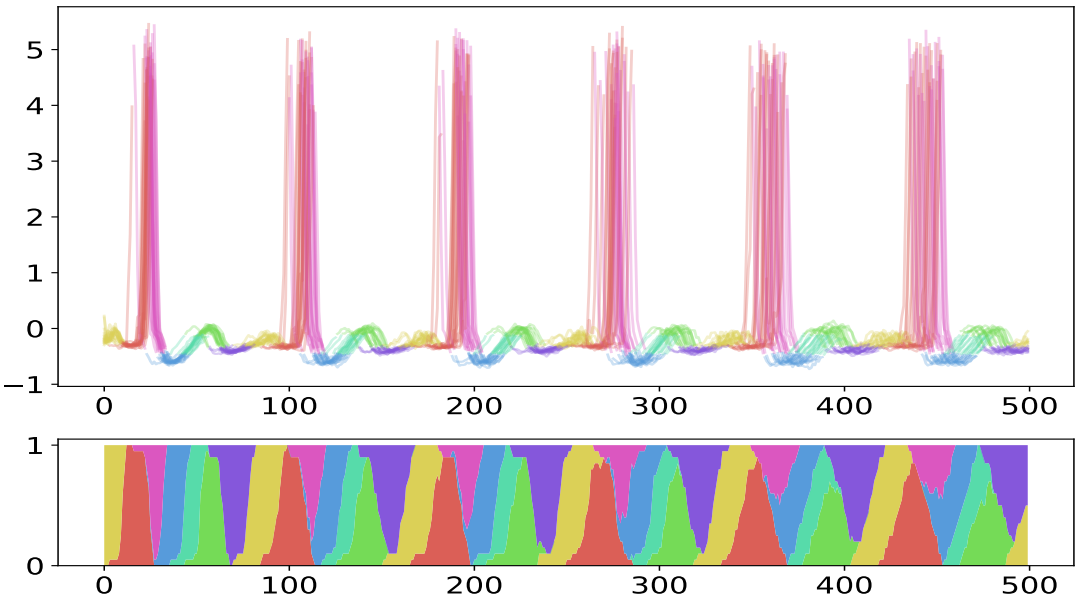
One characteristic of the Mackey-Glass chaotic system is that every quasi-cycle is distinguished by the presence of downward openings ("valleys"), such as the red motif observed at timestep 100 in Figure 4.3a, which may or may not be present in subsequent quasi-cycles. We note that our model captures this behaviour by modelling such presence or absence as distinct motif clusters. We take two examples: the pair of (light blue, green) motifs and (pink, red) motifs. The former pair describes the top "caps" in each quasi-period of the system, with the distinguishing feature being whether the cap is preceded by a "valley" or not. Deformations such as the length and depth of the valley and spread of the cap itself are modelled by different instances within the same cluster. In a similar fashion, both (light blue, green) motifs are always preceded by (pink, red) motifs, which describe the growth state of the system. The presence of another "valley" prior to reaching the top is described by the red motif cluster.

4.2.4 Concluding remarks

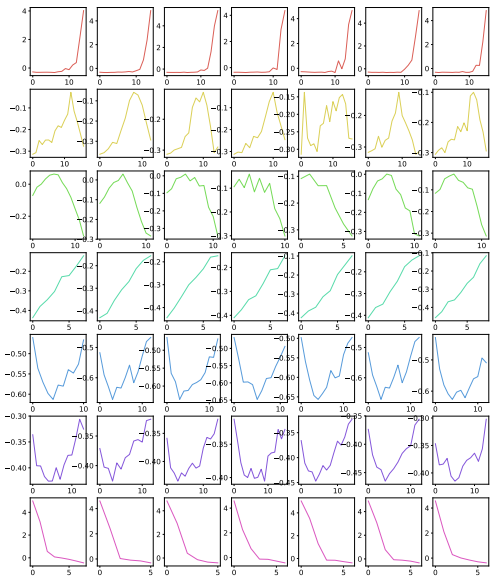
In this section, we introduced a framework to infer and visualise time series structure from MOrdReD's predictive densities. We showed how salient motifs and state predictive densities can be derived by clustering the temporal features produced by our model. Crucially, we provided empirical evidence on how morphological variations in our motifs can be captured by the learnt manifold.

Additionally, we demonstrated how characteristic traits of certain systems are encoded by our features. For instance, the chaotic behaviour of the Lorenz and Mackey-Glass systems is encoded as bridge areas in their embeddings, and quasi-periodicity of systems such as electrocardiograms is captured by learning a closed-loop manifold.

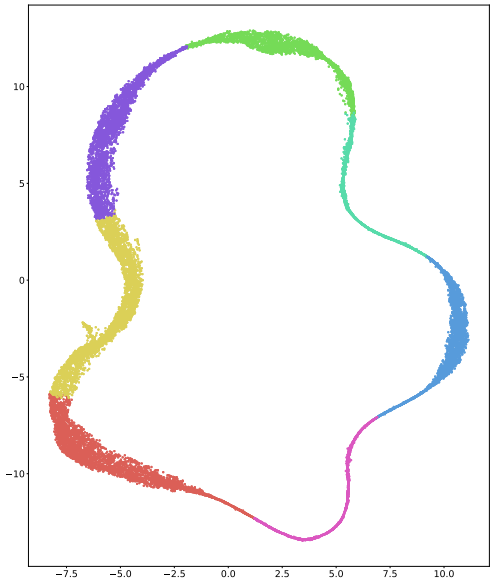
Although the results in this section are empirical and exploratory, we argue that the value of this presentation also lies in offering the reader a collection of algorithms that can provide some light for understanding the temporal features



(a) ECG motifs over time. Stacked area plots describe the probability of observing each motif at each timestep.

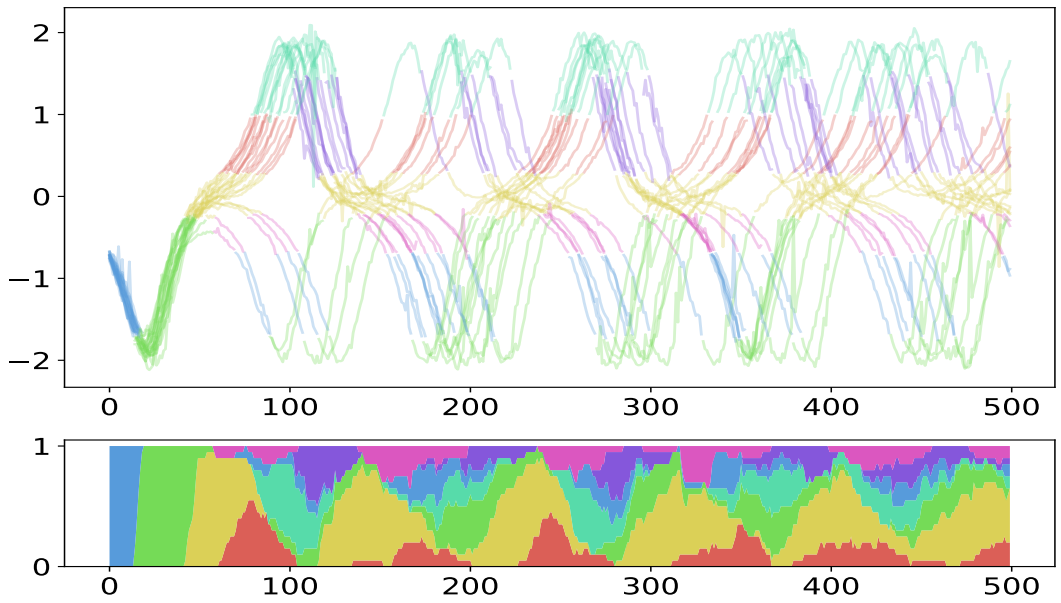


(b) ECG sample extracted motifs

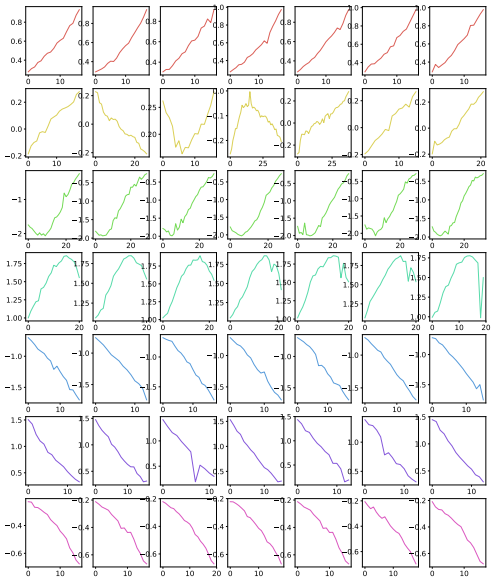


(c) ECG UMAP embedding

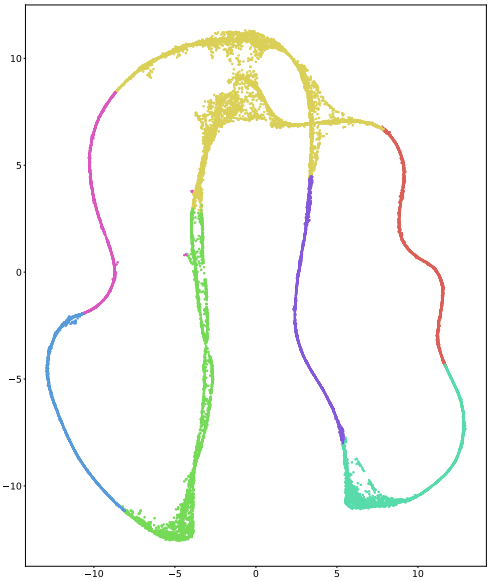
Figure 4.1: Electrocardiogram (ECG) cluster analysis and feature visualisation. Note that the lower-dimensional representation in Figure 4.1c forms a loop, highlighting the quasi-periodic nature of the time series. We also note that the obtained clusters distinguish the characteristic peak of each quasi-period as a discovered motif in Figure 4.1b.



(a) Lorenz motifs over time. Stacked area plots describe the probability of observing each motif at each timestep.

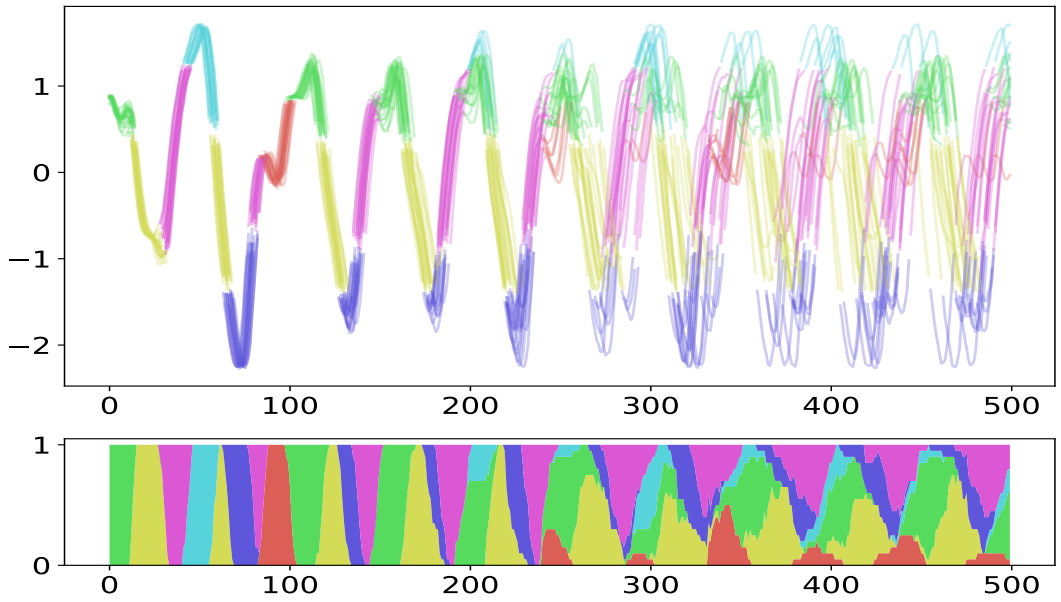


(b) Lorenz sample extracted motifs

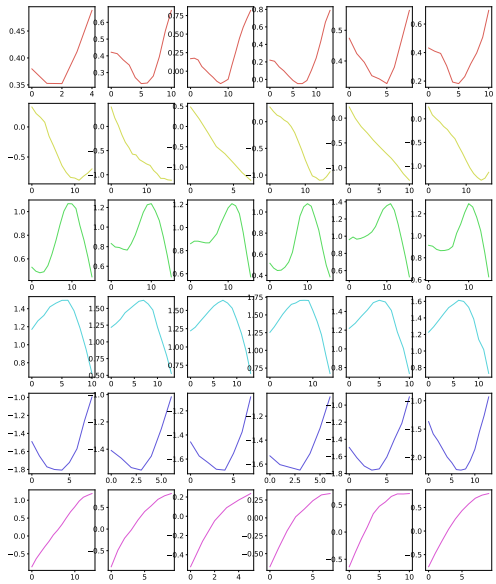


(c) Lorenz UMAP embedding

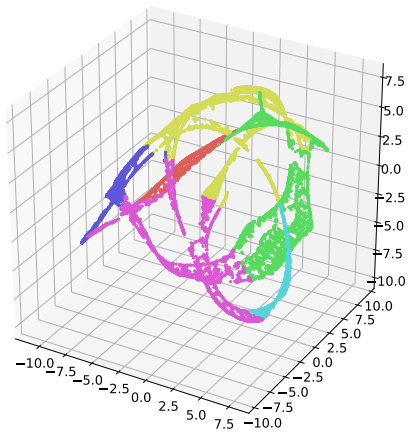
Figure 4.2: Lorenz attractor cluster analysis and feature visualisation. Note that a characteristic feature of this system is the alternation between two states. From the learnt embedding's perspective, this is encoded as a loop with a knot, corresponding to the yellow cluster in Figure 4.2c.



(a) Mackey-Glass motifs over time. Stacked area plots describe the probability of observing each motif at each timestep.



(b) Mackey-Glass sample extracted motifs



(c) Mackey-Glass UMAP embedding

Figure 4.3: Mackey-Glass attractor cluster analysis and feature visualisation. We note that motif clusters encompass patterns that vary in length, size and other morphological traits related to openings and closings.

learnt by our forecasting framework. We also explore further avenues for future work in Chapter 6.

4.3 Multivariate forecasting

The second part of this chapter develops on our Memory-endowed Ordinal Regression Deep neural network (MOrdReD) framework to allow for multiple-output prediction. Up to this point, we have dealt with univariate time series $\mathbf{x} \in \mathbb{R}^T$ with T scalar measurements $x \in \mathbb{R}$. In many practical applications, however, we encounter the task of simultaneously predicting a list of L *correlated* time series $\mathbf{x}^{[1]}, \ldots, \mathbf{x}^{[L]} \in \mathbb{R}^T$. To illustrate, this is the case of tasks such as predicting measurements in a multiple sensor network, or weather forecasting, where variables such as temperature and solar radiance may be correlated.

When modelling multivariate time series, the simplest approach is to assume that time series channels are mutually conditionally independent, in which case an individual regression model (such as MOrdReD) is fit to each channel separately. This captures the underlying relationships between inputs and outputs, but omits those between targets (Borchani et al., 2015). Modelling cross-channel knowledge transfer is desirable as it may benefit predictive accuracy, but for some models it also incurs a severe cost in computational complexity.

In this section, we directly address the multivariate time series forecasting task by introducing a MOrdReD multivariate extension in Section 4.3.1. In a similar line to the forecasting experiments presented in Section 3.5, we then introduce a multiple-output Gaussian Process (GP) baseline, which we describe in Section 4.3.2. We then compare the predictive performance of these models in Section 4.3.3.

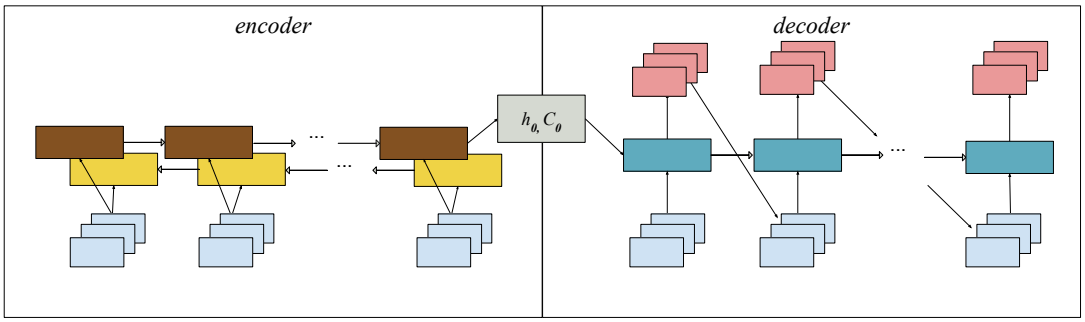


Figure 4.4: Visual depiction of MultiMOrdReD scanning a time series with $L = 3$ channels. MultiMOrdReD's LSTM layers infer a joint space over the multivariate time series channel.

4.3.1 Multivariate MOrdReD

Let us now show how we can perform multivariate forecasting with our MOrdReD model. We first introduce our multiple-output extension, the Multivariate Memory-endowed Ordinal Regression Deep neural network (MultiMOrdReD), which infers a joint embedding across the time series channels of interest. This feature space is the result of allowing our model to look at all L channels simultaneously at every timestep, thereby enabling information sharing across channels. Such features are subsequently exploited by separate fully-connected neural network layers for each channel. Combining this approach with Monte Carlo dropout results in a set of samples that we use to infer full joint predictive distributions at each timestep, in the shape of a Variational Bayesian Gaussian Mixture Model (VBGMM) (Penny and Roberts, 2000).

In Section 3.1, we introduced a quantisation scheme for univariate time series, which is based on partitioning the observed time series range into M equally-sized bins. In the univariate case, this is simply a partitioning of a subset of the real line which requires $\mathcal{O}(M)$ in memory to encode each time series measurement. As per the presentation in Chapter 3, this quantisation scheme also imposes a piece-wise uniform likelihood over the values of the original, real-valued time series, which

in turn enables us to model flexible, multi-modal predictive distributions in the form of a Variational Bayesian Gaussian Mixture Model (VBGMM).

Nevertheless, generalising this to higher dimensions so as to model *simultaneous* occurrences of time series values, e.g. where each bin is a hyper-cube in $\mathbb{R}^L$, would require a total number of bins that grows exponentially on the number of channels L, in $\mathcal{O}(M^L)$. This would rapidly become intractable or obsolete for large or small values of M, respectively. Indeed, even if this storage challenge could be overcome, its effect on the number of parameters of over-parameterised models such as neural networks could render the effort worthless.

In this section, we propose a multivariate extension to MOrdReD that achieves quantisation in $\mathcal{O}(M \cdot L)$ whilst also enabling the description of correlated outputs. Our approach is divided in two phases: firstly, inferring a joint feature space over the multivariate time series; this phase is then followed by the inference of joint predictive distributions.

In the first phase, we quantise each time series channel *independently*, as described in Section 3.1. We then infer a MOrdReD model with L input and (fully-connected) output layers, one for each channel, respectively. Crucially, these are all connected to a joint recurrent layer, which enables information sharing across tasks. To illustrate, in Figure 4.4 we provide an example of this MultiMOrdReD architecture for the case of a 3-dimensional time series.

After fitting a MOrdReD model as presented in Section 3.3, we construct a sample set of N_s L-dimensional samples by executing N_s stochastic forward passes. In order to capture cross-channel correlation, we infer a Variational Bayesian Gaussian Mixture Model (VBGMM) with full covariance matrix (Penny and Roberts, 2000) at every time step over the resulting set of samples. We refer the reader to Section 3.5.2 for a summary of this model, and to (Penny and Roberts, 2000) for a full presentation. Let us simply recall that VBGMMs can describe target correlations by allowing mixtures with full covariance matrices, and

crucially, inferring this model in an (approximate) Bayesian way guards against pathologies observed in point-inference methods (such as Maximum A Posteriori (MAP)), whilst also allowing for principled model selection, including natural shrinkage of the number of mixtures in the model.

4.3.2 Multiple-output Gaussian Processes

We now describe our Gaussian Process (GP) baseline for multivariate forecasting. Modelling channel cross-correlations requires that the GP covariance function characterise both the correlation structure of each channel, in addition to the cross-correlations between channels (Rasmussen and Williams, 2006). In the GP literature, this has been addressed by modelling the cross-covariance between outputs through a coregionalisation kernel. This is also known as co-kriging in the geostatistics literature (Cressie, 1993).

Additionally, correlated outputs can be modelled as linear mixtures of random samples drawn from uncorrelated latent GPs. This is known as a Linear Model of Coregionalisation (LMC) (Journel and Huijbregts, 1978), which generalises other approaches such as the Intrinsic Model of Coregionalisation (ICM) (Bonilla, Chai, and Williams, 2008) and the Semiparametric Latent Factor Model (SLFM) (Teh, Seeger, and Jordan, 2005). This is the approach we follow in our experiments in Section 4.3.3.

Constructing an input matrix for this approach requires augmenting each entry with an additional dimension in which the coregionalisation kernel operates, and subsequently stacking all of them. This new dimension simply identifies the time series channel each input entry is associated with, but effectively increases the dataset length by a factor of L, which in turn severely increases the cost of performing exact GP inference. To address this issue, we perform approximate Bayesian inference over a sparse GP (Matthews, Hensman, et al., 2016; Hensman, Matthews, and Ghahramani, 2015), using the open source implementation

available in GPFlow (Matthews, Alexander, et al., 2017).

4.3.3 Experiments and results

We now compare our Multivariate Memory-endowed Ordinal Regression Deep neural network (MultiMOrdReD) in the context of long-term forecasting, following a similar reasoning as in Section 3.5.

Experimental setup

We evaluate our models over 8 datasets, which are described fully in Appendix B.1. In summary, these eight datasets were chosen following similar desiderata as in Section 3.5.1, namely by drawing them from varied application domains and ensuring both synthetic and real-world data are represented. Two of these datasets are the Lorenz and Mackey-Glass chaotic attractors. To further assess the robustness of our method to noise, we also add 3 variants of the Lorenz attractor with various levels of additive white noise. The last 3 datasets were drawn from real-world data: two correspond to environmental multi-sensor recordings of air temperature, relative humidity and atmospheric pressure; the last dataset corresponds to an individual's heart and breathing recordings. Full details of all datasets are given in Appendix B.1.

We compare the performance of three different models across the following baselines:

- Multivariate Memory-endowed Ordinal Regression Deep neural network (MultiMOrdReD), as introduced in Section 4.3.1,
- a sparse Gaussian Process (GP) model with a coregionalisation kernel, as described in Section 4.3.2,
- and a MOrdReD baseline that models each time series channel separately, which we codename IndyMOrdReD.

The number of hidden units n_u, L_2 regularisation λ and dropout rate θ_{drop} of our MOrdReD models were optimised by grid search over $n_u \in \{64, 128\,256, 320, 512, 768\}$, $\lambda \in \{10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}, \theta_{\text{drop}} \in \{0.25, 0.5\}$. For our coregionalised GP baseline, we optimised over the Matérn 5/2 and the Rational Quadratic kernels with 5 restarts each with 1024 inducing points for the sparse approximation. These kernels model a wide variety of phenomena, such as those that are twice differentiable (i.e. show a degree of "roughness") in the case of the Matérn 5/2, or those that have processes at different characteristic length scales, in the case of the Rational Quadratic.

Predictive accuracy and uncertainty quantification are then assessed as introduced in Section 3.5.4, across three different metrics: the negative log-likelihood (NLL) and the predictive mean and median's Root Mean Squared Error (RMSE).

Results and discussion

Having introduced a multiple-output extension to our ordinal regression time series forecasting framework, we now show our results in Tables 4.2, 4.3 and 4.4. For each table, we also provide summary rank metrics. These suggest that both MOrdReD baselines achieve or surpass our multiple-output GP baseline.

We now examine more closely the results of both MOrdReD models. From the rank metric, it would seem like both achieve similar performance across all metrics and datasets. Nevertheless, upon closer inspection at the datasets, we notice that IndyMOrdReD seems to draw level with MultiMOrdReD as white noise levels increase. This becomes more evident in the case of the Lorenz datasets, where the performance gap between both models narrows as the amount of white noise increases. On the other hand, MultiMOrdReD performs better at systems with little-to-no noise. We argue this is the case because, in the limit of vast amounts of noise, the cross-correlation between channels tends to zero.

We conclude this subsection by noting that Multivariate Memory-endowed Or-

	MultiMOrdReD	IndyMOrdReD	MultiGP
TEMP-1	2288.0259	**2021.8028**	2139.9984
TEMP-2	1388.0796	**1320.8463**	1752.4292
ECG-AIR	882.2518	**-386.1975**	1429.8947
MACKEY-3D	**-21.0894**	406.3279	2857.9082
LORENZ-ORIG	**-258.8417**	-14.6267	966.5116
LORENZ-LOW	**673.5023**	1474.7758	1297.8398
LORENZ-MOD	**1881.0110**	1980.7656	2184.5955
LORENZ-HIGH	2223.7596	**1990.9901**	2168.0593
Average rank	1.7500	**1.6250**	2.6250

Table 4.2: Negative log-likelihood (NLL). The summary rank is obtained by ranking all baselines for each time series and then averaging across datasets.

dinal Regression Deep neural network (MultiMOrdReD) improves generalisation for multivariate time series with moderate levels of noise through a quantisation scheme that is $\mathcal{O}(M \cdot L)$, where L is the number of channels and M is the number of ordinal bins per channel. This, nevertheless, comes at the expense of inferring a VBGMM at every time step so as to model cross-channel correlations, which is computationally expensive in time. Future work in this avenue concerns exploring further quantisation schemes, which we discuss in Section 6.2.

	MultiMOrdReD	IndyMOrdReD	MultiGP
TEMP-1	0.9629	**0.9597**	0.9731
TEMP-2	**0.5515**	0.6827	0.8400
ECG-AIR	1.0524	**0.6088**	1.9593
MACKEY-3D	**0.5301**	0.6670	1.2750
LORENZ-ORIG	**0.3801**	0.5987	0.7493
LORENZ-LOW	**0.6865**	0.8296	0.7862
LORENZ-MOD	**0.8723**	0.9368	1.0144
LORENZ-HIGH	1.0140	**0.9338**	0.9972
Average rank	**1.5000**	1.7500	2.7500

Table 4.3: Mean Root Mean Squared Error (RMSE). The summary rank is obtained by ranking all baselines for each time series and then averaging across datasets.

	MultiMOrdReD	IndyMOrdReD	MultiGP
TEMP-1	0.9894	**0.9722**	0.9728
TEMP-2	**0.5494**	0.7039	0.8489
ECG-AIR	1.0457	**0.6829**	1.0199
MACKEY-3D	**0.5461**	0.7300	1.2760
LORENZ-ORIG	**0.4622**	0.5701	0.7816
LORENZ-LOW	**0.6544**	0.8196	0.7864
LORENZ-MODE	**0.8817**	0.9375	1.0146
LORENZ-HIGH	1.0244	**0.9369**	0.9728
Average rank	**1.7500**	**1.7500**	2.5000

Table 4.4: Median Root Mean Squared Error (RMSE). The summary rank is obtained by ranking all baselines for each time series and then averaging across datasets.

5

Towards a General Unified Model for time series forecasting

In Chapter 3, we introduced our Memory-endowed Ordinal Regression Deep neural network (MOrdReD) for univariate time series forecasting, and evaluated it in the context of long-term forecasting. We assumed that the time series of interest had "sufficient" data[1] to infer a recurrent neural network's parameters, as set out in our data requirements in Section 3.5.1. In many forecasting settings, however, we may only have access to one or two dozens of observations, contrary to the cases introduced in Section 3.5.1. This restricts the classes of models that we can learn without overfitting to data, especially in the case of over-parameterised models such as neural networks..

In the time series forecasting corpus, Bayesian nonparametric methods such as Gaussian Process Regression (GPR), introduced in Section 3.5.2, have dominated

[1]Without loss of generality, providing formal bounds for the amount of data needed to learn a concept with a specific class of models, such as neural networks, falls outside the scope of this thesis. The experiments in this chapter are performed with *comparatively* much less data than the ones available in Chapter 3. For further discussion on sample complexity measures, we refer the reader to Golowich, Rakhlin, and Shamir (2017), Mohri, Rostamizadeh, and Talwalkar (2018), and Shalev-Shwartz and Ben-David (2014).

the literature for forecasting this type of datasets. This may arguably stem from the fact that full predictive distributions can be inferred exactly for small datasets in the GPR model, in addition to the substantial amount of prior knowledge encapsulated through the handcrafted covariance function.

In this chapter, we propose a novel, neural network-based General Unified Model (GUM) for time series forecasting that directly addresses the problem of inferring accurate and reliable predictive distributions from scarce data. Our approach is based on an extension to our MOrdReD framework, introduced in Chapter 3. In particular, we showcase how predictive distributions can be inferred for time series with few available measurements by a model that has learnt a shared feature representation over an associated space of quantised time series, represented by a dataset of several auxiliary time series.

Our approach takes inspiration from the *transfer learning* corpus, a concept that we introduce in Section 5.1, by recognising that knowledge across forecasting tasks can be relevant to each other. By assuming an ordinal regression approach, we learn a shared representation over the space of *quantised* time series. Knowledge transfer is readily enabled by recognising that seemingly different time series can now be compared in terms of their ordinal bin sequences, in a scale-invariant manner.

The remainder of this chapter is structured as follows: in Section 5.1, we discuss relevant literature on transfer learning for time series forecasting, which paves the way to introduce our GUM approach in Section 5.2. We then evaluate GUM in two different settings: firstly, in Section 5.3, we assess GUM's predictive distributions over new time series *without* any parameters updates, in a zero-shot fashion; this is accompanied by an analysis of the latent embedding learnt by GUM in Section 5.4.

Lastly, in Section 5.5, we present the second part of the evaluation, where we focus on the case in which GUM's parameters serve as initial values. These

are then *adapted* to a new, unseen time series. We provide evidence that this approach achieves better results than a randomly-initialised MOrdReD model (also known as a *tabula rasa* model) trained from scratch.

5.1 Transfer learning

The General Unified Model (GUM) we introduce in Section 5.2 is based on the concept of transfer learning, which is closely related to Multitask Learning (MTL). In the MTL time series forecasting setting, each time series dataset is considered a distinct task. The aim is to learn a *shared representation* of features that are salient across all of these tasks (Caruana, 1997). To illustrate, consider the case of object detection in images, where a single model learns a shared representation of foundational, low-level, notions such as edges, shapes and textures. Such features can be relevant to learn recognition models of new object classes.

Considering an ignorant agent A_{ign} (for this purpose, a neural network initialised at random, before any parameter inference takes place) and an informed agent A_{inf} that has learnt to execute related tasks. The usefulness of such prior knowledge for new tasks may then be quantified, for instance, in three axes identified by Torrey and Shavlik (2010):

1. The performance of both agents in the new task *before* any further inference is done;

2. the time it takes both agents to fully learn the new task;

3. the final performance of both agents in the new task *after* the parameter inference stage.

These guidelines motivate our experiments in Sections 5.3 and 5.5.

In this transfer learning context, we often need to learn a model from limited labelled data. As a result, there exists a rich literature in transfer learning

for few-shot learning, especially in the Computer Vision community, where pre-trained models such as VGGNet (Simonyan and Zisserman, 2014) and AlexNet (Krizhevsky, Sutskever, and Hinton, 2012) are regularly used as either feature extractors, when followed by simpler bespoke classifiers; or whose parameters act as initial values to fine-tune a model (Xie et al., 2016; Huh, Agrawal, and Efros, 2016). In the case of Natural Language Processing (NLP), word embeddings obtained from models such as GloVe (Pennington, Socher, and Manning, 2014) and word2vec (Mikolov et al., 2013) are commonly used as features for a wide range of tasks (Conneau et al., 2017; Akata et al., 2015; Xian et al., 2016).

An extreme case of such *few-shot learning* tasks is *zero-shot learning* or *zero-data learning*, in which a model must generalise to new tasks in the absence of training data (Larochelle, Erhan, and Bengio, 2008). This is an extreme scenario where generalisation relies on crafting a purposeful shared representation space across tasks. For example, in the Computer Vision setting, Lei Ba, Swersky, Fidler, et al. (2015) introduced a zero-shot object recognition system that can recognise unseen image classes from encyclopedic descriptions. Their method was enabled by crafting a joint representation space between raw text and visual features learnt by a convolutional neural network.

Few-shot tasks are also commonplace in the time series forecasting literature. To the best of our knowledge, however, transfer learning literature in this setting is significantly more scarce than in the case of Computer Vision or NLP, although it has seen some success in predicting wind speed direction in newly-built farms by transferring knowledge from older farms (Hu, Zhang, and Zhou, 2016). It has similarly been applied to forecast cross-building energy (Ribeiro et al., 2018), port containers throughput (Jin et al., 2014) and wind and solar power (Qureshi et al., 2017; Shireen et al., 2018). Nevertheless, these and other similar works often provide evaluations solely based on the residuals of point forecasts. Instead, models that provide full probabilistic forecasts are desirable, as we motivated in

Section 2.1.4, since this enables practitioners to incorporate forecast uncertainty into their decision-making processes. This can be done, for example, with Gaussian Processes (GPs) (Rasmussen and Williams, 2006), which are state-of-the-art, non-parametric probabilistic models that generalise well from little data.

On the contrary, deep and recurrent neural networks, which have achieved great success across a wide range of sequential learning tasks in NLP (Sutskever, Vinyals, and Le, 2014; Graves, Mohamed, and Hinton, 2013) and time series forecasting, often have millions of parameters, making them likely to overfit to scarce data. Such models are unfit to solve time series forecasting tasks where only a handful of observations are available.

This motivates the core contribution of this chapter: the General Unified Model (GUM) for time series forecasting. GUM is a novel neural network-based approach to infer full predictive distributions for time series prediction tasks where there is little-to-no training data at hand. Using an ordinal regression approach, we learn a shared representation over the space of quantised time series, as we describe in Section 5.2, which thereafter enables forecasting of unseen time series. In Sections 5.3 and 5.5, we assess GUM's predictive performance over two different settings: zero-shot and few-shot learning, respectively.

5.2 A General Unified Model

We now introduce our General Unified Model (GUM) ordinal regression neural network for time series forecasting. Our strategy consists in inferring a shared representation space for time series forecasting tasks, which we do by learning a *joint* Memory-endowed Ordinal Regression Deep neural network (MOrdReD) embedding across a collection of varied auxiliary time series. This can then be transferred over to unseen time series for which there is little-to-no training data to fit a customised model. We briefly recapitulate MOrdReD's notation below,

and refer the reader to Chapter 3 for the full description.

5.2.1 Ordinal Regression Recurrent Neural Networks

MOrdReD is an ordinal regression approach to time series forecasting. In the ordinal setting, real-valued time series observations are firstly framed and quantised into a number M of ordinal bins $\mathcal{C} = \{\mathcal{C}_k\}_{k=1}^{M}$, using a 1-of-$M$ scheme as described in Section 3.1. Such type of discrete symbol sequences have been most efficiently dealt with in the NLP literature by deep recurrent neural networks, and in particular sequence-to-sequence extensions of the Long Short-Term Memory (LSTM) recurrent neural network layer (Sutskever, Vinyals, and Le, 2014).

Sequence-to-sequence models, which we introduced in Section 3.2.3, fit together two LSTM layers, an encoder $f^{(\text{enc})}$ and a decoder $f^{(\text{dec})}$. In the first stage of each *forward pass*, the encoder sequentially summarises the last P measurements in the time series $\mathbf{X}^{(\tau-1)} = (\mathbf{x}_{\tau-P}, \ldots, \mathbf{x}_{\tau-1})$ as salient temporal characteristics $\mathbf{h}_0^{(\text{dec})}, \mathbf{C}_0^{(\text{dec})}$. These summaries are then fed into the decoder to produce an output sequence iteratively. The $(t+1)$-th element of the output sequence, $\hat{\mathbf{x}}_{t+1}$, is thence given by:

$$
\begin{aligned}
(\mathbf{h}_0^{(\text{dec})}, \mathbf{C}_0^{(\text{dec})}) &= f^{(\text{enc})}(\mathbf{X}^{(\tau-1)}), \\
\hat{\mathbf{x}}_{t+1} &= \text{Softmax}\left(f^{(\text{dec})}(\mathbf{h}_{t-1}^{(\text{dec})}, \mathbf{C}_{t-1}^{(\text{dec})}, \mathbf{x}_t)\right).
\end{aligned}
$$

The softmax output $\hat{\mathbf{x}}_{t+1}$ can be readily interpreted as a categorical probability distribution at each timestep over the set of ordinal bins, $\mathcal{C} = \{\mathcal{C}_k\}_{k=1}^{M}$, associated with the time series. It can furthermore be fed back to the decoder in an autoregressive fashion in order to compute the next element in the output sequence, thus enabling long-term forecasting.

Finally, in Sections 2.3.3 and 3.3, we also discussed how to perform approximate Bayesian inference over this sort of models, based on recent findings by Gal

and Ghahramani (2016b) that link the dropout regularisation technique (Srivastava et al., 2014) to performing variational Bayes over deep GPs. In particular, we described how post-inference, dropout-enabled forward passes can be regarded as draws from the approximate predictive posterior distribution.

5.2.2 Inferring a unified time series embedding

Our General Unified Model (GUM) approach is based on inferring a shared representation space of time series. This shared embedding is learnt by fitting a joint MOrdReD model over a single compilation dataset $\mathcal{X}_{\text{aux}}$ of 20 different quantised auxiliary time series, which are assumed to possess sufficient observations for training. Indeed, these auxiliary time series correspond to some of those from our long-term forecasting experiment from Section 3.5. Full detail of these time series is given in Appendix B.

Ordinal regression enables us to learn such a shared representation space by reformulating time series prediction as a symbol sequence task, i.e. the embedding is inferred over symbol patterns that are invariant to the original scale of the time series, as all auxiliary time series are quantised independently. This space can thereafter serve to forecast previously unseen time series for which there is insufficient data to learn a bespoke model.

The description above signals the key role that the quantisation phase of MOrdReD plays in learning the shared embedding space. Let us recall, however, that the quantisers we proposed in Section 3.1, and that we recurred to for our experiments in Chapter 3, assume that the maximum and minimum values of a time series $\mathbf{x}$ of length T, $x_{\min}, x_{\max}$, will not change within a given predictive horizon P_h, i.e. that $x_{\min}, x_{\max}$ are the same at times $t = T$ and $t = T + P_h$.

This assumption may be too strong for time series with scarce data, in which the observed time series range may not be representative for forthcoming observations, even in the short-term. We are thus required to estimate bounds for

$x_{\min}, x_{\max}$ up to horizon $t = T + P_h$. Achieving this automatically from scarce data is a tough task and arguably merits an exhaustive exploration on its own (Kuznetsov and Mohri, 2015), which we leave for future possible extensions to GUM.

Within the scope of our experiments, we propose a simple linear heuristic in order to bound $x_{\min}, x_{\max}$ at $t = T + P_h$. Given a predictive horizon P_h and the largest first-order finite difference of the input sequence $\mathbf{X}^{(T)}$, $\Delta_{\max}$, our linear heuristic approximates $\tilde{\mathbf{X}}_{\min}, \tilde{\mathbf{X}}_{\max} = \mathbf{X}_{\min} - H\Delta_{\max}, \mathbf{X}_{\max} + H\Delta_{\max}$. This is the heuristic we use to evaluate GUM in the remainder of this chapter. We conclude by noting that the estimation of these limits can be improved, for instance, by relating it to the inference of Lipschitz constants, as shown by Calliess (2015).

5.3 Zero-shot forecasting

In this first experiment, we evaluate our General Unified Model (GUM) in a *zero-data* forecasting setting, i.e. assuming the model has no training data at hand to perform any parameter updates. We firstly infer a joint MOrdReD model over 20 auxiliary time series, all of which are quantised independently, but using the same number of bins $M = 150$. These auxiliary datasets are drawn from a number of varied application domains, such as meteorology, dynamical systems, physiology and astronomy, as detailed in Appendix B. Through this ordinal approach, MOrdReD is also capable of learning a unified embedding over the space of ordinal bin sequence patterns. Additional insights about this embedding space are provided in Section 5.4.

We now focus on evaluating the predictive performance of this method over unseen time series. In this zero-shot setting, we evaluate our approach over a subset $\mathcal{X}_{\text{M4}}$ of the yearly frequency M4 dataset (Makridakis, Spiliotis, and Assimakopoulos, 2018) in comparison with state-of-art models that can generalise and

Algorithm 1 A General Unified Model for Time Series Forecasting

inputs:
Auxiliary datasets $\mathcal{X}_{\text{aux}}$,
evaluation datasets $\mathcal{X}_{\text{M4}}$
configuration:
number of quantisation bins M,
lookback window length P,
predictive horizon P_h

Phase 1: inference
for all i, $\mathbf{X}_{\text{aux}}$ in enumerate($\mathcal{X}_{\text{aux}}$) **do**
  $\mathcal{X}_{\text{quantised}}[i]$, Aux_Quantisers$[i]$ = quantise($\mathbf{X}_{\text{aux}}$, M)
  $\mathcal{X}_{\text{framed}}[i]$ = frame($\mathcal{X}_{\text{quantised}}[i]$, P)
end for
$\mathcal{X}_{\text{train}} \leftarrow$ concatenate($\mathcal{X}_{\text{framed}}$, axis=0)
GUM $\leftarrow$ MOrdReD.fit($\mathcal{X}_{\text{train}}$)

Phase 2: evaluation
for all j, $\mathbf{X}_{\text{M4}}$ in enumerate($\mathcal{X}_{\text{M4}}$) **do**
  $\mathcal{X}_{\text{M4-quantised}}[j]$, M4_Quantisers$[j]$ = quantise($\mathbf{X}_{\text{M4}}$, M)
  pred $\leftarrow$ GUM.predict($\mathcal{X}_{\text{M4-quantised}}[j]$, P_h)
  run evaluation metrics
end for

be inferred from scarce data, which we introduce in Section 5.3.2. We summarise our GUM approach in Algorithm 1.

5.3.1 Data

We use two collections of datasets in our experiments. The first collection serves to fit our GUM ordinal regression neural network, and it corresponds to 20 auxiliary datasets drawn from Fulcher, Little, and Jones (2013), Rezek and Roberts (1998), and Bramble Bank (2018) that cover an ample range of application domains, such as dynamical systems, meteorology, physiological recordings, and others. A full list of these auxiliary datasets is provided in Appendix B. We truncate them to the first 30,000 observations and quantised individually over $M = 150$ ordinal bins, as described in Section 3.1.

We then evaluate our General Unified Model (GUM) ordinal regression neural

network and the benchmarks over a second collection of time series, which is the yearly-frequency segment of the M4 competition dataset (Makridakis, Spiliotis, and Assimakopoulos, 2018). We select all time series with no missing data (no interpolation needed) that have at least 36 observations available, which results in 150 distinct time series. We remark that benchmark models are trained on these M4 datasets, contrary to our GUM neural network, which only sees them at testing time and is inferred instead over the auxiliary datasets above.

5.3.2 Benchmarks

We compare our General Unified Model (GUM) neural network against two state-of-the-art time series forecasting models that can generalise from scarce data: Gaussian Process Regression (GPR) (Rasmussen and Williams, 2006) and State-space AR(p) models (Durbin and Koopman, 2012), both of which we introduced in Section 3.5.2. In the case of GPR, we fit two instances for every time series: one with the Mátern 5/2 kernel and one with the Rational Quadratic kernel. Both choices are commonly found in the literature to model a wide range of physical phenomena. In the case of State-space AR(p), we fit two instances with $p = 3, 4$.

5.3.3 Experimental setup

We split each time series into $P = 21$ lookback timesteps, and $P_h = 15$ timesteps for evaluation. In the case of GUM, a quantiser is fit for every time series aided by the heuristic described in Section 5.2.2 over the 21-input sequence. Their quantised representations are then fed into the LSTM encoder at testing time, and prediction continues as described in Phase 2 of Algorithm 1. Importantly, no neural network parameters are updated whilst scanning the 21-input sequence, in a zero-shot fashion. All parameters are inferred only over our auxiliary dataset, as described in Phase 1 of Algorithm 1.

In the case of the benchmarks, we fit a distinct model instance for each selected

M4 time series using the first $P = 21$ observations as training data. Each model is subsequently evaluated over the next $P_h = 15$ timesteps. We remark this difference in comparison with our GUM neural network, which only sees the M4 dataset as a collection of test examples and whose parameters are never updated with any of these time series.

5.3.4 Metrics

We evaluate our models across three axes, using the metrics introduced in Section 3.5.4:

- **Predictive accuracy.** We measure the deviation between the predictive mean of our models and the ground truth by computing the well-known Root Mean Squared Error (RMSE).
- **Uncertainty quantification.** We evaluate the quality of the probabilistic forecasts by measuring the negative log-likelihood (NLL) of the ground truth time series.
- **Forecast reliability.** We measure the robustness and calibration of predictive densities, i.e. the extent to which output densities can be interpreted as real-world probabilities, through the QQ distance metric introduced in Section 3.5.4.

5.3.5 Predictive performance comparison

We present our results in Tables 5.1 and 5.2. In Table 5.1, we show the percentage of datasets in which our GUM neural network outperforms each of our benchmarks for every metric. We show that across all metrics, our model consistently achieves better performance than all baselines for the majority of cases.

Then in Table 5.2, we provide the average rank of our models across all 150 filtered datasets. In the case of GPR and AR(p), the instance with the best

	AR3	AR4	Mat5/2	RatQ
NLL	65%	70%	59%	62%
RMSE	64%	68%	54%	58%
QQD	74%	75%	67%	67%

Table 5.1: Percentage of unseen evaluation datasets our GUM approach outperforms each of our benchmarks for each metric. In a pairwise fashion, GUM performs better with respect to our baselines, in all metrics, in the majority of the 150 evaluation dataset cases.

	GUM	GP	AR
NLL	**1.76**	2.04	2.20
RMSE	**1.83**	2.01	2.15
QQD	**1.65**	2.01	2.34

Table 5.2: Average rank achieved by each model. Ranks 1 (best) to 3 (worst) were assigned to each model for each metric and each one of the 150 evaluated datasets. The average provided above shows that we expect GUM to perform better than our optimised baselines across all metrics in a given task.

performance across kernels or model order p was chosen. Ranks from 1 (best) to 3 (worst) were assigned for every metric, dataset and model and averaged across all M4 evaluation datasets. This metric provides evidence that, on average, we expect GUM to outperform all other models; and additionally, when it does not, it can still be expected to come as a second competitor.

GUM's performance in this task relies on the inferred shared representation space. As we show in Section 5.2.2, this embedding allows quantised M4 time series to fire similar neurons to those from the auxiliary dataset, which enables our model to infer predictive densities based on prior time series forecasting tasks.

5.4 Feature embedding analysis

We now focus our attention on the feature embedding learnt by our GUM ordinal neural network, and in particular on the fixed-dimensional summaries $\mathbf{h}_0^{(\text{dec})}, \mathbf{C}_0^{(\text{dec})}$ produced by the LSTM encoder after scanning an input sequence, as described in Section 5.2. Through this analysis, we further investigate the inner mechanics of our approach, whilst also giving some intuition about why it is able to infer accurate predictive distributions for unseen time series without further parameter updates.

We first draw a random sample of 50 excerpts of length $P-1=20$ (excluding the decoder seed timestep) from each auxiliary time series and compute their fixed-dimensional summaries by executing a forward pass of GUM's encoder. We then do the same for all M4 evaluation input sequences. This amounts to $Q = 1150$ frames, 1000 from the auxiliary datasets and 150 from the queried yearly frequency M4 dataset.

Intra-group minimum variance clusters can then be efficiently computed through Ward's clustering method (Ward Jr, 1963). The results shown in both Figures 5.1 and 5.2 are based on the resulting groups after applying this method with $K = 36$ clusters. As per our experiments, we tried different values for the number of clusters ranging between 5 and 50, and kept the one with the maximum silhouette score ($K = 36$). From Section 4.2.1, we recall that the silhouette score S for a cluster assignment over Q objects $\mathbf{h}_1, \ldots, \mathbf{h}_Q$ that belong to one out of K classes $\mathcal{D}_1, \ldots, \mathcal{D}_K$ is given by:

$$S = \frac{1}{Q}\sum_{q=1}^{Q} S_q, \qquad S_q = \frac{b_q - a_q}{\max\left(a_q, b_q\right)}.$$

If $\mathbf{h}_q \in \mathcal{D}_k$ and $d(\mathbf{h}_q, \mathbf{h}_{q'})$ is the Euclidean distance between the objects $d(\mathbf{h}_q, \mathbf{h}_{q'})$,

then:

$$a_q = \frac{1}{|\mathcal{D}_k| - 1} \sum_{\mathbf{h}_{q'} \in \mathcal{D}_k} d(\mathbf{h}_q, \mathbf{h}_{q'}), \qquad b_q = \min_{i \neq k} \frac{1}{|\mathcal{D}_i|} \sum_{\mathbf{h}_{q'} \in \mathcal{D}_i} d(\mathbf{h}_q, \mathbf{h}_{q'}),$$

i.e. a_q is the intra-cluster average distance of object $\mathbf{h}_q$ with respect to all the other objects in its same cluster, and b_q is the average *dissimilarity* of $\mathbf{h}_q$ with respect to its nearest cluster. Note that larger values of s_q are achieved when b_q is much larger than a_q, i.e. $\mathbf{x}_q$ is in a tight cluster that is clearly separated from other clusters. We also remark that $S_q = 0$ if $\mathbf{h}_q \in \mathcal{D}_k$ is the only element in $\mathcal{D}_k$.

In Figure 5.1, we plot the sample excerpts and colour-code them blue for those from the auxiliary dataset, and red for those from the M4 evaluation dataset. We first observe that some of the M4 quantised time series, which are seen by our GUM neural network only at testing time, resemble many of the drawn auxiliary excerpts after quantisation, regardless of the real-valued measurements in their original scale. These previously unseen time series are thence able to fire similar neurons to those that auxiliary excerpts would activate. We also note that the excerpts in every cluster tend to group in the right edge of each subplot, i.e. every cluster contains time series with diverse features (such as degree of smoothness, number of optima and rate of change) that lead to a similar forecast.

Furthermore, we then map LSTM encoder activations into 2-D through Uniform Manifold Approximation and Projection for dimension reduction (UMAP) (McInnes, Healy, and Melville, 2018; McInnes, Healy, Saul, et al., 2018), as shown in Figure 5.2. We refer the reader to Section 4.2.2 for our review on UMAP. The criterion for picking a colour for each marker is based on the last time step's intra-cluster average, which is simply computed by taking the empirical mean of the ordinal bins at the last timestep across all time series excerpts of each cluster. These values are provided in the legend of Figure 5.2. The smooth colour degradation from left to right, with blue clusters grouping in the left edge and

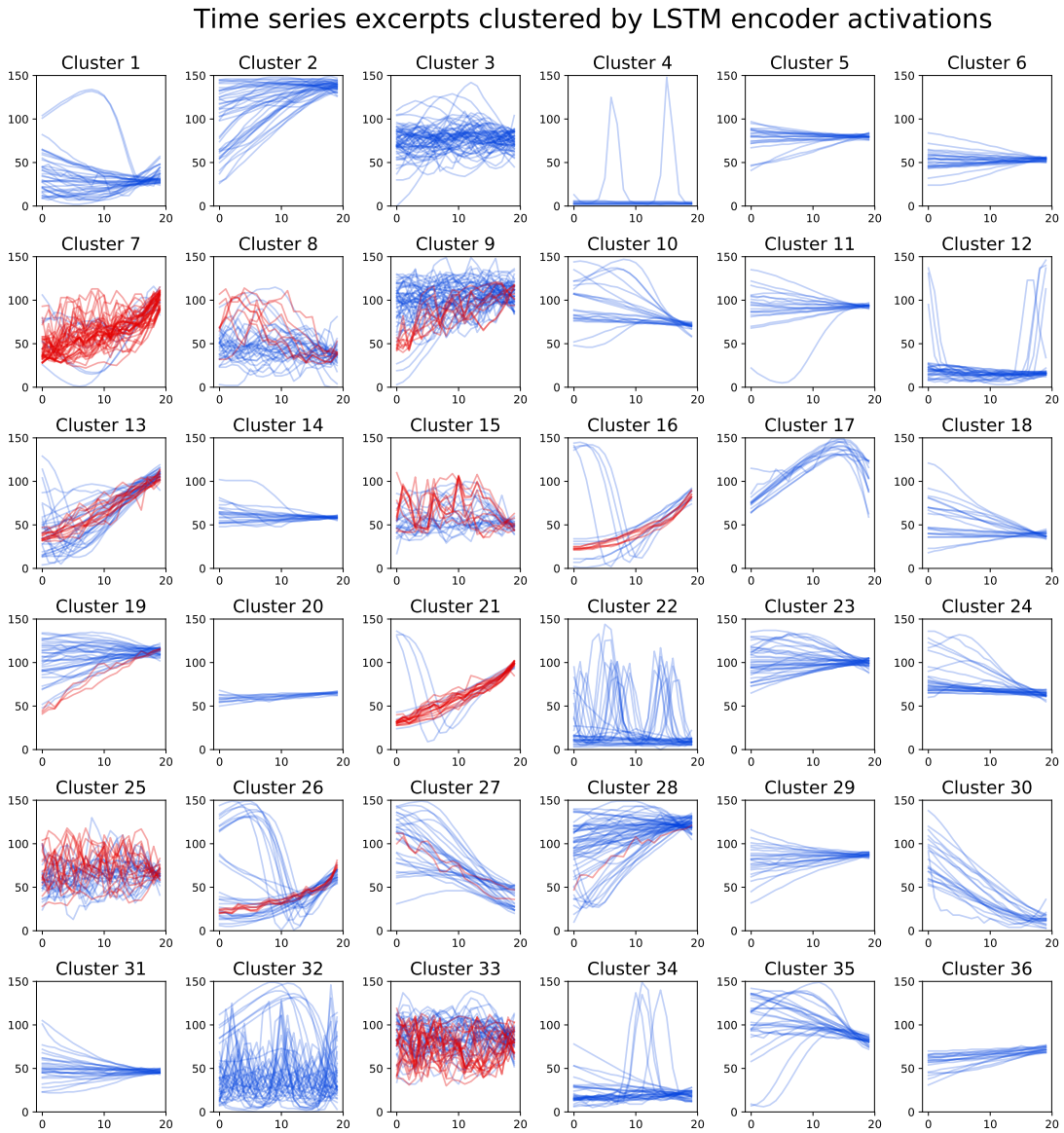


Figure 5.1: Sample excerpt groups resulting from clustering the LSTM encoder activations over both auxiliary (in blue) and M4 unseen time series (in red). After quantisation, unseen time series may trigger similar activations than other frames from the auxiliary dataset. This enables GUM to produce reliable forecasts even prior to seeing the M4 dataset.

red clusters in the right one, suggests that our model is learning an approximate ordering over the ordinal bins.

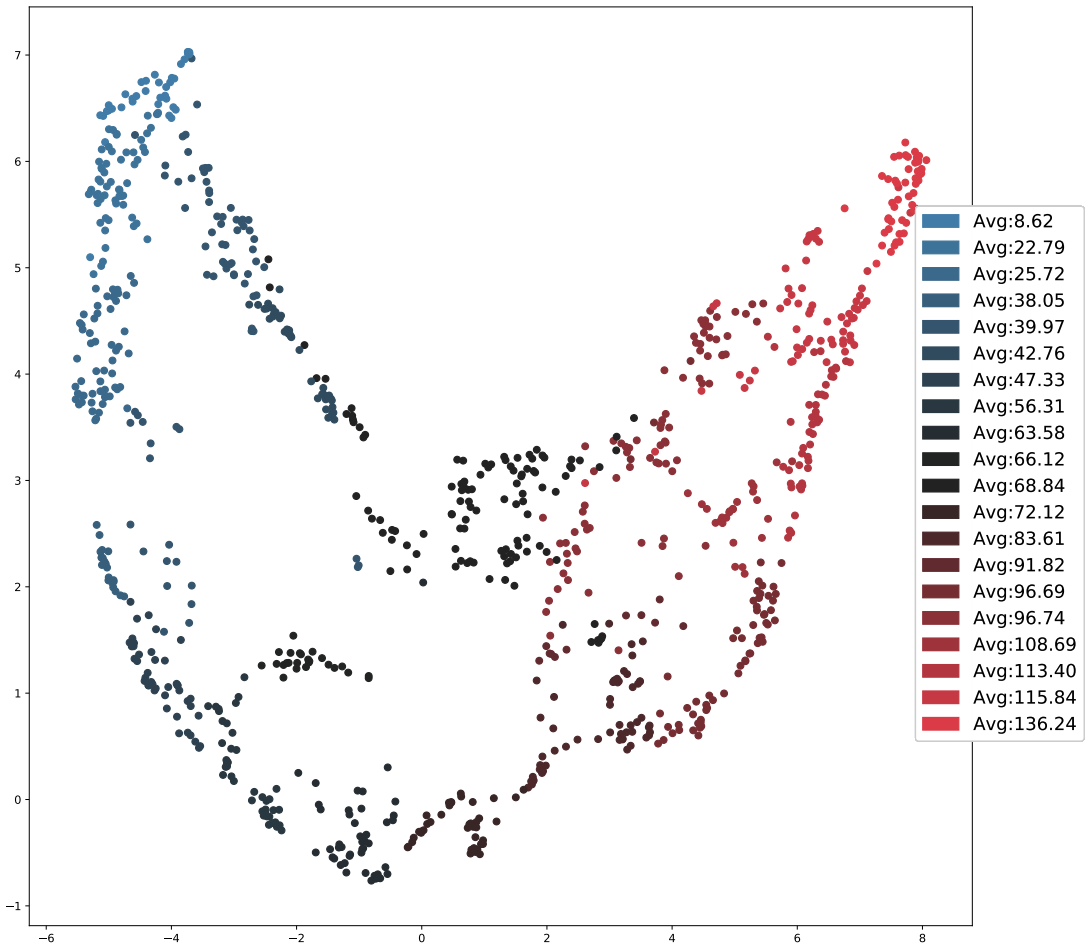


Figure 5.2: 2D UMAP representation of LSTM encoder activations, where each cluster is colour-coded by the average value of the last timestep across its excerpts. Bluer markers represent time series excerpts that converge to lower bins at the last timestep, and redder ones those that converge to higher bins. The smooth colour degradation from left to right suggests that our GUM approach is implicitly learning a quasi-ordering over ordinal bins.

5.5 Few-shot forecasting

In Section 5.3, we presented how to learn a shared representation across forecasting tasks, and showed how it can be transferred to infer predictive probability distributions over unseen time series without any further parameter inference. We enabled this transfer by learning a Memory-endowed Ordinal Regression Deep

neural network (MOrdReD) over a compilation dataset of 20 auxiliary time series drawn from various application domains. In Section 5.2.2 we then provided some empirical evidence of the properties of this shared feature representation.

We now turn our attention to further evaluations of the predictions offered by our approach. In this experiment, however, we look at time series where there is sufficient data to learn a variety of models. We compare these models with a version of our General Unified Model (GUM) that uses the parameters inferred over the auxiliary dataset $\mathcal{X}_{\text{aux}}$ as initial values, and then proceeds to perform further parameter updates over the new time series of interest.

5.5.1 Data

In a similar fashion to Section 5.3.1, we use the same 20 auxiliary datasets detailed in Appendix B to infer the initial state of our GUM neural network. In a second parameter inference phase, we look at 10 datasets drawn from the monthly-frequency segment of the M4 dataset (Makridakis, Spiliotis, and Assimakopoulos, 2018). We select all datasets that have no missing observations and that possess 15% of the minimum requirement of our original experiment from Section 3.5.1, i.e. 1500 measurements for model inference. This allows for exact and efficient inference of the baselines. Furthermore, all datasets were quantised over $M = 150$ bins as described in Section 3.1.

5.5.2 Baselines

In this experiment, we compare the following models: on one hand, the Autoregressive Gaussian Process (ARGP) and the state-space formulation of AR(p) models, both introduced in Section 3.5.2. In the case of the ARGP, we fit two instances with lookback $P = 50$, one with the Rational Quadratic kernel and one with the Matérn 5/2. As motivated in Section 3.5.2, these kernels can model a wide range of phenomena – those that are twice differentiable in the case of the Matérn 5/2,

and those that are smooth and vary at multiple length scales in the case of the Rational Quadratic. As for the linear-Gaussian AR(p) model, we optimise the lookback hyperparameter over $p \in \{3, 6, 12, 18\}$ for each dataset.

On the other hand, one further crucial comparison in this experiment is the one between our GUM MOrdReD and a *tabula rasa* MOrdReD, both of which use a lookback horizon of $P = 50$. The crucial difference is that GUM takes as initial parameters those that result from fitting over $\mathcal{X}_{\text{aux}}$, whereas those in the *tabula rasa* instance are initialised at random, following a Glorot uniform scheme. Furthermore, our GUM neural network is fit up to 50 epochs, where as the *tabula rasa* instance is fit for up to 100 epochs, twice as many. Both instances are equipped with early stopping and thus may not use up all available epochs. We additionally optimise over the number of hidden units $n_h \in \{64, 128, 256, 512\}$, dropout rate $\theta_{\text{drop}} \in \{0.25, 0.5\}$ and L_2 regularisation constant $\lambda \in \{10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$.

5.5.3 Predictive performance results

In this task, we forecast $P_h = 100$ measurements ahead from a lookback window of $P = 50$ measurements, as described in Section 5.5.2. Following the line of our evaluation in Section 5.3.4, we measure predictive accuracy, uncertainty quantification and forecast reliability through the Root Mean Squared Error (RMSE), negative log-likelihood (NLL) and QQDist metrics introduced in Section 3.5.4.

The baseline results per metric achieved after hyperparameter optimisation are presented in Tables 5.3, 5.4 and 5.5. We note that our GUM neural network consistently outperforms our baselines across all metrics. Crucially, GUM also outperforms the *tabula rasa* MOrdReD across all metrics, suggesting that transferring knowledge from prior tasks makes a substantial contribution to predictive performance in different metrics.

Dataset	**GUM**	**MOrdReD**	**ARGP**	**AR(p)**
monthly 1	**46.8032**	94.7700	481.9563	319.7136
monthly 2	**58.1034**	67.2500	534.3960	319.7124
monthly 3	**49.3639**	66.3570	510.7713	319.7151
monthly 4	**-71.7943**	119.5500	-31.4614	-6.9417
monthly 5	**-28.9168**	146.7181	54.4872	47.9708
monthly 6	**62.0636**	67.3323	116.7757	89.6751
monthly 7	148.9104	153.0409	172.1160	**139.8460**
monthly 8	**14.3592**	101.8852	47.6229	56.1512
monthly 9	**-9.4525**	143.6900	120.5116	70.4265
monthly 10	**-21.1821**	137.0737	31.8871	51.8744

Table 5.3: Negative log-likelihood (NLL) comparison across all 10 datasets for our 4 baselines. We observe that GUM consistently outperforms the rest of the baselines, and crucially, our own *tabula rasa* MOrdReD model.

Dataset	**GUM**	**MOrdReD**	**ARGP**	**AR(p)**
monthly 1	**0.6476**	0.9736	0.7104	0.7517
monthly 2	**0.6703**	0.7984	0.6972	0.7516
monthly 3	**0.6278**	0.8562	0.7090	0.7518
monthly 4	**0.0345**	0.9367	0.0480	0.0599
monthly 5	**0.0783**	0.1079	0.2747	0.2384
monthly 6	**0.2748**	0.3345	0.5945	0.4182
monthly 7	**0.9281**	1.0173	1.0757	0.9990
monthly 8	**0.1094**	0.1282	0.2343	0.2840
monthly 9	**0.0854**	0.6625	0.7080	0.5304
monthly 10	**0.0902**	0.4307	0.3594	0.4736

Table 5.4: Root Mean Squared Error (RMSE) comparison across all 10 datasets for our 4 baselines. We observe that GUM consistently outperforms the rest of the baselines, and crucially, our own *tabula rasa* MOrdReD model.

5.6 Concluding Remarks

In this chapter, we introduced a novel neural network-based General Unified Model (GUM) to infer probabilistic time series forecasts with scarce data. We showed how a shared representation over the space of time series forecasting

	GUM	**MOrdReD**	**ARGP**	**AR(p)**
monthly 1	0.0983	0.1284	0.0915	**0.0868**
monthly 2	0.0877	0.0995	0.0931	**0.0867**
monthly 3	0.0909	0.0367	0.0982	**0.0869**
monthly 4	0.0283	**0.0286**	0.0511	0.0554
monthly 5	**0.0066**	0.0526	0.056	0.0531
monthly 6	**0.0248**	0.0313	0.0485	0.0498
monthly 7	**0.0053**	0.0089	0.0186	0.0137
monthly 8	0.1188	**0.0309**	0.0526	0.0694
monthly 9	**0.0062**	0.0143	0.0551	0.0699
monthly 10	**0.0029**	0.0164	0.0115	0.0536

Table 5.5: QQDist metric results across all 10 datasets for our 4 baselines. Even though GUM achieves the best performance more times than the rest of the baselines, we observe a similar behaviour to the results from Section 3.5.5, where AR(p) models outperform the other baselines in a considerable amount of cases. This could be a result of the exact inference procedure available for this model, as described in Section 3.5.2.

tasks can be built through an ordinal regression approach, such as our Memory-endowed Ordinal Regression Deep neural network (MOrdReD). Moreover, we characterised a number of aspects of the inferred representation space by examining the neural activations of our model. For instance, we illustrated how class ordering is captured by this learnt embedding.

Furthermore, we evaluated the predictive performance of our GUM approach against state-of-the-art baselines in two different settings: zero-shot and few-shot forecasting. In the former, we demonstrated how accurate predictive distributions over unseen time series can be inferred without further GUM training after learning the shared representation space, which is particularly helpful for time series where there is insufficient data to perform any further parameter updates. We also provided insights on how these unseen time series can fire similar neural activations than those of certain frames observed in the auxiliary dataset, which enables our predictive inference phase.

Finally, we evaluated GUM in few-shot forecasting tasks. In this setting, we showed how it can be adapted to time series tasks where there exists some data to perform parameter updates. One crucial finding in this experiment was that our GUM neural network outperformed a MOrdReD model trained from scratch across all metrics and datasets. This key conclusion provides empirical evidence that GUM's transferred knowledge gives it a substantial advantage over *tabula rasa* models.

6 Conclusions

6.1 Summary of contributions

We now come to the conclusion of this thesis, where we introduced a novel recurrent neural network-based framework for time series forecasting. By recasting time series prediction as an ordinal regression task, we demonstrated how state-of-the-art Machine Learning methods can be employed to perform reliable and scalable long-term time series prediction in a general-purpose fashion. In this chapter, we briefly recapitulate our contributions and describe potential avenues for future work.

The core contribution of this thesis is the Memory-endowed Ordinal Regression Deep neural network (MOrdReD), introduced in Chapter 3. This ordinal regression approach is enabled by a quantisation scheme, which defines a discrete encoding scheme over time series measurements, effectively recasting time series forecasting as a symbolic sequence learning task. Such tasks have been most successfully dealt with in the NLP literature by Long Short-Term Memory (LSTM) models and extensions, such as the sequence-to-sequence model, introduced in Section 3.2. Furthermore, it has been shown how approximate predictive posterior distributions can be built from these models when they incorporate the dropout regularisation technique (Srivastava et al., 2014; Gal and Ghahramani,

2016a).

We then showcased the performance of our model in a large-scale comparison in Section 3.5, across 45 datasets drawn from a wide range of application domains, including meteorology, physiology, dynamical systems, astronomy, among others. MOrdReD's performance was compared against state-of-the-art baselines across several metrics to evaluate predictive accuracy and well-calibrated uncertainty quantification. Empirical evidence suggested that MOrdReD was able to achieve or surpass the performance of other state-of-the-art methods at these tasks. Crucially, our framework is also accompanied by an open source implementation, available at `https://github.com/bperezorozco/ordinal_tsf` and fully documented in Appendix C.

In Chapter 4, we motivated extensions that took our forecasting framework beyond the univariate case. In the first part of this chapter, we dealt with deriving densities that quantified the occurrence of events of interest in time series forecasts, focusing on the case of predicting the occurrence of optima. This framework assumed access to labelled instances of such events, which may be unrealistic in many scenarios. This motivated Section 4.2, where we introduced a prototypical motif extraction framework based on analysing clusters over MOrdReD's feature space. Furthermore, we also demonstrated how the encoding of characteristic time series aspects, such as quasi-periodicity and chaos, can be visualised in this feature representation.

Next, in the second part of Chapter 4, we introduced a multi-output extension of our framework. This extension was able to infer full joint predictive distributions by learning a joint feature representation over the time series channels. We showed that our model offers accurate and reliable forecasts when the time series has been corrupted by moderate levels of additive white noise.

Finally, in Chapter 5, we introduced a General Unified Model (GUM) for time series forecasting. This approach extends MOrdReD by inferring a shared repre-

sentation embedding over an auxiliary set of quantised time series, which in turn enables information sharing across different forecasting tasks. We showed how such inductive transfer enables the inference of accurate predictive distributions for time series where only a few lookback measurements are available, even prior to performing any further model parameter updates, i.e. in a *zero-shot* fashion.

6.2 Predictive horizon: future work

In this section, we briefly describe some avenues for future research. Firstly, in Chapter 3, we introduced our Memory-endowed Ordinal Regression Deep neural network (MOrdReD) model. In order to perform ordinal regression, in Section 3.1 we described an encoding scheme that uses equally-sized bins to quantise time series. Future work includes incorporating information-theoretic approaches to estimate both, the optimal number of bins M and their thresholds, which could be of variable width (MacKay, 2003; Bose, 2008).

Such an analysis must also take into consideration the trade-off between representation accuracy and computational cost. In particular, let us recall that time series quantisation introduces a distortion error that vanishes as the number of quantisation bins grows. However, MOrdReD also relies on LSTM layers, whose parameters grow quadratically on the number of bins. Not only is this problematic in terms of the availability of memory resources, but the growth in model parameters without an increase in data points may also prevent the model from generalising, and therefore actually attain worse predictive performance than with less quantisation bins.

One possible research avenue that spans from this discussion is the development of strategies to achieve the best representation accuracy after fixing a number of bins, for example by learning (uneven) discretisation bin thresholds. A possible approach to achieve this could be constructing these thresholds from

the empirical cumulative distribution of the data, for instance, by discretising observations based on which quantile they belong in.

Then, in Section 3.5, we performed a large-scale experiment to compare time series forecasting models. Given the relevance of this task, other recurrent neural network approaches have been developed in parallel to our own contribution, such as Amazon's Deep AR (Salinas, Flunkert, and Gasthaus, 2017), Facebook's Prophet (Taylor and Letham, 2018) and Neural Processes (Garnelo et al., 2018), amongst others (Laptev et al., 2017). This opens up an opportunity for further evaluation with respect to our experiments from Section 3.5.

When we performed these experiments, we highlighted how MOrdReD achieves long-term forecasting with comparatively less expert craftsmanship with respect to methods such as the Gaussian Process (GP). For our model, however, we still needed to tune hyperparameters such as the number of hidden units, L_2 regularisation and dropout rate, which we did through an extensive grid search. Not only is this computationally expensive, but also incorporating a different hyperparameter optimisation routine, such as Bayesian optimisation, could likely have a positive impact in predictive performance and further automate steps in our framework.

Next, in Chapter 4, we explored extensions to go beyond univariate time series forecasting. For example, we showed how prototypical time series motifs can be derived by performing a spectral cluster analysis of our inferred feature embeddings, and suggested these patterns could be used in applications such as indexing huge datasets and constructing critical event densities. The motifs we constructed in this thesis, however, cannot be readily manipulated by the user. We propose further user knowledge can be incorporated through pairwise-constrained clustering approaches, such as the one proposed by Lu and Carreira-Perpinan (2008). In the case of spectral clustering, for instance, such knowledge can be encoded in the manner of constructing the pairwise affinity matrix. For example, let us

look at the case of electrocardiogams. We could envision how discovered motifs could be refined by imposing constraints on the learnt embedding so that motif boundaries coincide with cardiac waveforms of interest, such as those described in a QRS complex.

In a similar fashion, this thesis leaves for future work a deeper analysis of the relationship between the number of quantisation bins and the motifs discovered by our model. In particular, we consider that a small number of bins may fail to capture the richness of motif deformations discussed in Section 4.2. As discussed in that section, one of the strengths of our approach is that the extracted motifs form a distribution whose instances vary in a manner akin to operations such as opening and closing from the morphological filtering literature. Since we wish to retain this advantage, but acknowledging that motif extraction can only happen after a potentially expensive training of a MOrdReD model, we would then benefit from having a strategy to choose the number of bins that will enable rich motif learning prior to training.

Let us note next that, when we motivated this motif discovery extension, we mentioned that these patterns could also serve to form the lexicon of a formal language. Transition diagrams can then readily be built so as to characterise the dynamics of a time series. We could then picture how such diagrams could serve as a basis to construct a metric to evaluate predictions structurally, by comparing the resulting diagrams over both real data and predictions. Such a metric would be more robust to misalignment errors than others, such as the Mean Squared Error (MSE) and the like. In a similar manner, these diagrams could then be used to generate synthetic data from the same distribution as the data.

We now move on to Section 4.3, where we proposed a multivariate prediction framework whose encoding scheme consisted in quantising each time series channel separately. This was motivated by remarking that a brute-force approach, such as defining a fine-grained hyper-grid of ordinal bins, would have exponential

complexity in memory. We argue that this can be further explored by adding a flexible quantisation approach that takes into account the sparsity of observed measurements.

To illustrate, consider a time series with only 3 dimensions and $10,000$ available observations. A hyper-grid that uses $M = 100$ bins per dimension would induce $100^3 = 1,000,000$ bins, which are far more than the number of observations available. Such an approach could also consider, for instance, that different channels in the time series could have a different optimal number of bins, e.g. that some channels may require more resolution than others, especially in the case of mixed-type time series, where some channels may be sampled from the real numbers, and others may be discrete variables over a finite set. MultiMOrdReD could handle this case as each channel corresponds to a different input layer with a potentially different number of input units.

In the case of continuous multivariate time series, one further possibility to explore is to learn quantisation bins through clustering over a maximum budget of bins. For example, in low-dimensional settings, efficient methods such as Lloyd's algorithm are able to learn Voronoi tesselations, which can then be used as codebooks. Likewise, probabilistic clustering can be performed through Variational Bayesian Gaussian Mixture Models (VBGMMs), which also provide a principled way to choose the number of clusters. This would effectively recast multivariate prediction as its univariate case, where we would now forecast the next cluster id, in a related fashion to models such as the Hidden Markov Model (HMM). We could then readily use our methods from Chapter 3.

Subsequently, in Chapter 5 we introduced our General Unified Model (GUM) to perform zero-shot and few-shot forecasting from short time series, and we showed how to learn a shared representation over the space of quantised time series so as to transfer knowledge across forecasting tasks. This space can then be used to infer accurate and reliable predictive distributions for time series with

scarce data. At present, however, this approach is highly sensitive to the associated time series quantisation scheme. This motivated the linear heuristic we provided in Section 5.2 to estimate quantisation bounds. The estimation of such limits can be improved, for instance, by relating it to the inference of Lipschitz constants, as shown in (Calliess, 2015).

Another aspect to investigate further is how the number and type of auxiliary time series used to fit GUM affect its performance. Whereas in Chapter 5 our choosing criterion relied on diversifying attributes such as domain application, roughness and length scales of a fixed number of auxiliary time series, a more formal analysis of how these attributes relate to GUM's performance and complexity remains to be done. Intuitively, adding more auxiliary time series would likely improve performance on unseen time series, but could also introduce data redundancy if different auxiliary time series have similar behaviours.

One possible manner to avoid such redundancy in the auxiliary datasets is to use our motif extraction method introduced in Section 4.2. This would aid to compare, find and discard similar patterns across auxiliary time series, potentially reducing the required model complexity for GUM to generalise.

How the auxiliary time series affect GUM's performance also depends on the attributes of the unseen time series themselves. In our experiments, we assumed we had no information at hand about the test time series; but, if there is any such information, then we hypothesise that choosing auxiliary time series that fit with these criteria (i.e. auxiliary time series whose distribution resembles more closely what we know about the distribution of the unseen time series) could yield better performance.

Finally, we can also extend the discussion on MOrdReD's hyperparameters to our GUM framework. At present, we are able to offer pre-trained models for specific settings of hyperparameters, all of which were inferred separately from scratch. Future work includes inferring models with different hyperparameter

settings, without having to train them from scratch, i.e. sharing information across architectures. One possible way to do this, for instance, is by inferring a sufficiently large network and then pruning it, as described by Hanson and Pratt (1989) and Han et al. (2015).

A Acronyms

AD Automatic Differentiation

AdaGrad Adaptive Gradient

ARGP Autoregressive Gaussian Process

BiLSTM Bidirectional LSTM

cdf cumulative density function

CEC Constant Error Carousel

CV Computer Vision

ELBO evidence lower bound

FC fully-connected

GLM Generalised Linear Model

GMM Gaussian Mixture Model

GP Gaussian Process

GPR Gaussian Process Regression

GPSSM Gaussian Process State Space Models

GPU Graphics Processing Unit

GRU Gated Recurrent Unit

GUM General Unified Model

HMM Hidden Markov Model

ICM Intrinsic Model of Coregionalisation

IndyMOrdReD Independent Memory-endowed Ordinal Regression Deep neural networks

KDE Kernel Density Estimation

KL Kullback-Leibler

LMC Linear Model of Coregionalisation

LPC Linear Predictive Coding

LSTM Long Short-Term Memory

MAP Maximum A Posteriori

MCMC Markov Chain Monte Carlo

MFCC Mel-Frequency Cepstral Coefficients

ML machine learning

MLE Maximum Likelihood Estimation

MLP Multilayer Perceptron

MOrdReD Memory-endowed Ordinal Regression Deep neural network

MultiMOrdReD Multivariate Memory-endowed Ordinal Regression Deep neural network

MSE Mean Squared Error

MTL Multitask Learning

NLL negative log-likelihood

NLP Natural Language Processing

pdf probability density function

pmf probability mass function

ReLU Rectified Linear Unit

RGP Recurrent Gaussian Process

RMSE Root Mean Squared Error

RNN Recurrent Neural Network

RQ Rational Quadratic

SE Squared Exponential

Seq-to-Seq sequence-to-sequence

SGD Stochastic Gradient Descent

SLFM Semiparametric Latent Factor Model

SMAPE Symmetric Mean Absolute Percentage Error

SNR signal-to-noise ratio

TPU Tensor Processing Unit

TSF time series forecasting

t-SNE t-distributed Stochastic Neighbour Embedding

UMAP Uniform Manifold Approximation and Projection for dimension reduction

VBGMM Variational Bayesian Gaussian Mixture Model

VI Variational Inference

B

Dataset description

In this supplementary material we provide a list of the 45 datasets used throughout this thesis. Those marked with an (*) were further used for the event occurrence forecasting task and those marked with a † were used as auxiliary datasets for our General Unified Model (GUM) experiments in Chapter 5. Unless otherwise stated, these were all drawn from the compilation made by Fulcher, Little, and Jones (2013). Finally, in Section B.1, we enlist the datasets we used for our multivariate prediction experiment from Section 4.3.3.

1. † {MACKEY} Mackey-Glass chaotic attractor, synthesised by the authors according to the system:

$$x_{t+1} = (1-b)x_t + a\frac{x_{t-\tau}}{1+x_{t-\tau}^n} \tag{B.1}$$

   with $a = 0.2, b = 0.1, \tau = 17, n = 10$ and discarding the first 1,000 burnout samples.

2. (*),† {ECG} a recording of the electrical activity of the human heart (Rezek and Roberts, 1998), which highlights forecasting of a quasi-periodic, complex-shaped signal where accurate timing is essential;

3. (*),† {AIRFLOW} a recording of the breathing activity of a human(Rezek

and Roberts, 1998), which highlights forecasting of a quasi-periodic signal where each quasi-cycle has a relatively long time period;

4. (*),† {TIDE} data taken from the Sotonmet (Bramble Bank, 2018) environmental sensors that record tide heights in an off-shore weather station.

5. {AS_s3.2_} birdsong excerpt from the Macaulay Animal Sounds Library,

6. (*),† {CM_air.s} air temperature time series taken by different sensors in the Eurasia region between 1948 and 2007,

7. † {CM_air.2} a second air temperature time series taken by different sensors in the Eurasia region between 1948 and 2007,

8. {CM_prate} precipitation rate time series taken by different sensors in the Eurasia region between 1948 and 2007,

9. (*), † {CM_slp19} a sea level pressure time series taken by different sensors in the Eurasia region between 1948 and 2007,

10. † {CM_SLP.2} a second sea level pressure time series taken by different sensors in the Eurasia region between 1948 and 2007,

11. (*) {CM_rhum1} relative humidity time series taken by different sensors in the Eurasia region between 1948 and 2007,

12. {CM_lwtla} Lamb/Jenkins weather type series measured from 1861 to 1997,

13. † {EMexptqp} Quasi-periodic output of Eric Weeks' Annulus experiment,

14. {EM_henon} Henon map

$$x' = \quad a + by - x^2 \tag{B.2}$$

$$y' = \qquad x \tag{B.3}$$

with $a = 1.4, b = 0.3$

15. † {EMlorenz} Lorenz map

$$x' = \sigma(y - x) \tag{B.4}$$
$$y' = rx - y - xz \tag{B.5}$$
$$z' = xy - bz \tag{B.6}$$

with $\sigma = 10, r = 28, b = 8/3$,

16. † {EM_rossl} Rössler attractor

$$x' = -z - y \tag{B.7}$$
$$y' = x + ay \tag{B.8}$$
$$z' = b + z(x - c) \tag{B.9}$$

with $a = 0.15, b = 0.20, c = 10.0$,

17. {FI_yahoo} Log returns of GPSC stock chart,

18. † {FL_ACT_L} $z-$channel of the ACT attractor:

$$x' = \alpha(x - y) \tag{B.10}$$
$$y' = -4\alpha y + xz + \mu x^3 \tag{B.11}$$
$$z' = -\delta\alpha z + xy + \beta z^2 \tag{B.12}$$

computed with $\alpha = 1.8, \beta = -0.07, \delta = 1.5, \mu = 0.02$,

19. {FL_chen_} $x-$channel of Chen's system

$$x' = a(y - x) \tag{B.13}$$
$$y' = (c - a)x - xz + cy \tag{B.14}$$
$$z' = xy - bz \tag{B.15}$$

computed with $a = 35, b = 3, c = 28$,

20. † {FL_dblsc} $y-$channel of the double-scroll system

$$x' = y \tag{B.16}$$

$$y' = z \tag{B.17}$$

$$z' = -a[z + y + x - \text{sgn}(x)] \tag{B.18}$$

computed with $a = 0.8$ and initial conditions $x_0 = 0.01, y_0 = 0.01, z_0 = 0$,

21. † {FL_hadley} $x-$channel of the Hadley circulation system

$$x' = -y^2 - z^2 - ax + aF \tag{B.19}$$

$$y' = xy - bxz - y + G \tag{B.20}$$

$$z' = bxy + xz - z \tag{B.21}$$

computed with $a = 0.25, b = 4, F = 8, G = 1$ and initial conditions $x_0 = 0, y_0 = 0, z_0 = 1.3$,

22. {FL_labyr} $y-$channel of the Labyrinth Chaos system

$$x' = \sin(y) \tag{B.22}$$

$$y' = -\sin(z) \tag{B.23}$$

$$z' = \sin(x) \tag{B.24}$$

computed with initial conditions $x_0 = 0.1, y_0 = 0, z_0 = 0$,

23. † {FL_moore} $z-$channel of the Moore-Spiegel oscillator

$$x' = y \tag{B.25}$$
$$y' = z \tag{B.26}$$
$$z' = -z - (T - R + Rx^2)y - Tx \tag{B.27}$$

computed with $T = 6, R = 20$,

24. {FL_noseh} $z-$channel of the Nosé-Hoover oscillator

$$x' = y \tag{B.28}$$
$$y' = -x + yz \tag{B.29}$$
$$z' = a - y^2 \tag{B.30}$$

computed with $a = 1$ and initial conditions $x_0 = 0, y_0 = 5, z_0 = 0$,

25. † {FL_ruckl} $z-$channel of the Rucklidge attractor

$$x' = -\kappa x + \lambda y - yz \tag{B.31}$$
$$y' = x \tag{B.32}$$
$$z' = -z + y^2 \tag{B.33}$$

computed with $\kappa = 2, \lambda = 6.7$ and initial conditions $x_0 = 1, y_0 = 0, z_0 = 4.5$,

26. † {FL_simpq} $y-$channel of the simplest quadratic flow

$$x' = y \tag{B.34}$$
$$y' = z \tag{B.35}$$
$$z' = -az + y^2 - x \tag{B.36}$$

computed with $a = 2.028$ and initial conditions $x_0 = 0.9, y_0 = 0, z_0 = 0.5$,

27. † {FL_thoma} $y-$channel of Thomas cyclically symmetric attractor

$$x' = -bx + \sin(y) \tag{B.37}$$

$$y' = -by + \sin(z) \tag{B.38}$$

$$z' - bz + \sin(x) \tag{B.39}$$

computed with $b = 0.18$ and initial conditions $x_0 = 0.1, y_0 = 0, z_0 = 0$,

28. {FL_windm} $y-$channel of the WINDMI attractor

$$x' = y \tag{B.40}$$

$$y' = z \tag{B.41}$$

$$z' = -az - y + b - \exp(x) \tag{B.42}$$

for $a = 0.7, b = 2.5$ and initial conditions $x_0 = 0, y_0 = 0.8, z_0 = 0$,

29. {K_standa} Log returns of the Standard & Poor index,

30. {MC_inttr} Internet traffic data from an ISP, provided by the Time Series Data Library,

31. {MP_Lozi_} Lozi map $x_{t+1} = 1 - a|x_t| + bx_{n-1}$ with $a = 1.7, b = 0.5$ and initial conditions $x_1 = -0.1, x_0 = 0.15$,

32. {MP_freit} Freitas' stochastic sine map $x_{t+1} = \mu \sin(x_t) + Y_t \eta_t$, with parameters $\mu = 2.4, b = 3, q = 0.2, \eta_t \sim \text{Uniform}(-b, b), Y_t \sim \text{Bernoulli}(q)$,

33. {MP_logis} Logistic map $x_{t+1} = Ax_t(1 - x_t)$ with $A = 3.2$ and initial condition $x_0 = 0.91$,

34. {MUS_Si_l} a music excerpt from B. Fulcher's personal collection, entitled *Si loin de vous*,

35. {MUS.3_78} a second music excerpt from B. Fulcher's personal collection, entitled *3*,
36. {SFX_mach} Sound Jay Mach Electric Drill sound effect,
37. † {SF_Acont} Santa Fe laser generated data excerpt,
38. {SF_B1_1} Santa Fe heart rate data excerpt,
39. {SF_D1} Santa Fe synthetically generated snippet,
40. {SL_perci} StatLib demeaned water level measurements,
41. {SPIDR_hp} Hemispheric Power Index excerpt provided by the Space Physics Interactive Data Resource,
42. † {SY_AR2_T} Timmer nonstationary autoregressive process with $\tau = 20, T_{mean} = 20, T_{mod} = 20, \sigma = 1, \mathcal{M}_T = 5, \eta = 1000$,
43. {SY_NLAR2} Faes nonlinear autoregressive process with $a_1 = 3.6, a_2 = 0.8$,
44. {TSAR_eqe} Earthquake and explosion seismic series provided in Stoffer's *Time Series Analysis and its applications, with R examples.*
45. {TXT_slc_} Project Gutenberg excerpt of Dickens' *Oliver Twist.*

B.1 Multivariate datasets

In this Section, we enlist all 8 datasets used in our experiments from Section 4.3.3.

1. (LORENZ-ORIG) Lorenz chaotic attractor (no noise):

$$x' = \quad \sigma(y - x) \tag{B.43}$$

$$y' = \quad rx - y - xz \tag{B.44}$$

$$z' = \quad xy - bz \tag{B.45}$$

2. (LORENZ-LOW) Lorenz chaotic attractor (white noise with variance $\sigma^2 = 1.0$)

3. (LORENZ-MOD) Lorenz chaotic attractor (white noise with variance $\sigma^2 = 5.0$)

4. (LORENZ-HIGH) Lorenz chaotic attractor (white noise with variance $\sigma^2 = 7.5$)

5. (MACKEY-3D) 3-D Mackey-Glass chaotic attractor. Obtained by stacking 3 lagged copies of the Mackey-Glass attractor, with lag= 10:

$$x_{t+1} = (1 - b)x_t + a\frac{x_{t-\tau}}{1 + x_{t-\tau}^n} \tag{B.46}$$

6. (ECG-AIR) Electrocardiogram and airflow recording simultaneously from the same individual. Corresponds to datasets ECG and AIR_FLOW in the previous list.

7. (TEMP-1) Three recordings of air temperature, sea level pressure and relative humidity, identified as 67_9 in (Fulcher, Little, and Jones, 2013).

8. (TEMP-2) Three recordings of air temperature, sea level pressure and relative humidity, identified as 56_6 in (Fulcher, Little, and Jones, 2013).

C

Python library documentation: Ordinal time series forecasting

We offer a time series forecasting open source library, available at `https://github.com/bperezorozco/ordinal_tsf`, that is implemented in an object-oriented fashion and following good sofwtare engineering practices wherever possible, such as the inclusion of design patterns (Gamma, 1995). Here we offer an implementation of our framework, in addition to interfaces for methods drawn from other libraries, such as GPFlow (Matthews, Alexander, et al., 2017).

This library an object-oriented approach for time series forecasting methods by encapsulating them according to the Strategy design pattern, as shown in Figure C.1. This enables users to include other methods under a unified environment. Dataset preprocessing steps, such as standardisation to zero-mean and unit-variance, in addition to quantisation methods as described in Section 3.1, are also designed in an object-oriented fashion, corresponding to the Command design pattern. UML diagrams of the core classes of our library are provided below. Finally, we then provide a copy of the first tutorial of our library, also available at `https://github.com/bperezorozco/ordinal_tsf`.

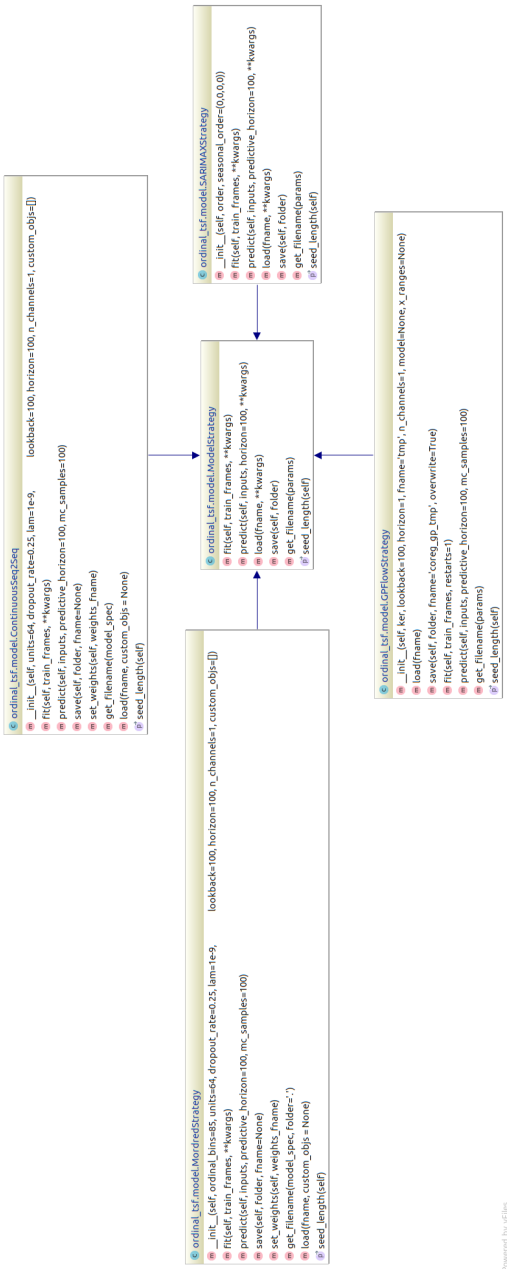


Figure C.1: UML diagram of Ordinal TSF's available models.

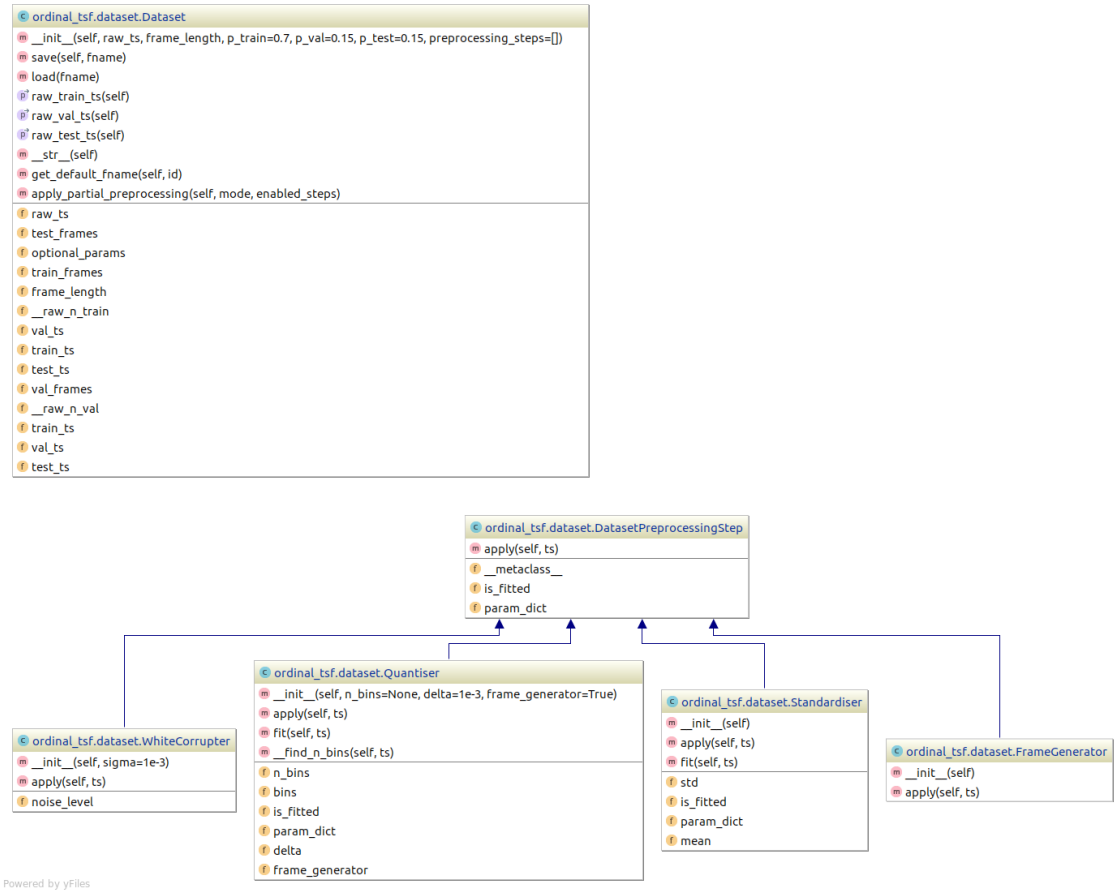


Figure C.2: UML diagram of Ordinal TSF's dataset structure and dataset preprocessing routines.

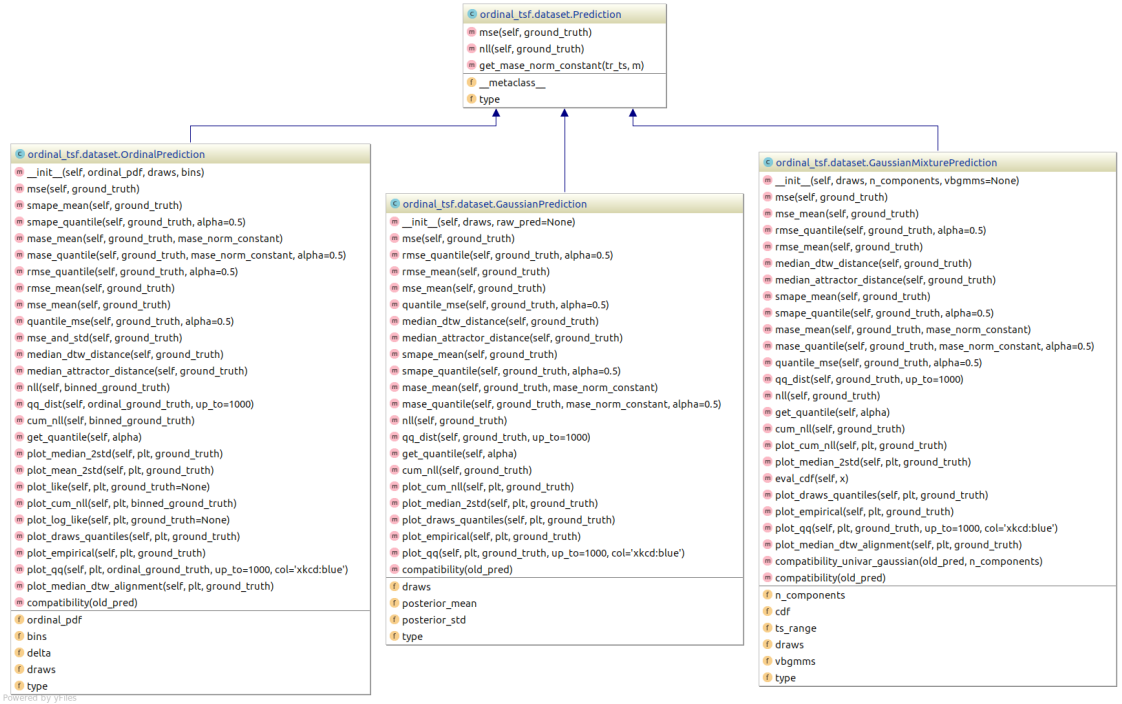


Figure C.3: UML diagram of Ordinal TSF's available prediction types.

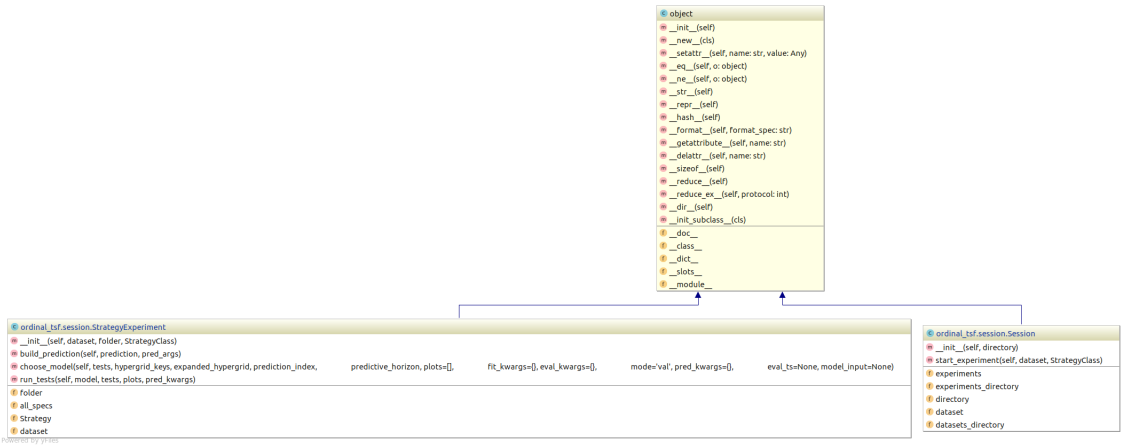


Figure C.4: UML diagram of Ordinal TSF's session management structures.

Mordred Example

October 8, 2019

1 MOrdReD: Ordinal autoregression with recurrent neural networks

1.0.1 Introduction

This Python library accompanies our work. MOrdReD enables time series forecasting in an autoregressive and ordinal fashion. This simply means two things: that a new sample x_{t+1} is forecasted by looking at previous observations $x_t, x_{t-1}..., x_{t-p}$ for some lookback p.

Our framework provides an implementation of our ordinal autoregression framework (via Keras) described in the paper above; however, it also provides a flexible and amicable interface to set up time series forecasting tasks (parameter optimisation, model selection, model evaluation, long-term prediction, plotting) with either our prediction framework, or other well-established techniques, such as Gaussian Processes (via GPy) or Dynamic AR models (via statsmodels).

The following notebook takes us through the most basic example: loading and preparing a time series dataset, choosing a model, and evaluating it.

In [5]:
```
%matplotlib inline

from ordinal_tsf.dataset import *
from ordinal_tsf.model import *
from ordinal_tsf.session import *
import os
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
os.environ["CUDA_VISIBLE_DEVICES"]="3"

plt.style.use('seaborn-ticks')
matplotlib.rcParams['figure.figsize'] = (20.0, 16.0)
matplotlib.rcParams['font.serif'] = 'times'
labelsize = 26
titlesize=32
params= {
    'text.usetex':True, "font.family": "serif", "font.serif": ["Computer Modern"],
    'font.family':'serif', 'font.serif': ["Times", "Times New Roman"],
    'legend.fontsize':labelsize, 'axes.labelsize':labelsize, 'axes.titlesize':titlesize,
    'xtick.labelsize' :labelsize, 'ytick.labelsize' : labelsize
}
matplotlib.rcParams.update(params)
```

1.1 Starting a session

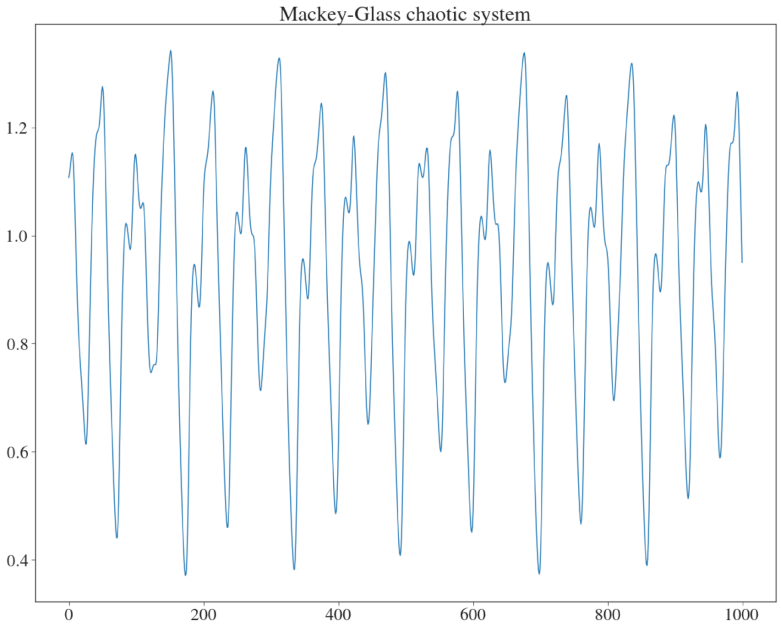
All interactions with this library are managed by a Session object. This guarantees that other objects such as preprocessed datasets, models, predictions and plots are kept in an orderly way. This also makes sure the project's structure is cohererent and queryable by different modules.

In practice, Sessions are meant to group experiments on the same time series, regardless of whether they are ordinal or real-valued. The id provided as a constructor argument is the name of the folder that will store all of the session's objects.

For this introductory example, we will work with the Mackey Glass chaotic system.

```
In [9]: sess = Session('mg')
        x = pd.read_feather('mg.feather').values
        plt.plot(x[:1000])
        plt.title('Mackey-Glass chaotic system')
        plt.show()
```

```
Opening session: mg/...
```



1.2 Preparing the dataset

All Ordinal TSF models work with the (autoregressive) Dataset interface. This provides a transparent way of framing a time series, as well as applying preprocessing transformations such as: standardising, adding white noise, quantising (binning/discretising), etc. Users can also add any additional data preprocessing

steps they may need by providing an object whose class inherits from DatasetPreprocessingStep, as defined in the .dataset package.

For this example, we will only make the time series zero-mean, unit-variance and quantise it in 85 equally-distributed bins over the time series' range. The first 70% of the time series' samples will then be used as the training set for our model, and the validation and test sets are made by equally splitting the remaining 30%.

```
In [10]: lookback = 101
         horizon = 100
         ordinal_bins = 85

         stand = Standardiser()
         quant = Quantiser(ordinal_bins)

         dataset = Dataset(x, lookback + horizon,
                           p_val=0.15, p_test=0.15,
                           preprocessing_steps=[stand, quant])

         print('Time series framed into {} training frames with {},{} \
               -feature observations at each timestep\n'.format(*dataset.train_frames.shape))

         print dataset
```

```
Time series framed into 14559 training frames with 201 85-feature observations at each timestep

Dataset with properties:
zero_mean_unit_var:True
bin_delta:0.0494988267428
ts_std:0.234664381011
bins:[-2.37251619 -2.32301736 -2.27351853 -2.22401971 -2.17452088 -2.12502205
 -2.07552323 -2.0260244  -1.97652557 -1.92702675 -1.87752792 -1.82802909
 -1.77853027 -1.72903144 -1.67953261 -1.63003379 -1.58053496 -1.53103613
 -1.48153731 -1.43203848 -1.38253965 -1.33304082 -1.283542   -1.23404317
 -1.18454434 -1.13504552 -1.08554669 -1.03604786 -0.98654904 -0.93705021
 -0.88755138 -0.83805256 -0.78855373 -0.7390549  -0.68955608 -0.64005725
 -0.59055842 -0.5410596  -0.49156077 -0.44206194 -0.39256312 -0.34306429
 -0.29356546 -0.24406664 -0.19456781 -0.14506898 -0.09557016 -0.04607133
  0.0034275   0.05292632  0.10242515  0.15192398  0.2014228   0.25092163
  0.30042046  0.34991928  0.39941811  0.44891694  0.49841576  0.54791459
  0.59741342  0.64691224  0.69641107  0.7459099   0.79540873  0.84490755
  0.89440638  0.94390521  0.99340403  1.04290286  1.09240169  1.14190051
  1.19139934  1.24089817  1.29039699  1.33989582  1.38939465  1.43889347
  1.4883923   1.53789113  1.58738995  1.63688878  1.68638761  1.73588643
  1.78538526]
is_ordinal:True
ts_mean:0.925427526852
```

1.3 Starting an experiment

```
In [11]: experiment = sess.start_experiment(dataset, MordredStrategy)
```

1.4 Model creation and fitting

Our library offers seamless integration with many Machine Learning libraries (Keras, Tensorflow, GPy, Scikit-Learn) to perform autoregressive (ordinal or real-valued) time series forecasting. This is achieved via

the Strategy wrapper, and custom ML models can be added to the library straightforwardly by implementing this interface.

In this example, we setup an autoregressive ordinal neural network. This is a sequence-to-sequence architecture assembled by the wrapper using Keras layers. Any additional arguments to be passed on to the Keras fit method should be given via a dictionary.

```
In [ ]: model = MordredStrategy(ordinal_bins=ordinal_bins, units=75)

        train_args = {'epochs':50, 'batch_size':256, 'validation_split':0.15}

        model.fit(dataset.train_frames, **train_args)
```

1.5 Forecasting

All model strategies must implement the predict method, but the return object will be different, depending on the dataset's representation (is it real-valued or ordinal? is it mono- or multi-channel?)

The session's

```
In [27]: first_prediction_index = 121
         seed_start = first_prediction_index - model.seed_length
         validation_predictive_horizon = 300
         val_seed = dataset.val_ts[np.newaxis, seed_start:first_prediction_index]
         model_output = model.predict(val_seed,
                                      predictive_horizon=validation_predictive_horizon,
                                      mc_samples=20)
         prediction = experiment.build_prediction(model_output)
```

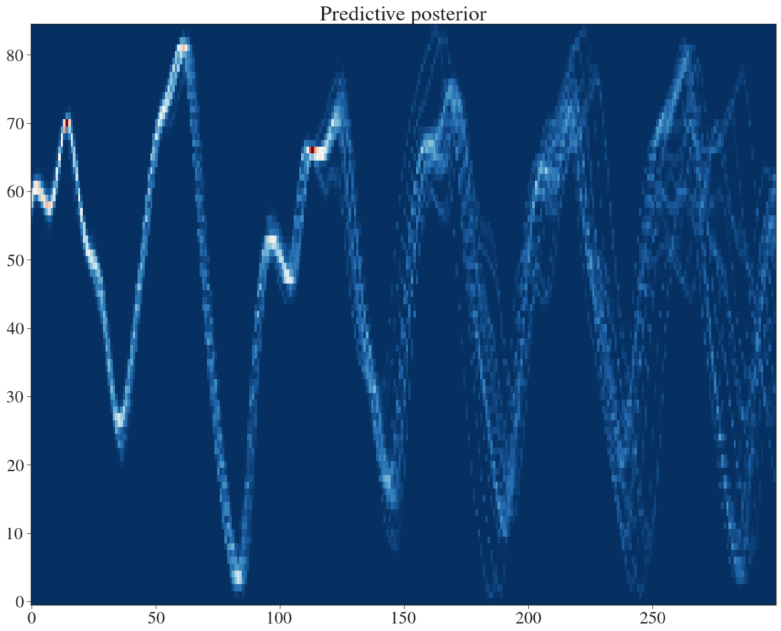
1.6 Test design

```
In [28]: selector = Selector(first_prediction_index, validation_predictive_horizon)
         continuous_ground_truth = dataset.apply_partial_preprocessing('val',
                                                                        [selector, stand])
         ordinal_ground_truth = dataset.apply_partial_preprocessing('val',
                                                                    [selector, stand, quant])

         validation_tests = [TestDefinition('mse', continuous_ground_truth),
                             TestDefinition('nll', ordinal_ground_truth),
                             TestDefinition('cum_nll', ordinal_ground_truth)]

         validation_plots = {'plot_median_2std': {'ground_truth':continuous_ground_truth},
                             'plot_cum_nll': {'binned_ground_truth': ordinal_ground_truth},
                             'plot_like': {}}
```

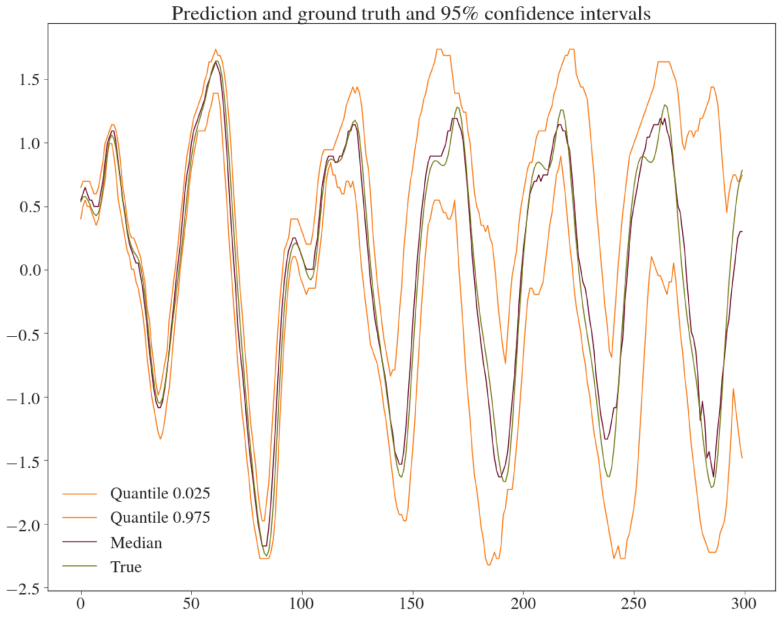
1.7 Prediction analysis

```
In [29]: print 'MSE: {} +- {}'.format(*prediction.mse_and_std(continuous_ground_truth))
         print 'NLL: {}'.format(prediction.nll(ordinal_ground_truth))

         prediction.plot_like(plt)
         plt.title('Predictive posterior')
         plt.show()

         prediction.plot_median_2std(plt, continuous_ground_truth)
         plt.title('Prediction and ground truth and 95\% confidence intervals')
         plt.show()
```

```
MSE: 0.198050307877 +- 0.0262471117184
NLL: 29.2952095558
```



5


Prediction and ground truth and 95% confidence intervals
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
0
50
100
150
200
250
300
Quantile 0.025
Quantile 0.975
Median
True

6

D

Further charts and tables

In this appendix, we provide the full results of our long-term forecasting task from Chapter 3. The tables below include the results for all 45 datasets from Section 3.5.1 comparing all baselines described in Section 3.5.2 across each of the 6 metrics presented in Section 3.5.4.

	MOrdReD	**Best GP**	**AR(p)**	**Seq2Seq**
AS_s3.2_	**1,067.0291**	1,459.8154	1,154.4474	1,148.4572
CM_air.s	1,647.5834	1,747.9861	**1,300.4001**	5,490.6812
CM_lwtla	**-1,678.6070**	1,450.3402	1,449.5522	7,615.6543
CM_prate	**-1,857.4666**	1,704.9511	1,632.7339	12,773.7439
CM_rhum1	**674.6915**	1,505.7443	1,415.9282	2,952.0419
CM_slp19	1,450.5504	**1,385.1744**	1,432.9088	4,218.5500
EM_henon	**1,327.4453**	1,435.1775	1,438.2578	1,445.6591
EM_rossl	896.0987	**-892.0153**	1,184.9238	1,165.5323
EMexptqp	1,216.7753	**432.4752**	1,393.8399	1,378.7560
EMlorenz	1,591.2022	**121.5318**	1,373.5663	1,357.3732
FI_yahoo	**1,199.2036**	1,278.7070	1,275.6439	4,873.6909
FL_ACT_L	3,208.1393	2,697.9846	**1,319.7379**	1,738.1959
FL_chen_	**1,338.2304**	1,382.0458	1,410.6686	1,440.1776
FL_dblsc	3,950.2313	9,972.5421	**1,184.7026**	5,146.1878
FL_hadle	-1,299.9369	**-1,475.3231**	1,186.5957	1,322.2664
FL_labyr	5,146.8124	147,438.9667	**276.0968**	24,374.2497
FL_moore	**1,227.1144**	1,260.6459	1,404.3013	1,454.9362
FL_noseh	1,597.5401	2,593.2756	1,347.5664	**1,251.4174**
FL_ruckl	5,883.6799	11,849.8276	**1,348.7245**	1,943.1374
FL_simpq	849.7241	**422.4309**	1,364.1362	1,138.5684
FL_thoma	1,640.3127	1,734.6156	**1,411.1365**	1,590.3507
FL_windm	1,274.1142	**667.4089**	1,319.0198	1,400.5067
K_standa	**1,074.6649**	1,440.4257	1,134.9302	29,788.0239
MC_inttr	**-1,467.4769**	1,372.3784	1,045.9586	30,051.7516

MP_Lozi_	**1,389.3785**	1,443.7876	1,430.1584	1,445.1244
MP_freit	**544.8592**	1,439.2089	1,438.4814	1,738.2226
MP_logis	-4,955.7417	-3,700.6026	**-5,814.7114**	-639.4380
MUS.3_78	**920.0800**	1,390.8033	1,359.8414	25,326.5354
MUS_Si_l	1,232.7973	**1,194.7616**	1,223.9410	2,804.7069
SFX_mach	**1,461.8245**	1,549.1978	1,508.8975	1,726.2633
SF_Acont	**875.0483**	1,825.6995	1,343.4891	1,342.5143
SF_B1_1	2,551.0437	**1,784.4138**	1,808.9434	3,106.4496
SF_D1	2,232.7716	1,711.1560	**1,588.1358**	1,594.8293
SL_perci	1,205.1593	1,088.5775	1,122.7930	**993.8790**
SPIDR_hp	**685.5992**	1,621.4097	1,650.6923	3,572.2350
SY_AR2_T	1,410.4564	1,505.1960	**1,399.2487**	1,508.9031
SY_NLAR2	**1,378.5653**	1,419.7959	1,421.6656	1,418.8368
TSAR_eqe	**493.5048**	812.4271	887.1088	1,229.8399
TXT_slc_	**1,035.6897**	1,533.3507	1,559.1041	28,889.8597
AIRFLOW	**435.9626**	2,877.8574	968.6876	4,062.9142
ECG	**296.9145**	1,287.2807	1,396.4655	1,686.4036
MACKEY	1,030.1611	**-76.3993**	1,359.9969	1,274.3958
TIDE	1,480.2779	**965.8456**	1,726.2864	1,921.4516
CM_air.2	2,536.3782	**1,181.4982**	1,332.4977	7,671.7171
CM_slp2	**1,057.9162**	1,209.9395	1,376.0466	1,785.7876
# BEST	**22**	12	9	2

Table D.1: Negative log-likelihood results for each task in our comparison. Bold indicates best performance.

	MOrdReD	**Best GP**	**AR(p)**	**Seq2Seq**
AS_s3.2_	**537,774.43**	730,981.81	583,011.52	602,465.25
CM_air.s	797,310.66	849,920.62	**595,495.74**	1,987,045.65
CM_lwtla	**-850,655.88**	720,115.91	720,560.30	3,475,892.35
CM_prate	**-962,237.35**	827,452.57	785,376.89	inf
CM_rhum1	**318,715.95**	744,527.62	698,875.16	1,318,162.92
CM_slp19	708,956.41	**676,033.24**	708,960.39	1,956,641.34
EM_henon	**666,538.09**	721,627.81	722,434.22	723,671.53
EM_rossl	234,903.22	**-799,240.14**	525,264.26	501,136.69
EMexptqp	603,202.42	**233,318.58**	679,917.50	666,458.95
EMlorenz	670,342.93	**-352,288.80**	660,739.26	633,814.42
FI_yahoo	**610,691.18**	649,524.68	652,466.75	2,614,821.68
FL_ACT_L	1,695,738.37	**202,184.74**	549,821.00	859,863.90
FL_chen_	**643,972.99**	676,464.02	702,070.44	712,645.43
FL_dblsc	1,409,267.68	3,042,435.39	**319,559.15**	1,735,099.88
FL_hadle	-745,731.68	**-808,478.23**	499,370.76	584,166.13
FL_labyr	2,547,025.42	69,599,173.17	**7,320.57**	11,599,495.08
FL_moore	594,250.96	**-7,825.35**	651,279.99	703,499.42
FL_noseh	733,863.00	1,138,630.16	647,597.99	**576,340.79**
FL_ruckl	3,393,325.76	10,305,030.96	**728,687.10**	1,329,705.18

FL_simpq	425,452.28	**-28,921.56**	643,127.84	503,836.93
FL_thoma	755,466.71	1,073,910.49	**721,636.05**	884,462.56
FL_windm	655,510.60	**-38,691.89**	625,721.84	675,112.24
K_standa	**521,541.77**	705,862.87	552,889.65	inf
MC_inttr	**-806,169.24**	681,920.77	519,199.89	inf
MP_Lozi_	**698,082.80**	725,860.12	717,822.77	725,154.91
MP_freit	**282,108.11**	716,132.16	717,761.94	861,123.80
MP_logis	-2,479,885.56	-1,851,887.05	**-2,937,775**	-320,044.53
MUS.3_78	**587,264.63**	768,461.15	777,514.25	inf
MUS_Si_l	605,535.44	**580,569.85**	606,199.76	1,374,118.90
SFX_mach	**738,330.97**	816,355.99	787,286.37	952,185.44
SF_Acont	**326,603.37**	737,650.13	652,492.95	639,344.90
SF_B1_1	1,043,778.68	**817,815.99**	820,812.04	1,262,284.01
SF_D1	1,210,384.29	891,784.86	817,067.07	**793,949.75**
SL_perci	582,674.39	514,702.66	532,024.91	**459,786.51**
SPIDR_hp	**326,188.71**	804,120.49	806,853.54	1,803,166.55
SY_AR2_T	**680,901.30**	726,950.41	682,292.96	739,537.19
SY_NLAR2	**696,355.54**	712,570.85	712,999.74	712,314.28
TSAR_eqe	**282,982.33**	408,908.60	445,354.10	671,606.17
TXT_slc_	**478,537.97**	737,348.58	739,450.48	inf
AIRFLOW	**20,871.60**	1,108,695.50	377,589.07	1,974,141.39
ECG	**202,695.99**	570,928.62	689,755.10	734,267.29
MACKEY	438,812.51	**-555,196.36**	658,548.58	626,990.61
TIDE	586,436.90	**378,848.89**	790,991.96	933,903.84
CM_air.2	897,901.41	**546,902.59**	640,305.14	2,829,558.40
CM_slp2	**426,658.49**	561,996.39	661,139.25	831,550.65
# BEST	**22**	14	6	3

Table D.2: Integrated negative log-likelihood results for each task in our comparison. Bold indicates best performance.

	MOrdReD	**Best GP**	**AR(p)**	**Seq2Seq**
AS_s3.2_	**0.0066**	0.0211	0.0096	0.0073
CM_air.s	0.0295	0.0170	**0.0014**	0.0748
CM_lwtla	0.0110	**0.0066**	0.0075	0.0906
CM_prate	0.2647	0.0554	**0.0549**	0.2777
CM_rhum1	0.0103	0.0166	**0.0078**	0.0347
CM_slp19	0.0042	0.0007	**0.0006**	0.0587
EM_henon	**0.0001**	0.0023	0.0029	0.0026
EM_rossl	0.0024	0.0032	0.0005	**0.0002**
EMexptqp	0.0023	0.0080	**0.0007**	0.0042
EMlorenz	0.0024	0.0062	**0.0003**	0.0021
FI_yahoo	**0.0009**	0.0032	0.0025	0.0225
FL_ACT_L	0.0611	0.0196	**0.0112**	0.0802
FL_chen_	0.0001	**5.01e-5**	0.0009	0.0018
FL_dblsc	0.0340	0.0489	**0.0020**	0.0593

FL_hadle	0.0084	0.0028	**0.0016**	0.0065
FL_labyr	0.3317	0.3317	**0.0512**	0.3316
FL_moore	**0.0023**	0.0040	0.0033	0.0062
FL_noseh	0.0065	0.0440	**0.0006**	0.0007
FL_ruckl	**0.0076**	0.0132	0.0170	0.1098
FL_simpq	**0.0002**	0.0003	0.0018	0.0004
FL_thoma	0.0328	0.0041	**0.0041**	0.0165
FL_windm	**0.0007**	0.0136	0.0035	0.0062
K_standa	**0.0012**	0.0179	0.0055	0.0756
MC_inttr	0.2047	0.0550	**0.0520**	0.2041
MP_Lozi_	**0.0002**	0.0005	0.0007	0.0007
MP_freit	**0.0004**	0.0116	0.0125	0.0386
MP_logis	0.3004	**0.0453**	0.0527	0.0498
MUS.3_78	**0.0059**	0.0230	0.0203	0.0649
MUS_Si_l	0.0021	0.0019	**0.0012**	0.0306
SFX_mach	**3.78e-5**	0.0009	0.0006	0.0056
SF_Acont	**0.0036**	0.0198	0.0052	0.0056
SF_B1_1	0.0301	**0.0011**	0.0021	0.0194
SF_D1	0.0362	0.0376	0.0163	**0.0140**
SL_perci	0.0266	**0.0013**	0.0059	0.0016
SPIDR_hp	**0.0021**	0.0253	0.0156	0.0657
SY_AR2_T	**0.0011**	0.0041	0.0024	0.0135
SY_NLAR2	0.0027	0.00014	**0.0001**	0.0006
TSAR_eqe	**0.0079**	0.0201	0.0235	0.0123
TXT_slc_	**0.0003**	0.0100	0.0061	0.0870
AIRFLOW	0.0048	0.0633	**0.0042**	0.0711
ECG	0.0431	0.0622	0.0328	**0.0183**
MACKEY	**0.0005**	0.0011	0.0023	0.0027
TIDE	0.0645	0.1086	**0.0142**	0.0253
CM_air.2	0.0231	0.0243	**0.0086**	0.0954
CM_slp2	0.0110	**0.0008**	0.0018	0.0212
# BEST	**18**	6	18	3

Table D.3: Euclidean distance between the identity function and the reliability plot for each task in our comparison. Bold indicates best performance.

	MOrdReD	**Best GP**	**AR(p)**	**Seq2Seq**
AS_s3.2_	**0.0094**	0.0235	0.0122	0.0300
CM_air.s	0.0169	0.0541	**0.0048**	0.0728
CM_lwtla	**0.0061**	0.0085	0.0080	0.0568
CM_prate	0.2856	0.0575	**0.0567**	0.2890
CM_rhum1	**0.0012**	0.0016	0.0013	0.0185
CM_slp19	0.0055	**0.0017**	0.0045	0.0678
EM_henon	**0.0002**	0.0022	0.0021	0.0019
EM_rossl	0.0123	0.0502	0.0040	**0.0030**
EMexptqp	0.0053	0.0443	**0.0009**	0.0086

EMlorenz	0.0056	0.0097	**0.0013**	0.0056
FI_yahoo	**0.0005**	0.0020	0.0013	0.0262
FL_ACT_L	0.2183	**0.0180**	0.0245	0.1056
FL_chen_	**0.0004**	0.0007	0.0013	0.0009
FL_dblsc	0.1264	0.1586	**0.0593**	0.2006
FL_hadle	0.0034	0.0035	**0.0011**	0.0105
FL_labyr	0.3317	0.3317	**0.0540**	0.3314
FL_moore	0.0163	**0.0058**	0.0101	0.0171
FL_noseh	0.0069	0.0374	**0.0007**	0.0007
FL_ruckl	0.1852	0.1319	**0.0236**	0.1248
FL_simpq	0.0110	0.0072	**0.0005**	0.0033
FL_thoma	0.1390	0.1033	**0.1021**	0.2193
FL_windm	**0.0014**	0.0294	0.0037	0.0054
K_standa	**0.0036**	0.0153	0.0060	0.0779
MC_inttr	0.2329	**0.0604**	0.0614	0.2397
MP_Lozi_	**0.0004**	0.0006	0.0007	0.0007
MP_freit	**0.0018**	0.0069	0.0068	0.0261
MP_logis	0.2977	**0.0443**	0.0525	0.0489
MUS.3_78	0.0039	0.0013	**0.0005**	0.0770
MUS_Si_l	**0.0003**	0.0006	0.0024	0.0457
SFX_mach	**0.0006**	0.0082	0.0044	0.0156
SF_Acont	0.0097	**0.0068**	0.0070	0.0080
SF_B1_1	0.0069	0.0041	0.0053	**0.0017**
SF_D1	0.0576	0.0528	0.0239	**0.0078**
SL_perci	0.0543	0.0280	0.0246	**0.0153**
SPIDR_hp	**0.0006**	0.0264	0.0153	0.0557
SY_AR2_T	**0.0025**	0.0069	0.0042	0.0173
SY_NLAR2	0.0151	**0.0054**	0.0065	0.0135
TSAR_eqe	**0.0065**	0.0227	0.0250	0.0082
TXT_slc_	**0.0054**	0.0280	0.0192	0.1169
AIRFLOW	0.1135	0.2125	**0.0126**	0.0559
ECG	0.0812	0.0361	0.0335	**0.0108**
MACKEY	0.0036	0.0079	**0.0013**	0.0037
TIDE	0.1674	0.1899	**0.0283**	0.0594
CM_air.2	**0.0171**	0.0644	0.0376	0.0809
CM_slp2	**0.0023**	0.0045	0.0071	0.0210
# BEST	**18**	7	15	5

Table D.4: Euclidean distance between the identity function and the reliability plot for each task in our comparison. Reliability plots for this table were computed from only the first 250 out-f-sample predictions generated by each model for each dataset. Bold indicates best performance.

	MOrdReD	**Best GP**	**AR(p)**	**Seq2Seq**

CM_LWTLA	1.6479	1.7955	1.8053	1.5809
AS_S3.2_	1.8184	1.5639	1.6351	1.3664
CM_AIR.S	1.0860	1.5918	1.4570	1.3134
CM_PRATE	1.9693	1.4023	1.4006	1.9711
CM_RHUM1	1.2107	1.6082	1.6819	1.4002
CM_SLP19	1.6019	1.6147	1.7388	1.3921
EM_HENON	1.7452	1.7493	1.7747	1.7425
EM_ROSSL	0.7261	0.2959	1.1970	1.2620
EMEXPTQP	1.1335	0.7291	1.7299	1.5902
EMLORENZ	1.5931	0.9476	1.6717	1.5241
FI_YAHOO	1.7011	1.6991	1.6885	1.8315
FL_ACT_L	1.0296	1.1732	1.6232	1.5676
FL_CHEN_	1.5556	1.7506	1.7470	1.6514
FL_DBLSC	1.6725	1.4025	1.1624	1.3952
FL_HADLE	0.7364	0.1298	1.2714	1.5234
FL_LABYR	0.8739	1.9941	0.0353	1.0676
FL_MOORE	1.6304	1.1501	1.6711	1.5540
FL_NOSEH	1.5364	1.3409	1.6407	1.5371
FL_RUCKL	1.4703	1.1251	1.5250	1.5019
FL_SIMPQ	0.9065	0.8744	1.5257	1.2256
FL_THOMA	1.3819	1.5836	1.7531	1.4419
FL_WINDM	1.8784	0.9487	1.6585	1.6944
K_STANDA	1.7612	1.5711	1.6532	1.8597
MC_INTTR	1.8507	1.4128	1.5416	1.7998
MP_LOZI_	1.7297	1.7608	1.7244	1.7612
MP_FREIT	1.5571	1.8101	1.8015	1.6288
MP_LOGIS	0.1045	0.0008	0.0002	0.0304
MUS.3_78	1.4647	1.5726	1.5610	1.7256
MUS_SI_L	1.8446	1.6841	1.7280	1.3604
SFX_MACH	1.8968	1.6381	1.6852	1.7161
SF_ACONT	1.4930	1.5342	1.6801	1.7120
SF_B1_1	1.4653	1.6519	1.7227	1.4735
SF_D1	1.4589	1.6169	1.7781	1.1392
SL_PERCI	1.1605	1.5407	1.5464	1.2299
SPIDR_HP	1.7020	1.7115	1.6681	1.6401
SY_AR2_T	1.7808	1.7461	1.7413	1.5590
SY_NLAR2	1.5911	1.7534	1.7044	1.7295
TSAR_EQE	1.5655	1.5945	1.5434	1.8148
TXT_SLC_	1.7458	1.7244	1.7330	1.9347
AIR	0.5546	1.4398	1.1710	1.3795
HEART	1.0511	1.7276	1.5423	1.3119
MG	0.8810	0.9571	1.6755	1.1923
TIDE	0.7162	0.5699	1.5541	1.7027
WEBTSSLP	0.8221	1.4685	1.6949	1.6707
WEBTSAIR	1.2155	1.1183	1.6023	1.3777

# BEST	**15**	**15**	7	8

Table D.5: Symmetric mean absolute percentage error between the out-of-sample ground truth and the mean of the predictive distribution in each task. Bold indicates best performance.

	MOrdReD	**Best GP**	**AR(p)**	**Seq2Seq**
AS_s3.2_	1.8147	1.5639	1.6351	**1.3664**
CM_air.s	**1.0681**	1.5650	1.4570	1.3134
CM_lwtla	**1.4926**	1.7521	1.8053	1.5809
CM_prate	**0.7965**	1.4023	1.4006	1.9711
CM_rhum1	**1.1057**	1.5822	1.6819	1.4002
CM_slp19	1.5448	1.6020	1.7388	**1.3921**
EM_henon	**1.3646**	1.7034	1.7747	1.7425
EM_rossl	0.7428	**0.2904**	1.1970	1.2620
EMexptqp	1.0935	**0.6750**	1.7299	1.5902
EMlorenz	1.5430	**0.9303**	1.6717	1.5241
FI_yahoo	**1.6647**	1.6732	1.6885	1.8315
FL_ACT_L	**0.9427**	1.1732	1.6232	1.5676
FL_chen_	**1.4854**	1.6635	1.7470	1.6514
FL_dblsc	1.5428	1.4025	**1.1624**	1.3952
FL_hadle	0.7312	**0.1298**	1.2714	1.5234
FL_labyr	0.7911	1.9941	**0.0353**	1.0676
FL_moore	1.5840	**1.1501**	1.6711	1.5540
FL_noseh	1.4959	**1.2923**	1.6407	1.5371
FL_ruckl	1.5515	**1.1251**	1.5250	1.5019
FL_simpq	0.7904	**0.7676**	1.5257	1.2256
FL_thoma	**1.2368**	1.4740	1.7531	1.4419
FL_windm	1.3481	**0.9162**	1.6585	1.6944
K_standa	1.6337	**1.5711**	1.6532	1.8597
MC_inttr	**0.3040**	1.4128	1.5416	1.7998
MP_Lozi_	**1.5819**	1.7303	1.7244	1.7612
MP_freit	**1.3093**	1.7869	1.8015	1.6288
MP_logis	**0.0000**	0.0008	0.0002	0.0304
MUS.3_78	1.7685	**1.5454**	1.5610	1.7256
MUS_Si_l	1.8375	1.6673	1.7280	**1.3604**
SFX_mach	1.9651	**1.6139**	1.6852	1.7161
SF_Acont	**1.0772**	1.5164	1.6801	1.7120
SF_B1_1	1.4776	1.6354	1.7227	**1.4735**
SF_D1	1.4701	1.5502	1.7781	**1.1392**
SL_perci	**1.1529**	1.5393	1.5464	1.2299
SPIDR_hp	**1.1855**	1.6951	1.6681	1.6401
SY_AR2_T	1.7251	1.7359	1.7413	**1.5590**
SY_NLAR2	**1.5561**	1.7345	1.7044	1.7295

TSAR_eqe	1.5632	1.5945	**1.5434**	1.8148
TXT_slc_	**1.1753**	1.7161	1.7330	1.9347
AIRFLOW	**0.5783**	1.4398	1.1710	1.3795
ECG	**0.8779**	1.7244	1.5423	1.3119
MACKEY	**0.8303**	0.9571	1.6755	1.1923
TIDE	0.6660	**0.5474**	1.5541	1.7027
CM_air.2	1.1996	**1.0856**	1.6023	1.3777
CM_slp2	**0.8115**	1.4028	1.6949	1.6707
# BEST	**22**	14	3	6

Table D.6: Symmetric mean absolute percentage error between the out-of-sample ground truth and the median of the predictive distribution in each task. Bold indicates best performance.

	MOrdReD	**Best GP**	**AR(p)**	**Seq2Seq**
CM_LWTLA	1.0921	1.0264	1.0246	1.1300
AS_S3.2_	0.6258	0.6551	0.6424	0.6421
CM_AIR.S	0.8817	1.1635	0.9058	1.2285
CM_PRATE	1.2644	1.2853	1.2704	1.2636
CM_RHUM1	1.0259	1.0447	0.9934	0.9744
CM_SLP19	1.0581	0.9683	1.0104	1.0727
EM_HENON	0.9985	1.0086	1.0102	1.0124
EM_ROSSL	0.5694	0.2584	0.8099	0.8342
EMEXPTQP	0.6944	0.3742	0.9889	0.9724
EMLORENZ	0.9248	0.7308	0.9892	0.9329
FI_YAHOO	0.8401	0.8438	0.8469	0.8377
FL_ACT_L	1.0792	0.8801	1.0010	1.2759
FL_CHEN_	0.9604	0.9935	0.9969	1.0106
FL_DBLSC	1.2945	1.2080	1.0318	1.3579
FL_HADLE	0.5162	0.0598	0.8562	0.9772
FL_LABYR	2.3082	3.7961	0.1742	2.6423
FL_MOORE	1.0398	0.7934	1.0399	1.0527
FL_NOSEH	1.0740	1.1971	0.9342	0.8885
FL_RUCKL	1.1837	1.0766	1.0197	1.0914
FL_SIMPQ	0.7233	0.7320	0.9594	0.8135
FL_THOMA	1.0762	1.1046	0.9974	1.1071
FL_WINDM	0.9661	0.9527	0.9353	0.9402
K_STANDA	0.6886	0.7058	0.6939	0.6889
MC_INTTR	0.3268	0.2963	0.2817	0.2519
MP_LOZI_	1.0021	1.0184	1.0088	1.0177
MP_FREIT	0.9901	1.0152	1.0145	1.0925
MP_LOGIS	0.1011	0.0010	0.0002	0.0352
MUS.3_78	0.8578	0.8560	0.8638	0.8482
MUS_SI_L	0.7826	0.7805	0.7851	0.8269
SFX_MACH	1.0263	1.0746	1.0628	1.0522

SF_ACONT	0.8604	0.8941	0.9254	0.9200
SF_B1_1	1.3821	1.3189	1.3177	1.3802
SF_D1	1.2350	1.2493	1.1524	1.1876
SL_PERCI	0.7112	0.7043	0.6972	0.6617
SPIDR_HP	1.1848	1.1870	1.1884	1.2112
SY_AR2_T	0.9703	0.9883	0.9801	1.0558
SY_NLAR2	1.0080	0.9991	0.9949	1.0013
TSAR_EQE	0.3302	0.3369	0.3369	0.3261
TXT_SLC_	1.1100	1.1179	1.1272	1.1155
AIR	0.3858	0.8196	0.6740	1.1156
HEART	0.8462	1.0434	0.9977	0.9965
MG	0.7073	0.8587	0.9586	0.8335
TIDE	0.7283	0.4963	1.2515	1.2381
WEBTSSLP	0.6390	0.8730	0.9690	0.9883
WEBTSAIR	1.1141	0.7993	0.9186	1.2268
# BEST	**17**	10	10	8

Table D.7: Root Mean Squared Error between the out-of-sample ground truth and the mean of the predictive distribution in each task. Bold indicates best performance.

	MOrdReD	**Best GP**	**AR(p)**	**Seq2Seq**
CM_LWTLA	1.2220	1.0229	1.0246	1.1300
AS_S3.2_	0.6233	0.6551	0.6424	0.6421
CM_AIR.S	0.9013	1.1635	0.9058	1.2285
CM_PRATE	1.2708	1.2853	1.2704	1.2636
CM_RHUM1	1.1193	1.0447	0.9934	0.9744
CM_SLP19	1.0952	0.9666	1.0104	1.0727
EM_HENON	1.0294	1.0086	1.0102	1.0124
EM_ROSSL	0.5826	0.2581	0.8099	0.8342
EMEXPTQP	0.6629	0.3361	0.9889	0.9724
EMLORENZ	0.9012	0.7043	0.9892	0.9329
FI_YAHOO	0.8438	0.8438	0.8469	0.8377
FL_ACT_L	1.2954	0.8801	1.0010	1.2759
FL_CHEN_	0.9666	0.9935	0.9969	1.0106
FL_DBLSC	1.3801	1.2080	1.0318	1.3579
FL_HADLE	0.5325	0.0593	0.8562	0.9772
FL_LABYR	2.1533	3.7961	0.1742	2.6423
FL_MOORE	1.0418	0.7934	1.0399	1.0527
FL_NOSEH	1.1721	1.1971	0.9342	0.8885
FL_RUCKL	1.2522	1.0741	1.0197	1.0914
FL_SIMPQ	0.7428	0.7308	0.9594	0.8135
FL_THOMA	1.3525	1.1046	0.9974	1.1071
FL_WINDM	0.9843	0.9527	0.9353	0.9402
K_STANDA	0.6895	0.7032	0.6939	0.6889

MC_INTTR	0.2367	0.2963	0.2817	0.2519
MP_LOZI_	1.0091	1.0170	1.0088	1.0177
MP_FREIT	1.0137	1.0108	1.0145	1.0925
MP_LOGIS	0.0000	0.0010	0.0002	0.0352
MUS.3_78	0.8482	0.8560	0.8638	0.8482
MUS_SI_L	0.7829	0.7805	0.7851	0.8269
SFX_MACH	1.0261	1.0746	1.0628	1.0522
SF_ACONT	0.9184	0.8941	0.9254	0.9200
SF_B1_1	1.3761	1.3159	1.3177	1.3802
SF_D1	1.2356	1.2493	1.1524	1.1876
SL_PERCI	0.7450	0.7043	0.6972	0.6617
SPIDR_HP	1.2116	1.1870	1.1884	1.2112
SY_AR2_T	0.9688	0.9883	0.9801	1.0558
SY_NLAR2	1.0129	0.9991	0.9949	1.0013
TSAR_EQE	0.3302	0.3369	0.3369	0.3261
TXT_SLC_	1.1920	1.1179	1.1272	1.1155
AIR	0.4521	0.8196	0.6740	1.1156
HEART	0.9896	1.0434	0.9977	0.9965
MG	0.6975	0.8587	0.9586	0.8335
TIDE	0.7275	0.4751	1.2515	1.2381
WEBTSSLP	0.6462	0.8561	0.9690	0.9883
WEBTSAIR	1.1240	0.7993	0.9186	1.2268
# BEST	11	**17**	8	9

Table D.8: Root Mean Squared Error between the out-of-sample ground truth and the median of the predictive distribution in each task. Bold indicates best performance.

	Order		Order
AS_s3.2_	16	**MP_freit**	16
CM_air.s	64	**MP_logis**	64
CM_lwtla	16	**MUS.3_78**	16
CM_prate	64	**MUS.3_78**	16
CM_rhum1	64	**MUS.3_78**	16
CM_slp19	64	**SFX_mach**	16
EM_henon	16	**SF_Acont**	64
EM_rossl	16	**SF_B1_1**	64
EMexptqp	16	**SF_D1**	16
EMlorenz	16	**SL_perci**	64
FI_yahoo	64	**SPIDR_hp**	64
FL_ACT_L	16	**SY_AR2_T**	64
FL_chen_	16	**SY_NLAR2**	64
FL_dblsc	16	**TSAR_eqe**	16
FL_hadle	16	**TXT_slc_**	64
FL_labyr	16	**AIR**	16

FL_moore	16	**ECG**	32
FL_noseh	16	**MACKEY**	16
FL_ruckl	16	**TIDE**	16
FL_simpq	16	**CM_air 2**	64
FL_thoma	16	**CM_slp 2**	64
FL_windm	16		
K_standa	64		
MC_inttr	64		
MP_Lozi_	16		

Table D.9: Best value for the p hyperparameter of each AR(p) model found by grid search on $p \in \{16, 32, 64\}$

	Hidden Units h_u	Dropout rate p_{dropout}	L2 regularisation λ	Ordinal bins
AS_s3.2_	64.0	0.25	1e-08	300.0
CM_air.s	64.0	0.25	1e-08	300.0
CM_lwtla	128.0	0.25	1e-07	300.0
CM_prate	128.0	0.25	1e-07	300.0
CM_rhum1	128.0	0.5	1e-07	300.0
CM_slp19	256.0	0.5	1e-06	300.0
EM_henon	64.0	0.5	1e-06	300.0
EM_rossl	320.0	0.25	1e-07	300.0
EMexptqp	256.0	0.5	1e-08	300.0
EMlorenz	64.0	0.5	1e-06	300.0
FI_yahoo	320.0	0.5	1e-08	300.0
FL_ACT_L	256.0	0.5	1e-07	300.0
FL_chen_	128.0	0.5	1e-07	300.0
FL_dblsc	128.0	0.25	1e-08	300.0
FL_hadle	320.0	0.25	1e-07	300.0
FL_labyr	64.0	0.25	1e-08	300.0
FL_moore	320.0	0.5	1e-06	300.0
FL_noseh	128.0	0.25	1e-08	300.0
FL_ruckl	320.0	0.5	1e-07	300.0
FL_simpq	64.0	0.5	1e-08	300.0
FL_thoma	64.0	0.5	1e-06	300.0
FL_windm	256.0	0.5	1e-08	300.0
K_standa	256.0	0.5	1e-06	300.0
MC_inttr	128.0	0.25	1e-06	300.0
MP_Lozi_	64.0	0.5	1e-08	300.0
MP_freit	64.0	0.25	1e-07	300.0
MP_logis	128.0	0.25	1e-06	300.0
MUS.3_78	320.0	0.5	1e-07	300.0
MUS_Si_l	320.0	0.5	1e-08	300.0
SFX_mach	64.0	0.5	1e-08	300.0
SF_Acont	64.0	0.25	1e-06	236.0

SF_B1_1	128.0	0.5	1e-08	300.0
SF_D1	320.0	0.5	1e-08	300.0
SL_perci	256.0	0.5	1e-06	300.0
SPIDR_hp	128.0	0.5	1e-08	300.0
SY_AR2_T	64.0	0.25	1e-06	300.0
SY_NLAR2	320.0	0.5	1e-06	300.0
TSAR_eqe	128.0	0.25	1e-07	300.0
TXT_slc_	256.0	0.25	1e-07	300.0
AIRFLOW	256.0	0.5	1e-06	226.0
ECG	256.0	0.5	1e-07	134.0
MACKEY	256.0	0.25	1e-07	300.0
TIDE	256.0	0.5	1e-06	300.0
CM_air.2	128.0	0.25	1e-07	300.0
CM_SLP.2	320.0	0.25	1e-07	259.0

Table D.10: Best value for the hyperparameters of our MOrdReD models found by grid search on $h_u \in \{64, 128, 256, 320\}, \lambda \in \{1e-6, 1e-7, 1e-8\}, p_{\text{dropout}} \in \{0.25, 0.35, 0.5\}$.

	Hidden Units h_u	Dropout rate p_{dropout}	L2 regularisation λ
AS_s3.2_	128.0	0.5	1e-08
CM_air.s	320.0	0.5	1e-07
CM_lwtla	256.0	0.5	1e-08
CM_prate	64.0	0.25	1e-06
CM_rhum1	128.0	0.5	1e-07
CM_slp19	128.0	0.5	1e-06
EM_henon	64.0	0.5	1e-06
EM_rossl	128.0	0.25	1e-07
EMexptqp	256.0	0.25	1e-07
EMlorenz	128.0	0.25	1e-07
FI_yahoo	128.0	0.25	1e-08
FL_ACT_L	128.0	0.25	1e-06
FL_chen_	320.0	0.5	1e-08
FL_dblsc	256.0	0.25	1e-08
FL_hadle	128.0	0.25	1e-06
FL_labyr	320.0	0.25	1e-07
FL_moore	128.0	0.25	1e-07
FL_noseh	256.0	0.5	1e-08
FL_ruckl	256.0	0.25	1e-06
FL_simpq	128.0	0.25	1e-08
FL_thoma	320.0	0.25	1e-07
FL_windm	256.0	0.5	1e-08
K_standa	64.0	0.25	1e-06
MC_inttr	64.0	0.25	1e-06

MP_Lozi_	64.0	0.25	1e-08
MP_freit	64.0	0.25	1e-06
MP_logis	64.0	0.25	1e-07
MUS.3_78	64.0	0.25	1e-06
MUS_Si_l	320.0	0.5	1e-08
SFX_mach	320.0	0.25	1e-08
SF_Acont	128.0	0.25	1e-07
SF_B1_1	320.0	0.25	1e-06
SF_D1	256.0	0.5	1e-07
SL_perci	128.0	0.25	1e-06
SPIDR_hp	320.0	0.25	1e-08
SY_AR2_T	320.0	0.25	1e-08
SY_NLAR2	128.0	0.25	1e-07
TSAR_eqe	128.0	0.5	1e-06
TXT_slc_	64.0	0.25	1e-06
AIRFLOW	64.0	0.25	1e-06
ECG	128.0	0.25	1e-07
MACKEY	128.0	0.25	1e-07
TIDE	320.0	0.25	1e-07
CM_air.2	256.0	0.5	1e-06
CM_SLP.2	320.0	0.5	1e-06

Table D.11: Best value for the hyperparameters of our direct regression neural-network models found by grid search on $h_u \in \{64, 128, 256, 320\}$, $\lambda \in \{1e-6, 1e-7, 1e-8\}$, $p_{\text{dropout}} \in \{0.25, 0.35, 0.5\}$.

Bibliography

Akata, Z. et al. (2015). "Evaluation of output embeddings for fine-grained image classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2927–2936.

Amodei, D. et al. (2016). "Deep speech 2: End-to-end speech recognition in english and mandarin". In: *International conference on machine learning*, pp. 173–182.

Ardalani-Farsa, M. and S. Zolfaghari (2010). "Chaotic time series prediction with residual analysis method using hybrid Elman–NARX neural networks". In: *Neurocomputing* 73.13. Pattern Recognition in Bioinformatics Advances in Neural Control, pp. 2540–2553. ISSN: 0925-2312.

Bahdanau, D., K. Cho, and Y. Bengio (2014). "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473*.

Barber, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press.

Barber, D. and C. M. Bishop (1998). "Ensemble learning in Bayesian neural networks". In:

Bergstra, J. et al. (2010). "Theano: a CPU and GPU math compiler in Python". In: *Proceedings of the Python for Scientific Computing Conference (SciPy)* 9th.Scipy, pp. 1–7.

Bernardo, J. and A. Smith (2007). *Bayesian Theory*. Wiley Series in Probability and Statistics. Wiley. ISBN: 9780470028735.

Bernardo, J., M. Bayarri, et al. (2003). "The variational Bayesian EM algorithm for incomplete data: with application to scoring graphical model structures". In: *Bayesian statistics*.

Bishop, C. M. (1994). "Training with Noise is Equivalent to Tikhonov Regularization". In: *Neural Computation* 7, pp. 108–116.

— (1995). "Bayesian methods for neural networks". In: *Aston University Neural Computing Research Group Journal* 7.1, pp. 1–11.

— (2006). *Pattern Recognition and Machine Learning*. Cambridge: Springer.

Bluche, T., C. Kermorvant, and J. Louradour (2015). "Where to apply dropout in recurrent neural networks for handwriting recognition?" In: *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, pp. 681–685.

Bonilla, E. V., K. M. Chai, and C. Williams (2008). "Multi-task Gaussian process prediction". In: *Advances in neural information processing systems*, pp. 153–160.

Borchani, H. et al. (2015). "A survey on multi-output regression". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5.5, pp. 216–233.

Bose, R. (2008). *Information theory, coding and cryptography*. Tata McGraw-Hill Education.

Bottou, L. (2010). "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *Proceedings of COMPSTAT'2010*, pp. 177–186.

Box, G. E. et al. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

Bramble Bank, W. reports from (2018). *Chimet Support Group*.

Bücker, H. M. et al., eds. (2005). *Automatic Differentiation: Applications, Theory, and Implementations*. Vol. 50. Lecture Notes in Computational Science and Engineering. New York, NY: Springer.

Calliess, J.-P. (2015). "Bayesian Lipschitz Constant Estimation and Quadrature". In: *Advances in Neural Information Processing Systems*.

Caruana, R. (1997). "Multitask Learning". In: *Machine Learning* 28.1, pp. 41–75. ISSN: 1573-0565.

Chiu, B., E. Keogh, and S. Lonardi (2003). "Probabilistic discovery of time series motifs". In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 493–498.

Chollet, F. (2015). *Keras*.

Collobert, R. (2011). "Torch7: A Matlab-like environment for machine learning". In: *BigLearn, NIPS Workshop*, pp. 1–6.

Conneau, A. et al. (2017). "Supervised learning of universal sentence representations from natural language inference data". In: *arXiv preprint arXiv:1705.02364*.

Cox, R. T. (1946). "Probability, frequency and reasonable expectation". In: *American journal of physics* 14.1, pp. 1–13.

Cressie, N. (1993). *Statistics for spatial data*. Wiley series in probability and mathematical statistics: Applied probability and statistics. J. Wiley. ISBN: 9780471002550.

Cybenko, G. (1993). "Degree of approximation by superpositions of a sigmoidal function". In: *Approximation Theory and its Applications* 9.3, pp. 17–28. ISSN: 10009221.

Damianou, A. and N. Lawrence (2013). "Deep gaussian processes". In: *Artificial Intelligence and Statistics*, pp. 207–215.

Dellaportas, P. and D. A. Stephens (1995). "Bayesian analysis of errors-in-variables regression models". In: *Biometrics*, pp. 1085–1095.

Duchi, J., E. Hazan, and Y. Singer (2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12, pp. 2121–2159. ISSN: 15324435. arXiv: `arXiv:1103.4296v1`.

Durbin, J. and S. Koopman (2012). *Time Series Analysis by State Space Methods: Second Edition*. Oxford Statistical Science Series. OUP Oxford. ISBN: 9780199641178.

Faruk, D. Ö. (2010). "A hybrid neural network and ARIMA model for water quality time series prediction". In: *Engineering Applications of Artificial Intelligence* 23.4, pp. 586–594. ISSN: 0952-1976.

Fulcher, B. D., M. A. Little, and N. S. Jones (2013). "Highly comparative time-series analysis: the empirical structure of time series and their methods". In: *Journal of The Royal Society Interface* 10.83. ISSN: 1742-5689.

Gal, Y. and Z. Ghahramani (2016a). "A theoretically grounded application of dropout in recurrent neural networks". In: *Advances in neural information processing systems*, pp. 1019–1027.

— (2016b). "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by M. F. Balcan and K. Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, pp. 1050–1059.

Gales, M., S. Young, et al. (2008). "The application of hidden Markov models in speech recognition". In: *Foundations and Trends® in Signal Processing* 1.3, pp. 195–304.

Gamma, E. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Education India.

Garnelo, M. et al. (2018). "Neural processes". In: *arXiv preprint arXiv:1807.01622*.

Ghahramani, Z. (2001). "An introduction to hidden Markov models and Bayesian networks". In: *International journal of pattern recognition and artificial intelligence* 15.01, pp. 9–42.

— (2013). "Bayesian non-parametrics and the probabilistic approach to modelling". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 371.1984, p. 20110553.

Girard, A. and R. Murray-Smith (2005). "Gaussian processes: Prediction at a noisy input and application to iterative multiple-step ahead forecasting of time-series". In: *Switching and Learning in Feedback Systems*. Springer, pp. 158–184.

Golowich, N., A. Rakhlin, and O. Shamir (2017). "Size-independent sample complexity of neural networks". In: *arXiv preprint arXiv:1712.06541*.

Gomes, E. F., A. M. Jorge, and P. J. Azevedo (2013). "Classifying heart sounds using multiresolution time series motifs: an exploratory study". In: *Proceedings of the International C* Conference on Computer Science and Software Engineering*. ACM, pp. 23–30.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press.

GPy (2012). *GPy: A Gaussian process framework in Python*.

Graves, A. (2013). "Generating sequences with recurrent neural networks". In: *CoRR*, pp. 1–43. ISSN: 18792782. arXiv: `arXiv:1308.0850v5`.

Graves, A., A.-r. Mohamed, and G. Hinton (2013). "Speech Recognition With Deep Recurrent Neural Networks". In: *ICASSP*. ISSN: 1520-6149. arXiv: `arXiv:1303.5778v1`.

Graves, A. and J. Schmidhuber (2005). "Framewise phoneme classification with bidirectional LSTM networks". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 4, 2047–2052 vol. 4.

Gurevich, T. Y. et al. (2004). "R-R interval variation in Parkinson's disease and multiple system atrophy". In: *Acta neurologica scandinavica* 109.4, pp. 276–279.

Han, S. et al. (2015). "Learning both weights and connections for efficient neural network". In: *Advances in neural information processing systems*, pp. 1135–1143.

Hanson, S. J. and L. Y. Pratt (1989). "Comparing biases for minimal network construction with back-propagation". In: *Advances in neural information processing systems*, pp. 177–185.

Harrell Jr, F. E. (2015). *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis*. Springer.

Hausknecht, M. and P. Stone (2015). "Deep Recurrent Q-Learning for Partially Observable MDPs". In: *CoRR*. arXiv: 1507.06527.

Hensman, J., A. Matthews, and Z. Ghahramani (2015). "Scalable variational Gaussian process classification". In:

Hinton, G. E. et al. (2012). "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580. arXiv: 1207.0580.

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory." In: *Neural computation* 9.8, pp. 1735–80. ISSN: 0899-7667. arXiv: 1206.2944.

Hu, Q., R. Zhang, and Y. Zhou (2016). "Transfer learning for short-term wind speed prediction with deep neural networks". In: *Renewable Energy*. ISSN: 18790682.

Huh, M., P. Agrawal, and A. A. Efros (2016). "What makes ImageNet good for transfer learning?" In: *arXiv preprint arXiv:1608.08614*.

Jaynes, E. T. (1996). *Probability theory: the logic of science*.

Jin, X. et al. (2014). "A Transfer Forecasting Model for Container Throughput guided by discrete PSO". In: *J Syst Sci Complex* 27, pp. 181–192.

Journel, A. and C. Huijbregts (1978). *Mining Geostatistics*. Academic Press. ISBN: 9780123910509.

Jurafsky, D. and J. H. Martin (2009). *Speech and Language Processing*. 2nd. Prentice Hall.

Karpathy, A. (2015). *The Unreasonable Effectiveness of Recurrent Neural Networks*.

Khashei, M. and M. Bijari (2011). "A novel hybridization of artificial neural networks and ARIMA models for time series forecasting". In: *Applied Soft Computing* 11.2. The Impact of Soft Computing for the Progress of Artificial Intelligence, pp. 2664–2675. ISSN: 1568-4946.

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

Kingma, D. and J. Ba (2014). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations*, pp. 1–13. arXiv: 1412.6980.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

Kumar, N. et al. (2005). "Time-series bitmaps: a practical visualization tool for working with large time series databases". In: *Proceedings of the 2005 SIAM international conference on data mining*. SIAM, pp. 531–535.

Kuznetsov, V. and M. Mohri (2015). "Learning theory and algorithms for forecasting non-stationary time series". In: *Advances in neural information processing systems*, pp. 541–549.

Laptev, N. et al. (2017). "Time-series extreme event forecasting with neural networks at uber". In:

Larochelle, H., D. Erhan, and Y. Bengio (2008). "Zero-data learning of new tasks." In:

LeCun, Y. et al. (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551.

Lei Ba, J., K. Swersky, S. Fidler, et al. (2015). "Predicting deep zero-shot convolutional neural networks using textual descriptions". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4247–4255.

Li, Y. and J. Lin (2010). "Approximate Variable-length Time Series Motif Discovery Using Grammar Inference". In: *Proceedings of the Tenth International Workshop on Multimedia Data Mining*. MDMKDD '10. Washington, D.C.: ACM, 10:1–10:9. ISBN: 978-1-4503-0220-3.

Lin, J. et al. (2004). "Visually mining and monitoring massive time series". In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 460–469.

Lu, Z. and M. A. Carreira-Perpinan (2008). "Constrained spectral clustering through affinity propagation". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8.

Lu, Z., H. Pu, et al. (2017). "The expressive power of neural networks: A view from the width". In: *Advances in neural information processing systems*, pp. 6231–6239.

Luxburg, U. V. (2007). *A Tutorial on Spectral Clustering*.

Maaten, L. v. d. and G. Hinton (2008). "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov, pp. 2579–2605.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Makridakis, S., E. Spiliotis, and V. Assimakopoulos (2018). "The M4 Competition: Results, findings, conclusion and way forward". In: *International Journal of Forecasting* 34.4, pp. 802–808.

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org.

Matthews, A. G. d. G., J. Hensman, et al. (2016). "On sparse variational methods and the Kullback-Leibler divergence between stochastic processes". In: *Artificial Intelligence and Statistics*, pp. 231–239.

Matthews, D. G., G. Alexander, et al. (2017). "GPflow: A Gaussian process library using TensorFlow". In: *The Journal of Machine Learning Research* 18.1, pp. 1299–1304.

Mattos, C. et al. (2016). "Recurrent Gaussian Processes". In: *arXiv*.

McInnes, L., J. Healy, and J. Melville (2018). "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction". In: *ArXiv e-prints*. arXiv: `1802.03426 [stat.ML]`.

McInnes, L., J. Healy, N. Saul, et al. (2018). "UMAP: Uniform Manifold Approximation and Projection". In: *The Journal of Open Source Software* 3.29, p. 861.

Mikolov, T. et al. (2013). "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*, pp. 3111–3119.

Minsky, M. and S. Papert (1969). *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press.

Mohri, M., A. Rostamizadeh, and A. Talwalkar (2018). *Foundations of machine learning*.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. Cambridge: MIT Press.

Neal, R. M. (1996a). *Bayesian Learning for Neural Networks*. Berlin, Heidelberg: Springer-Verlag. ISBN: 0387947248.

Neal, R. M. (1996b). "Sampling from multimodal distributions using tempered transitions". In: *Statistics and computing* 6.4, pp. 353–366.

Ng, A. Y., M. I. Jordan, and Y. Weiss (2002). "On spectral clustering: Analysis and an algorithm". In: *Advances in neural information processing systems*, pp. 849–856.

Nocedal, J. and S. Wright (2006). *Numerical optimization*. Springer Science & Business Media.

Oldford, R. W. (2016). "Self-Calibrating Quantile–Quantile Plots". In: *The American Statistician* 70.1, pp. 74–90.

Pachitariu, M. and M. Sahani (2013). "Regularization and nonlinearities for neural language models: when are they needed?" In: *arXiv preprint arXiv:1301.5650*.

Paoli, C. et al. (2010). "Forecasting of preprocessed daily solar radiation time series using neural networks". In: *Solar Energy* 84.12, pp. 2146–2160. ISSN: 0038-092X.

Pascanu, R., T. Mikolov, and Y. Bengio (2013). "On the difficulty of training recurrent neural networks". In: *Proceedings of The 30th International Conference on Machine Learning* 28.2, pp. 1310–1318. ISSN: 1045-9227. arXiv: `arXiv:1211.5063v2`.

Paszke, A. et al. (2017). "Automatic differentiation in PyTorch". In:

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Pennington, J., R. Socher, and C. Manning (2014). "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Penny, W. and S. J. Roberts (2000). "Bayesian methods for autoregressive models". In: *Neural Networks for Signal Processing X, 2000. Proceedings of the 2000 IEEE Signal Processing Society Workshop*. Vol. 1. IEEE, pp. 125–134.

Pradeepkumar, D. and V. Ravi (2017). "Forecasting financial time series volatility using Particle Swarm Optimization trained Quantile Regression Neural Network". In: *Applied Soft Computing* 58, pp. 35–52. ISSN: 1568-4946.

Qureshi, A. S. et al. (2017). "Wind power prediction using deep neural network based meta regression and transfer learning". In: *Applied Soft Computing* 58, pp. 742–755.

Rasmussen, C. E. and C. Williams (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

Rezek, I. and S. J. Roberts (1998). "Stochastic complexity measures for physiological signal analysis". In: *IEEE Transactions on Biomedical Engineering* 45.9, pp. 1186–1191.

Ribeiro, M. et al. (2018). "Transfer learning with seasonal and trend adjustment for cross-building energy forecasting". In: *Energy and Buildings*. ISSN: 03787788.

Rumelhart, D. E., G. E. Hinton, R. J. Williams, et al. (1988). "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3, p. 1.

Rutkauskas, A. V., A. Maknickas, and N. Maknickienė (2011). "Investigation of financial market prediction by recurrent neural network". In: *Innovative Infotechnologies for Science, Business and Education* 2.687, pp. 3–8.

Salinas, D., V. Flunkert, and J. Gasthaus (2017). "DeepAR: Probabilistic forecasting with autoregressive recurrent networks". In: *arXiv preprint arXiv:1704.04110*.

Schmidhuber, J. et al. (2007). "Training recurrent networks by Evolino." In: *Neural computation* 19.3, pp. 757–779.

Seabold, S. and J. Perktold (2010). "Statsmodels: Econometric and statistical modeling with python". In: *9th Python in Science Conference*.

Senin, P. et al. (2014). "Grammarviz 2.0: a tool for grammar-based pattern discovery in time series". In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer, pp. 468–472.

Shalev-Shwartz, S. and S. Ben-David (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Shi, J. and J. Malik (2000). "Normalized cuts and image segmentation". In: *Departmental Papers (CIS)*, p. 107.

Shireen, T. et al. (2018). "Iterative multi-task learning for time-series modeling of solar panel PV outputs". In: *Applied energy* 212, pp. 654–662.

Sietsma, J. and R. J. Dow (1991). "Creating artificial neural networks that generalize". In: *Neural networks* 4.1, pp. 67–79.

Silverman, B. W. (1986). *Density estimation for statistics and data analysis*. Routledge.

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

— (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *ICLR 2015*, pp. 1–14. arXiv: `arXiv:1409.1556v6`.

Snelson, E. and Z. Ghahramani (2006). "Sparse Gaussian processes using pseudo-inputs". In: *Advances in neural information processing systems*, pp. 1257–1264.

Soille, P. (2013). *Morphological image analysis: principles and applications*. Springer Science & Business Media.

Srivastava, N. et al. (2014). "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Sutskever, I., J. Martens, et al. (2013). "On the importance of initialization and momentum in deep learning". In: *JMLR: W&CP* 28.2010, pp. 1139–1147. ISSN: 15206149.

Sutskever, I., O. Vinyals, and Q. V. Le (2014). "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Z. Ghahramani et al., pp. 3104–3112.

Taylor, S. J. and B. Letham (2018). "Forecasting at scale". In: *The American Statistician* 72.1, pp. 37–45.

Teh, Y.-W., M. Seeger, and M. Jordan (2005). "Semiparametric Latent Factor Models". In: *Artificial Intelligence and Statistics 10*.

Titsias, M. and M. Lázaro-Gredilla (2014). "Doubly stochastic variational Bayes for non-conjugate inference". In: *International conference on machine learning*, pp. 1971–1979.

Torrey, L. and J. Shavlik (2010). "Transfer learning". In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI Global, pp. 242–264.

Turner, R., M. Deisenroth, and C. Rasmussen (2010). "State-space inference and learning with Gaussian processes". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 868–875.

Vahdatpour, A., N. Amini, and M. Sarrafzadeh (2009). "Toward unsupervised activity discovery using multi dimensional motif detection in time series". In: *Twenty-First International Joint Conference on Artificial Intelligence*.

Venugopalan, S. et al. (2015). "Sequence to sequence-video to text". In: *Proceedings of the IEEE international conference on computer vision*, pp. 4534–4542.

Vinyals, O. et al. (2015). "Show and tell: A neural image caption generator". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, pp. 3156–3164. ISBN: 978-1-4673-6964-0.

Ward Jr, J. H. (1963). "Hierarchical grouping to optimize an objective function". In: *Journal of the American statistical association* 58.301, pp. 236–244.

Wen, T.-H. et al. (2015). "Semantically conditioned lstm-based natural language generation for spoken dialogue systems". In: *arXiv preprint arXiv:1508.01745*.

Xian, Y. et al. (2016). "Latent embeddings for zero-shot classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 69–77.

Xie, M. et al. (2016). "Transfer learning from deep features for remote sensing and poverty mapping". In: *Thirtieth AAAI Conference on Artificial Intelligence*.

Xu, K. et al. (2014). "Show , Attend and Tell : Neural Image Caption Generation with Visual Attention". In: *Proceedings of Machine Learning Research*. arXiv: `arXiv:1502.03044v3`.

You, Q. et al. (2016). "Image Captioning with Semantic Attention". In: *CoRR* abs/1603.03925.

Zaremba, W., I. Sutskever, and O. Vinyals (2014). "Recurrent neural network regularization". In: *arXiv preprint arXiv:1409.2329*.

Zhang, G. (2003). "Time series forecasting using a hybrid ARIMA and neural network model". In: *Neurocomputing* 50, pp. 159–175. ISSN: 0925-2312.

Zhu, L. and N. Laptev (2017). "Deep and Confident Prediction for Time Series at Uber". In: *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*. IEEE, pp. 103–110.